Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Инженерно-экономический факультет

Кафедра экономической информатики

Дисциплина Программирование сетевых приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

на тему

**РАЗРАБОТКА АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ РЕАЛИЗАЦИИ АВИАБИЛЕТОВ**

Студент Ващабрович А.Ю.

гр. 472301

Руководитель ассистент каф. ЭИ Салапура М.Н.

Минск 2016

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc469631554)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc469631555)

[1.1 Тенденции мирового развития авиаперевозок 6](#_Toc469631556)

[1.2 Объект исследования 7](#_Toc469631557)

[2 ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЁ РЕШЕНИЯ 8](#_Toc469631558)

[3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ СТАНДАРТА IDEF0 10](#_Toc469631559)

[4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЁ ОПИСАНИЕ 14](#_Toc469631560)

[5 ОБОСНОВАНИЕ ОРИГИНАЛЬНЫХ РЕШЕНИЙ ПО ИСПОЛЬЗОВАНИЮ ТЕХНИЧЕСКИХ И ПРОГРАММНЫХ СРЕДСТВ, НЕ ВКЛЮЧЕННЫХ В ТРЕБОВАНИЯ 17](#_Toc469631561)

[6 ОПИСАНИЕ АЛГОРИТМОВ РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ ПРОЕКТИРУЕМОЙ СИСТЕМЫ 18](#_Toc469631562)

[6.1 Схема работы всей программы сервера 18](#_Toc469631563)

[6.2 Алгоритм покупки билетов с сохранением в БД 19](#_Toc469631564)

[7 СПЕЦИФИКАЦИЯ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ СИСТЕМЫ 20](#_Toc469631565)

[8 МОДЕЛИ ПРЕДСТАВЛЕНИЯ СИСТЕМЫ НА ОСНОВЕ UML 22](#_Toc469631566)

[8.1 Диаграмма последовательности процесса покупки билетов 22](#_Toc469631567)

[8.2 Диаграмма состояний просмотра статистики 23](#_Toc469631568)

[8.3 Диаграмма компонентов 23](#_Toc469631569)

[8.4 Диаграмма развертывания 24](#_Toc469631570)

[8.5 Диаграмма классов 25](#_Toc469631571)

[9 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 26](#_Toc469631572)

[10 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ 36](#_Toc469631573)

[ЗАКЛЮЧЕНИЕ 38](#_Toc469631574)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 39](#_Toc469631575)

[ПРИЛОЖЕНИЕ А 40](#_Toc469631576)

[Модели представления системы реализации билетов 40](#_Toc469631578)

[ПРИЛОЖЕНИЕ Б 47](#_Toc469631579)

[Блок-схемы методов, реализующих бизнес-логику 47](#_Toc469631581)

[ПРИЛОЖЕНИЕ В 49](#_Toc469631582)

[Листинг программы 49](#_Toc469631584)

# ВВЕДЕНИЕ

В современных рыночных условиях одной из основных задач, которые ставят перед собой предприятия, является эффективное ведение своей деятельности.

Для более удобного, эффективного и современного решения данной задачи на помощь приходят автоматизированные системы, в данном случае электронный документооборот.

Данная автоматизированная система предназначена для комплексной автоматизации бизнес-процессов продажи авиабилетов с помощью приложения, которое бронирует их через web – системы.

Внедрение системы позволяет:

* повысить эффективность и снизить трудоемкость продажи авиабилетов;
* повысить прозрачность процесса продаж для управляющего персонала компании.
* повысить качество оказываемых услуг;
* увеличить охват клиентов компании – пассажиров;

Данная автоматизированная система предоставляет возможность интеграции с другими системами, в частности – с системами электронной коммерции, с системами электронных платежей.

# 1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

# 1.1 Тенденции мирового развития авиаперевозок

Существующее положение и будущее мирового рынка авиаперевозок связано с формированием альянсов авиакомпаний, опирающихся на круг аэропортов, на основе которых строится сеть перевозок и которые обеспечат рост перевозок, базирующимся в них авиакомпаниям - членам альянса. Конкуренция на мировом рынке авиаперевозок - это борьба альянсов авиакомпаний и крупных узловых аэропортов за захват возможно большей части потребительского рынка. В мире сейчас налицо две тенденции развития авиаперевозок:

1) система point-to-point, которую можно увидеть на примере США, предполагает выполнение прямых перелетов между пунктом вылета и пунктом назначения;

2) европейским странам ближе схема hub-and-spoke (дословно - узел и спица), которая заключается в том, что пассажир летит в пункт назначения не напрямую, а с пересадкой в узловом аэропорту. При этом расписание должно быть составлено так, чтобы пассажир мог как можно быстрее пересесть на состыкованный рейс.

То есть аэропорты-хабы собирают пассажиропотоки из большого числа городов и перераспределяют их на состыкованные рейсы. В аэропорту-хабе также должна быть отработана специальная технология обслуживания пассажиров и обработки багажа, обеспечивающая как можно более высокий уровень обслуживания пассажиров.

Эти тенденции накладывают свой отпечаток на формирование политики ведущих авиастроительных корпораций: американской компании "Boeing" и западноевропейского авиастроительного альянса "Airbus". Американцы исходят из того, что будущее за скоростными, высокоэкономичными лайнерами средней пассажировместимости, позволяющими организовать беспосадочное воздушное сообщение между множеством городов мира, минуя крупные аэропорты, так как часто они перегружены. Европейцы делают ставку на суперлайнер А-380, перевозящий с высочайшим уровнем комфорта более полутысячи пассажиров за рейс. Принимать такие воздушные суда смогут лишь крупнейшие аэропорты мира, откуда пассажирам предстоит добираться до конечного пункта путешествия на самолетах меньшей вместимости.

Исходя из этих предпосылок ведущие самолетостроительные компании формируют линейки воздушных судов (ВС). Так, компания "Boeing" заканчивает летные испытания и готовит к серийному производству модель 787, способную перевозить от 200 до 300 человек на рекордные 16 тыс.км. А "Airbus" показал авиакомпаниям сверхвместимый двухпалубный А380, способный перевозить до 550 человек. При этом европейский производитель не сбрасывает со счетов спрос на сверхдальние самолеты, поэтому в пику Boeing и его 787 модели создает конкурирующий самолет А350. Ожидать, что лайнеры средней пассажировместимости (по американской модели) будут летать в каждый аэропорт нельзя, исходя из возможностей аэропортов и наличия пассажиропотока. Скорее всего, такие самолеты смогут доставлять пассажиров в региональные аэропорты, из которых пассажиры в зависимости от расстояния будут лететь до конечного пункта самолетами местных авиалиний или доставляться автотранспортом (в условиях РФ при наличии дорог).

Таким образом, в обоих направлениях развития сохраняется потребность в узловых аэропортах, емкости и характеристики которых зависят от пассажиропотока и типа используемой авиационной техники. Не стоит забывать и о том, что и при наличии американской модели в самих США успешно функционируют крупнейшие в мире узловые аэропорты-хабы, например такие как Атланта, Чикаго, Лос-Анжелес, Ньюарк, Хьюстон и другие с пассажиропотоком в несколько десятков млн. человек в год каждый.

# 1.2 Объект исследования

Объектом исследования является деятельность авиа-агентства. Агентство предоставляет услуги по заказу билетов на авиарейсы различных авиакомпаний. Каждый рейс следует из пункта отправления в пункт назначения. Рейс имеет дату и время вылета, дату и время прибытия. Каждый рейс выполняется самолетом определенной модели, в салоне которого есть места первого, бизнес и эконом класса. В зависимости от класса, билет имеет разную цену.

Для каждой модели самолета имеется определенное число мест каждого класса. Каждая модель самолета характеризуется авиакомпанией-производителем.

Ставится задача разработки многопользовательской системы, предназначенной для поиска и заказа билетов на авиарейсы. Каждому пользователю должна предоставляться возможность найти интересующие его рейсы, получить информацию о времени вылета и прибытия, авиакомпании, обслуживающей данный рейс, а также сделать заказ определенного количества билетов на выбранный рейс. Так же система должна предоставлять администратору системы WEB-интерфейс для её сопровождения [1].

# 2 ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЁ РЕШЕНИЯ

Определив проблему, которая, в свою очередь, заключается в потребности автоматизации процесса реализации авиабилетов, сформируем главную задачу, которую должно решить разрабатываемое приложение. Она состоит в том, чтобы на выходе получилось приложение с удобным пользовательским интерфейсом, простое в использовании и обладающее следующими характеристиками:

* обладающее возможностью хранения и обработки данных;
* работоспособный, эффективный, надежный и отказоустойчивый программный продукт;
* совместимое с любой операционной системой;
* построенное на архитектуре клиент-сервер;
* выступающее не только средством покупки или продажи авиабилетов, но и информационным ресурсом, то есть предоставляющее актуальные сведения о дате и времени рейсов, находящиеся в данное время в зарегистрированными авиакомпанией;
* содержащее в себе аналитические данные.

Для достижения поставленной цели разработано приложение, написанное на языке программирования Java, так как он является кросплатформенным, соответственно совместим с любой операционной системой. Программа представляет собой клиент-серверное приложения, состоящего из 3 частей: консольного сервера, мобильного клиента на платформе Android версии 4.0.3 и выше и десктопного клиента-администратора на Swing. Взаимодействие между пользователем и программным приложением осуществляется с помощью пользовательского графического интерфейса. Был выбран язык программирования Java. Для хранения информации была выбрана реляционная база данных MySQL. Для сборки проекта используется сборщик Java-проектов Maven.

Так как правильное проектирование системы играет наиболее важную роль при создании программного продукта, то на практике были применены такие паттерны или шаблоны проектирования как Model-View-Controller (MVC), Наблюдатель (Observer), Команда и Фабрика. Последний является порождающим, так как он представляет собой интерфейс для создания компонентов системы. Паттерн Наблюдатель (Observer) служит для осведомления всех частей программы (всех Наблюдателей), об изменениях наблюдаемого объекта (Observable). Паттерн команда служит для унифицирования и абстрагирования различного рода действий, с общими чертами, но различной составляющей. Шаблон MVC, который переводится как Модель-Представление-Контроллер относится к так называемым архитектурным шаблонам, с помощью которого модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные [2].

# 3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ СТАНДАРТА IDEF0

Построение модели информационной системы начинается с описания функционирования системы в целом в виде контекстной диаграммы. На верхнем уровне произведем описание системы в целом и её взаимодействие с окружающим миром. На рисунке 3.1 изображена контекстная диаграмма, отображающая основной процесс системы – продажу авиабилетов билетов.

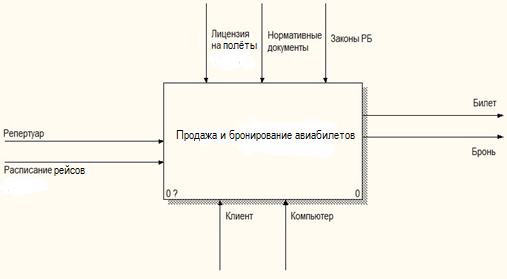


Рисунок 3.1 – Контекстная диаграмма верхнего уровня

Взаимодействие системы с окружающей средой описывается с помощью входов «Репертуар» и «Расписание рейсов»), выходов («Билет» и «Бронь»), управления («Лицензия на полёты», «Нормативные документы» и «Законы РБ») и механизмов («Клиент» и «Компьютер»).

Клиенты в данном случае – это люди, создающие спрос на услуги авиакомпании, а репертуар – выбор различного рода самолётов или иной техники авиакомпании, расписание рейсов – список всех доступных рейсов.

Компьютер – это некая программа на компьютере или мобильном устройстве, с помощью которой пользователь может покупать либо бронировать билеты.

Законы РБ – законы по защите прав потребителя, и различные нормы на осуществление коммерческой деятельности.

Билет – это право клиента на полет на конкретном рейсе.

Бронь – закрепление места на рейсе за клиентом.

После описания контекстной диаграммы переходим к процессу функциональной декомпозиции, т.е. разбиваем систему на подсистемы до степени, достаточной для понимания роли проектируемого ПО и написания спецификаций процессов (смотреть рисунок 3.2).

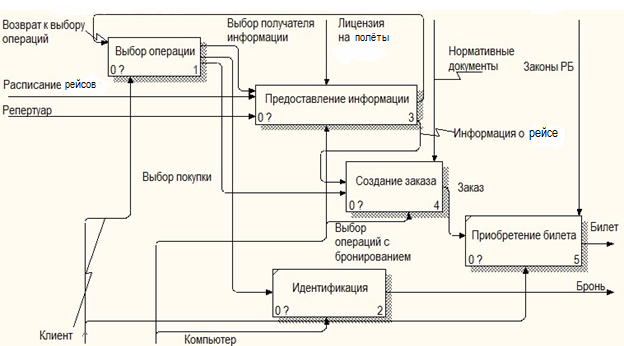


Рисунок 3.2 – Декомпозиция контекстной диаграммы

Как видно из диаграммы, весь процесс функционирования авиакомпании разбивается на шесть блоков:

выбор операции, т. е. ветвление, позволяющее клиенту выбрать интересующую операцию с системой авиакомпании;

предоставление информации, т. е. предоставление пользователю всей доступной информации о расписании и рейсов;

создание заказа, т. е. сведение всех требований клиента в один заказ;

приобретение билета, т. е. совершение операции купли-продажи между клиентом и кассиром и закрепления за клиентом билета;

идентификация, т. е. подтверждение компьютером личности клиента и его права на осуществление постановки или снятия брони.

Произведем дальнейшее разбиение на подсистемы. Декомпозиция блока «Выбор операции» может выглядеть следующим образом (рис. 3.3):

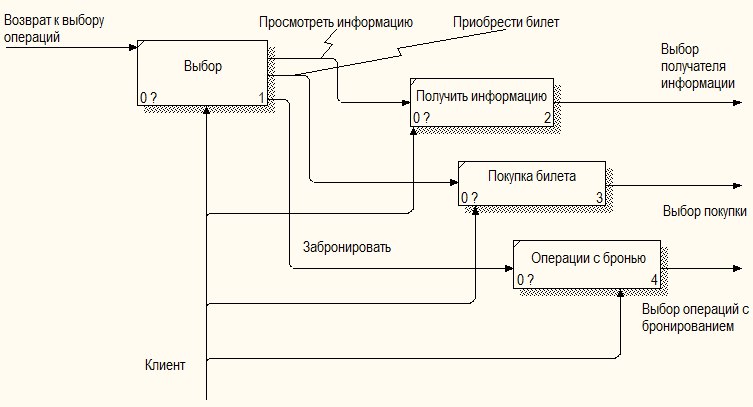


Рисунок 3.3 – Декомпозиция блока «Выбор операции»

Опишем процессы, представленные на данной диаграмме декомпозиции.

Выбор – акт в котором пользователь выбирает то, что он хочет.

Получение информации – принятие клиентом решения получить информацию.

Покупка билета – принятие клиентом решения приобрести билет.

Операции с бронью – принятие клиентом решения осуществить операцию с бронью, то есть забронировать его с последующим выкупом.

После того, как клиент сделал свой выбор, просмотрев всю действующую информацию, ему необходимо создать заказ. Диаграмма, отражающая процесс создания заказа приведена на рисунке 3.4.

Первым этапом клиент должен заполнить форму, в которой должен указать требуемый рейс из расписания рейсов и требуемое место в зале. Клиенту к этому моменту уже известна стоимость билета – она входит в информацию о рейсе. Далее, если клиент согласен с введенной информацией, он должен подтвердить заказ. Система сформирует заказ в виде, принятом в нормах авиакомпании.

Требования Клиента – набор, заполненных клиентом данных о рейсе которые однозначно его (рейс) позиционирует.

Генерирование заказа – система формирует заказ исходя из требований клиента и норм предприятия.

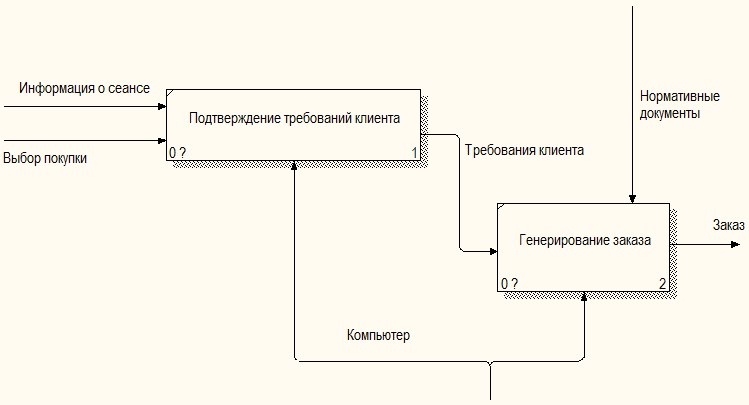


Рисунок 3.4 – Декомпозиция блока «Создание заказа»

На выходе получается сформированный системой в электронном виде билет, который можно как распечатать самостоятельно, так и в кассах аэропорта.

Таким образом, были рассмотрены основные бизнес-процессы, которые происходят при бронировании или покупке клиентом авиабилета. При этом заранее забронированный билет можно сразу оплатить, либо в кассе аэропорта перед вылетом.

# 4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЁ ОПИСАНИЕ

Первым шагом при проектировании информационной модели базы данных (БД) является нормализация. Нормализация данных — одно из самых важных понятий и концепций реляционной системы. Нормализованная система сводит к минимуму количество избыточных данных, при этом сохраняя их целостность. Нормализованной можно назвать базу данных, в которой все таблицы следуют правилам нормальных форм. Нормальная форма — набор правил, которые показывают, как надо организовать данные, чтобы они были нормализованными. Конечной целью нормализации является уменьшение потенциальной противоречивости, хранимой в базе данных информации.

