Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет инженерно-экономический

Кафедра экономической информатики

Дисциплина «Программирование сетевых приложений»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  старший преподаватель  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Сторожев |
|  | \_\_\_.\_\_\_\_.2022 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«Программная поддержка деятельности автосалона»»**

БГУИР КР 1-40 01 02-08 015 ПЗ

|  |  |
| --- | --- |
|  | Выполнил студент группы 073601  Синицкая Ксения Сергеевна  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовая работа представлена на проверку \_\_\_.\_\_\_\_.2023  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2022

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 6](#_Toc532506648)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 8](#_Toc532506649)

[2 ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЕ РЕШЕНИЯ 10](#_Toc532506650)

[2.1 Постановка задачи 10](#_Toc532506651)

[2.2 Обзор методов решения задачи 10](#_Toc532506652)

[2.2.1 MySQL 10](#_Toc532506653)

[2.2.2 Механизм сокетов 11](#_Toc532506654)

[2.2.3 Команды серверу 12](#_Toc532506655)

[2.2.4 Потоки 12](#_Toc532506656)

[2.2.5 JDBC 12](#_Toc532506657)

[2.2.6 Потоки FileWriter и FileReader 13](#_Toc532506658)

[2.2.7 Механизм авторизации 14](#_Toc532506659)

[2.2.8 JavaFX 14](#_Toc532506660)

[3 ФУНКЦИОНАЛЬНАЯ МОДЕЛЬ И ЕЕ ОПИСАНИЕ 16](#_Toc532506661)

[4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЕ ОПИСАНИЕ 23](#_Toc532506662)

[5 МОДЕЛИ ПРЕДСТАВЛЕНИЯ СИСТЕМЫ И ИХ ОПИСАНИЕ 25](#_Toc532506663)

[5.1 Язык UML 25](#_Toc532506664)

[5.2 Диаграмма вариантов использования 26](#_Toc532506665)

[5.3 Диаграмма развертывания 27](#_Toc532506666)

[5.4 Диаграмма состояний 28](#_Toc532506667)

[5.5 Диаграмма последовательностей 29](#_Toc532506668)

[5.6 Диаграмма компонентов 30](#_Toc532506669)

[5.7 Диаграмма классов 31](#_Toc532506670)

[6 ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ ПРОЕКТИРУЕМОЙ СИСТЕМЫ 33](#_Toc532506671)

[6.1 Команды серверу 33](#_Toc532506672)

[6.2 Потоки 34](#_Toc532506673)

[7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 36](#_Toc532506674)

[7.1 Серверная часть 36](#_Toc532506675)

[7.2 Клиентская часть 36](#_Toc532506676)

[8 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ 47](#_Toc532506677)

[ЗАКЛЮЧЕНИЕ 48](#_Toc532506678)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 49](#_Toc532506679)

[ПРИЛОЖЕНИЕ А 50](#_Toc532506680)

[ПРИЛОЖЕНИЕ Б 53](#_Toc532506681)

[ПРИЛОЖЕНИЕ В 55](#_Toc532506682)

[ПРИЛОЖЕНИЕ Г 66](#_Toc532506683)

# ВВЕДЕНИЕ

В настоящее время число автомобилей на улицах увеличивается, следовательно, и спрос на них растет соответственно. Во все времена автомобиль был и остается популярным средством передвижения. Существует множество фирм по продаже автомобилей. При покупке нового автомобиля клиент желает получить качественный товар как можно быстрее и с наименьшим количеством бумажной волокиты. При выполнении этих условий клиент остается доволен, а это значит, что у компании будет на одного довольного клиента больше, возможно, этот довольный клиент приведет еще одного, а может даже и не одного клиента, репутация компании улучшится, количество клиентов будет расти, а, соответственно, и прибыль. А ведь главное для фирмы – это клиенты, не будет клиентов, не будет и фирмы. Если качество товара в большей степени зависит от производителя, то скорость получения машины клиентом и количество бумаг для покупки в большей степени зависит от компании-продавца. В эпоху информационных технологий одним из самых эффективных способов ускорения выполнения тривиальных задач является внедрение программного обеспечения, задачей которого будет ускорение и упрощение выполнения этих задач. А поскольку от качества данного программного обеспечения зависит репутация целой кампании, то и требования к качеству, быстродействию, отказоустойчивости данной программы велики. Программа также будет работать с огромным количеством информации: данные о клиентах, сотрудниках, товарах и т.д., поэтому ее работу невозможно представить без использования баз данных.

Современный мир информационных технологий трудно представить себе без использования баз данных. Практически все системы в той или иной степени связаны с функциями долговременного хранения и обработки информации. Фактически информация становиться фактором, определяющим эффективность любой сферы деятельности. Увеличились информационные потоки и повысились требования к скорости обработки данных, и теперь уже большинство операций не может быть выполнено вручную, они требуют применения наиболее перспективных компьютерных технологий. Любые административные решения требуют четкой и точной оценки текущей ситуации и возможных перспектив ее изменения. И, если раньше в оценке ситуации участвовало несколько десятков факторов, которые могли быть вычислены вручную, то теперь таких факторов сотни и сотни тысяч, и ситуация меняется не в течение года, а через несколько минут, а обоснованность принимаемых решений требуется большая, потому что и реакция на неправильные решения более серьезная, более быстрая и более мощная, чем раньше. И, конечно, обойтись без информационной модели производства, хранимой в базе данных, в этом случае невозможно.

Как правило, базы данных создаются обычно не для решения какой-либо одной задачи для одного пользователя, а для многоцелевого использования. Базы данных отражают определенную часть реального мира. Эта информация должна по возможности фиксироваться в базе данных однократно, и все пользователи, которым эта информация нужна, должны иметь возможность работать с ней. Отсутствие централизованных методов управления доступом к информации послужило причиной разработки систем управления базой данных, а сами хранилища информации, которые работали под управлением данных систем, назывались базами или банками данных.

*Задачей* курсового проекта является изучение предметной области, разработка функциональной модели системы, диаграммы классов, приятного и понятного интерфейса, программного обеспечения, руководства пользователя, описание принципов работы приложения, тестирование программного обеспечения.

*Целью* является улучшение показателей автосалона и привлечение новых покупателей.

# 1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Предметной областью выбранного мною проекта является продажа автомобилей в автосалоне. Автосалон – это некий аналог магазина, в котором представлены существующие образцы автомобилей. Основными элементами предметной области являются поставляемые автомобили, которые обозначены следующими признаками – модель, комплектация, объем двигателя, цвет, цена.

Тема покупок автомобилей очень популярна в наше время, так как сам предмет – автомобиль пользуется большой популярностью у населения. Автомобиль давно перестал быть роскошью и стал, скорее, необходимостью для комфортной жизни. В наше время для покупки нового авто даже не обязательно иметь на руках всю сумму – огромное количество банков предлагает неплохие условия кредитования, на рынке имеется большое количество компаний, предлагающих покупку в лизинг, не стоит забывать и про трэйд ин. Кроме этого есть и другие факторы, влияющие на решение поменять старый авто на новый.

Автомобиль стал слишком старым. Некоторые умельцы говорят, что ни одна машина не может стать слишком старой, главное – правильно ее содержать. Однако легковой автомобиль, как и любое другое устройство или механизм, со временем изнашивается и начинает требовать более частого и дорогостоящего ремонта. Поэтому покупатели стараются поменять предыдущую машину на абсолютно новую, как только заканчивается гарантийный срок, – так обычно поступают предприятия, пользующиеся служебным транспортом.

Автомобили различных марок и классов стареют по-разному, однако рано или поздно наступает момент, когда прогнозировать эксплуатационные издержки становится невозможно. Несмотря на то что запчасти для старых и очень популярных моделей дешевы, тем не менее, в каждом автомобиле есть неадекватно дорогой узел, ремонт или замена которого не является целесообразной.

Следует помнить о том, что легковой автомобиль, в отличие от коммерческого транспорта, не сконструирован для больших пробегов, и миллион километров, скорее всего, не проедет. И вообще: если машина начинает требовать слишком много внимания, значит, пришло время с ней расстаться.

Машина не соответствует жизненной ситуации. В жизни все течет и меняется. Довольно часто случается так, что имеющийся автомобиль не соответствует новому месту работы, семейному положению или новому хобби. Чаще всего предыдущая машина просто оказывается слишком мала для подрастающих детей или горных велосипедов, однако случается и наоборот – ранее полноценно используемый минивэн, микроавтобус или универсал теперь преимущественно возит только водителя и воздух. Иногда смене машины способствует переезд за город – когда ежедневно приходится проделывать долгий путь, это мотивирует выбирать двигатель маленький, более экономичный и, возможно, работающий на другом топливе.

Содержание автомобиля требует слишком больших вложений. Автомобиль может превратиться в пожирателя денег – как из-за старения, так и по другим причинам. Например, может оказаться, что приобретенная машина мечты вам совсем не по карману. У новых автомобилей это обнаруживается при первом же большом техосмотре, а во сколько обойдется данное мероприятие, в момент покупки машины вам скажет только очень самоотверженный продавец.

О том, сколько будет стоить техническое содержание машины, например, до пробега 100 000 км, обычно интересуются только водители автопарков, потому, что это их работа. Бывает, что даже у маленького автомобиля расход горючего намного выше, чем тот, что указан в технической информации, и изменить это не сможет ни посещение сервиса, ни уроки эковождения. Как бы ситуация ни развивалась до того, ясно одно – если автомобиль оказался слишком дорог, не надо откладывать принятие неизбежного решения, потому что тянуть резину значит еще больше разорять финансовый огород и красть средства для покупки новой машины.

Не устраивает выбранная марка или моделью. Независимо от возраста автомобиля – будь это десять дней или десять лет – всегда существует возможность приобрести то, что вам не нравится или не подходит. Самое смешное, что неприязнь начинается с абсолютно неважных и внешне незаметных мелочей. Например, со временем выясняется, что сиденья невозможно отрегулировать так, как нравится вашей спине. Казалось бы, пустяк, но во время длительной поездки неудобство отлично чувствуется, и тянущее ощущение не компенсирует никакая динамика или хорошая аудиосистема.

# 2 ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЕ РЕШЕНИЯ

## 2.1 Постановка задачи

Для успешной реализации системы необходимо, выделить основные задачи, которые будет решать приложение, а также те задачи, которые необходимо выполнить для правильной работы программы.

Основными задачами являются следующие:

1. разработка базы данных для MySQL Server, используя пакеты программы MySQL Workbench;
2. программа должна работать в рамках архитектуры клиент-сервер;
3. бизнес-логика системы должна быть реализована только на серверной части приложения;
4. на сервере должна быть предусмотрена возможность параллельной обработки запросов;
5. доступ к данным в СУБД должен осуществляться через драйвер, предоставленный производителем СУБД или через использование специальных технологий;
6. предусмотреть сохранение данных в файл;
7. предусмотреть механизм авторизации пользователей;
8. программа должна предоставлять удобный и минималистичный интерфейс для конечного пользователя.

## 2.2 Обзор методов решения задачи

### 2.2.1 MySQL

Для разработки базы данных использовался MySQL Server версии 8.0.13 предоставляет удобный графический интерфейс для работы с базой данных: подключение к базе данных, создание таблиц, полей, ключей, связей, внесение данных в таблицы и т.д. На рисунке 2.1 представлен интерфейс программы MySQL.