Для проектирования БД необходимо, чтобы она соответствовала первым трем нормальным формам. Первая нормальная форма (1НФ) требует, чтобы каждое поле таблицы: было неделимым; не содержало повторяющихся групп. Вторая нормальная форма (2НФ) требует соответствия 1НФ, и чтобы все поля зависели от первичного ключа, т.е. первичный ключ должен однозначно определять запись и не быть избыточным. Те поля, которые зависят только от части первичного ключа, должны быть выделены в составе отдельных таблиц. Третья нормальная форма (3НФ) требует, соответствия 2НФ и отсутствия в таблице транзитивных зависимостей между неключевыми полями, то есть, чтобы значение любого поля таблицы, не входящего в первичный ключ, не зависело от значения другого поля, не входящего в первичный ключ [3].

В ходе нормализации были выделены следующие сущности:

* Координаты;
* Рейс;
* Место;
* Пользователь;
* Билет.

Модель базы данных представлена на рисунке 4.1 ниже. Каждая сущность имеет набор атрибутов, а также обязательный первичный ключ. Рассмотрим подробнее каждую сущность.

Сущность Пользователь содержит уникальное поле ID, логин и пароль пользователя, зарегистрированного в системе, а также атрибут статус, который идентифицирует его как администратора, модератора, зарегистрированного, но ещё непрошедшего проверку подлинности, модератора, либо как заурядного клиента. Связана неидентифицирующей связью с сущностью Билет по полю логин, для того, чтобы каждый билет хранил в себе информацию о том, кто его приобрел.

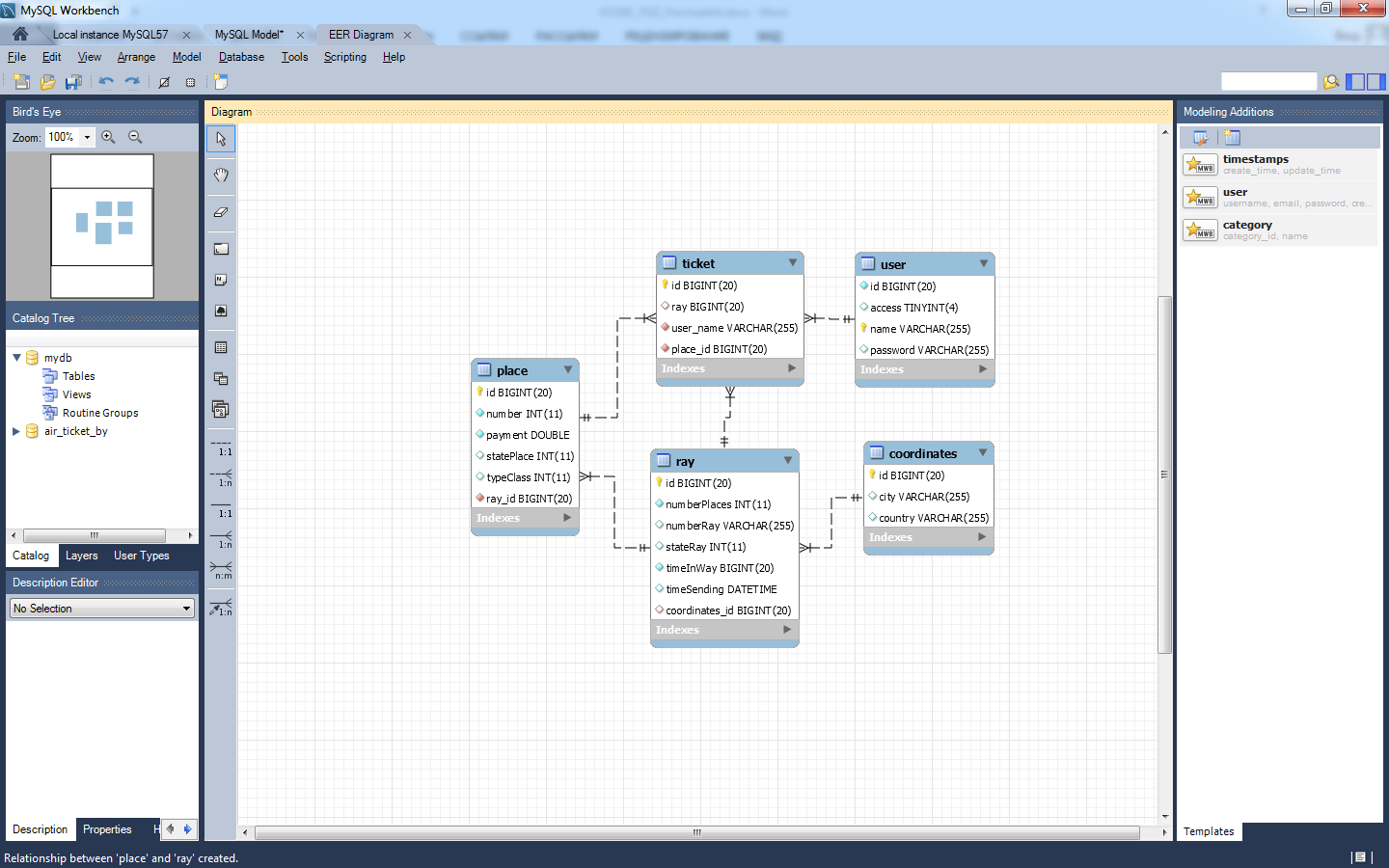


Рисунок 4.1 – Модель базы данных

Следующая сущность Координаты содержит уникальное поле ID, а также обязательное ненулевое поле, такое как название города и страны. Данная таблица содержит набор записей, то есть координат, на основе которых формируется репертуар авиакомпании и соответствующие им рейсы.

Что касается сущности Рейс, то она содержит уникальное поле ID, а так же набор атрибутов, таких как номер рейса, места рейса и их кол-во, состояние рейса, дата отправления и продолжительность в пути. Эта таблица имеет 3 неидентифицирующие связи с сущностями Место, Билет и Координаты. Так как у одних координат может быть множество рейсов, то это связь многие к одному, у одного рейса имеется много билетов, значит это связь один ко многим. И оставшаяся связь один ко многим относится к сущности Место, так как один рейс включает в себя набор мест. Поле состояние рейса может принимать такие значения, как «Доступен для покупки билетов», «Готовиться к отправлению», «Отправлен», «Завершён», «Отменён».

Таблица Билет несет в себе следующие атрибуты: уникальное поле ID, внешние ключи по связям с другими таблицами Рейс, Пользователь, Место: ID рейса, логин пользователя и номер места соответственно.

И оставшаяся сущность Место хранит первичный ключ ID места и поля: стоимость, зависящая от типа места на рейсе, которое может принимать значения «Первый класс», «Бизнес класс», «Эконом класс», статус, принимающий значения «Свободно», «Куплено» и «Забронировано», если место доступно, оплачено или на него оформлена бронь соответственно и номер места на рейсе. ID рейса является внешним ключом по связи многие к одному с таблицей Рейс.

Все таблицы приведены к 1НФ, так как не содержат повторяющихся групп и их поля являются неделимыми. Так как у каждой таблицы присутствует уникальное поле ID, которое не является избыточным и полностью идентифицируют каждую запись в таблице, можно сделать вывод, что все таблицы также находятся во второй нормальной форме.

Для доказательства приведения к 3НФ разберем подробно каждую сущность.

Сущности пользователь и координаты содержат лишь зависящие от первичного ключа атрибуты и не содержат неуникальных полей, зависящих друг от друга. Следовательно, они находятся в 3НФ.

Таблица Билет также находится в 3НФ, потому что не имеет ни одно неуникального поля, которое не зависит ни от чего, а все поля зависят от других сущностей.

У сущности Место все атрибуты абсолютно самостоятельны. Например, стоимость, которая регулируется администрацией и зависит только от расположения на рейсе и категории самого места. Статус также независим, а номер места отражает только порядковую характеристику. Значит, данная сущность в 3НФ базы данных.

И, наконец, Рейс, включающий в себя дату отправления, которая не зависит от поля продолжительность в пути, что верно и в обратную сторону. Можно прийти к выводу, что и эта таблица находится в 3НФ.

Таким образом, база данных является правильно спроектированной, приведенной к 3НФ и имеет четко сформированную структуру.

# 5 ОБОСНОВАНИЕ ОРИГИНАЛЬНЫХ РЕШЕНИЙ ПО ИСПОЛЬЗОВАНИЮ ТЕХНИЧЕСКИХ И ПРОГРАММНЫХ СРЕДСТВ, НЕ ВКЛЮЧЕННЫХ В ТРЕБОВАНИЯ

Оригинальным решением в данной задании стало создание приложений, взаимодействующих по принципу «Клиент - Сервер», где клиент представляет из себя 2 различных приложения: Мобильное приложение пользователя написанное под систему Android версии 4.0.3 и выше и десктопное приложение написанное на Swing для клиента-администратора (модератора). Это было сделано для того чтобы пользователю не приходилось заходить в систему авиакомпании через веб-браузер или иные компьютер, достаточно загрузить только мобильное приложение себе в смартфон и можно пользоваться, помимо основного функционала по работе с билетами, там представлены также дополнительные настройки приложения в разделе “Настройки”. Что касается клиента-администратора, то решение реализовать его возможности в отдельном приложении, было принято на основании того, что ПО данного типа должно распространяться и устанавливаться в приделах самой компании, которая занимается реализацией авиабилетов, и чтобы администраторам не приходилось выполнять свои обязанности с мобильного устройства, было реализовано десктопное приложение, с удобным графическим интерфейсом.

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

# 6.1 Схема работы всей программы сервера

Данная схема представляет собой алгоритм работы всей программы от начала и до ее завершения. Она представлена на рисунке Б.1 приложения Б. Для начала номер порта достаётся из property файла environment.xml, по которому сервер будет принимать запросы на соединение. Порт идентифицирует программу в сети. Адрес же идентифицирует компьютер или другое устройство в сети. После ввода порта и проверки его корректности выводится уведомление, что сервер запущен на соответствующем порту и готов принимать запросы. Для того чтобы сервер мог это делать нужно создать объект серверного сокета на основе соответствующего порта. Далее запускается главный цикл, который будет работать до тех пор, пока пользователь не завершит программу.

Для инициирования соединения с клиентом у объекта созданного ранее сокета вызывается метод accept(), который возвращает ссылку на созданный между клиентом и сервером канал передачи данных, то есть клиентский сокет. Далее по нему данные программы смогут обмениваться сообщениями и пересылать различные данные. Так как метод может бросать IOException, то есть исключения ввода/вывода, необходимо обернуть вызов такого метода в try-catch блок, и в случае ошибки управление перейдет в блок catch, где оно будет обработано и выполнится соответствующий блок кода. В данном случае закроется текущий серверный сокет, выведется сообщение и главная нить программы будет ожидать следующего подключения клиента [4].

Если ошибки не произошло, будет создана новая нить программы, выполнится метод run() вложенного класса Handler, в котором будут обслуживаться все последующие запросы клиента.

Каждое выполнение цикла влечет за собой выделение нового потока для работы с новым клиентом. Таких нитей может быть неограниченное количество.

После того, как пользователь решит завершить программу, основной цикл прекратит работу и выполнение метода main() будет закончено. На этом описание данного алгоритма окончено.

# 6.2 Алгоритм покупки билетов с сохранением в БД

Начало алгоритма, представленного на рисунке Б.2 приложения Б, происходит с того, что в функцию приходит id рейса, забронированные места для пользователя, с которым происходит взаимодействие, должны быть сохранены в БД, как купленные. Получив все записи рейсов из БД необходимо отыскать тот единственный рейс, который выбрал пользователь. Следующим действием нужно отыскать данный рейс в базе данных, и, если он найден, получить список всех доступных в нем мест.

Все эти манипуляции происходят в синхронизированном блоке по объекту замку, что означает, что любой, кто в данный момент попытается купить билеты, будет ожидать, пока не отработает данный блок кода. Это делается для того, чтобы избежать некорректной обработки данных, если клиент попытается произвести какие-либо действия с местами, которые обрабатываются в данный момент [5].

Следующим действием необходимо отфильтровать по циклу неинтересующие места, то есть все, кроме тех, что указал пользователь для покупки. Для каждого из них устанавливает статус как «Куплено». Далее ID выбранного рейса и данный рейс передаются методу updateById() класса RayDAO, имеющему доступ к БД, который, в свою очередь, обновит у соответствующего сеанса все выбранные места.

Если метод отработает корректно и все изменения успешно сохранятся, произведем запрос в базу данных на обновленный список рейсов. Далее добавим в таблицу Билет записи, соответствующие каждому купленному месту и получим новый список билетов. Теперь можно отправить всем клиентам широковещательное сообщение, в котором содержатся новые данные. Клиентская часть отобразит полученные результаты на экран, уведомив клиента об успешной покупке.

В случае, если сеанс, выбранный пользователем не будет найден, метод завершит свою работу. Листинг кода данного алгоритма приведен в обязательном приложении В.

# СПЕЦИФИКАЦИЯ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ СИСТЕМЫ

Диаграмма вариантов использования системы реализации авиабилетов представлена на рисунке 7.1.

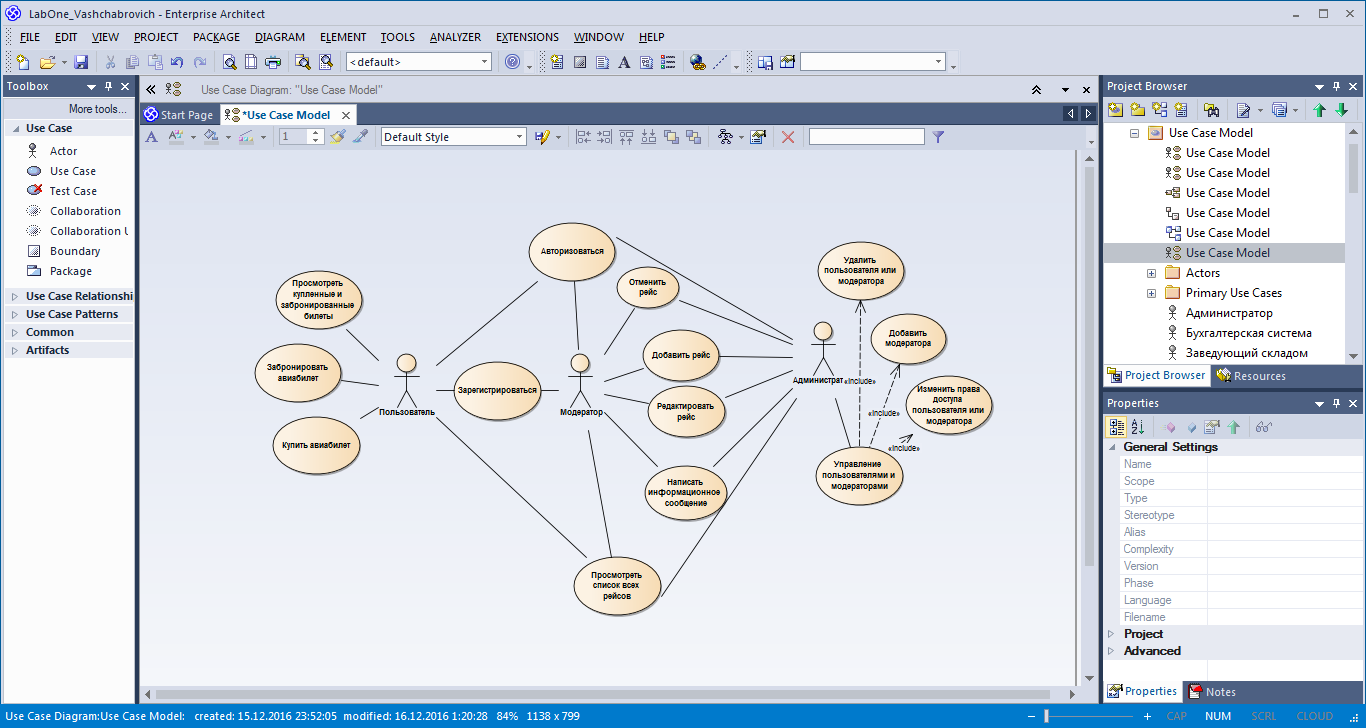


Рисунок 7.1 – Диаграмма вариантов использования

Для организации работы данной системы необходимы Администратор и Пользователь, так же может присутствовать модератор, который должен зарегистрироваться и после подтверждения прав доступа со стороны администратора, может приступать к своим обязанностям.

В основные функции Администратора/Модератора входит не только контроль за корректной работой системы, но и такие функции как:

* 1. добавление, отмена, редактирование рейсов;
  2. управление пользователями и модераторами;
  3. просмотр статистической и аналитической информации;

В обязанности администратора/модератора входит контроль за репертуаром авиакомпании, что означает регулирование находящихся доступных координат полётов рейсов. Что касается управления пользователями, то данная функция позволяет лицу, осуществляющему администрирование системы, выполнять контроль над зарегистрированными пользователями и модераторами, а именно их добавление или удаление, а также изменение роли доступа [6].

К универсальным функциям пользователя относятся покупка либо бронирование мест на рейс и регистрация.

Для того, чтобы не было ситуации, когда один человек забронирует все места на рейс, в системе предусмотрена проверка на это. То есть за 12 часов до любого рейса служебная нить программы проверит, все ли места выкуплены, и если нет, то снимет с них бронь, дабы другие пользователи получили возможность купить данные места.

Так как администратор существует в системе с момента ее создания, ему нет необходимости регистрироваться, в отличие от пользователя, который для того, чтобы пользоваться возможностями системы и для подтверждения соблюдения правил системы обязан зарегистрироваться.

Что касается модераторов, то им необходимо регистрироваться в системе, и только после подтверждения запроса на регистрацию модератора администратором, первый в свою очередь может приступить к обязанностям, а до этого момента, с момента регистрации модератора, тот считается простым пользователем, но с запросом посланным администратору.

Также в системе реализованы общие как для пользователя, так и для администратора функции:

* авторизация;
* просмотр рейсов;

# МОДЕЛИ ПРЕДСТАВЛЕНИЯ СИСТЕМЫ НА ОСНОВЕ UML

В данном курсовом проекте рассматриваются некоторые модели представления разрабатываемой системы, а именно:

* диаграмма последовательности;
* диаграмма состояния объектов;
* диаграмма компонентов;
* диаграмма развертывания;
* диаграмма классов;

# 8.1 Диаграмма последовательности процесса покупки билетов

Данная диаграмма отображена на рисунке А.1 Приложения А.

Главным окном, в котором происходит выбор рейса, и в последующем выбор мест с отправкой на обработку их покупки, является класс MainActivity и соответствующее ему представление activity\_main.xml, в который помещаются фрэймы MainWindowFrame, в котором представлен список рейсов и ItemListRaysFragment, в котором непосредственно происходит выбор мест для оформления брони или приобретения всех уже забронированных мест. Данный класс отображает пользовательский интерфейс для клиента Android. После нажатия клиентом «Купить» вызывается обработчик нажатия, который отправляет все предварительно считанные данные на сервер.

После отправления необходимых данных серверу, их обработка происходит в классе ConnectionAndroid сервера, который и формирует запрос в базу данных на сохранение с последующей отправкой ответа об успешном сохранении клиенту на мобильное устройство. В мобильном приложении отслеживанием поступающих данных занимается класс BackgroundAsyncTask, это специализированный класс, который является наследником AsyncTask класса определённом в библиотеках Android. Он служит для реализации работы в фоновом режиме, то есть создаётся нить, которая отделена от основной и следит за поступающими данными не воздействуя на Пользовательский интерфейс. После получения данных этим классом, он извещает пользовательскую нить (UI-thread), о пришедших данных и вызывает их обновление. Забронированные места выделяются галочкой, то есть checkbox становиться выбранным, а статус места изменяется на «Забронировано», только в случае если они забронированы этим пользователем, в противном случае и в случае покупки выбранных мест, их статус меняется на «Продано» и checkbox исчезает, без возможности повторно выбрать место. Модель обновляет представление, где пользователю выводятся уведомления о только что совершенной покупке и отображаются все изменения в сеансах, то есть пропадают места, приобретенные пользователем.

# 8.2 Диаграмма состояний просмотра статистики

Диаграмма состояний изображена на рисунке А.2 Приложения А.

На вход приходит объект статистики билетов пользователя, который следует отобразить пользователю в виде списка, на основе полученных от сервера актуальных данных. Для этого необходимо запросить данные у сервера, переходим в состояние «Запрос отправлен», далее, когда сервер отправил всю расчётную информацию, перемещаемся в состояние «Данные получены».

После получения данных необходимо «Создать список», если же данных нет, то список окажется пуст.

Теперь мы находимся в состоянии «График построен». Совершив действие «Отобразить статистику», перемещаемся в следующее, конечное состояние под названием «Статистика отображена» и завершаем описание диаграммы, так как теперь мы оказались в финальной точке.

# 8.3 Диаграмма компонентов

Данная диаграмма представлена в трёх экземплярах: для сервера, для клиента пользователя (Android) и для клиента администратора/модератора (Swing). Ее изображение можно найти в приложении А на рисунках А.3, А.4 и A.5 соответственно. Так как все три части затрагивают одну и ту же предметную область, то и их реализации будут не во многом отличаться. Рассмотрим вначале их общие составляющие, затем отличия.

В всех трёх проектах существуют пакеты user, connection, ray, в которых присутствуют общие для них классы. Для осуществления сериализации необходимо полное соответствие классов, объекты которых будут пересылаться между клиентом и сервером, однако из-за невозможности этого сделать на клиенте Android, так как там назначается пакет по умолчанию в виде названия applicationId, который используется сборщиком проекта Gradle, то передача данных осуществляется JSON объектами, поэтому это даёт некую гибкость, так как уже нету строгой привязки к полной идентичности классов. Для трансформации классов в JSON и обратно используется библиотека Jackson версии 2.8.2.

Что касается отличий, то у каждого из проектов есть свои ключевые классы, в которых описаны основная логика их взаимодействия. Для сервера это пакет server с классами Server, а для клиента пакет client с классами ClientModel, ClientView, ClientController, реализующие паттерн MVC и собственная реализация класса Connection, причем на сервере представлены еще два класса, расширяющие его. Также на сервере существует свой пакет для взаимодействия с базой данных, носящий название dao и представляющий собой уровень ДАО, для взаимодействия с провайдером базы данных Hibernate. Он содержит все классы, необходимые для доступа в БД для всех типов данных. Так же имеется пакет db, в котором лежат классы классы помогающие в различных действиях с базой данных, реализующие паттерн Команда. На клиенте же присутствует пакет frames, состоящий из набора классов для пользовательского интерфейса. Что касается Android приложения то там помимо общих классов имеются классы активностей, основной из них это MainActivity, активность настроек PrefActivity, так же имеется класс для выполнения асинхронных задач в фоновом режиме BackgroundAsyncTask, остальные классы являются лишь фрэймами, частями которые встраиваются в основную активность.

# 8.4 Диаграмма развертывания

Эта диаграмма показывает все существующие аппаратные компоненты приложения, а именно сервер на одном персональном компьютере, клиент администратор/модератор на другом, а так же пользователь на мобильном устройстве, и какие программные компоненты работают на каждом узле, например, база данных – MySQL Server c базой данных air\_ticket\_by, и как различные части этого комплекса соединяются друг с другом, в данном случае через протокол TCP/IP. Запускаемыми модулями являются Course\_Project\_Admin.jar, CourseProject.jar и AirTicketBy.apk для клиента администратора/модератора, сервера и пользователя мобильного устройства соответственно. Данную диаграмму можно найти в приложении А на рисунке А.6 [7].

# 8.5 Диаграмма классов

В приложении А на рисунках А.7, А.8 и А.9 продемонстрированы диаграммы классов для проектов CourseProject, Course\_Project\_Admin и AirTicketBy соответственно.

Каждый из изображенных классов находится в каком-либо отношении с другим классом.

Как видно из диаграммы серверной части класс Handler является статически вложенным в класс Server. На клиенте же такого отношения придерживаются класс ClientModel и перечисление ConnectionState.

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для того чтобы запустить клиентскую программу, необходимо дважды нажать на файл Curse\_Project\_Admin.jar, а серверную программу следует запустить с помощью командной строки. Для этого необходимо запустить ее, проложить путь в папку, где находится файл CourseProject.jar, и написать следующую команду: java -jar CourseProject.jar. Теперь запустим клиент. На рисунке 9.1 изображено начальное окошко. Введем необходимые параметры запуска, а именно IP-адрес и персональный порт.

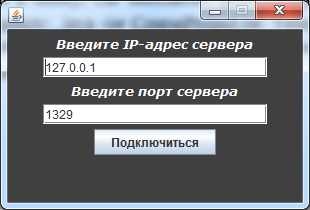


Рисунок 9.1 – Стартовое окно запуска

Если данные корректны, то после нажатия кнопки «Подключиться» откроется окно, изображенное на рисунке 9.2, где необходимо аутентифицироваться и авторизоваться чтобы войти в учётную запись администратора/модератора или же зарегистрироваться модератором, рисунок 9.3, а после регистрации будет получено уведомление, представленное на рисунке 9.4:

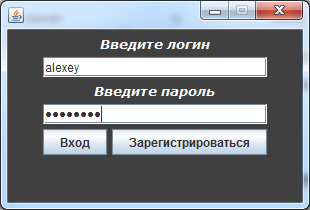


Рисунок 9.2 – Аутентификация и авторизация администратора/модератора

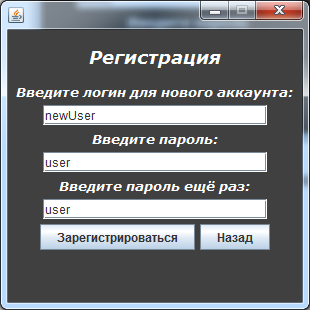


Рисунок 9.3 – Регистрация модератора в системе

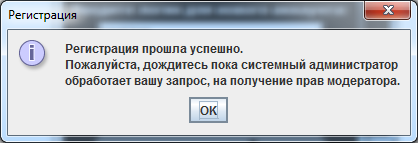


Рисунок 9.4 – Уведомление об успешной регистрации

При успешной авторизации выведется главное окно программы. Для администратора оно изображено на рисунке 9.5, для модератора на рисунке 9.6. Функционал для каждого из режимов частично отличается, рассмотрим подробнее некоторые функции.

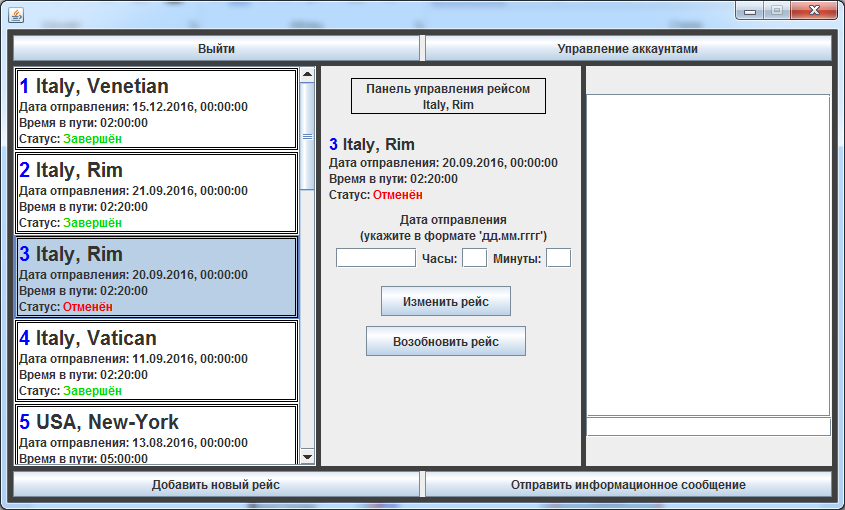


Рисунок 9.5 – Главное окно программы в режиме администратора

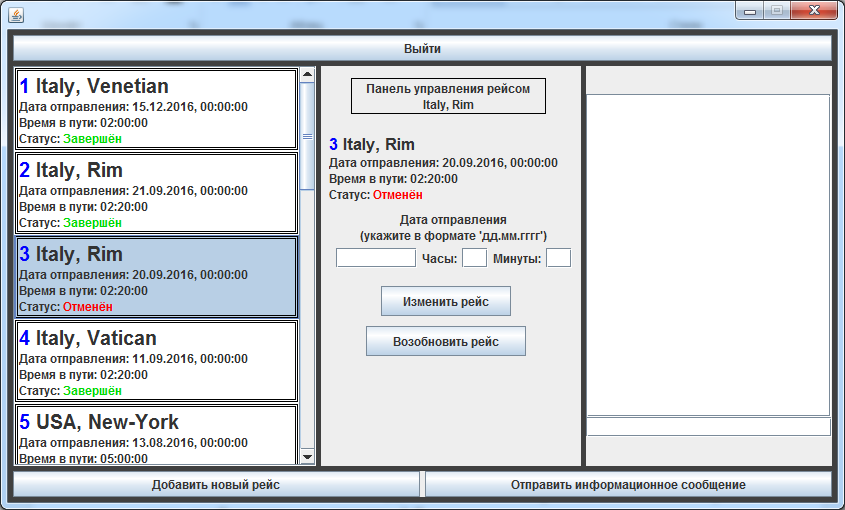


Рисунок 9.6 – Главное окно программы в режиме модератора

Слева в окнах располагается список доступных рейсов, и администратору и модератору могут управлять этим списком, а именно добавлять, редактировать и отменять рейсы. Посередине обоих окон находится окно в котором отображается информация по выбранному рейсу. Редактирование рейс можно в этом окне.

После нажатия кнопки «Добавить новый рейс» переходим в соответствующее окно, изображенное на рисунке 9.7.

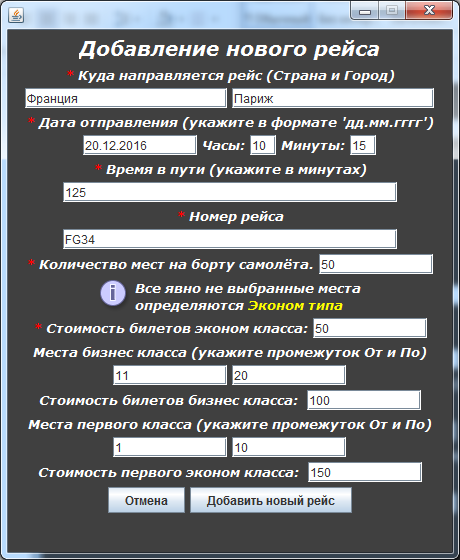


Рисунок 9.7 – Главное окно программы в режиме

Администратор/модератор заполняет форму добавления нового рейса. Система проверяет данные формы на корректность, и, если все верно, то по нажатию кнопки «Добавить новый рейс», система производит сохранение данной записи. На рисунке 9.8 изображен результат добавления нового рейса. Если нажать на данный рейс, посреди окна отобразится информация по рейсу. Что касается пользовательского мобильного приложения AirTicketBy.apk, то достаточно его установить на устройство под операционной системой Android версии 4.0.3 и выше. После входа в приложение, при успешном подключении к серверу будет отображено окно аутентификации, изображённое на рисунке 9.9, а на рисунке 9.10 представлено окно регистрации пользователя.

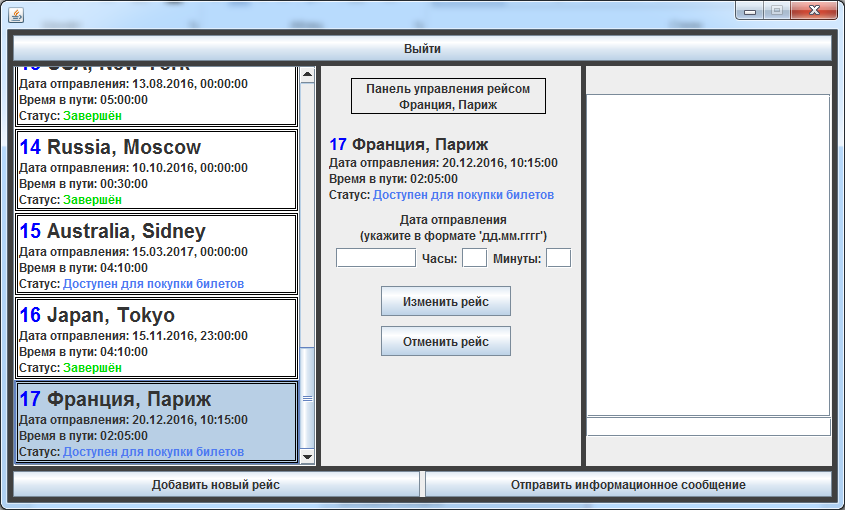


Рисунок 9.8 – Просмотр выбранного рейса



Рисунок 9.9 – Окно аутентификации пользователя мобильного приложения



Рисунок 9.10 – Окно регистрации пользователя мобильного приложения

После входа или регистрации пользователя моб. приложения перед пользователем появиться окно со списком рейсов представленного на рисунке 9.11, выбрав один из рейсов пользователь перейдёт в окно бронирования и покупки мест, представленное на рисунках 9.12, 9.13с забронированными местами и 9.15 с купленными местами, а на рисунке 9.14 показано окно покупки авиабилетов на выбранные места.

Для всех забронированных билетов есть возможность их выкупа не позднее, чем за 12 часов до отправления рейса.

Статистика купленных и забронированных авиабилетов пользователя моб. приложения отображается в окне приведённом на рисунке 9.16.