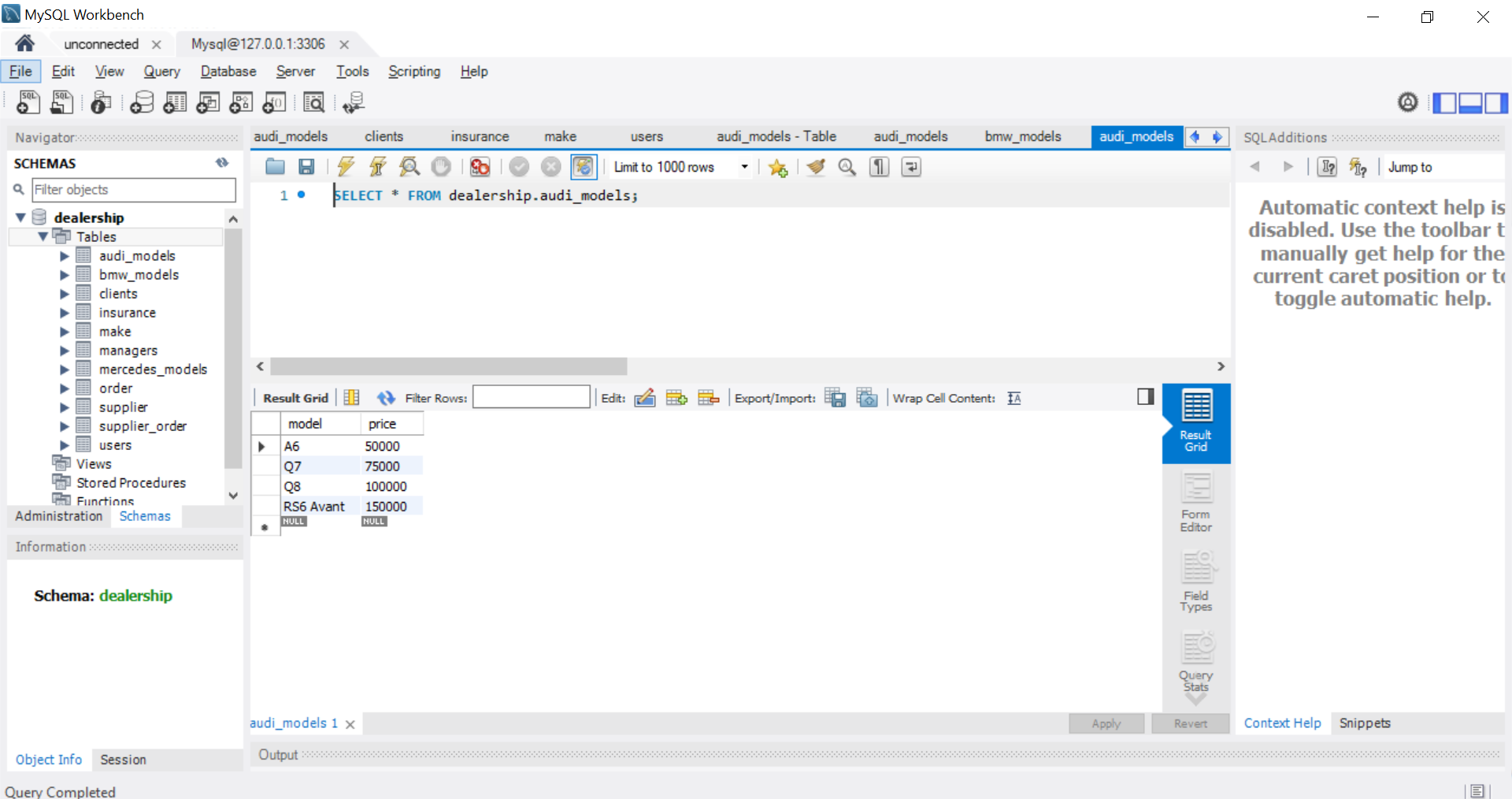


Рисунок 2.1 - Интерфейс программы MySQL

### 2.2.2 Механизм сокетов

Для реализации архитектуры клиент-сервер используется встроенный в язык программирования Java механизм сокетов. Сокет (гнездо, разъем) – это программная абстракция, используемая для представления «терминалов» соединений между двумя машинами.

Каждый из сокетов определяется типом и ассоциированным с ним процессом. Реально для передачи организуются определенные дескрипторы *TCP*-соединения, так называемые гнезда (*socket*): гнездо сервера и гнездо клиента, которые в Internet-домене включают в себя *IP*-адреса сервера и клиента и номера портов, через которые они взаимодействуют. Сервер обычно имеет закрепленный и постоянный во взаимодействии номер порта, а клиенту, обращающемуся по этому номеру для связи к серверу, назначается некоторый другой (эфемерный) номер порта после установления соединения с сервером на сеанс их взаимодействия. Таким образом основной порт освобождается для установления последующих связей (номер порта выбирается сервером из числа не занятых в диапазоне от 1024 до 65 535). Эта комбинация (*IP*-адрес и номера портов) однозначно определяет отдельные сетевые процессы в сети *Internet* (номера портов до 1024, как правило, резервируются для широко известных приложений, например, 80 – для связывания с серверами *Web* по протоколу *HTTP*).

### 2.2.3 Команды серверу

Для реализации бизнес-логики исключительно на серверной части приложения используется механизм, когда на сервере хранится набор команд, клиент отправляет эти команды, когда сервер получает ее, то выполняет предопределенный набор действий и отправляет клиенту результат.

### 2.2.4 Потоки

Для реализации механизма параллельной обработки запросов используются потоки. В отличие от большинства других машинных языков *Java* обеспечивает встроенную поддержку для многопоточного программирования. Многопоточная программа содержит две и более части, которые могут выполняться одновременно, конкурируя друг с другом. Каждая часть такой программы называется потоком, а каждый поток определяет отдельный путь выполнения (в последовательности операторов программы).

Потоки существуют в нескольких состояниях. Поток может быть в состоянии выполнения. Может находиться в состоянии готовности к выполнению, как только он получит время *CPU*. Выполняющийся поток может быть приостановлен, что временно притормаживает его действие. Затем приостановленный поток может быть продолжен (возобновлен) с того места, где он был остановлен. Поток может быть блокирован в ожидании ресурса. В любой момент выполнение потока может быть завершено, что немедленно останавливает его выполнение. После завершения поток не может быть продолжен.

Многопоточная система Java построена на классе *Thread*, его методах и связанном с ним интерфейсе *Runnable*. *Thread* инкапсулирует поток выполнения. Так как невозможно непосредственно обращаться к внутреннему состоянию потока выполнения, то взаимодействие с ним осуществляется через его полномочного представителя — экземпляр (объект) класса *Thread*, который его породил. Чтобы создать новый поток, выбранная программа должна будет или расширять Класс *Thread* или реализовывать интерфейс *Runnable*.

### 2.2.5 JDBC

Для работы с базой данный используется API JDBC (Java Database Connectivity) – платформенно-независимый промышленный стандарт взаимодействия Java-приложений с различными [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94), реализованный в виде пакета java.sql, входящего в состав [Java SE](https://ru.wikipedia.org/wiki/Java_SE). JDBC основан на концепции так называемых драйверов, позволяющих получать соединение с базой данных по специально описанному URL. Драйверы могут загружаться динамически (во время работы программы). Загрузившись, драйвер сам регистрирует себя и вызывается автоматически, когда программа требует URL, содержащий протокол, за который драйвер отвечает. Компоненты JDBC приведены на рисунке 2.2.

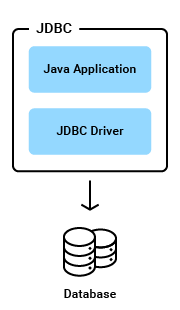


Рисунок 2.2 - Компоненты Java Database Connectivity

### 2.2.6 Потоки FileWriter и FileReader

Для реализации сохранения данных в файл используются потоки FileWriter и BufferedFileWriter. Схема записи информации в файл используя класс FileWriter приведена на рисунке 2.3.

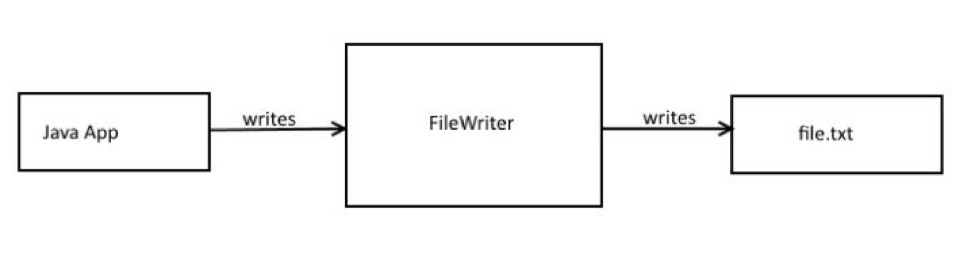


Рисунок 2.3 - Схема записи информации в файл

### 2.2.7 Механизм авторизации

Для реализации механизма авторизации пользователей используется заданный в отдельной таблице базы данных тип учетной записи. Таблица состоит из четырех полей:

* + - id;
    - account\_type;
    - login;
    - password.

При правильном вводе логина и пароля сервер посылает клиенту тип его аккаунта.

### 2.2.8 JavaFX

Для создания графического интерфейса используется платформа JavaFX. JavaFX — платформа на основе Java для создания приложений с насыщенным графическим интерфейсом. Может использоваться как для создания настольных приложений, запускаемых непосредственно из-под операционных систем, так и для интернет-приложений (RIA), работающих в браузерах, и для приложений на мобильных устройствах. JavaFX призвана заменить использовавшуюся ранее библиотеку Swing. Платформа JavaFX конкурирует с Microsoft Silverlight, Adobe Flash и аналогичными системами. Технология JavaFX была впервые продемонстрирована корпорацией Sun Microsystems на конференции JavaOne в мае 2007 года. 4 декабря 2008 года вышла версия 1.0 платформы, содержащая следующие компоненты: средства разработки — компилятор и среда исполнения JavaFX, язык программирования JavaFX Script, а также графические, медийные и веб-библиотеки для создания RIA-приложений для настольных компьютеров, веб-сайтов и мобильных устройств; интегрированная среда разработки NetBeans IDE (версии 6.\*) — средства для кодирования и отладки приложений, написанных на JavaFX Script. В редакторе JavaFX Script есть возможность быстрого добавления объектов JavaFX с уже готовыми геометрическими фигурами, компонентами интерфейса пользователя, средствами преобразования и анимацией; Production Suite — набор инструментов и плагинов для импорта графических объектов в приложения JavaFX. Включает следующие компоненты: плагины для графических редакторов Adobe Photoshop CS3, CS4 и Adobe Illustrator CS3, CS4. С помощью плагинов можно экспортировать графические объекты из этих приложений в код JavaFX Script; Media Factory: набор инструментов для конвертирования SVG-графики в код JavaFX и просмотра графических объектов, импортированных в JavaFX из других форматов. Также включает примеры приложений, учебные курсы, статьи, API-документацию и примеры кода.

Приложения JavaFX создаются с помощью декларативного языка программирования JavaFX Script. Для разработки приложений на языке JavaFX Script необходимо установить JavaFX SDK, который входит в поставку с Java SE 7. Из кода, написанного на языке JavaFX Script, можно обращаться к любым библиотекам Java. Поэтому совместное использование языков Java и JavaFX Script позволяет решать разнообразные задачи, например, логика бизнес-приложения может быть написана на Java, а графический интерфейс пользователя — на JavaFX Script. Пример графического интерфейса, написанного на JavaFX, приведен на рисунке 2.5.