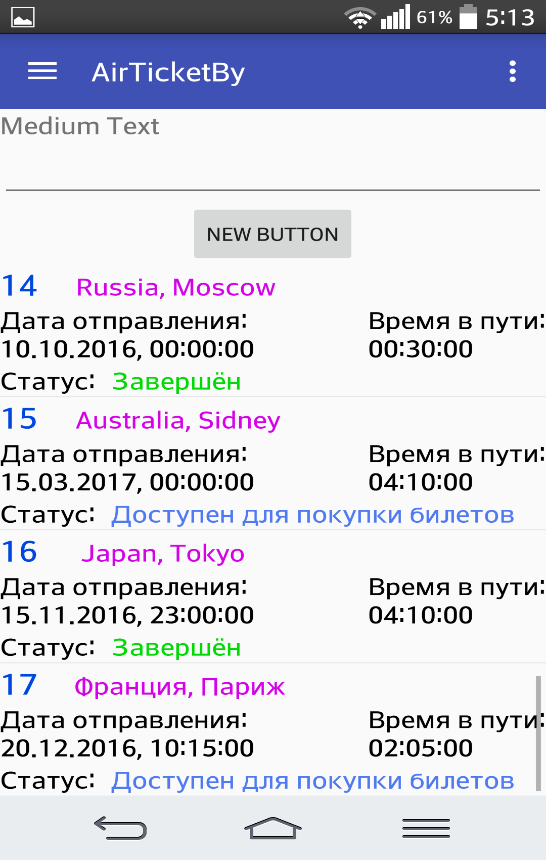


Рисунок 9.11 – Список рейсов пользовательского моб. Приложения

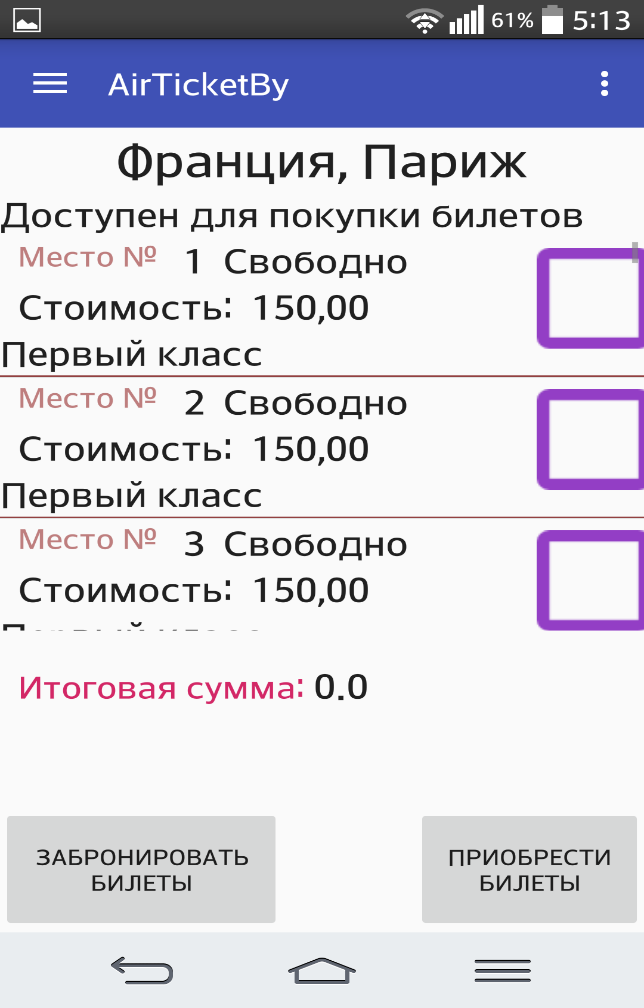


Рисунок 9.12 – Окно бронирования и покупки мест на рейс пользовательского моб. приложения

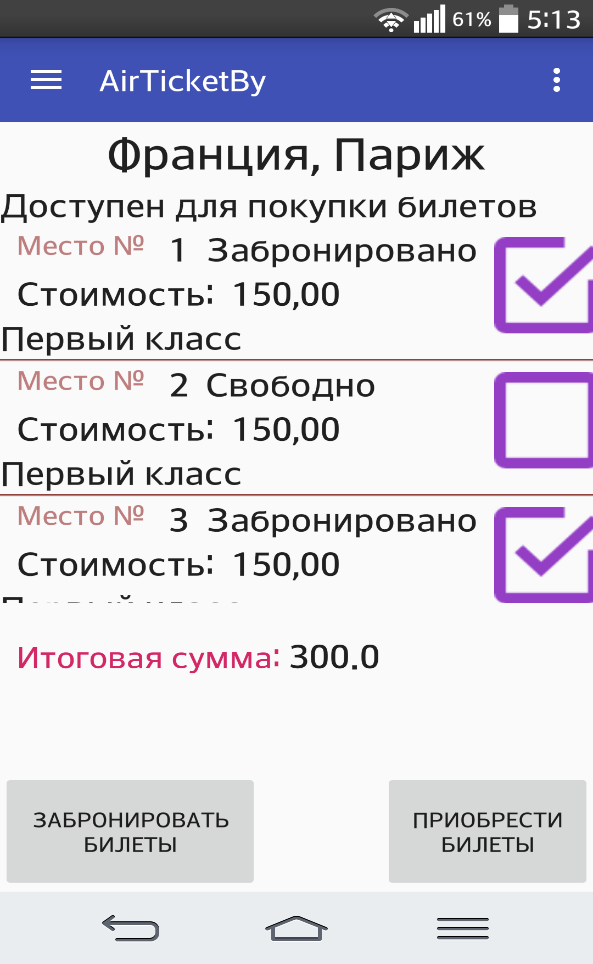


Рисунок 9.13 – Окно бронирования и покупки мест на рейс пользовательского моб. приложения с забронированными местами

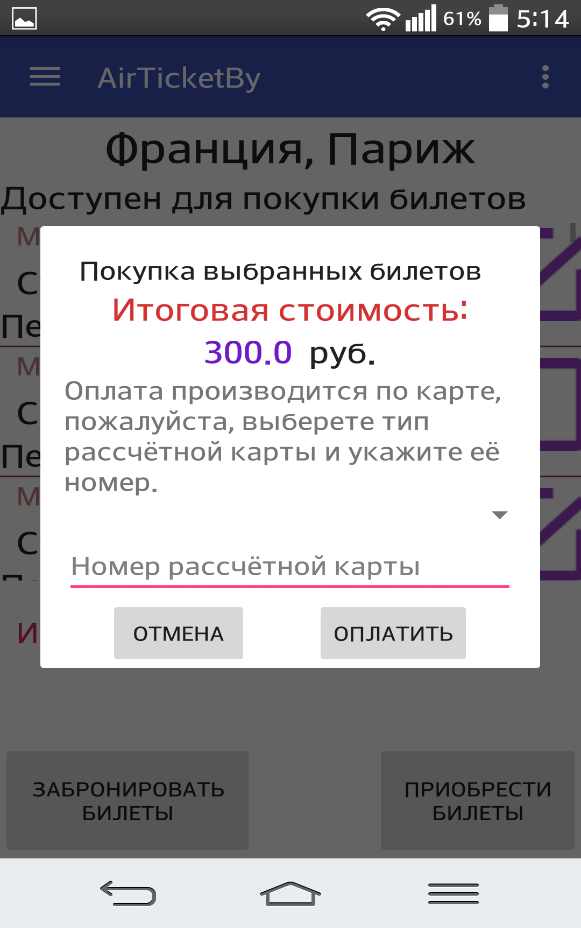


Рисунок 9.14 – Окно покупки авиабилетов выбранных мест на рейс пользовательского моб. приложения

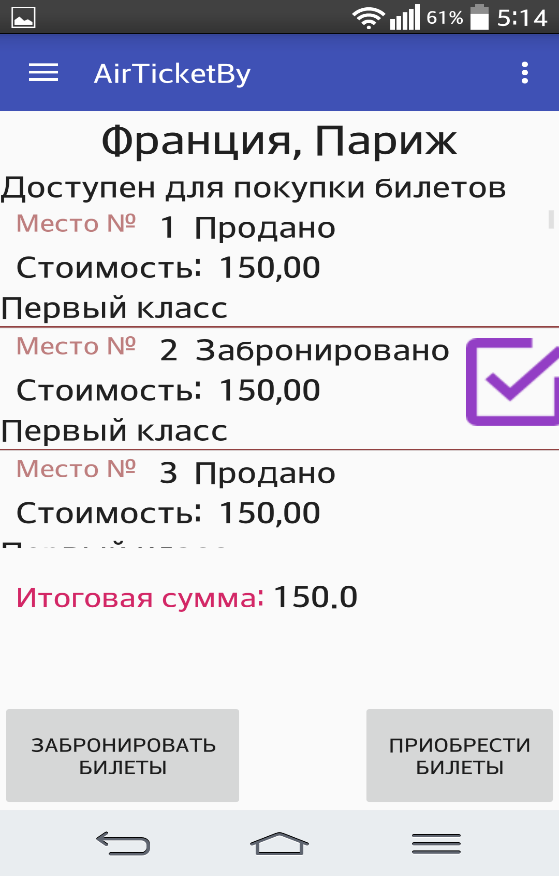


Рисунок 9.15 – Окно бронирования и покупки мест на рейс пользовательского моб. приложения с купленными местами



Рисунок 9.16 – Окно отображения статистики купленных и забронированных авиабилетов пользователя моб. приложения

Управление пользователями администраторской части выглядит следующим образом - рисунок 9.17. Чтобы изменить роль выбранного пользователя, достаточно просто выбрать необходимую роль из выпадающего списка и нажать кнопку «Внести изменения», а для удаления достаточно выбрать необходимый элемент и нажать кнопку «Удалить». Для отмены либо редактирования рейса необходимо прежде всего выделить нужный рейс и нажать соответствующие кнопки.

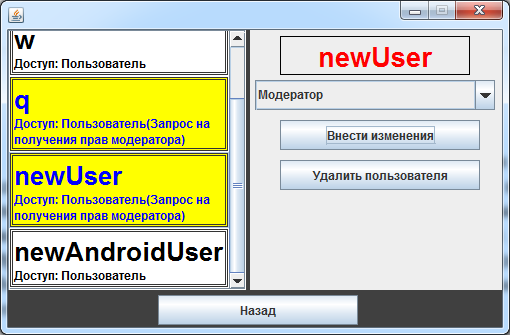


Рисунок 9.17 – Пункт управления пользователями

Если доступных билетов на какой-либо сеанс не осталось, то он не отображается в режиме пользователя, однако администратору этот сеанс по-прежнему будет доступен.

Для выхода из своего аккаунта пользователю достаточно нажать кнопку «выйти» управление вернется в самое начало работы, то есть в окно подключения к серверу.

В текущем разделе были показаны основные принципы работы со всеми компонентами системы, в следующем девятом разделе будут рассмотрены механизмы обработки ошибок, возникающих при работе с системой.

# РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ

В данном разделе будут продемонстрированы все механизмы для обработки различных исключительных ситуаций.

С запуском сервера свою работу начинает логгер фасат slf4j на основе logback и статический логгер, встроенный в ORM-фреймворк Hibernate. После подключения первого клиента он начинает свою работу. Пример присутствует на рисунке 10.1.

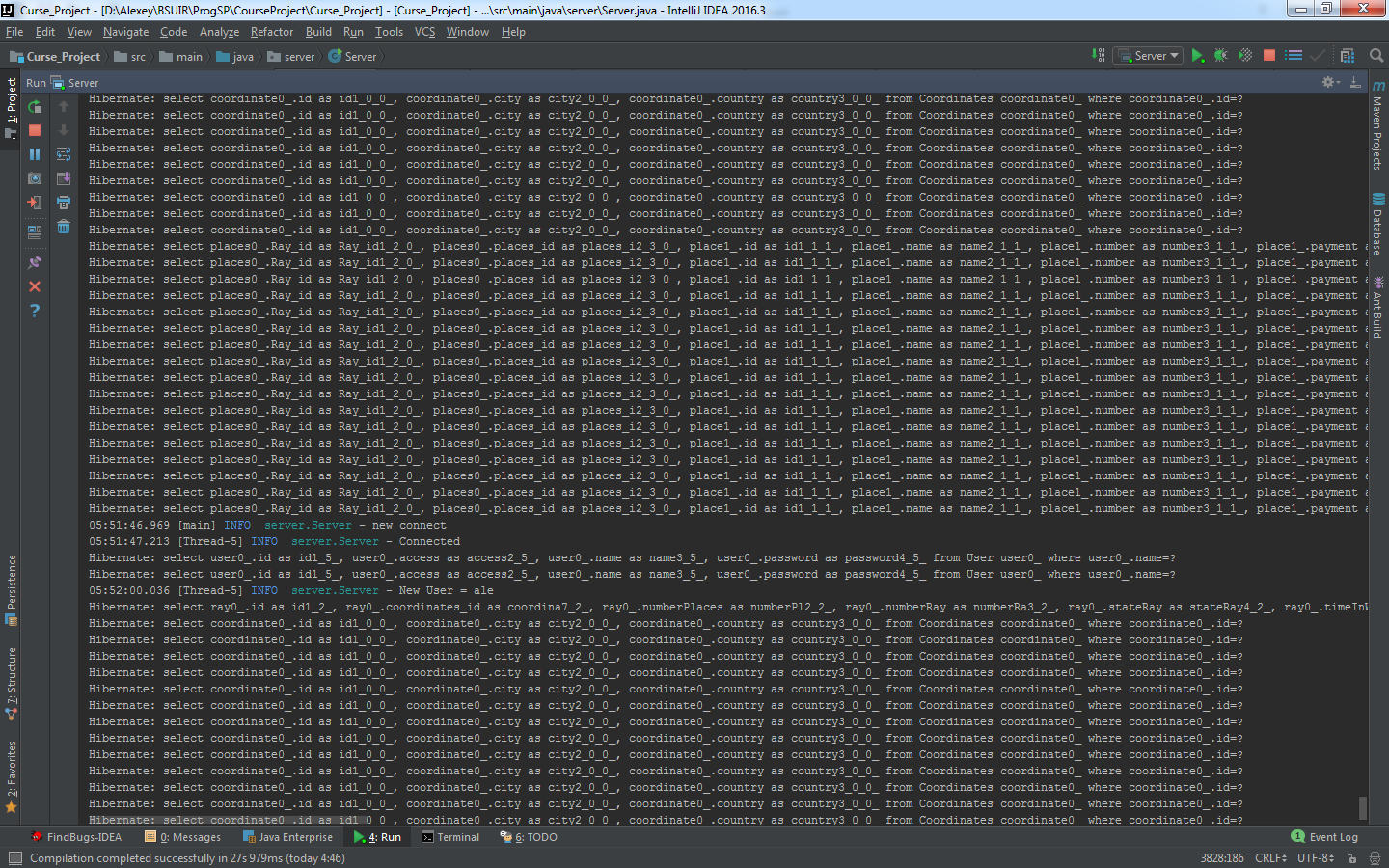


Рисунок 10.1 – Демонстрация работы логгера

На скриншоте видно, что при подключении нового пользователя, сервер выводит в консоль уведомление, что новый клиент подключен. Далее происходят запросы в базу данных с выводом результатов на экран клиенту.

Рассмотрим некоторые механизмы обработки ошибок. При попытке подключиться к одному аккаунту с двух устройств программа оповестит об этом и запретит повторный вход в систему, что видно на рисунке 10.2.

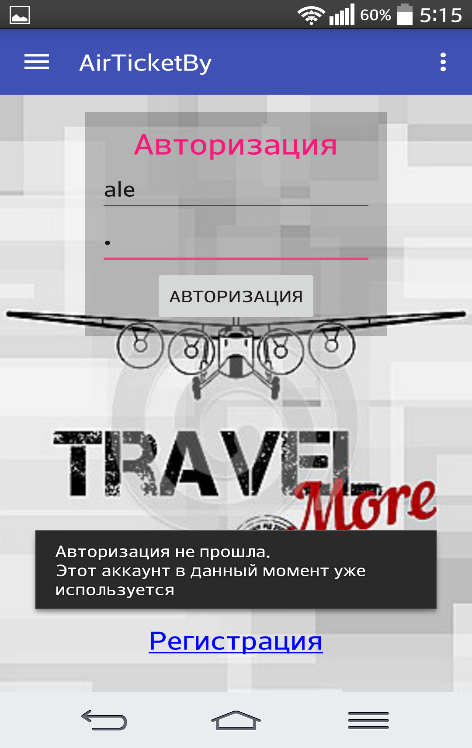


Рисунок 10.2 – Демонстрация обработки ошибок

Так как в системе не может быть пользователей с одинаковыми логинами, программа не позволит зарегистрироваться клиенту с логином, существующим в системе рисунок 10.3.

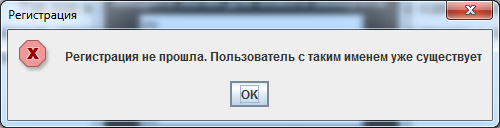


Рисунок 10.3 – Демонстрация проверки на одинаковые логины

При добавлении нового рейса, если в форме есть неверно заполненные поля, правила к заполнению которых написаны рядом с полем, также должны быть обязательно заполнены все поля помеченные красной «\*», остальные могут остаться пустыми, программа не позволит добавить такую запись (рисунок 10.4).

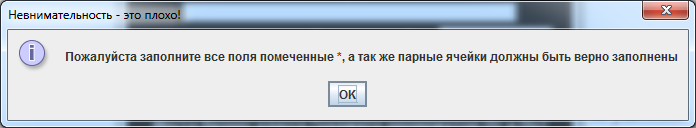


Рисунок 10.4 – Демонстрация проверки правильное заполнение

Таким образом, в данном разделе были продемонстрированы различные обработки исключительных ситуаций, возникающих в процессе работы с программой.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта была детально исследована предметная область системы реализации авиабилетов, а также реализовано программное приложение, позволяющее покупать либо бронировать места на рейс. Реализация программного продукта обеспечивает точный и полный учет продаж авиабилетов. Следовательно, основная задача курсового проекта была успешно выполнена.

Приложение удовлетворяет основным характеристикам, которые были заявлены. Оно удобно в эксплуатации, целостно, конкретизировано в рамках заданной предметной области. Также приложение хорошо структурировано с помощью разбиения на отдельные задачи и подзадачи.

Следует отметить, что предметная область тщательно изучена, исследован тенденции авиа индустрии, а также выявлены следующие сущности БД: Координаты, Рейс, Пользователь, Билет, Место.

В качестве моделей представления некоторого количества классов и связей между ними были построены различные UML-диаграммы, то есть можно говорить о выполнении второстепенных задач проектирования курсового проекта.

При разработке данного программного продукта была учтена логика пользователя и проанализированы вероятные ошибки ввода, с которыми можно столкнуться. Поэтому в программе реализовано множество проверок на различные виды ввода, и при неправильном вводе пользователь получает оповещение-подсказку, помогающую ему сориентироваться и исправить ввод. Программа легко адаптируется под другую предметную область.

Это приложение может полезным для аэропортов, авиа-агентств и туристических компаний, и устанавливаться на терминалы, а также компьютеры кассиров.

В будущем приложение может совершенствоваться посредством добавления в него новых функций (возможность страхования клиентов), редактирование структуры каталога рейсов, организация сортировок и фильтраций по дополнительным полям (в зависимости от предпочтений предприятия).

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Автоматизированная система бронирования авиабилетов [Электронный ресурс]. –Режим доступа: https://studsell.com/view/5832/30000

[2] Йенер, М., Фидом, А. Java ЕЕ. Паттерны проектирования для профессионалов: Практикум. – Спб: Питер, 2016. – 240 c.

[3] I-novice [Электронный ресурс]. – Электронные данные. – Режим доступа: http://i-novice.net/6-normalnyx-form-bd/

[4] Блинов, И., Романчик, В. Java Промышленное программирование: Практическое пособие: УниверсалПресс. – Минск, 2007. – 704 с.