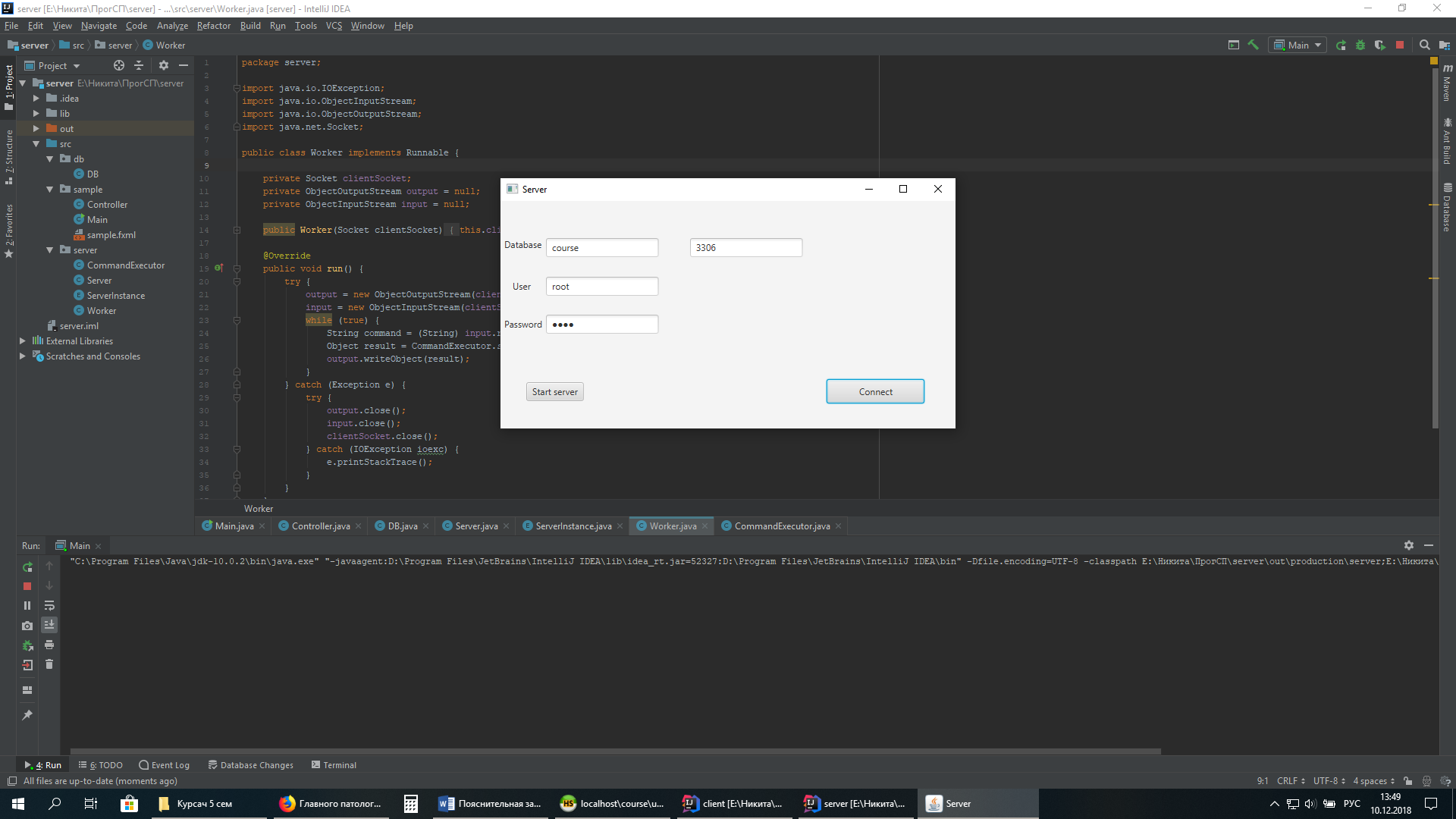


Рисунок 2.5 - Пример графического интерфейса JavaFX

# 3 ФУНКЦИОНАЛЬНАЯ МОДЕЛЬ И ЕЕ ОПИСАНИЕ

Построим контекстную диаграмму, которая предоставит общее описание работы системы по продаже автомобилей автосалона. На рисунке 3.1 представлена контекстная диаграмма программы по автоматизации системы учета продаж автомобилей в автосалонах.

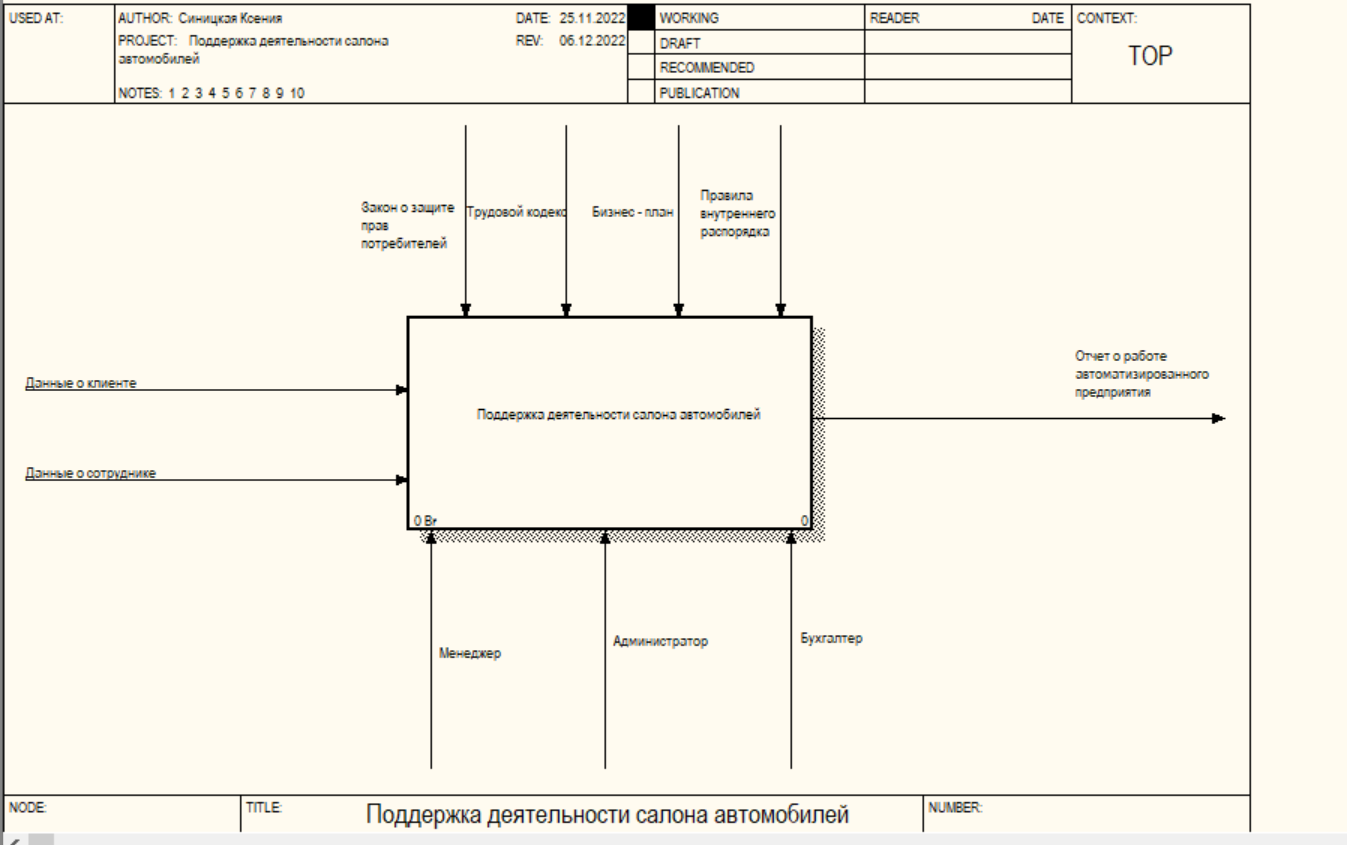


Рисунок 3.1 - Контекстная диаграмма программы по автоматизации системы учета продаж автомобилей в автосалонах

На вход идут «Данные о клиенте» и «Данные о сотруднике». На выходе получаем «Отчет о работе автоматизированного предприятия». Ресурсами для выполнения являются: «Менеджер», который осуществляет работу с клиентами, «Администратор», который осуществляет работу с сотрудниками, «Бухгалтер», задача которого составить отчет о предметной области фирмы.

Затем идет декомпозиция на четыре блока: «Зарегистрировать заказ», «Осуществлять работу с сотрудниками», «Выполнить заказ», «Управлять финансами». На рисунке 3.2 представлена декомпозиция системы по продаже автомобилей в автосалонах.

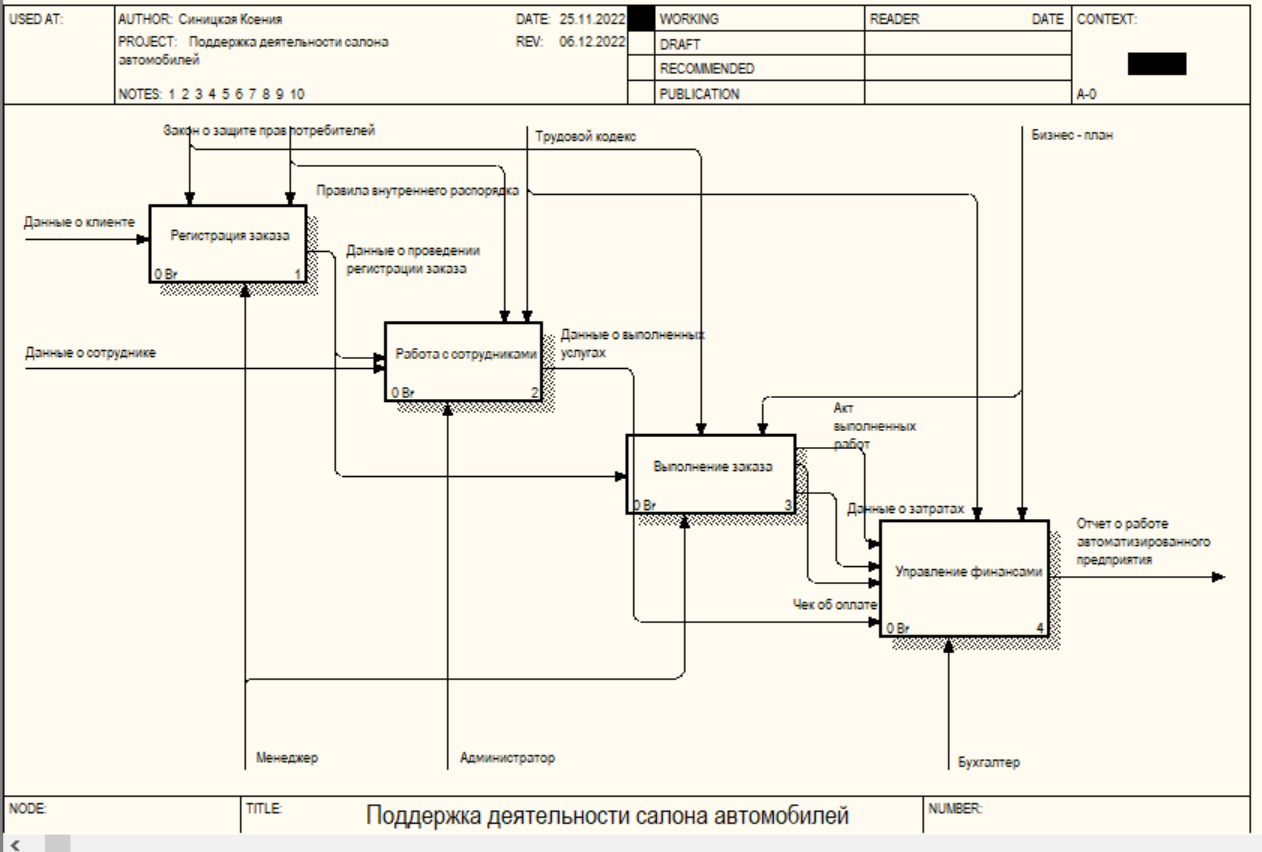


Рисунок 3.2 - Декомпозиция системы по продаже автомобилей в автосалонах

Для регистрации заказа: на входе имеем «Данные о клиенте», на выходе получаем «Данные о проведении заказа», которые переходят в блок «Осуществлять работу с сотрудниками» и «Выполнить заказ». Регистрирует заказ «Менеджер», опираясь на «Правила внутреннего распорядка» и «Закон о защите прав потребителей».

В блоке «Осуществлять работу с сотрудниками»: на входе имеем «Данные о сотруднике» и «Данные о проведении заказа», на выходе «Данные о выполненных услугах», которые переходят в блок «Управлять финансами». Осуществляет работу «Администратор», опираясь на «Правила внутреннего распорядка» и «Трудовой кодекс».

Блок «Выполнить заказ»: на входе имеем «Данные о проведении заказа», на выходе «Данные о затратах на проведение услуги», «Акт выполненных работ», «Чек об оплате» и «Отчет о качестве проделанной работы», которые переходят в блок «Управлять финансами». Работу осуществляет «Менеджер», опираясь на «Закон о защите прав потребителей», «Закон о бухгалтерском учете и составление отчетов» и «Бизнес-план».

Блок «Управлять финансами»: на входе имеет «Данные о выполненных услугах», «Данные о затратах на проведение услуги», «Акт выполненных работ», «Чек об оплате» и «Отчет о качестве проделанной работы», на выходе «Отчет о работе автоматизированного предприятия». Управляет финансами бухгалтер, опираясь на «Закон о бухгалтерском учете», «Составление отчетов», «Трудовой кодекс» и «Бизнес-план».

Блок «Зарегистрировать заказ» разбивается на еще на три блока. На рисунке 3.3 представлена декомпозиция блока зарегистрировать заказ.

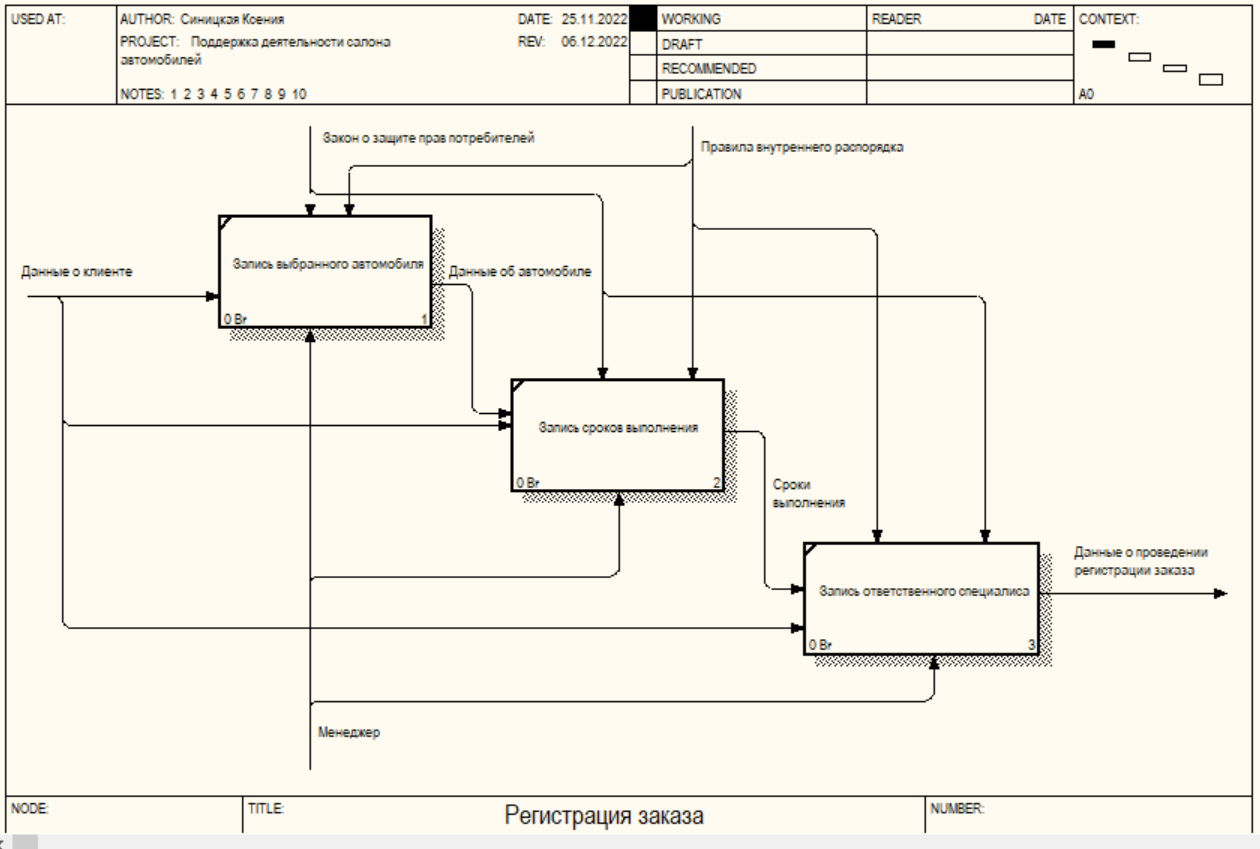


Рисунок 3.3 - Декомпозиция блока «Регистрация заказа»

Для того, чтобы записать выбранный автомобиль, на входе имеем «Данные о клиенте», на выходе получаем «Данные об автомобиле», который переходит в блок «Записать сроки выполнения». Процесс выполняется «Менеджером», опираясь на «Правила внутреннего распорядка» и «Закон о защите прав потребителей».

В блоке «Записать сроки выполнения» на входе имеется «Данные о клиенте» и «Данные об автомобиле». Процесс выполняет «Менеджер», который руководствуется «Правилами внутреннего распорядка» и «Законом о защите прав потребителей». На выходе получаем «Сроки выполнения заказа», которые переходят в блок «Записать ответственного специалиста».

Блок «Записать ответственного специалиста»: на входе имеем «Сроки выполнения заказа» и «Данные о клиенте». Выполняет подбор «Менеджер», опираясь на «Правила внутреннего распорядка» и «Закон о защите прав потребителей». На выходе «Данные о проведении заказа»

Блок «Осуществлять работу с сотрудниками» также разбивается на три блока. Декомпозиция блока «Осуществлять работу с сотрудниками» представлена на рисунке 3.4.

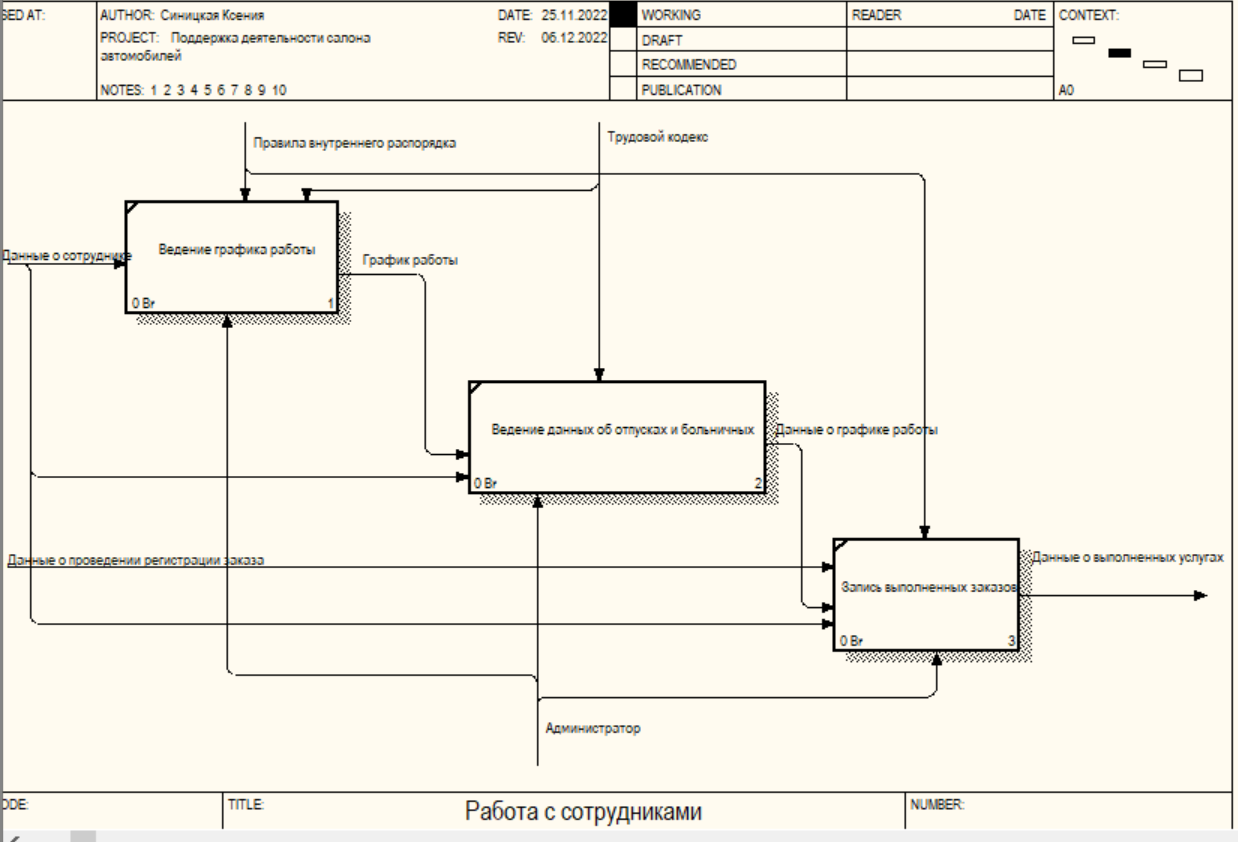


Рисунок 3.4 - Декомпозиция блока «Осуществлять работу с сотрудниками»

Блок «Вести график работы»: на входе имеем «Данные о сотруднике» и «Данные о проведении заказа», на выходе «График работы», который переходит в блок «Вести данные об отпусках и больничных». Работу выполняет «Администратор», опираясь на «Правила внутреннего распорядка» и «Трудовой кодекс».

Блок «Вести данные об отпусках и больничных»: на входе имеем «Данные о сотруднике» и «График работы», на выходе «Данные о графике работы», которые переходят в блок «Записать выполненные заказы». Работу выполняет «Администратор», опираясь на «Трудовой кодекс».

Блок «Записать выполненные заказы»: на входе имеем «Данные о проведении заказа» и «Данные о графике работы», на выходе «Данные о выполненных услугах». Выполняет работу «Администратор», опираясь на «Правила внутреннего распорядка».

Блок выполнить заказ разбивается на четыре блока. Декомпозиция блока «Выполнить заказ» представлена на рисунке 3.5.

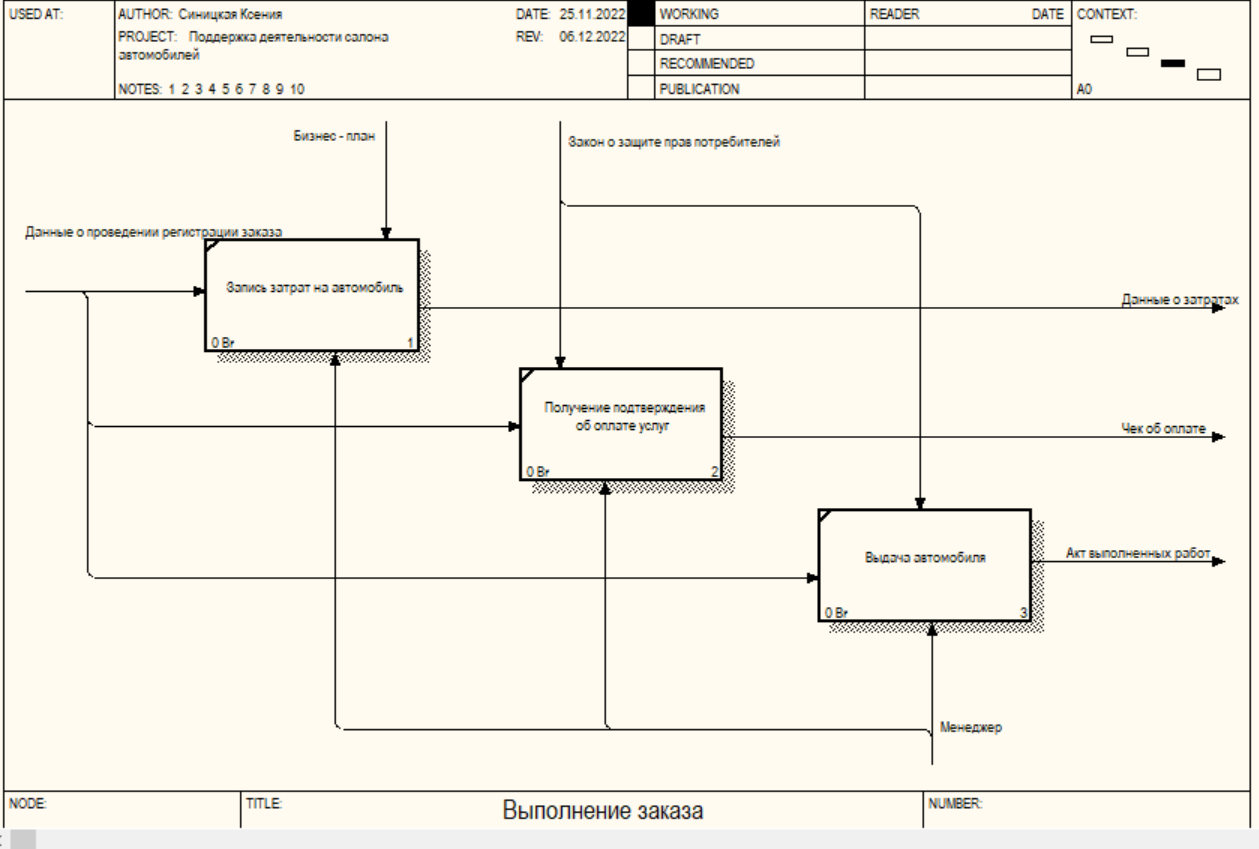


Рисунок 3.5 - Декомпозиция блока «Выполнить заказ»

Блок «Записать затраты на исполнение заказа»: на входе имеем «Данные о проведении заказа», на выходе «Данные о затратах на проведение услуги». Выполняет запись «Менеджер», опираясь на «Бизнес-план».