[5] Habrahabr [Электронный ресурс]. – Электронные данные. – Режим доступа: https://habrahabr.ru/post/150041/

[6] Пашкевич, А., Чумаков, О. Современные технологии программирования: Конспект лекций. – Минск: БГУИР, 2007. – 64 с.

[7] Эккель, Брюс. Философия Java: Практическое пособие. – Спб: Питер, 2015. – 1168 с.

# ПРИЛОЖЕНИЕ А

# (обязательное)

# Модели представления системы реализации билетов



Рисунок А.1 – Диаграмма последовательности покупки билета

****

Рисунок А.2 – Диаграмма состояний просмотра статистики



Рисунок А.3 – Диаграмма компонентов серверной части



Рисунок А.4 – Диаграмма компонентов клиентской части (Android)



Рисунок А.5 – Диаграмма компонентов клиентской части (Admin)



Рисунок А.6 – Диаграмма развертывания

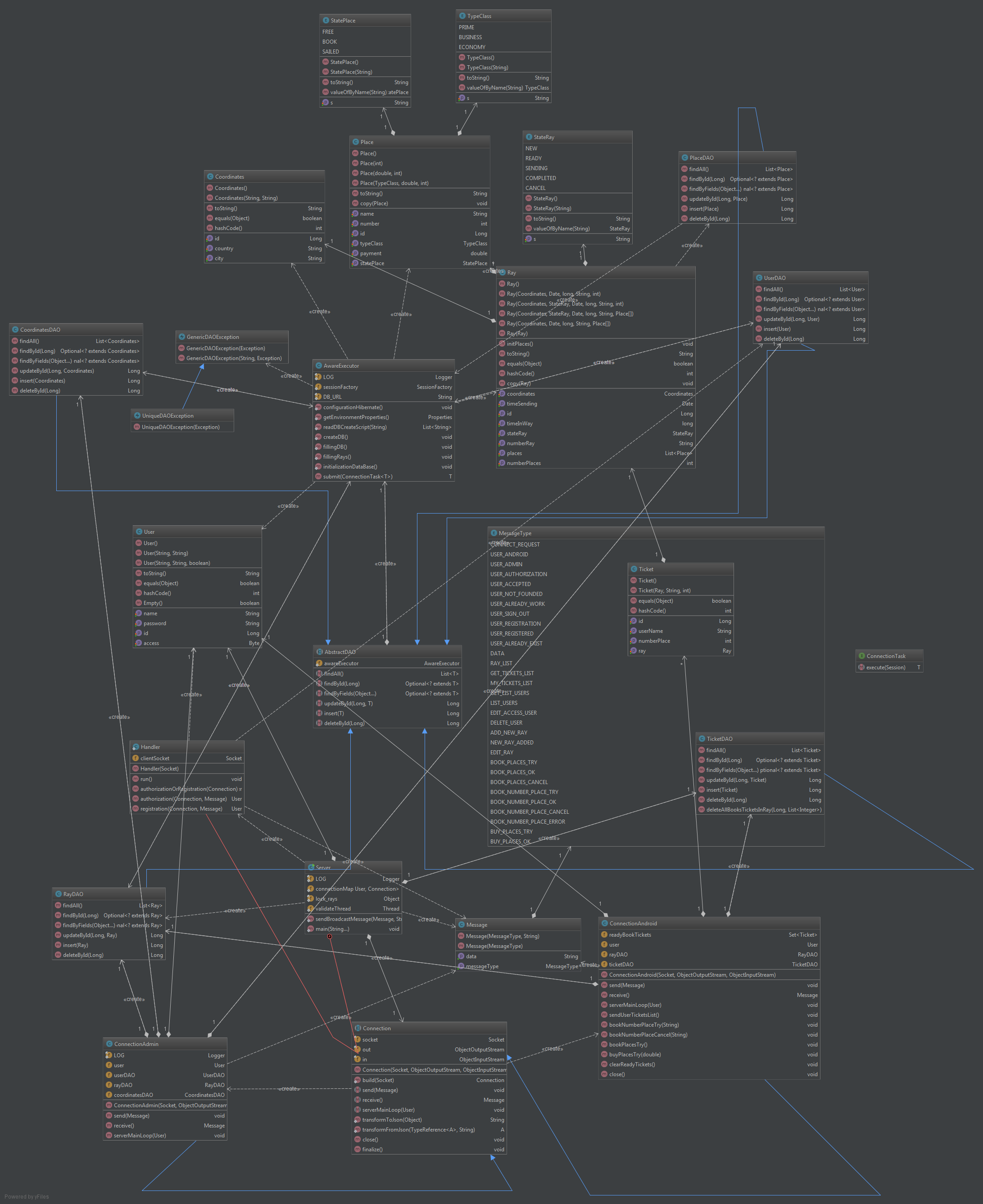
****

Рисунок А.7 – Диаграмма классов серверной части

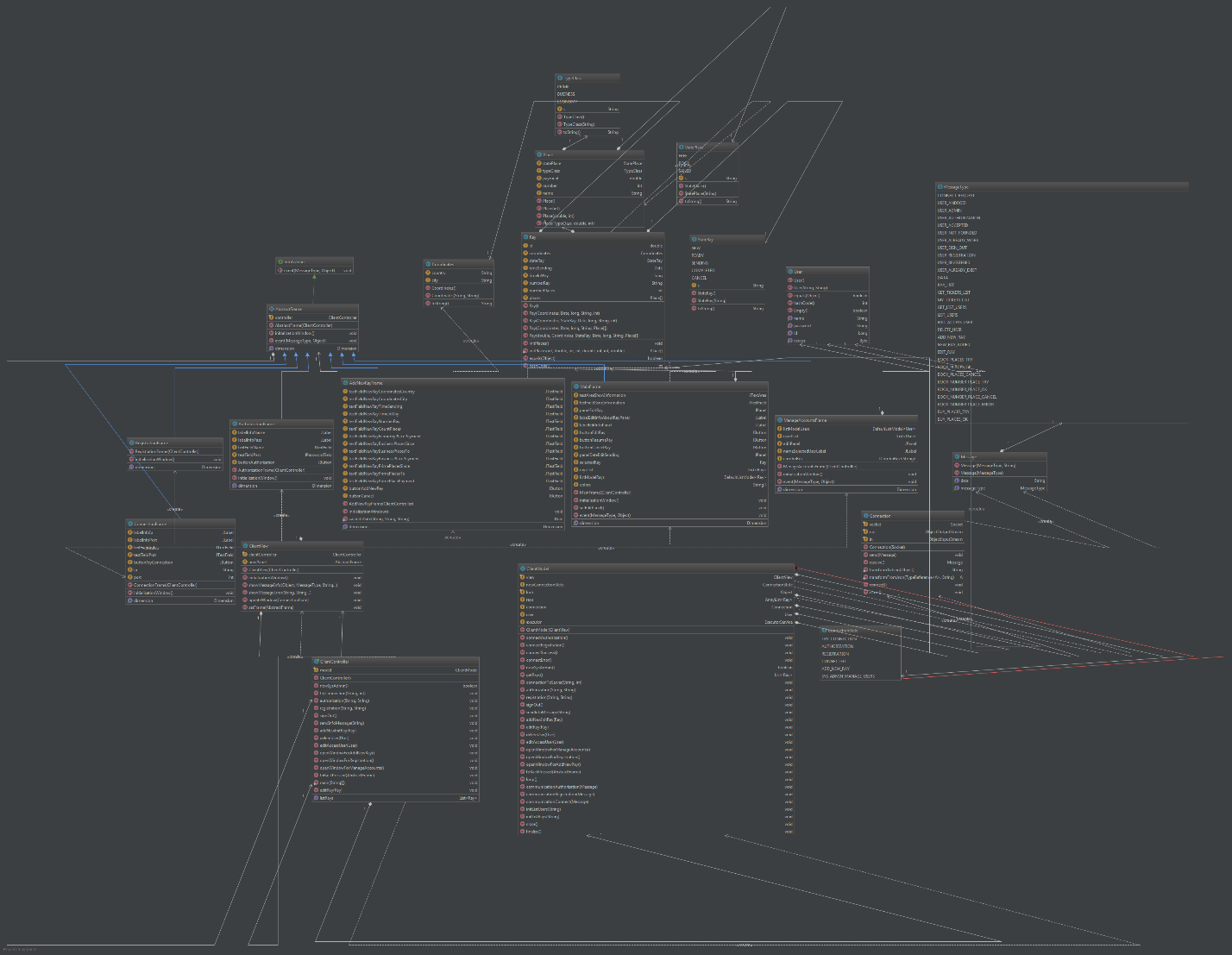
****

Рисунок А.8 – Диаграмма классов клиентской администраторской части

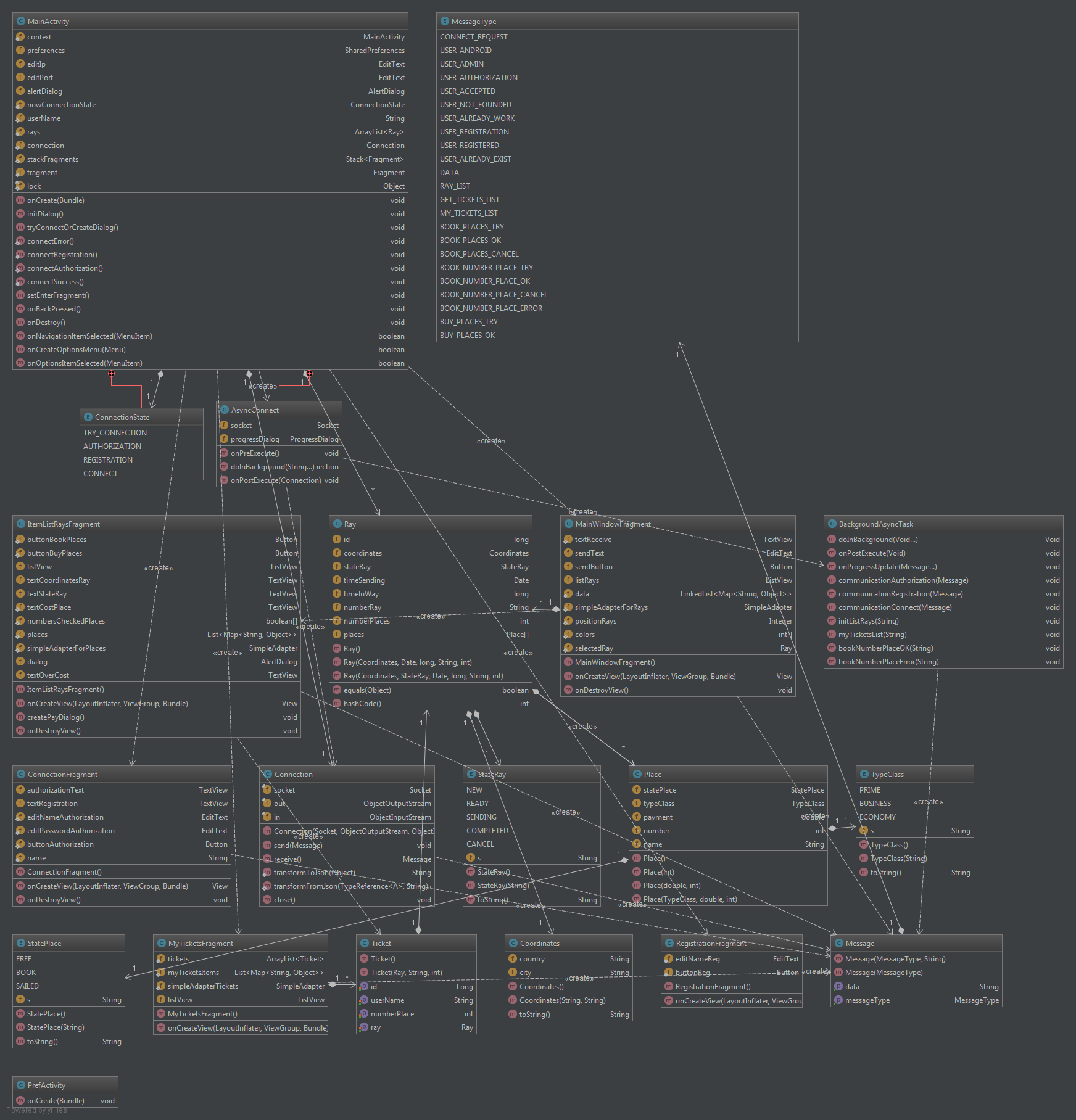
****

Рисунок А.9 – Диаграмма классов пользовательской мобильной части

# ПРИЛОЖЕНИЕ Б

# (обязательное)

# Блок-схемы методов, реализующих бизнес-логику



Рисунок Б.1 – Схема работы всей программы

Продолжение Приложения Б



Рисунок Б.2 – Схема алгоритма покупки билетов

# ПРИЛОЖЕНИЕ В

# (обязательное)

# Листинг программы

// Скрипт создания базы данных

*CREATE DATABASE IF NOT EXISTS `air\_ticket\_by` DEFAULT CHARACTER SET utf8 ;*

*USE `air\_ticket\_by` ;*

*CREATE TABLE IF NOT EXISTS `air\_ticket\_by`.`coordinates` (*

*`id` BIGINT(20) NOT NULL AUTO\_INCREMENT,*

*`city` VARCHAR(255) NULL DEFAULT NULL,*

*`country` VARCHAR(255) NULL DEFAULT NULL,*

*PRIMARY KEY (`id`))*

*ENGINE = InnoDB*

*AUTO\_INCREMENT = 15*

*DEFAULT CHARACTER SET = utf8;*

*CREATE TABLE IF NOT EXISTS `air\_ticket\_by`.`place` (*

*`id` BIGINT(20) NOT NULL AUTO\_INCREMENT,*

*`name` VARCHAR(255) NULL DEFAULT NULL,*

*`number` INT(11) NOT NULL,*

*`payment` DOUBLE NOT NULL,*

*`statePlace` INT(11) NULL DEFAULT NULL,*

*`typeClass` INT(11) NULL DEFAULT NULL,*

*PRIMARY KEY (`id`))*

*ENGINE = InnoDB*

*AUTO\_INCREMENT = 407*

*DEFAULT CHARACTER SET = utf8;*

*CREATE TABLE IF NOT EXISTS `air\_ticket\_by`.`ray` (*

*`id` BIGINT(20) NOT NULL AUTO\_INCREMENT,*

*`numberPlaces` INT(11) NOT NULL,*

*`numberRay` VARCHAR(255) NULL DEFAULT NULL,*

*`stateRay` INT(11) NULL DEFAULT NULL,*

*`timeInWay` BIGINT(20) NOT NULL,*

*`timeSending` DATETIME NULL DEFAULT NULL,*

*`coordinates\_id` BIGINT(20) NULL DEFAULT NULL,*

*PRIMARY KEY (`id`),*

*INDEX `FK\_7ryvk7p461ah6esu4irc307eu` (`coordinates\_id` ASC),*

*CONSTRAINT `FK\_7ryvk7p461ah6esu4irc307eu`*

*FOREIGN KEY (`coordinates\_id`)*

*REFERENCES `air\_ticket\_by`.`coordinates` (`id`))*

*ENGINE = InnoDB*

*AUTO\_INCREMENT = 17*

*DEFAULT CHARACTER SET = utf8;*

*CREATE TABLE IF NOT EXISTS `air\_ticket\_by`.`ray\_place` (*

*`Ray\_id` BIGINT(20) NOT NULL,*

*`places\_id` BIGINT(20) NOT NULL,*

*UNIQUE INDEX `UK\_2cduk4namj16xra5u59e7csym` (`places\_id` ASC),*

*INDEX `FK\_ftacvk785k62xnrcsb8yr93s8` (`Ray\_id` ASC),*

Продолжение Приложения В

*CONSTRAINT `FK\_2cduk4namj16xra5u59e7csym`*

*FOREIGN KEY (`places\_id`)*

*REFERENCES `air\_ticket\_by`.`place` (`id`),*

*CONSTRAINT `FK\_ftacvk785k62xnrcsb8yr93s8`*

*FOREIGN KEY (`Ray\_id`)*

*REFERENCES `air\_ticket\_by`.`ray` (`id`))*

*ENGINE = InnoDB*

*DEFAULT CHARACTER SET = utf8;*

*CREATE TABLE IF NOT EXISTS `air\_ticket\_by`.`ticket` (*

*`id` BIGINT(20) NOT NULL AUTO\_INCREMENT,*

*`numberPlace` INT(11) NOT NULL,*

*`userName` VARCHAR(255) NULL DEFAULT NULL,*

*`ray` BIGINT(20) NULL DEFAULT NULL,*

*PRIMARY KEY (`id`),*

*INDEX `FK\_6cbdvpb52cgruwoxf9sxjkdl` (`ray` ASC),*

*CONSTRAINT `FK\_6cbdvpb52cgruwoxf9sxjkdl`*

*FOREIGN KEY (`ray`)*

*REFERENCES `air\_ticket\_by`.`ray` (`id`))*

*ENGINE = InnoDB*

*AUTO\_INCREMENT = 7*

*DEFAULT CHARACTER SET = utf8;*

*CREATE TABLE IF NOT EXISTS `air\_ticket\_by`.`user` (*

*`id` BIGINT(20) NOT NULL AUTO\_INCREMENT,*

*`access` TINYINT(4) NULL DEFAULT NULL,*

*`name` VARCHAR(255) NULL DEFAULT NULL,*

*`password` VARCHAR(255) NULL DEFAULT NULL,*

*PRIMARY KEY (`id`),*

*UNIQUE INDEX `UK\_syftr7gx86fwf7ox7bgvnnta7` (`name` ASC))*

*ENGINE = InnoDB*

*AUTO\_INCREMENT = 6*

*DEFAULT CHARACTER SET = utf8;*

//Server.java

public class Server {  
  
  
 private static final Logger *LOG* = LoggerFactory.*getLogger*(Server.class);  
  
 public static Map<User, Connection> *connectionMap* = new ConcurrentHashMap<>();  
  
 public static final Object *lock\_rays* = new Object();  
  