Далее идет блок «Получить подтверждение об оплате услуг»: на выходе имеем «Данные о проведении заказа», на выходе «Чек об оплате». Получает подтверждение «Менеджер», опираясь на «Закон о бухгалтерском учете» и «Составление отчетов».

Блок «Выдать автомобиль»: на входе «Данные о проведении заказа», на выходе «Акт выполненных работ». Работу выполняет «Менеджер», опираясь на «Закон о защите прав потребителей».

Блок «Управлять финансами» разбивается на три блока. Декомпозиция блока «Управлять финансами» представлена на рисунке 3.6.

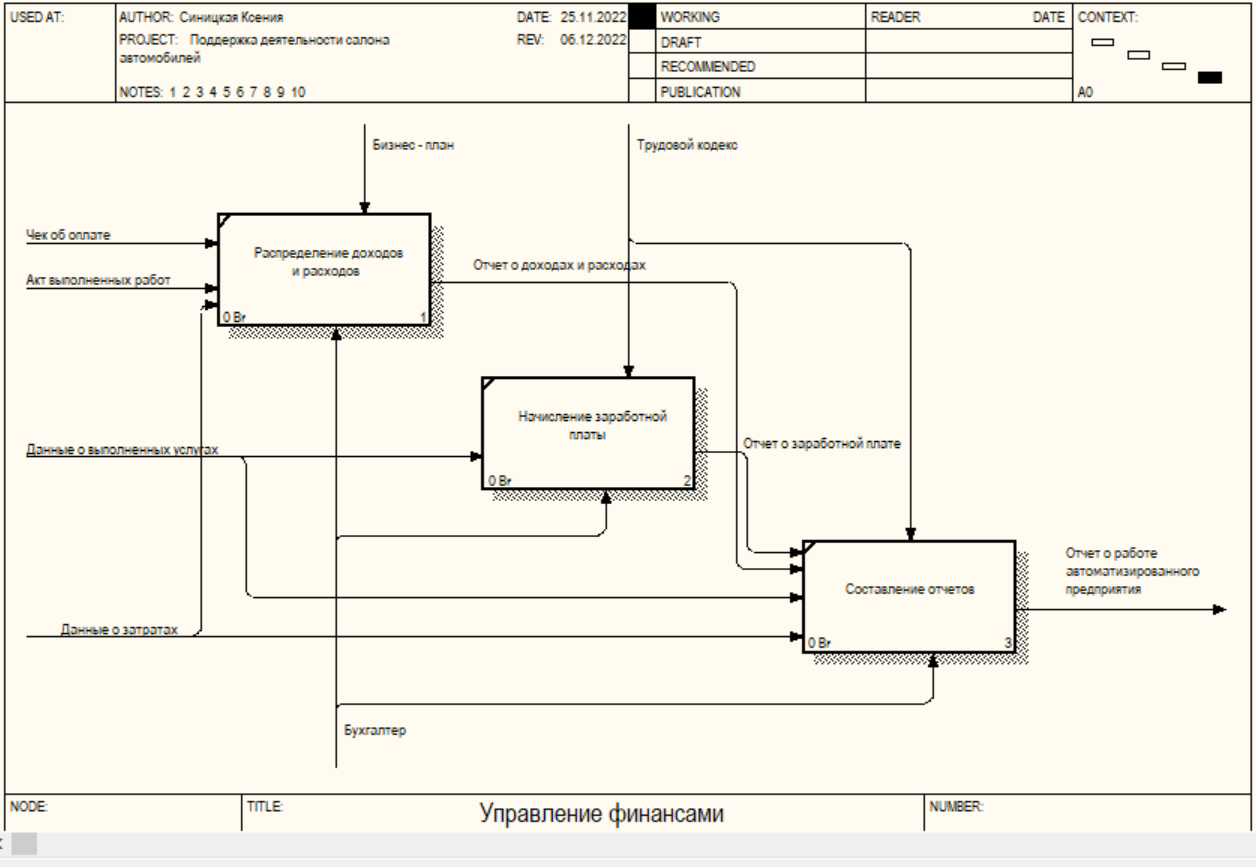


Рисунок 3.6 - Декомпозиция блока «Управлять финансами»

Блок «Распределить доходы и расходы»: на входе имеем «Данные о затратах на проведение услуги», «Акт выполненных работ» и «Чек об оплате», на выходе «Отчет о доходах и расходах», который переходит в блок «Составить отчеты». Доходы и расходы распределяет «Бухгалтер», опираясь на «Бизнес-план».

Блок «Начислить заработную плату»: на входе имеем «Данные о выполненных услугах», на выходе «Отчет о заработной плате», который переходит в блок «Составить отчеты». Начисляет заработную плату «Бухгалтер», опираясь на «Трудовой кодекс» и «Закон о бухгалтерском учете и составление отчетов».

Блок «Составить отчеты»: на входе имеем «Отчет о заработной плате» и «Отчет о доходах и расходах», на выходе «Отчет о работе автоматизированного предприятия». Составляет отчеты «Бухгалтер», опираясь на «Трудовой кодекс» и «Закон о бухгалтерском учете и составление отчетов».

# 4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЕ ОПИСАНИЕ

Информационная модель системы представляет собой базу данных на основе MySQL Server версии 8.0.13. Информационная модель представлена на рисунке 4.1.

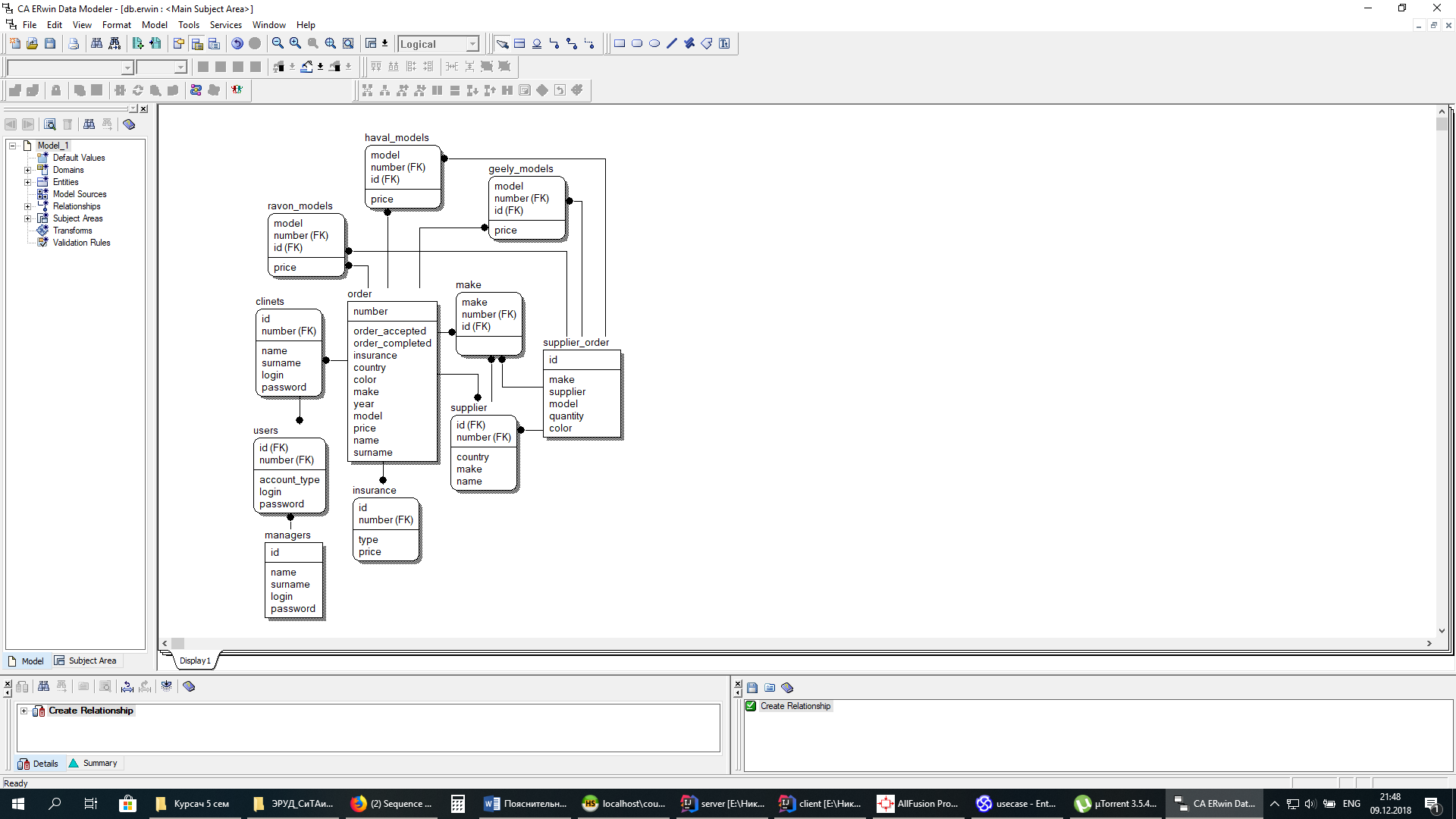


Рисунок 4.1 – Информационная модель

База данных состоит из одиннадцати таблиц. Таблицы geely\_models, haval\_models и ravon\_models содержат два поля: model и price, в них хранится название модели автомобиля и ее стоимость. Поле model является ключевым.

Таблица make хранит названия марок автомобилей. Ключевым полем является make.

Таблица insurance содержит три поля: id, type и price. В ней хранится информация о цене и типе страховки. Ключевым полем является id.

Таблицы managers и clients содержат пять полей: id, name, surname, login и password. В них хранится имя, фамилия, логин и пароль клиента/менеджера. Поле id является ключевым. Таблицы связаны с таблицей users.

Таблица users состоит из четырех полей: id, account\_type, login и password. В ней хранятся все учетные записи пользователей (логин и пароль), а также тип учетной записи конкретного пользователя.

Таблица supplier содержит четыре поля: id, country, make и name. Ключевым полем является id. В таблице хранится информация о поставщиках. Таблица связана с таблицей make.

Таблица supplier\_order состоит из шести полей: id, make, supplier, model, quantity и color. В ней хранится информация о заказах к поставщикам. Поле id является ключевым. Таблица связана с таблицами make, geely\_models, haval\_models, ravon\_models и supplier.

Таблица order состоит из двенадцати полей: number, order\_accepted, unsurance, country, color, make, year, order\_completed, model, price, name и surname. В ней хранится информация о заказах от клиентов. Таблица связана с таблицами make, geely\_models, haval\_models, ravon\_models, supplier, insurance и clients.

# 5 МОДЕЛИ ПРЕДСТАВЛЕНИЯ СИСТЕМЫ И ИХ ОПИСАНИЕ

## 5.1 Язык UML

Унифицированный язык моделирования (UML) является стандартным инструментом для создания «чертежей» программного обеспечения. С помощью UML можно визуализировать, специфицировать, конструировать и документировать артефакты программных систем.

UML пригоден для моделирования любых систем: от информационных систем масштаба предприятия до распределенных Web-приложений и даже встроенных систем реального времени. UML – это язык для визуализации, специфицирования, конструирования и документирования артефактов программных систем. Язык моделирования, подобный UML, является стандартным средством для составления «чертежей» программного обеспечения.

Для понимания любой нетривиальной системы приходится разрабатывать большое количество взаимосвязанных моделей. В применении к программным системам это означает, что необходим язык, с помощью которого можно с различных точек зрения описать представления архитектуры системы на протяжении цикла ее разработки.

UML – это графический язык специфицирования, что означает построение точных и полных графических моделей, касающиеся анализа, проектирования и реализации, которые должны приниматься в процессе разработки и развертывания системы программного обеспечения.

UML – это язык конструирования, и модели, созданные с его помощью, могут быть непосредственно переведены на различные языки программирования. Иными словами, UML-модель можно отобразить на такие языки, как Java, C++, Visual Basic, и даже на таблицы реляционной базы данных.

Такое отображение модели на язык программирования позволяет осуществлять прямое проектирование: генерацию кода из модели UML в какой-то конкретный язык. Можно решить и обратную задачу: реконструировать модель по имеющейся реализации.

UML позволяет решить проблему документирования системной архитектуры и всех ее деталей, предлагает язык для формулирования требований к системе и определения тестов и, наконец, предоставляет средства для моделирования работ на этапе планирования проекта и управления версиями.

Сфера применения UML не ограничивается моделированием ПО. Он позволяет моделировать, скажем, документооборот в юридических системах, структуру и функционирование системы обслуживания пациентов в больницах, осуществлять проектирование аппаратных средств.

## 5.2 Диаграмма вариантов использования

Этот вид диаграмм позволяет создать список операций, которые выполняет система. Часто этот вид диаграмм называют диаграммой функций, потому что на основе набора таких диаграмм создается список требований к системе и определяется множество выполняемых системой функций.

Диаграммы вариантов использования описывают функциональное назначение системы или то, что система должна делать. Разработка диаграммы преследует следующие цели:

* определить общие границы и контекст моделируемой предметной области;
* сформулировать общие требования к функциональному поведению проектируемой системы;
* разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
* подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть диаграммы вариантов использования состоит в следующем. Проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью вариантов использования. При этом актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. Вариант использования служит для описания сервисов, которые система предоставляет актеру.

Цель варианта использования заключается в том, чтобы определить законченный аспект или фрагмент поведения некоторой сущности без раскрытия её внутренней структуры. В качестве такой сущности может выступать система или любой элемент модели, который обладает собственным поведением.

Каждый вариант использования соответствует отдельному сервису, который предоставляет моделируемая сущность по запросу актера, то есть определяет способ применения этой сущности. Сервис, который инициализируется по запросу актера, представляет собой законченную неделимую последовательность действий. Это означает, что после того как система закончит обработку запроса, она должна возвратиться в исходное состояние, чтобы быть готовой к выполнению следующих запросов.

Варианты использования могут применяться как для спецификации внешних требований к проектируемой системе, так и для спецификации функционального поведения уже существующей системы. Множество вариантов использования в целом должно определять все возможные стороны ожидаемого поведения системы. Кроме этого, варианты использования неявно устанавливают требования, определяющие, как актеры должны взаимодействовать с системой, чтобы иметь возможность корректно работать с предоставляемыми сервисами. Для удобства множество вариантов использования может рассматриваться как отдельный пакет.

Примерами вариантов использования могут являться следующие действия: проверка состояния текущего счета клиента, оформление заказа на покупку товара, получение дополнительной информации о кредитоспособности клиента, отображение графической формы на экране монитора и другие действия.

Диаграмма вариантов использования размещена в приложении А на рисунке А.1.

Программа подразумевает три пользователя: администратор, менеджер и клиент.

Администратор добавляет менеджеров, страховку и поставщиков. Менеджер добавляет заказы поставщикам, подтверждает заказы клиентов. Клиент может добавить заказ и просмотреть свои заказы.

## 5.3 Диаграмма развертывания

Этот вид диаграмм предназначен для анализа аппаратной части системы, то есть "железа", а не программ. В прямом переводе с английского Deployment означает "развертывание", но термин "топология" точнее отражает сущность этого типа диаграмм.

Физическое представление программной системы не может быть полным, если отсутствует информация о том, на какой платформе и на каких вычислительных средствах она реализована. Если разрабатывается программа, выполняющаяся локально на компьютере пользователя и не использующая периферийных устройств и ресурсов, то в разработке дополнительных диаграмм нет необходимости. При разработке же корпоративных приложений наличие таких диаграмм может быть крайне полезным для решения задач рационального размещения компонентов в целях эффективного использования распределенных вычислительных и коммуникационных ресурсов сети, обеспечения безопасности и других.

Для представления общей конфигурации и топологии распределенной программной системы в UML предназначены диаграммы развертывания.

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения (runtime). При этом представляются только компоненты-экземпляры программы, являющиеся исполняемыми файлами или динамическими библиотеками. Те компоненты, которые не используются на этапе исполнения, на диаграмме развертывания не показываются. Так, компоненты с исходными текстами программ могут присутствовать только на диаграмме компонентов. На диаграмме развертывания они не указываются.

Диаграмма развертывания содержит графические изображения процессоров, устройств, процессов и связей между ними. В отличие от диаграмм логического представления, диаграмма развертывания является единой для системы в целом, поскольку должна всецело отражать особенности ее реализации. Разработка диаграммы развертывания, как правило, является последним этапом спецификации модели программной системы.

При разработке диаграммы развертывания преследуют следующие цели:

* определить распределение компонентов системы по ее физическим узлам;
* показать физические связи между всеми узлами реализации системы на этапе ее исполнения;
* выявить узкие места системы и реконфигурировать ее топологию для достижения требуемой производительности.

Диаграммы развертывания разрабатываются совместно системными аналитиками, сетевыми инженерами и системотехниками.

Диаграмма развертывания представлена в приложении А на рисунке А.2.

## 5.4 Диаграмма состояний

Каждая диаграмма состояний в UML описывает все возможные состояния одного экземпляра определенного класса и возможные последовательности его переходов из одного состояния в другое, то есть моделирует все изменения состояний объекта как его реакцию на внешние воздействия.

Диаграммы состояний чаще всего используются для описания поведения отдельных объектов, но также могут быть применены для спецификации функциональности других компонентов моделей, таких как варианты использования, актеры, подсистемы, операции и методы.

Диаграмма состояний является графом специального вида, который представляет некоторый автомат. Вершинами графа являются возможные состояния автомата, изображаемые соответствующими графическими символами, а дуги обозначают его переходы из состояния в состояние. Диаграммы состояний могут быть вложены друг в друга для более детального представления отдельных элементов модели.

Диаграмма состояний приведена в приложении А на рисунке А.3. Диаграмма описывает процесс авторизации пользователя. Для того чтобы зайти в систему необходимо иметь учетную запись. После ввода данных они отправляются на сервер и сравниваются с записями в базе данных, если совпадения найдены, пользователь успешно входит и может пользоваться системой, в противном случае появится сообщение об ошибке и для входа потребуется ввод правильных учетных данных.

## 5.5 Диаграмма последовательностей

При рассмотрении диаграмм состояния и деятельности, было отмечено, что хотя эти диаграммы и используются для спецификации динамики поведения систем, время в явном виде в них не присутствует. Временной же аспект поведения может иметь существенное значение при моделировании синхронных процессов, описывающих взаимодействия объектов. Для моделирования взаимодействия объектов во времени в языке UML используются диаграммы последовательности.

На диаграмме последовательности изображаются только те объекты, которые непосредственно участвуют во взаимодействии. Ключевым моментом для диаграмм последовательности является динамика взаимодействия объектов во времени.

В UML диаграмма последовательности имеет как бы два измерения. Первое слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии. Крайним слева на диаграмме изображается объект, который является инициатором взаимодействия. Правее изображается другой объект, который непосредственно взаимодействует с первым. Таким образом, все объекты на диаграмме последовательности образуют некоторый порядок, определяемый очередностью или степенью активности объектов при взаимодействии друг с другом.

Графически каждый объект изображается прямоугольником и располагается в верхней части своей линии жизни. Внутри прямоугольника записываются имя объекта и имя класса, разделенные двоеточием. При этом вся запись подчеркивается, что является признаком объекта.

Вторым измерением диаграммы последовательности является вертикальная временная ось, направленная сверху вниз. Начальному моменту времени соответствует самая верхняя часть диаграммы. Взаимодействия объектов реализуются посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения, а их порядок определяется временем возникновения. То есть, сообщения, расположенные на диаграмме последовательности выше, инициируются раньше тех, которые расположены ниже.

Диаграмма последовательностей приведена в приложении А на рисунке А.4. При авторизации пользователя с клиента на сервер посылаются учетные данные. Сервер сверяет их с записями в базе данных, при нахождении совпадения, сервер отправляет клиенту сообщение об успешной авторизации.

## 5.6 Диаграмма компонентов

Этот тип диаграмм предназначен для распределения классов и объектов по компонентам при физическом проектировании системы. Часто данный тип диаграмм называют диаграммами модулей.

Полный проект программной системы представляет собой совокупность моделей логического и физического уровней, которые должны быть согласованы между собой. В языке UML для физического представления моделей систем используются диаграммы реализации (implementation diagrams), которые включают в себя диаграмму компонентов и диаграмму развертывания.

Диаграмма компонентов, в отличие от ранее рассмотренных диаграмм, описывает особенности физического представления системы. Она позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный и исполняемый код. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

Диаграмма компонентов разрабатывается для следующих целей:

* визуализации общей структуры исходного кода программной системы;
* спецификации исполняемого варианта программной системы;
* обеспечения многократного использования отдельных фрагментов программного кода;
* представления концептуальной и физической схем баз данных.

В разработке диаграмм компонентов участвуют как системные аналитики и архитекторы, так и программисты. Диаграмма компонентов обеспечивает согласованный переход от логического представления к конкретной реализации проекта в форме программного кода. Одни компоненты могут существовать только на этапе компиляции программного кода, другие на этапе его исполнения. Диаграмма компонентов отражает общие зависимости между компонентами, рассматривая последние в качестве классификаторов.

Для представления физических сущностей в языке UML применяется специальный термин - компонент (component). Компонент реализует некоторый набор интерфейсов и служит для общего обозначения элементов физического представления модели. Для графического представления компонента используется специальный символ – прямоугольник со вставленными слева двумя более мелкими прямоугольниками. Внутри большого прямоугольника записывается имя компонента и, при необходимости, некоторая дополнительная информация. Изображение этого символа может незначительно варьироваться в зависимости от характера ассоциируемой с компонентом информации.

Диаграмма компонентов приведена в приложении А на рисунке А.5.

## 5.7 Диаграмма классов

Центральное место в объектно-ориентированном программировании занимает разработка логической модели системы в виде диаграммы классов. Диаграмма классов (class diagram) служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывать их внутреннюю структуру и типы отношений.

Значки диаграммы позволяют отображать сложную иерархию систем, взаимосвязи классов (Classes) и интерфейсов (Interfaces). Данный тип диаграмм противоположен по содержанию диаграмме Collaboration, на котором отображаются объекты системы. Rational Rose позволяет создавать классы при помощи данного типа диаграмм в различных нотациях. похожего на облако. Таким образом класс - это лишь шаблон, по которому в дальнейшем будет создан конкретный объект.

Диаграмма классов представляет собой граф, вершинами которого являются элементы типа "классификатор", связанные различными типами структурных отношений. Диаграмма классов может также содержать интерфейсы, пакеты, отношения и даже отдельные экземпляры, такие как объекты и связи.

Класс (class) в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами других классов. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции. В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

Диаграмма классов клиентской части приложения представлена в приложении Б на рисунке Б.1. На клиенте entity-класс «Order» отвечает за вывод заказов в таблицы. Класс «AlertBox» отвечает за вывод всплывающего окна с пользовательским сообщением. Класс «SceneLoader» отвечает за загрузку сцен. Класс «LoginController» – один из контроллеров для сцен.