  
  
 private static class Handler extends Thread {  
 private Socket clientSocket;  
  
 Handler(Socket clientSocket) {  
 this.clientSocket = clientSocket;  
 }  
 Продолжение Приложения В

@Override  
 public void run() {  
 User user = null;  
 try (Connection connection = Connection.*build*(clientSocket)) {  
 *LOG*.info("Connected");  
 while (true) {  
 user = null;  
 user = authorizationOrRegistration(connection);  
 if (user == null) continue;  
  
 *LOG*.info("New User = " + user.getName());  
 connection.serverMainLoop(user);  
 }  
 } catch (Exception e) {  
 *LOG*.error(e.getMessage());  
 System.*out*.println("Uoops! Run = " + e);  
 }  
 if (user != null) {  
 *connectionMap*.remove(user);  
 *LOG*.info("End run " + user.getName());  
 }  
 }  
  
 private User authorizationOrRegistration(Connection connection) throws IOException, ClassNotFoundException, GenericDAOException {  
 do {  
 Message answer = connection.receive();  
 if (answer.getMessageType() == MessageType.*USER\_AUTHORIZATION*) {  
 return authorization(connection, answer);  
 } else if (answer.getMessageType() == MessageType.*USER\_REGISTRATION*) {  
 return registration(connection, answer);  
 }  
 } while (true);  
 }  
  
 private User authorization(Connection connection, Message answer) throws IOException, GenericDAOException {  
 User user = Connection.*transformFromJson*(new TypeReference<User>() {  
 }, answer.getData());  
 UserDAO userDAO = new UserDAO();  
 User userFromDB;  
 if (user != null && !user.Empty()  
 && userDAO.findByFields(user.getName()).isPresent() && user.equals(userFromDB = userDAO.findByFields(user.getName()).get())  
 && (!(connection instanceof ConnectionAdmin) || (userFromDB.getAccess() != 0 && userFromDB.getAccess() != 2))) {  
 if (!*connectionMap*.containsKey(userFromDB)) {  
 *connectionMap*.put(userFromDB, connection);  
 if (connection instanceof ConnectionAdmin)  
 connection.send(new Message(MessageType.*USER\_ACCEPTED*, Connection.*transformToJson*(userFromDB)));  
 else  
 connection.send(new Message(MessageType.*USER\_ACCEPTED*, answer.getData()));  
 return userFromDB;  
 } else {  
 connection.send(new Message(MessageType.*USER\_ALREADY\_WORK*));  
 Продолжение Приложения В }  
 } else {  
 connection.send(new Message(MessageType.*USER\_NOT\_FOUNDED*));  
 }  
 return null;  
 }  
  
 private User registration(Connection connection, Message answer) throws IOException, GenericDAOException {  
 User user = Connection.*transformFromJson*(new TypeReference<User>() {  
 }, answer.getData());  
 UserDAO userDAO = new UserDAO();  
 if (user != null && !user.Empty() && !userDAO.findByFields(user.getName()).isPresent()) {  
 if (connection instanceof ConnectionAdmin) {  
 if (userDAO.findAll().stream().filter(userFromDB -> !userFromDB.getName().equalsIgnoreCase("alexey") && (userFromDB.getAccess() == -1))  
 .findFirst().orElse(null) == null)  
 user = new User(user.getName(), user.getPassword(), true);  
 else user = new User(user.getName(), user.getPassword(), false);  
  
 } else user = new User(user.getName(), user.getPassword());  
  
 userDAO.insert(user);  
  
 if (connection instanceof ConnectionAdmin) {  
 connection.send(new Message(MessageType.*USER\_REGISTERED*, Connection.*transformToJson*(user)));  
 return null;  
 } else {  
 *connectionMap*.put(user, connection);  
 connection.send(new Message(MessageType.*USER\_REGISTERED*, answer.getData()));  
 return user;  
 }  
 } else {  
 connection.send(new Message(MessageType.*USER\_ALREADY\_EXIST*));  
 }  
 return null;  
 }  
 }  
  
  
 public static Thread *validateThread* = new Thread() {  
 {  
 setDaemon(true);  
 }  
  
 private RayDAO rayDAO = new RayDAO();  
 private TicketDAO ticketDAO = new TicketDAO();  
  
 @Override  
 public void run() {  
 try {  
 synchronized (*lock\_rays*) {  
 while (!isInterrupted()) {  
 Date nowDate = new Date();  
 Продолжение Приложения В rayDAO.findAll().forEach(ray -> {

try {  
 if (ray.getStateRay() != StateRay.*CANCEL*) {  
 if (ray.getTimeSending().getTime() + ray.getTimeInWay() < nowDate.getTime())  
 ray.setStateRay(StateRay.*COMPLETED*);  
 else if (ray.getTimeSending().getTime() < nowDate.getTime())  
 ray.setStateRay(StateRay.*SENDING*);  
 else if (ray.getTimeSending().getTime() - 36000000 < nowDate.getTime()) {  
 ray.setStateRay(StateRay.*READY*);  
 List<Integer> placeNumbers = new LinkedList<>();  
 ray.getPlaces().stream().filter(place -> place.getStatePlace() == StatePlace.*BOOK*)  
 .forEach(place -> {  
 placeNumbers.add(place.getNumber());  
 place.setName(null);  
 place.setStatePlace(StatePlace.*FREE*);  
 });  
 ticketDAO.deleteAllBooksTicketsInRay(ray.getId(), placeNumbers);  
 } else ray.setStateRay(StateRay.*NEW*);  
 rayDAO.updateById(ray.getId(), ray);  
 }  
 } catch (GenericDAOException e) {  
 e.printStackTrace();  
 }  
 });  
 try {  
 *sendBroadcastMessage*(new Message(MessageType.*RAY\_LIST*, Connection.*transformToJson*(rayDAO.findAll())));  
 } catch (IOException e) {  
 }  
 *lock\_rays*.wait(60000);  
 }  
 }  
 } catch (InterruptedException | GenericDAOException ignored) {  
 }  
 }  
 };  
  
 public static void sendBroadcastMessage(Message message, String... ignoreNames) {  
 *connectionMap*.entrySet().stream()  
 .filter(pair -> ignoreNames.length < 1 || !pair.getKey().getName().equals(ignoreNames[0]))  
 .forEach(pair -> {  
 try {  
 pair.getValue().send(message);  
 } catch (IOException ignored) {  
 *LOG*.warn("Ошибка. Сообщение не было доставлено.");  
 }  
 });  
 }  
  
 public static void main(String... args) throws IOException {  
 int port =

Продолжение Приложения В

Integer.*parseInt*(AwareExecutor.*getEnvironmentProperties*().getProperty("server\_port"));  
 try (ServerSocket serverSocket = new ServerSocket(port)) {  
 AwareExecutor.*initializationDataBase*();  
 *validateThread*.start();  
 do {  
 Socket clientSocket = serverSocket.accept();  
 *LOG*.info("new connect");  
 new Handler(clientSocket).start();  
 } while (true);  
 } catch (Exception e) {  
 *LOG*.error("Uoops! Main\n" + e.getMessage());  
 }  
 }  
}

// Connection.class

public abstract class Connection implements Closeable {  
  
 protected final Socket socket;  
 protected final ObjectOutputStream out;  
 protected final ObjectInputStream in;  
  
 protected Connection(Socket socket, ObjectOutputStream out, ObjectInputStream in) {  
 this.socket = socket;  
 this.out = out;  
 this.in = in;  
 }  
  
 public static Connection build(Socket socket) throws IOException, ClassNotFoundException {  
 ObjectOutputStream outputStream = new ObjectOutputStream(socket.getOutputStream());  
 ObjectInputStream inputStream = new ObjectInputStream(socket.getInputStream());  
  
 outputStream.writeObject(MessageType.*CONNECT\_REQUEST*.toString());  
 outputStream.flush();  
 switch (MessageType.*valueOf*((String)inputStream.readObject())){  
 case *USER\_ANDROID*:  
 return new ConnectionAndroid(socket, outputStream, inputStream);  
 case *USER\_ADMIN*:  
 return new ConnectionAdmin(socket, outputStream, inputStream);  
 }  
 outputStream.close();  
 inputStream.close();  
 socket.close();  
 throw new IOException();  
 }  
  
 public abstract void send(Message message) throws IOException;  
  
 public abstract Message receive() throws IOException,

Продолжение Приложения В

ClassNotFoundException;

public abstract void serverMainLoop(User user) throws IOException, ClassNotFoundException, GenericDAOException;  
  
 public static String transformToJson(Object object) throws IOException {  
 StringWriter writer = new StringWriter();  
 new ObjectMapper().writeValue(writer, object);  
 return writer.toString();  
 }  
  
 public static <A> A transformFromJson(final TypeReference<A> type, final String json) throws IOException {  
 return new ObjectMapper().readValue(json, type);  
 }

@Override

public void close() throws IOException {  
 out.close();  
 in.close();  
 socket.close();  
 }  
  
 @Override  
 protected void finalize() throws Throwable {  
 close();  
 }  
}

// ConnectionAdmin.class

public class ConnectionAdmin extends Connection {  
  
 protected ConnectionAdmin(Socket socket, ObjectOutputStream out, ObjectInputStream in) {  
 super(socket, out, in);  
 }  
  
 public void send(Message message) throws IOException {  
 synchronized (out) {  
 out.writeObject(message.getMessageType().toString());  
 out.writeObject(message.getData());  
 out.flush();  
 }  
 }  
  
 public Message receive() throws IOException, ClassNotFoundException {  
 synchronized (in) {  
 MessageType type = MessageType.*valueOf*((String) in.readObject());  
 String data = (String) in.readObject();  
 return new Message(type, data);  
 }  
 }  
  
  
 private static final Logger *LOG* =

Продолжение Приложения В

LoggerFactory.*getLogger*(ConnectionAdmin.class);  
  
 private User user;  
 private UserDAO userDAO = new UserDAO();  
 private RayDAO rayDAO = new RayDAO();  
 private CoordinatesDAO coordinatesDAO = new CoordinatesDAO();  
  
 @Override  
 public void serverMainLoop(User user) throws IOException, ClassNotFoundException, GenericDAOException {  
 send(new Message(MessageType.*RAY\_LIST*, *transformToJson*(rayDAO.findAll())));  
 this.user = user;  
 do {  
 Message message = receive();  
 switch (message.getMessageType()) {  
 case *DATA*: {  
 if (!message.getData().trim().equals(""))  
 *sendBroadcastMessage*(new Message(message.getMessageType(), "Info от " + user.getName() + ": " + message.getData().trim()));  
 break;  
 }  
 case *ADD\_NEW\_RAY*: {  
 Ray ray = *transformFromJson*(new TypeReference<Ray>() {  
 }, message.getData());  
 synchronized (*lock\_rays*) {  
 coordinatesDAO.insert(ray.getCoordinates());  
 rayDAO.insert(ray);  
 send(new Message(MessageType.*NEW\_RAY\_ADDED*));  
 *lock\_rays*.notify();  
 }  
 break;  
 }  
 case *EDIT\_RAY*: {  
 Ray ray = *transformFromJson*(new TypeReference<Ray>() {  
 }, message.getData());  
 synchronized (*lock\_rays*) {  
 Optional<? extends Ray> result = rayDAO.findById(ray.getId());  
 if (result.isPresent()) {  
 result.get().setStateRay(ray.getStateRay());  
 result.get().setTimeSending(ray.getTimeSending());  
 rayDAO.updateById(result.get().getId(), result.get());  
 }  
 *lock\_rays*.notify();  
 }  
 break;  
 }  
 case *USER\_SIGN\_OUT*: {  
 *connectionMap*.remove(user);  
 System.*out*.println("End run " + user.getName());  
 return;  
 }  
 case *GET\_LIST\_USERS*: {  
 send(new Message(MessageType.*LIST\_USERS*, *transformToJson*(userDAO.findAll())));  
 break;

Продолжение Приложения В

}

case *DELETE\_USER*: {  
 User userEditable = *transformFromJson*(new TypeReference<User>() {  
 }, message.getData());  
 Optional<? extends User> result = userDAO.findByFields(userEditable.getName());  
 if (result.isPresent())  
 userDAO.deleteById(result.get().getId());  
 if (*connectionMap*.containsKey(userEditable)) {  
 *connectionMap*.get(userEditable).close();  
 }  
 send(new Message(MessageType.*LIST\_USERS*, *transformToJson*(userDAO.findAll())));  
 break;  
 }  
 case *EDIT\_ACCESS\_USER*: {  
 User userEditable = *transformFromJson*(new TypeReference<User>() {  
 }, message.getData());  
 Optional<? extends User> result = userDAO.findByFields(userEditable.getName());  
  
 if (result.isPresent()) {  
 User userFromDB = result.get();  
 byte oldAccess = userFromDB.getAccess();  
 userFromDB.setAccess(userEditable.getAccess());  
 userDAO.updateById(userFromDB.getId(), userFromDB);  
 if (oldAccess == 1 && (userFromDB.getAccess() == 0 || userFromDB.getAccess() == 2)  
 && *connectionMap*.containsKey(userFromDB)  
 && *connectionMap*.get(userFromDB) instanceof ConnectionAdmin) {  
 *connectionMap*.get(userFromDB).close();  
 }  
 }  
 send(new Message(MessageType.*LIST\_USERS*, *transformToJson*(userDAO.findAll())));  
 break;  
 }  
 }  
 } while (true);  
 }  
}

// ConnectionAndroid.class

class ConnectionAndroid extends Connection {  
  
 protected ConnectionAndroid(Socket socket, ObjectOutputStream out, ObjectInputStream in) {  
 super(socket, out, in);  
 }  
  
 @Override  
 public void send(Message message) throws IOException {

Продолжение Приложения В

synchronized (out) {  
 out.writeObject(message.getMessageType().toString());  
 out.writeObject(message.getData());  
 out.flush();  
 }  
 }  
  
 @Override  
 public Message receive() throws IOException, ClassNotFoundException {  
 synchronized (in) {  
 MessageType type = MessageType.*valueOf*((String) in.readObject());  
 String data = (String) in.readObject();  
 return new Message(type, data);  
 }  
 }  
  
  
 private Set<Ticket> readyBookTickets = new LinkedHashSet<>();  
 private User user;  
 private RayDAO rayDAO = new RayDAO();  
 private TicketDAO ticketDAO = new TicketDAO();  
  
 @Override  
 public void serverMainLoop(User user) throws IOException, ClassNotFoundException, GenericDAOException {  
 synchronized (*lock\_rays*) {  
 send(new Message(MessageType.*RAY\_LIST*, *transformToJson*(rayDAO.findAll())));  
 }  
 this.user = user;  
 while (true) {  
 Message message = receive();  
 System.*out*.println(message.getMessageType());  
 switch (message.getMessageType()) {  
 case *DATA*: {  
 if (!message.getData().trim().equals(""))  
 *sendBroadcastMessage*(new Message(message.getMessageType(), "От " + user.getName() + ": " + message.getData().trim()));  
 break;  
 }  
 case *BOOK\_NUMBER\_PLACE\_TRY*: {  
 bookNumberPlaceTry(message.getData());  
 break;  
 }  
 case *BOOK\_NUMBER\_PLACE\_CANCEL*: {  
 bookNumberPlaceCancel(message.getData());  
 break;  
 }  
 case *BOOK\_PLACES\_TRY*: {  
 bookPlacesTry();  
 break;  
 }  
 case *BOOK\_PLACES\_CANCEL*: {  
 clearReadyTickets();  
 break;  
 }  
 case *BUY\_PLACES\_TRY*: {  
 buyPlacesTry(Double.*parseDouble*(message.getData()));

Продолжение Приложения В

break;  
 }  
 case *GET\_TICKETS\_LIST*: {  
 sendUserTicketsList();  
 break;  
 }  
 }  
 }  
 }  
  
 private void sendUserTicketsList() throws IOException {  
 send(new Message(MessageType.*MY\_TICKETS\_LIST*, *transformToJson*(ticketDAO.findAll())));  
 }  
  
 private void bookNumberPlaceTry(String json) throws IOException, GenericDAOException {  
 Ticket ticket = null;  
 try {  
 ticket = *transformFromJson*(new TypeReference<Ticket>() {  
 }, json);  
 } catch (IOException ignore) {  
 }  
 if (ticket != null) {  
 boolean ok = false;  
 synchronized (*lock\_rays*) {  
 for (Ray ray : rayDAO.findAll()) {  
 if (ray.equals(ticket.getRay())) {  
 Place place = ray.getPlaces().get(ticket.getNumberPlace());  
 if (place.getStatePlace() == StatePlace.*FREE*) {  
 place.setStatePlace(StatePlace.*BOOK*);  
 place.setName(user.getName());  
 rayDAO.updateById(ray.getId(), ray);  
  
 ticket.setRay(ray);  
 readyBookTickets.add(ticket);  
 send(new Message(MessageType.*BOOK\_NUMBER\_PLACE\_OK*, json));  
 ok = true;  
 }  
 break;  
 }  
 }  
 if (!ok) {  
 send(new Message(MessageType.*BOOK\_NUMBER\_PLACE\_ERROR*, json));  
 }  
 *sendBroadcastMessage*(new Message(MessageType.*RAY\_LIST*, *transformToJson*(rayDAO.findAll())));  
 }  
 }  
 }  
  
 private void bookNumberPlaceCancel(String json) throws IOException, GenericDAOException {  
 Ticket ticket = null;  
 try {  
 ticket = *transformFromJson*(new TypeReference<Ticket>() {