Диаграмма классов серверной части приложения приведена в приложении Б на рисунке Б.2. Класс «DB» отвечает за работу с базой данных, включая подключение. Класс «Server» отвечает за запуск и остановку сервера. Класс «Worker» отвечает за создание потоков для клиентов и создание потоков ввода-вывода. Класс «CommandExecutor» отвечает за выполнение команд от клиента. Класс «Controller» – контроллер сцены.

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

## 6.1 Команды серверу

Поскольку бизнес-логика реализована на сервере, клиент не может напрямую вносить изменения в базу данных и обращаться к ней, для этого реализован механизм команд серверу.

На сервере хранится определенный набор команд, который также известен клиенту, когда клиенту необходимо произвести какие-либо действия, он посылает серверу команду и необходимые аргументы. Команды представлены в виде строк.

Когда сервер получает определенную строку, он отделяет первый набор символов до пробела, используя метод split() класса String. Далее этот набор символов проходит через оператор switch, который вызывает необходимый метод для выполнения команды и также отделяет аргументы от основной строки. После выполнения команды возвращается результат выполнения, в зависимости от метода это может быть String, boolean или ArrayList<String>. Листинг кода класса CommandExecutor, который хранит команды, представлен на рисунке 6.1.

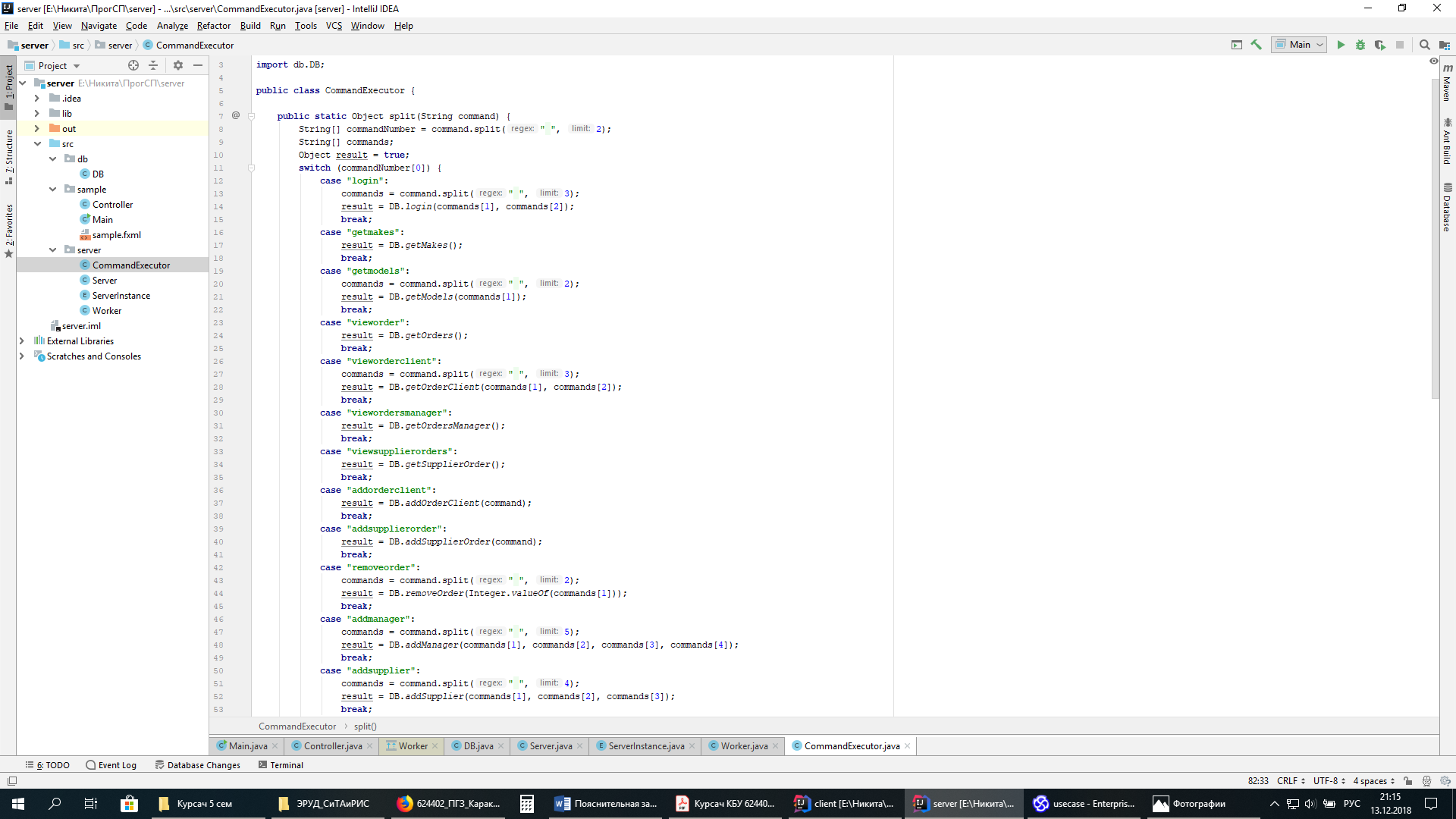


Рисунок 6.1 - Листинг кода класса CommandExecutor

## 6.2 Потоки

К большинству современных распределенных приложений (Rich Client) и Web-приложений (Thin Client) выдвигаются требования одновременной поддержки многих пользователей, каждому из которых выделяется отдельный поток, а также разделения и параллельной обработки информационных ресурсов. Потоки – средство, которое помогает организовать одновременное выполнение нескольких задач, каждую в независимом потоке. Потоки представляют собой классы, каждый из которых запускается и функционирует самостоятельно, автономно (или относительно автономно) от главного потока выполнения программы.

Потоки реализованы путем имплементации интерфейса Runnable. При подключении нового клиента, создается новый поток. Также создаются потоки ввода-вывода и сервер переходит в режим ожидания команд от клиента. При получении команды, сервер выполняет ее, отсылает результат выполнения и опять переход в режим ожидания команды. Сервер также имеет свой поток. Листинг кода с запуском потока сервера представлен на рисунке 6.2.

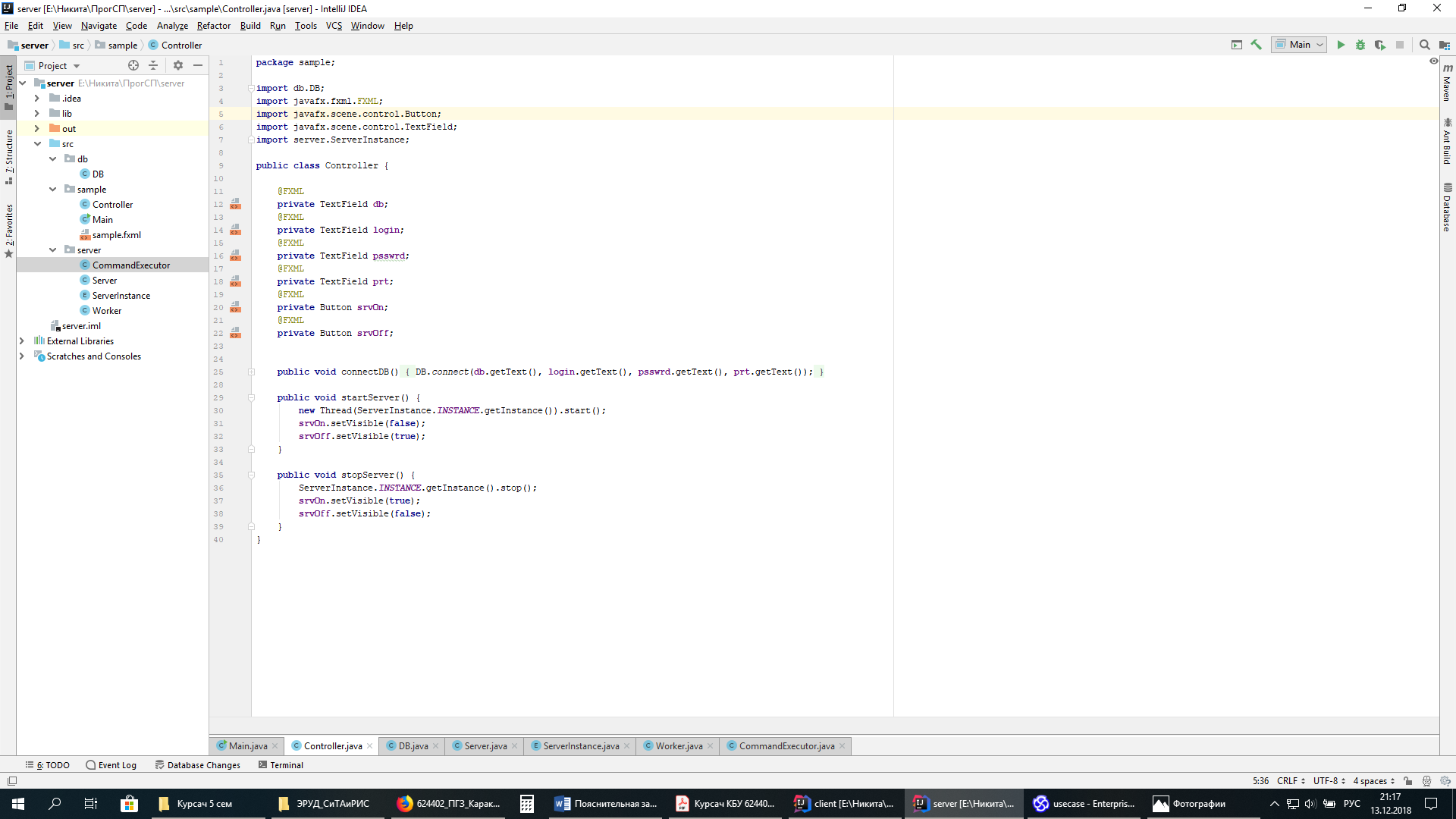


Рисунок 6.2 - Запуск потока сервера

Как видно на рисунке выше, помимо запуска сервера и его потока реализована возможность остановка сервера.

# 7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 7.1 Серверная часть

При запуске серверной части приложения пользователя встречает окно, на котором имеются две кнопки и четыре поля для ввода информации. Скриншот окна серверной части приложения приведен на рисунке 7.1.

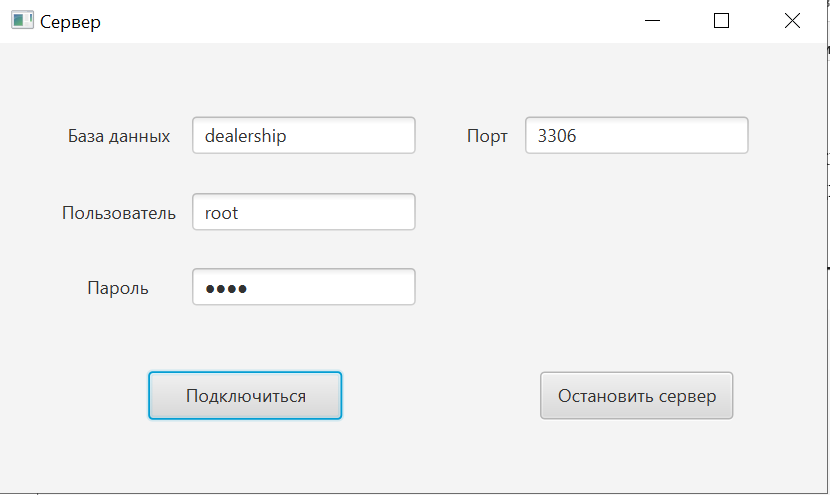


Рисунок 7.1 - Серверная часть приложения

Для запуска сервера необходимо нажать на кнопку «Запустить сервер». При успешном запуске сервера надпись на кнопке изменится на «Остановить сервер». Теперь, при нажатии на кнопку «Остановить сервер», сервер выключится, и клиенты не смогут к нему подключиться.