Продолжение Приложения В

}, json);  
 } catch (IOException ignore) {  
 }  
 if (ticket != null) {  
 synchronized (*lock\_rays*) {  
 for (Ray ray : rayDAO.findAll()) {  
 if (ray.equals(ticket.getRay())) {  
 Place place = ray.getPlaces().get(ticket.getNumberPlace());  
 if (place.getStatePlace() == StatePlace.*BOOK* && place.getName() != null && place.getName().equals(user.getName())) {  
 place.setStatePlace(StatePlace.*FREE*);  
 place.setName(null);  
 rayDAO.updateById(ray.getId(), ray);  
  
 if (readyBookTickets.contains(ticket))  
 readyBookTickets.remove(ticket);  
 else {  
 Optional<? extends Ticket> result = ticketDAO.findByFields(ray.getId(),  
 ticket.getNumberPlace(), ticket.getUserName());  
 if (result.isPresent())  
 ticketDAO.deleteById(result.get().getId());  
 }  
  
 *sendBroadcastMessage*(new Message(MessageType.*RAY\_LIST*, *transformToJson*(rayDAO.findAll())));  
 }  
 break;  
 }  
 }  
 }  
 }  
 }  
  
 private void bookPlacesTry() throws IOException, GenericDAOException {  
 if (readyBookTickets.size() > 0)  
 synchronized (*lock\_rays*) {  
 for (Ray ray : rayDAO.findAll()) {  
 Iterator<Ticket> iterator = readyBookTickets.iterator();  
 boolean foundRay = false;  
 while (iterator.hasNext()) {  
 Ticket ticket = iterator.next();  
 if (ray.equals(ticket.getRay())) {  
 Place place = ray.getPlaces().get(ticket.getNumberPlace());  
 if (place.getName() != null && place.getName().equalsIgnoreCase(user.getName())  
 && place.getStatePlace() == StatePlace.*BOOK*) {  
 ticketDAO.insert(ticket);  
 }  
 foundRay = true;  
 } else break;  
 }  
 if (foundRay)  
 break;

Продолжение Приложения В

}  
 readyBookTickets.clear();  
 send(new Message(MessageType.*BOOK\_PLACES\_OK*));  
 *sendBroadcastMessage*(new Message(MessageType.*RAY\_LIST*, *transformToJson*(rayDAO.findAll())));  
 }  
 }  
  
 private void buyPlacesTry(double idRay) throws IOException, GenericDAOException {  
 boolean buy = false;  
 synchronized (*lock\_rays*) {  
 for (Ray ray : rayDAO.findAll()) {  
 if (ray.getId() == idRay) {  
  
 for (Place place : ray.getPlaces()) {  
 if (place.getName() != null && place.getName().equalsIgnoreCase(user.getName())  
 && place.getStatePlace() == StatePlace.*BOOK*) {  
 buy = true;  
 place.setStatePlace(StatePlace.*SAILED*);  
 Iterator<Ticket> iterator = readyBookTickets.iterator();  
 while (iterator.hasNext()) {  
 Ticket ticket = iterator.next();  
 if (ticket.getNumberPlace() == place.getNumber()) {  
 ticketDAO.insert(ticket);  
 iterator.remove();  
 break;  
 }  
 }  
 }  
 }  
 rayDAO.updateById(ray.getId(), ray);  
 break;  
 }  
 }  
 readyBookTickets.clear();  
 if (buy)  
 send(new Message(MessageType.*BUY\_PLACES\_OK*));  
 *sendBroadcastMessage*(new Message(MessageType.*RAY\_LIST*, *transformToJson*(rayDAO.findAll())));  
 }  
 }  
  
 private void clearReadyTickets() throws IOException, GenericDAOException {  
 if (readyBookTickets.size() > 0) {  
 synchronized (*lock\_rays*) {  
 for (Ray ray : rayDAO.findAll()) {  
 Iterator<Ticket> iterator = readyBookTickets.iterator();  
 boolean foundRay = false;  
 while (iterator.hasNext()) {  
 Ticket ticket = iterator.next();  
 if (ray.equals(ticket.getRay())) {

Продолжение Приложения В

Place place = ray.getPlaces().get(ticket.getNumberPlace());  
 if (place.getName() != null && place.getName().equalsIgnoreCase(user.getName())  
 && place.getStatePlace() == StatePlace.*BOOK*) {  
 place.setStatePlace(StatePlace.*FREE*);  
 place.setName(null);  
 rayDAO.updateById(ray.getId(), ray);  
 }  
 foundRay = true;  
 } else break;  
 }  
 if (foundRay)  
 break;  
 }  
 readyBookTickets.clear();  
 *sendBroadcastMessage*(new Message(MessageType.*RAY\_LIST*, *transformToJson*(rayDAO.findAll())));  
 }  
 }  
 }  
  
 @Override  
 public void close() throws IOException {  
 try {  
 clearReadyTickets();  
 } catch (GenericDAOException e) {  
 }  
 super.close();  
 }  
}

// ClientModel.class (Swing)

public class ClientModel {  
 private final ClientView view;  
  
 ClientModel(ClientView view) {  
 this.view = view;  
 }  
  
 enum ConnectionState {  
 *TRY\_CONNECTION*,  
 *AUTHORIZATION*,  
 *REGISTRATION*,  
 *CONNECTED*,  
 // Others in connected  
 *ADD\_NEW\_RAY*,  
 *SYS\_ADMIN\_MANAGE\_USERS* }  
  
 private ConnectionState nowConnectionState = ConnectionState.*TRY\_CONNECTION*;  
  
 private void connectAuthorization() {

Продолжение Приложения В

nowConnectionState = ConnectionState.*AUTHORIZATION*;

}  
 private void connectRegistration() {  
 nowConnectionState = ConnectionState.*REGISTRATION*;  
 }  
 private void connectSuccess() {  
 nowConnectionState = ConnectionState.*CONNECTED*;  
 }  
 private void connectError() {  
 synchronized (lock) {  
 if (nowConnectionState != ConnectionState.*TRY\_CONNECTION*) {  
 nowConnectionState = ConnectionState.*TRY\_CONNECTION*;  
 try {  
 if (connection != null)  
 connection.close();  
 } catch (IOException e) {  
 }  
 connection = null;  
 user = null;  
 view.updateWindow(nowConnectionState);  
 }  
 }  
 }  
  
 private final Object lock = new Object();  
  
  
 boolean nowSysAdmin() {  
 return user.getAccess() == -1;  
 }  
  
 List<Ray> getRays() {  
 return rays;  
 }  
  
 private ArrayList<Ray> rays = null;  
 private Connection connection = null;  
 private User user = null;  
  
 private ExecutorService executor = Executors.*newFixedThreadPool*(1);  
  
 void connectionToServer(String ip, int port) {  
 // Try connect  
 Future<Connection> future = executor.submit(() -> {  
 Socket socket = new Socket();  
 socket.connect(new InetSocketAddress(InetAddress.*getByName*(ip), port), 5000);  
 Connection connection = new Connection(socket);  
 connection.connect();  
 return connection;  
 });  
 // Wait  
 while (!future.isDone()) {  
 Thread.*yield*();  
 }  
 try {  
 // Test connection  
 connection = future.get();

Продолжение Приложения В

//view.showMessageInfo("Соединение установлено", null);

connectAuthorization();  
 view.updateWindow(nowConnectionState);  
  
 // Background listener start  
 executor.submit(() -> {  
 try {  
 loop();  
 } catch (Exception exception) {  
 System.*out*.println("Бай-бай");  
 connectError();  
 }  
 });  
  
 } catch (Exception exception) {  
 view.showMessageError("Не удалось установить соединение", "Неудачная попытка соединения");  
 close();  
 }  
 }  
  
 /////////////////////////////////////////////////////////////////////////  
 // Events Methods  
  
 void authorization(String name, String password) {  
 try {  
 connection.send(new Message(MessageType.*USER\_AUTHORIZATION*, *transformToJson*(new User(name, password))));  
 } catch (Exception exception) {  
 connectError();  
 }  
 }  
 void registration(String name, String password) {  
 try {  
 connection.send(new Message(MessageType.*USER\_REGISTRATION*, *transformToJson*(new User(name, password))));  
 } catch (IOException e) {  
 connectError();  
 }  
 }  
 void signOut() {  
 try {  
 connection.send(new Message(MessageType.*USER\_SIGN\_OUT*));  
 user = null;  
 connectAuthorization();  
 view.updateWindow(nowConnectionState);  
 } catch (IOException ignored) {  
 connectError();  
 }  
 }  
  
  
 void sendInfoMessage(String text) {  
 try {  
 connection.send(new Message(MessageType.*DATA*, text));  
 } catch (IOException e) {  
 connectError();  
 }  
 }

Продолжение Приложения В

void addNewInitRay(Ray ray) {  
 try {  
 connection.send(new Message(MessageType.*ADD\_NEW\_RAY*, *transformToJson*(ray)));  
 } catch (IOException ignored) {  
 connectError();  
 }  
 }  
 public void editRay(Ray ray) {  
 try{  
 connection.send(new Message(MessageType.*EDIT\_RAY*, *transformToJson*(ray)));  
 } catch (IOException e) {  
 connectError();  
 }  
 }  
  
 void deleteUser(User user) {  
 try{  
 connection.send(new Message(MessageType.*DELETE\_USER*, *transformToJson*(user)));  
 } catch (IOException ignored) {  
 connectError();  
 }  
 }  
 void editAccessUser(User user) {  
 try{  
 connection.send(new Message(MessageType.*EDIT\_ACCESS\_USER*, *transformToJson*(user)));  
 } catch (IOException ignored) {  
 connectError();  
 }  
 }  
  
 // Manage windows  
  
 void openWindowForManageAccounts() {  
 try {  
 view.updateWindow(ConnectionState.*SYS\_ADMIN\_MANAGE\_USERS*);  
 connection.send(new Message(MessageType.*GET\_LIST\_USERS*));  
 } catch (Exception e) {  
 connectError();  
 }  
 }  
 void openWindowForRegistration() {  
 connectRegistration();  
 view.updateWindow(nowConnectionState);  
 }  
 void openWindowForAddNewRay() {  
 view.updateWindow(ConnectionState.*ADD\_NEW\_RAY*);  
 }  
  
 void toBackPressed(AbstractFrame abstractFrame) {  
 if (abstractFrame instanceof RegistrationFrame)  
 connectAuthorization();  
 view.updateWindow(nowConnectionState);  
 }

Продолжение Приложения В

//////////////////////////////  
 // Loop and other methods  
  
 private void loop() throws IOException, ClassNotFoundException {  
 while (!Thread.*currentThread*().isInterrupted() && connection != null) {  
 Message message = connection.receive();  
 if (nowConnectionState == ConnectionState.*AUTHORIZATION*) {  
 communicationAuthorization(message);  
 } else if (nowConnectionState == ConnectionState.*REGISTRATION*) {  
 communicationRegistration(message);  
 } else if (nowConnectionState == ConnectionState.*CONNECTED*) {  
 communicationConnect(message);  
 }  
 }  
 }  
  
 ////////////////////////////////////// Authorization  
 private void communicationAuthorization(Message message) throws IOException {  
 switch (message.getMessageType()) {  
 case *USER\_ACCEPTED*: {  
 user = *transformFromJson*(new TypeReference<User>() {  
 }, message.getData());  
 connectSuccess();  
 view.updateWindow(nowConnectionState);  
 break;  
 }  
 case *USER\_NOT\_FOUNDED*: {  
 view.showMessageError("Увы, логин или пароль неверен", "Авторизация");  
 break;  
 }  
 case *USER\_ALREADY\_WORK*: {  
 view.showMessageError("Увы, данный аккаунт сейчас используется", "Авторизация");  
 break;  
 }  
 }  
 }  
  
 /////////////////////////////////////////////////// Registration  
 private void communicationRegistration(Message message) throws IOException {  
 switch (message.getMessageType()){  
 case *USER\_REGISTERED*: {  
 user = *transformFromJson*(new TypeReference<User>() {  
 }, message.getData());  
 view.showMessageInfo("<html>Регистрация прошла успешно.<br>Пожалуйста, дождитесь пока системный администратор<br> обработает вашу запрос, на получение прав модератора.</html>", null, "Регистрация");  
 connectAuthorization();  
 view.updateWindow(nowConnectionState);  
 break;  
 }  
 case *USER\_ALREADY\_EXIST*: {  
 view.showMessageError("Регистрация не прошла. Пользователь с таким именем уже существует", "Регистрация");  
 break;  
 }

Продолжение Приложения В

}

}  
  
 ////////////////////////////////////////////////// Connected  
 private void communicationConnect(Message message) {  
 switch (message.getMessageType()) {  
 case *DATA*: {  
 view.showMessageInfo(message.getData(), MessageType.*DATA*);  
 break;  
 }  
 case *RAY\_LIST*: {  
 initListRays(message.getData());  
 break;  
 }  
 case *NEW\_RAY\_ADDED*: {  
 view.showMessageInfo("Рейс успешно добавлен.", null, "Добавление рейса");  
 view.updateWindow(nowConnectionState);  
 break;  
 }  
 case *LIST\_USERS*: {  
 initListUsers(message.getData());  
 break;  
 }  
 }  
 }  
  
 private void initListUsers(String json) {  
 try {  
 view.showMessageInfo(*transformFromJson*(new TypeReference<ArrayList<User>>() {}, json), MessageType.*LIST\_USERS*);  
 } catch (IOException ignored) {}  
 }  
  
 private void initListRays(String json) {  
 try {  
 rays = *transformFromJson*(new TypeReference<ArrayList<Ray>>() {  
 }, json);  
 view.showMessageInfo(null, MessageType.*RAY\_LIST*);  
 } catch (IOException ignored) {}  
 }  
  
  
 ///////////////////////////////////////////////////////// Close  
 protected void close() {  
 try {  
 if (connection != null)  
 connection.close();  
 } catch (IOException ignored) {  
 System.*out*.println("Finish " + ignored);  
 }  
 }  
  
 @Override  
 protected void finalize() throws Throwable {  
 close();  
 }  
}

Продолжение Приложения В

*// MainActivity.class (Android)*

**public class** MainActivity **extends** AppCompatActivity  
 **implements** NavigationView.OnNavigationItemSelectedListener {  
  
  
 **static** MainActivity *context*;  
 SharedPreferences **preferences**;  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
  
 Toolbar toolbar = (Toolbar) findViewById(R.id.***toolbar***);  
 setSupportActionBar(toolbar);  
 DrawerLayout drawer = (DrawerLayout) findViewById(R.id.***drawer\_layout***);  
 ActionBarDrawerToggle toggle = **new** ActionBarDrawerToggle(  
 **this**, drawer, toolbar, R.string.***navigation\_drawer\_open***, R.string.***navigation\_drawer\_close***);  
 drawer.setDrawerListener(toggle);  
 toggle.syncState();  
 NavigationView navigationView = (NavigationView) findViewById(R.id.***nav\_view***);  
 navigationView.setNavigationItemSelectedListener(**this**);  
 *//////////////////////////////////  
 context* = **this**;  
 **preferences** = PreferenceManager.*getDefaultSharedPreferences*(**this**);  
 initDialog();  
 setEnterFragment();  
 tryConnectOrCreateDialog();  
 }  
  
 EditText **editIp**;  
 EditText **editPort**;  
 AlertDialog **alertDialog**;  
  
 **private void** initDialog() {  
 AlertDialog.Builder dialog = **new** AlertDialog.Builder(**this**);  
 dialog.setTitle(**"Не удалось подключиться к серверу по данным параметрам"**);  
 View view = getLayoutInflater().inflate(R.layout.***connection\_dialog***, **null**);  
 **editIp** = (EditText) view.findViewById(R.id.***editIP***);  
 **editPort** = (EditText) view.findViewById(R.id.***editPort***);  
 **editIp**.setText(**preferences**.getString(**"ip"**, **""**));  
 **editPort**.setText(**preferences**.getString(**"port"**, **""**));  
 view.findViewById(R.id.***buttonConnect***).setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **alertDialog**.dismiss();  
 **new** AsyncConnect().execute(**editIp**.getText().toString(), **editPort**.getText().toString()); *// Asynchronous connection* }  
 });  
 view.findViewById(R.id.***buttonExit***).setOnClickListener(**new** View.OnClickListener() {

Продолжение Приложения В

@Override

**public void** onClick(View v) {  
 finish();  
 }  
 });  
 dialog.setView(view);  
 dialog.setCancelable(**false**);  
 **alertDialog** = dialog.create();  
 }  
 **void** tryConnectOrCreateDialog() {  
 **if** (**preferences**.getBoolean(**"autoConnect"**, **false**)) {  
 **new** AsyncConnect().execute(**preferences**.getString(**"ip"**, **""**), **preferences**.getString(**"port"**, **""**));  
 } **else** {  
 **alertDialog**.show();  
 }  
 }  
  
 **class** AsyncConnect **extends** AsyncTask<String, String, Connection> {  
  
 **private** Socket **socket** = **new** Socket();  
 **private** ProgressDialog **progressDialog**;  
  
 @Override  
 **protected void** onPreExecute() {  
 **progressDialog** = **new** ProgressDialog(*context*);  
 **progressDialog**.setTitle(**"Подключение к серверу"**);  
 **progressDialog**.setMessage(**"Пожалуйста подождите"**);  
 **progressDialog**.setCancelable(**false**);  
 **progressDialog**.setButton(Dialog.***BUTTON\_POSITIVE***, **"Прервать"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialog, **int** which) {  
 **try** {  
 **socket**.close();  
 } **catch** (IOException e) {  
 Toast.*makeText*(*context*, **"Ошибка в прерывании"**, Toast.***LENGTH\_SHORT***).show();  
 }  
 }  
  
 ;  
 });  
 **alertDialog**.dismiss();  
 **progressDialog**.show();  
 }  
 @Override  
 **protected** Connection doInBackground(String... params) {  
 **if** (params == **null** || params.**length** != 2) **return null**;  
 Connection connection = **null**;  
 **try** {  
 **socket**.connect(**new** InetSocketAddress(InetAddress.*getByName*(params[0]), Integer.*parseInt*(params[1])), 5000);  
 ObjectOutputStream outputStream = **new** ObjectOutputStream(**socket**.getOutputStream());  
 ObjectInputStream inputStream = **new** ObjectInputStream(**socket**.getInputStream());  
  
 connection = **new** Connection(**socket**, outputStream,

Продолжение Приложения В

inputStream);  
 **if** (MessageType.*valueOf*((String) inputStream.readObject()) == MessageType.***CONNECT\_REQUEST***) {  
 outputStream.writeObject(MessageType.***USER\_ANDROID***.toString());  
 outputStream.flush();  
 **return** connection;  
 }  
 } **catch** (Exception ignore) {  
 }  
 **try** {  
 **if** (connection != **null**)  
 connection.close();  
 } **catch** (IOException ignored) {  
 }  
 **return null**;  
 }  
 @Override  
 **protected void** onPostExecute(Connection connection) {  
 **progressDialog**.dismiss();  
 **if** (connection != **null**) {  
 MainActivity.*connection* = connection;  
 Toast.*makeText*(MainActivity.*context*, **"Соединение установленно"**, Toast.***LENGTH\_SHORT***).show();  
  
 *connectAuthorization*();  
 **new** BackgroundAsyncTask().execute(); *// Start listener* ConnectionFragment.*editNameAuthorization*.setEnabled(**true**);  
 ConnectionFragment.*editPasswordAuthorization*.setEnabled(**true**);  
 ConnectionFragment.*buttonAuthorization*.setEnabled(**true**);  
 } **else** {  
 Toast.*makeText*(MainActivity.*context*, **"Соединение не удалось установить"**, Toast.***LENGTH\_SHORT***).show();  
 **alertDialog**.show();  
 }  
 }  
 }  
  
 **enum** ConnectionState {  
 ***TRY\_CONNECTION***,  
 ***AUTHORIZATION***,  
 ***REGISTRATION***,  
 ***CONNECT*** }  
  
 **static** ConnectionState *nowConnectionState* = ConnectionState.***TRY\_CONNECTION***;  
 **static** String *userName* = **null**;  
 **static** ArrayList<Ray> *rays* = **null**;  
 **static** Connection *connection* = **null**;  
 **static** Stack<Fragment> *stackFragments* = **new** Stack<>();  
 **static** Fragment *fragment* = **null**;  
  
 **final static** Object ***lock*** = **new** Object();  
  
 **static void** connectError() {  
 **synchronized** (***lock***) {  
 **if** (*nowConnectionState* != ConnectionState.***TRY\_CONNECTION***) {

Продолжение Приложения В

*nowConnectionState* = ConnectionState.***TRY\_CONNECTION***;

**if** (*connection* != **null**) {  
 **try** {  
 *connection*.close();  
 } **catch** (Exception ignore) {  
 }  
 }  
 *connection* = **null**;  
 *rays* = **null**;  
 *userName* = **null**;  
 *stackFragments*.clear();  
 *context*.setEnterFragment();  
 *context*.tryConnectOrCreateDialog();  
 Toast.*makeText*(MainActivity.*context*, **"Соединение разорвано"**, Toast.***LENGTH\_SHORT***).show();  
 }  
 }  
 }  
 **static void** connectRegistration() {  
 *nowConnectionState* = ConnectionState.***REGISTRATION***;  
 }  
 **static void** connectAuthorization() {  
 *nowConnectionState* = ConnectionState.***AUTHORIZATION***;  
 }  
 **static void** connectSuccess() {  
 *nowConnectionState* = ConnectionState.***CONNECT***;  
 }  
  
 **void** setEnterFragment() {  
 FragmentManager fragmentManager = getSupportFragmentManager();  
 **switch** (*nowConnectionState*) {  
 **case *TRY\_CONNECTION***: {  
 }  
 **case *AUTHORIZATION***: {  
 *fragment* = **new** ConnectionFragment();  
 **break**;  
 }  
 **case *REGISTRATION***: {  
 *fragment* = **new** RegistrationFragment();  
 **break**;  
 }  
 **case *CONNECT***: {  
 *fragment* = **new** MainWindowFragment();  
 **break**;  
 }  
 }  
 fragmentManager.beginTransaction().replace(R.id.***content\_frame***, *fragment*).commit();  
 }  
  
  
 @Override  
 **public void** onBackPressed() {  
 DrawerLayout drawer = (DrawerLayout) findViewById(R.id.***drawer\_layout***);  
 **if** (drawer.isDrawerOpen(GravityCompat.***START***)) {  
 drawer.closeDrawer(GravityCompat.***START***);  
 } **else** {  
 **if** (*nowConnectionState* == ConnectionState.***CONNECT*** && *fragment*

Продолжение Приложения В

**instanceof** MainWindowFragment) {

*connectError*();  
 } **else if** (*nowConnectionState* == ConnectionState.***REGISTRATION*** && *fragment* **instanceof** RegistrationFragment) {  
 *connectAuthorization*();  
 setEnterFragment();  
 } **else if**(*nowConnectionState* == ConnectionState.***CONNECT*** && *fragment* **instanceof** MyTicketsFragment) {  
 *context*.setEnterFragment();  
 } **else if** (!*stackFragments*.isEmpty()) {  
 *fragment* = *stackFragments*.pop();  
 getSupportFragmentManager().beginTransaction().replace(R.id.***content\_frame***, *fragment*).commit();  
 } **else super**.onBackPressed();  
 }  
 }  
 @Override  
 **protected void** onDestroy() {  
 **super**.onDestroy();  
 *nowConnectionState* = ConnectionState.***TRY\_CONNECTION***;  
 **if** (*connection* != **null**)  
 **try** {  
 *connection*.close();  
 } **catch** (IOException ignore) {  
 }  
 }  
  
 @SuppressWarnings(**"StatementWithEmptyBody"**)  
 @Override  
 **public boolean** onNavigationItemSelected(MenuItem item) {  
 *// Handle navigation view item clicks here.* **int** id = item.getItemId();  
  
 **switch** (id) {  
 **case** R.id.***nav\_rays***:  
 *stackFragments*.clear();  
 setEnterFragment();  
 **break**;  
 **case** R.id.***nav\_my\_tickets***:  
 **if** (*nowConnectionState* == ConnectionState.***CONNECT***) {  
 **if**(!(*fragment* **instanceof** MyTicketsFragment)) {  
 *stackFragments*.clear();  
 *fragment* = **new** MyTicketsFragment();  
 getSupportFragmentManager().beginTransaction().replace(R.id.***content\_frame***, *fragment*).commit();  
 }  
 } **else** Toast.*makeText*(*context*, **"Необходимо аворизоваться"**, Toast.***LENGTH\_SHORT***).show();  
 **break**;  
 **case** R.id.***nav\_manage***:  
 startActivity(**new** Intent(**this**, PrefActivity.**class**));  
 **break**;  
 }  
  
 DrawerLayout drawer = (DrawerLayout) findViewById(R.id.***drawer\_layout***);

Продолжение Приложения В

drawer.closeDrawer(GravityCompat.***START***);

**return true**;  
 }  
  
  
 @Override  
 **public boolean** onCreateOptionsMenu(Menu menu) {  
 *// Inflate the menu; this adds items to the action bar if it is present.* getMenuInflater().inflate(R.menu.***main***, menu);  
 **return true**;  
 }  
 @Override  
 **public boolean** onOptionsItemSelected(MenuItem item) {  
 *// Handle action bar item clicks here. The action bar will  
 // automatically handle clicks on the Home/Up button, so long  
 // as you specify a parent activity in AndroidManifest.xml.* **int** id = item.getItemId();  
  
 *//noinspection SimplifiableIfStatement* **if** (id == R.id.***action\_settings***) {  
 **return true**;  
 }  
  
 **return super**.onOptionsItemSelected(item);  
 }  
}

*// BackgroundAsyncTask.class*

**public class** BackgroundAsyncTask **extends** AsyncTask<Void, Message, Void> {  
  
 @Override  
 **protected** Void doInBackground(Void... params) {  
 **try** {  
 **while** (!Thread.*currentThread*().isInterrupted() && MainActivity.*connection* != **null**) {  
 Message message = MainActivity.*connection*.receive();  
 **if** (message.getMessageType() != **null**) {  
 publishProgress(message);  
 }  
 }  
 } **catch** (Exception ignore) {  
 }  
 **return null**;  
 }  
  
 @Override  
 **protected void** onPostExecute(Void aVoid) {  
 Toast.*makeText*(MainActivity.*context*, **"Слушатель бай-бай"**, Toast.***LENGTH\_SHORT***).show();  
 MainActivity.*connectError*();  
 }  
  
 @Override  
 **protected void** onProgressUpdate(Message... values) {  
 **switch** (MainActivity.*nowConnectionState*) {  
 **case *AUTHORIZATION***:

Продолжение Приложения В

communicationAuthorization(values[0]);

**break**;  
 **case *REGISTRATION***:  
 communicationRegistration(values[0]);  
 **break**;  
 **case *CONNECT***:  
 communicationConnect(values[0]);  
 **break**;  
 }  
 }  
  
 *// AUTHORIZATION* **private void** communicationAuthorization(Message message) {  
 **switch** (message.getMessageType()) {  
 **case *USER\_ACCEPTED***: {  
 **try** {  
 MainActivity.*userName* = Connection.*transformFromJson*(**new** TypeReference<User>() {  
 }, message.getData()).getName();  
 } **catch** (IOException ignored) {  
 }  
 Toast.*makeText*(MainActivity.*context*, **"Авторизация прошла успешно."**, Toast.***LENGTH\_SHORT***).show();  
 Toast.*makeText*(MainActivity.*context*, **"Добро пожаловать "** + MainActivity.*userName*, Toast.***LENGTH\_SHORT***).show();  
 MainActivity.*connectSuccess*();  
 MainActivity.*context*.setEnterFragment();  
 **break**;  
 }  
 **case *USER\_NOT\_FOUNDED***: {  
 Toast.*makeText*(MainActivity.*context*, **"Авторизация не прошла.\n"** +  
 **"Пользователь с таким именем не найден"**, Toast.***LENGTH\_SHORT***).show();  
 **break**;  
 }  
 **case *USER\_ALREADY\_WORK***: {  
 Toast.*makeText*(MainActivity.*context*, **"Авторизация не прошла.\n"** +  
 **"Этот аккаунт в данный момент уже используется"**, Toast.***LENGTH\_SHORT***).show();  
 **break**;  
 }  
 }  
 ConnectionFragment.*editNameAuthorization*.setEnabled(**true**);  
 ConnectionFragment.*editPasswordAuthorization*.setEnabled(**true**);  
 ConnectionFragment.*buttonAuthorization*.setEnabled(**true**);  
 }  
  
  
 *// REGISTRATION* **private void** communicationRegistration(Message message) {  
 **switch** (message.getMessageType()) {  
 **case *USER\_REGISTERED***: {  
 **try** {  
 MainActivity.*userName* = Connection.*transformFromJson*(**new** TypeReference<User>() {  
 }, message.getData()).getName();  
 } **catch** (IOException ignored) {

Продолжение Приложения В

}

Toast.*makeText*(MainActivity.*context*, **"Регистрация успешно прошла.\n"** +  
 **"Теперь вы зарегистрированы"**, Toast.***LENGTH\_SHORT***).show();  
 Toast.*makeText*(MainActivity.*context*, **"Добро пожаловать "** + MainActivity.*userName*, Toast.***LENGTH\_SHORT***).show();  
 MainActivity.*connectSuccess*();  
 MainActivity.*context*.setEnterFragment();  
 **break**;  
 }  
 **case *USER\_ALREADY\_EXIST***: {  
 Toast.*makeText*(MainActivity.*context*, **"Регистрация не прошла.\n"** +  
 **"Пользователь с таким именем уже существует"**, Toast.***LENGTH\_SHORT***).show();  
 **break**;  
 }  
 }  
 RegistrationFragment.*buttonReg*.setEnabled(**true**);  
 }  
  
  
 *// CONNECT* **private void** communicationConnect(Message message) {  
 **switch** (message.getMessageType()) {  
 **case *DATA***: {  
 MainWindowFragment.*textReceive*.setText(message.getData());  
 **break**;  
 }  
 **case *RAY\_LIST***: {  
 initListRays(message.getData());  
 **break**;  
 }  
 **case *MY\_TICKETS\_LIST***: {  
 myTicketsList(message.getData());  
 **break**;  
 }  
 **case *BOOK\_NUMBER\_PLACE\_OK***: {  
 bookNumberPlaceOK(message.getData());  
 **break**;  
 }  
 **case *BOOK\_NUMBER\_PLACE\_ERROR***: {  
 bookNumberPlaceError(message.getData());  
 **break**;  
 }  
 **case *BOOK\_PLACES\_OK***: {  
 Toast.*makeText*(MainActivity.*context*, **"Выбранные билеты забронированы"**, Toast.***LENGTH\_SHORT***).show();  
 MainActivity.*context*.onBackPressed();  
 **break**;  
 }  
 **case *BUY\_PLACES\_OK***: {  
 Toast.*makeText*(MainActivity.*context*, **"Выбранные билеты куплены"**, Toast.***LENGTH\_SHORT***).show();  
 MainActivity.*context*.onBackPressed();  
 **break**;  
 }  
 }  
 }

Продолжение Приложения В

**private void** initListRays(String json) {  
 **try** {  
 MainActivity.*rays* = Connection.*transformFromJson*(**new** TypeReference<ArrayList<Ray>>() {  
 }, json);  
  
 **if** (MainActivity.*fragment* **instanceof** ItemListRaysFragment && MainWindowFragment.*selectedRay* != **null**) {  
 **boolean** exist = **false**;  
 **for** (Ray ray : MainActivity.*rays*) {  
 **if** (ray.equals(MainWindowFragment.*selectedRay*)) {  
 MainWindowFragment.*selectedRay* = ray;  
 ItemListRaysFragment.*places*.clear();  
 **for** (Place statePlace : ray.**places**) {  
 ItemListRaysFragment.*places*.add(**new** HashMap<String, Object>());  
 }  
 ItemListRaysFragment.*numbersCheckedPlaces* = **new boolean**[MainWindowFragment.*selectedRay*.**places**.**length**];  
 ItemListRaysFragment.*simpleAdapterForPlaces*.notifyDataSetChanged();  
  
 **if** (ray.**stateRay** != StateRay.***NEW***) {  
 ItemListRaysFragment.*buttonBookPlaces*.setVisibility(View.***INVISIBLE***);  
 **if** (ray.**stateRay** != StateRay.***READY***)  
 ItemListRaysFragment.*buttonBuyPlaces*.setVisibility(View.***INVISIBLE***);  
 }  
  
 ItemListRaysFragment.*textCostPlace*.setText(String.*valueOf*(0d));  
 **for** (Place place : MainWindowFragment.*selectedRay*.**places**) {  
 **if** (place.**statePlace** == StatePlace.***BOOK*** && place.**name**.equalsIgnoreCase(MainActivity.*userName*)) {  
 **try** {  
 **double** payment = Double.*parseDouble*(ItemListRaysFragment.*textCostPlace*.getText().toString()) + place.**payment**;  
 ItemListRaysFragment.*textCostPlace*.setText(String.*valueOf*(payment));  
 } **catch** (Exception e) {  
 Toast.*makeText*(MainActivity.*context*, **""** + e.getClass(), Toast.***LENGTH\_SHORT***).show();  
 }  
 }  
 }  
 exist = **true**;  
 **break**;  
 }  
 }  
 **if** (!exist) { *// if ray isn`t exist now* MainActivity.*context*.onBackPressed();  
 }  
 }  
  
 MainWindowFragment.*data*.clear();  
 **for** (Ray ray : MainActivity.*rays*) {  
 MainWindowFragment.*data*.add(**new** HashMap<String, Object>());

Продолжение Приложения В

}

MainWindowFragment.*simpleAdapterForRays*.notifyDataSetChanged();  
  
 **if** (MainActivity.*fragment* **instanceof** MyTicketsFragment) {  
 MainActivity.*connection*.send(**new** Message(MessageType.***GET\_TICKETS\_LIST***));  
 }  
 } **catch** (Exception ignored) {  
 }  
 }  
  
 **private void** myTicketsList(String json) {  
 **try** {  
 MyTicketsFragment.*tickets* = Connection.*transformFromJson*(**new** TypeReference<ArrayList<Ticket>>() {  
 }, json);  
  
 MyTicketsFragment.*myTicketsItems*.clear();  
 **for** (Ticket ticket : MyTicketsFragment.*tickets*) {  
 MyTicketsFragment.*myTicketsItems*.add(**new** HashMap<String, Object>());  
 }  
 MyTicketsFragment.*simpleAdapterTickets*.notifyDataSetChanged();  
 } **catch** (Exception ignored) {  
 Toast.*makeText*(MainActivity.*context*, **""** + ignored, Toast.***LENGTH\_SHORT***).show();  
 }  
 }  
  
 **private void** bookNumberPlaceOK(String json) {  
 **try** {  
 **int** numberPlace = Connection.*transformFromJson*(**new** TypeReference<Ticket>() {  
 }, json).**numberPlace**;  
 Toast.*makeText*(MainActivity.*context*, **"Место №"** + (numberPlace + 1)  
 + **" забронировано"**, Toast.***LENGTH\_SHORT***).show();  
 } **catch** (IOException ignored) {  
 }  
 }  
  
 **private void** bookNumberPlaceError(String json) {  
 **try** {  
 Ticket ticket = Connection.*transformFromJson*(**new** TypeReference<Ticket>() {  
 }, json);  
 Toast.*makeText*(MainActivity.*context*, **"Место №"** + (ticket.**numberPlace** + 1)  
 + **" не удалось забронировать"**, Toast.***LENGTH\_SHORT***).show();  
 } **catch** (IOException ignored) {  
 }  
 }  
  
  
}