Для подключения к базе данных необходимо заполнить все поля для ввода информации. В поле «База данных» необходимо ввести название базы данных, к которой нужно подключиться. В поле «Пользователь» вводится имя пользователя для подключения к базе данных. В поле «Пароль» вводится пароль, соответствующий имени пользователя для подключения к базе данных. В последнем поле для ввода информации вводится порт для подключения к базе данных.

## 7.2 Клиентская часть

При запуске клиентской части приложения пользователя встречает окно, на котором находятся две кнопки и два поля для ввода информации. Скриншот клиентской части приложения приведен на рисунке 7.2.

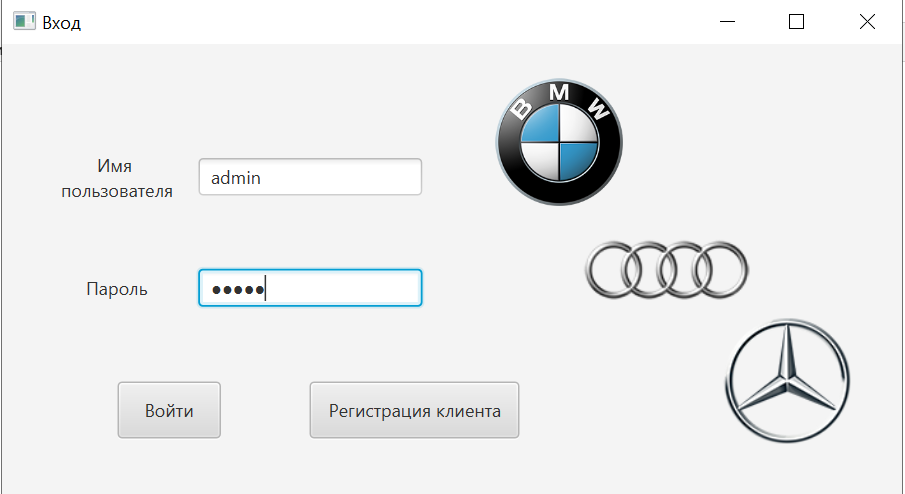


Рисунок 7.2 - Клиентская часть приложения

Для регистрации в качестве клиента необходимо нажать на кнопку «Регистрация клиента». Откроется новое окно, в котором необходимо ввести имя, фамилию, логин и пароль пользователя. После ввода данных, необходимо нажать на кнопку «Зарегистрироваться» для внесения пользователя в базу данных. Для возвращения на предыдущее окно предусмотрена кнопка «Назад». Скриншот окна с регистрацией нового пользователя приведен на рисунке 7.3.

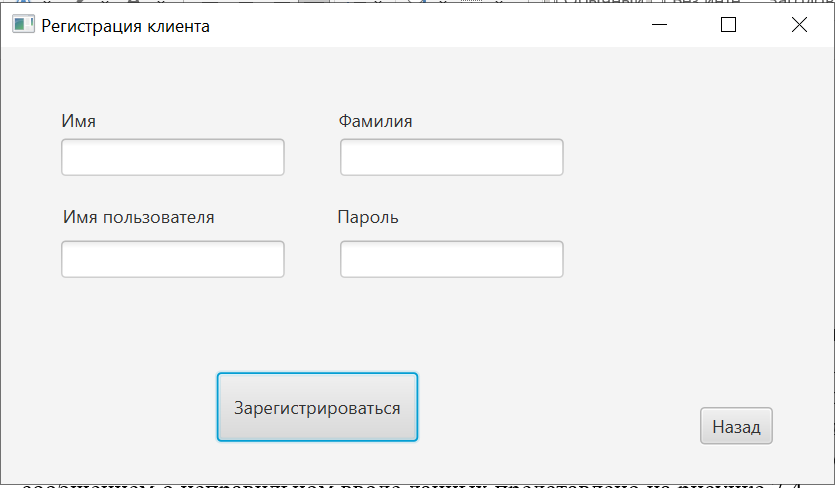


Рисунок 7.3 - Регистрация нового пользователя

Для входа в систему необходимо ввести в поле «Пользователь» имя пользователя и пароль в поле «пароль», после чего нажать на кнопку «Войти». При правильном вводе учетных данных будет открыто новое окно, соответствующее типу введенной учетной записи, в противном случае будет выведено окно с сообщением о неправильном вводе данных. Окно с сообщением о неправильном вводе данных представлено на рисунке 7.4.

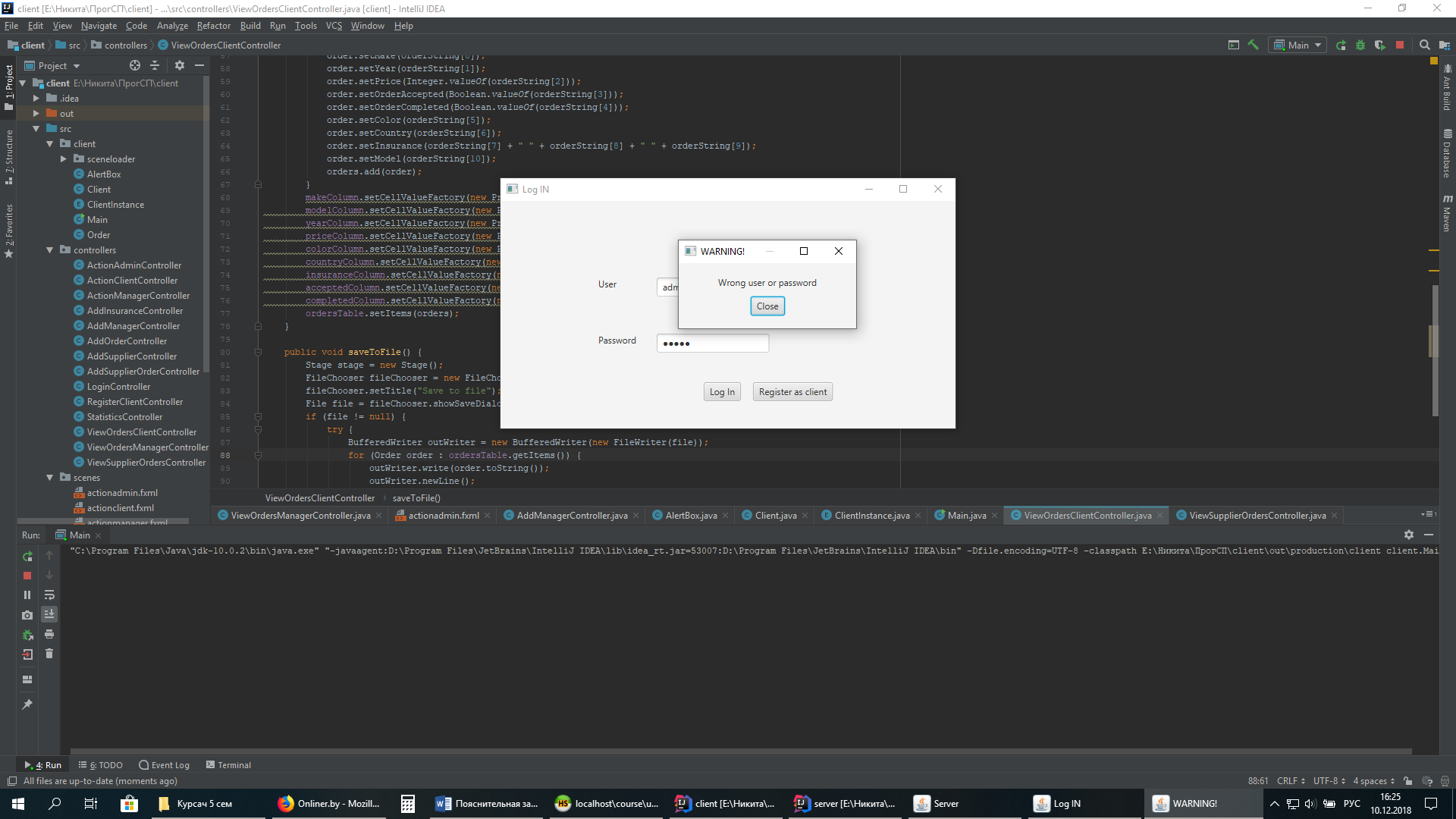


Рисунок 7.4 - Сообщение о неправильном вводе данных

При входе под учетной записью администратора появляется окно выбора с 5 кнопками: «Добавить менеджера», «Добавить поставщика», «Добавить страховку» и «Отобразить список вип-клиентов». Скриншот окна при входе под учетной записью администратора приведен на рисунке 7.5.

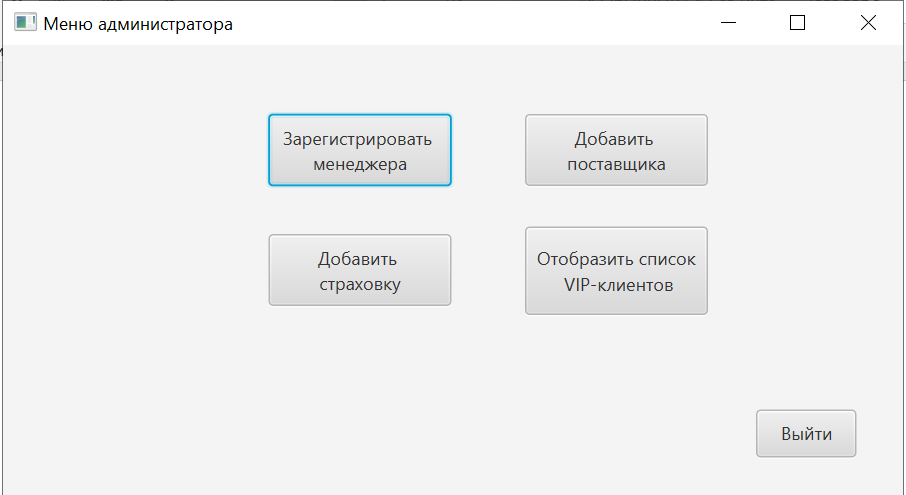


Рисунок 7.5 - Учетная запись администратора

При нажатии на кнопку «выйти» происходит возврат на окно ввода учетной записи.

При нажатии на кнопку «Добавить менеджера» появится новое окно для добавления менеджера, на котором есть две кнопки и четыре поля для ввода информации. Скриншот окна для добавления менеджера приведен на рисунке 7.6.

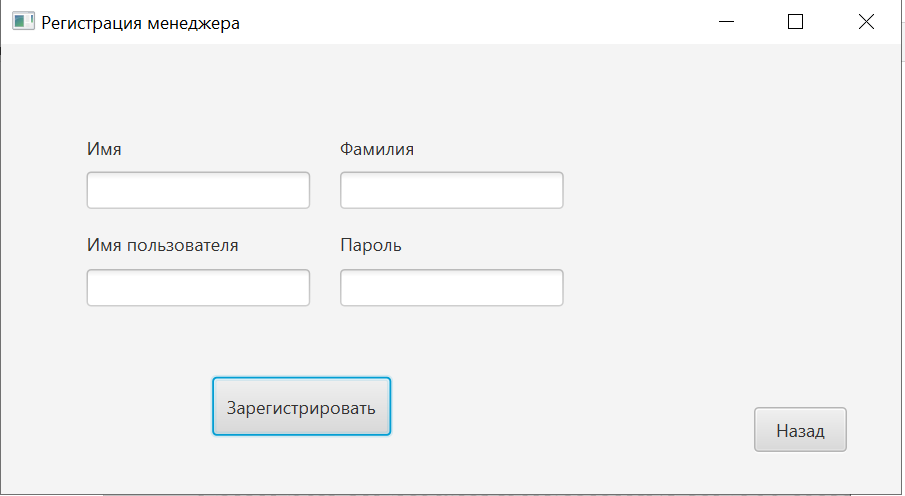


Рисунок 7.6 - Окно добавления менеджера

Для добавления менеджера необходимо заполнить все поля и нажать на кнопку «Зарегистрировать». Для возврата на предыдущее окно предусмотрена кнопка «Назад».

При нажатии на кнопку «Добавить поставщика» откроется новое окно для добавления поставщика, на котором есть две кнопки, два поля для ввода информации и один выпадающий список. Скриншот окна для добавления менеджера приведен на рисунке 7.7.

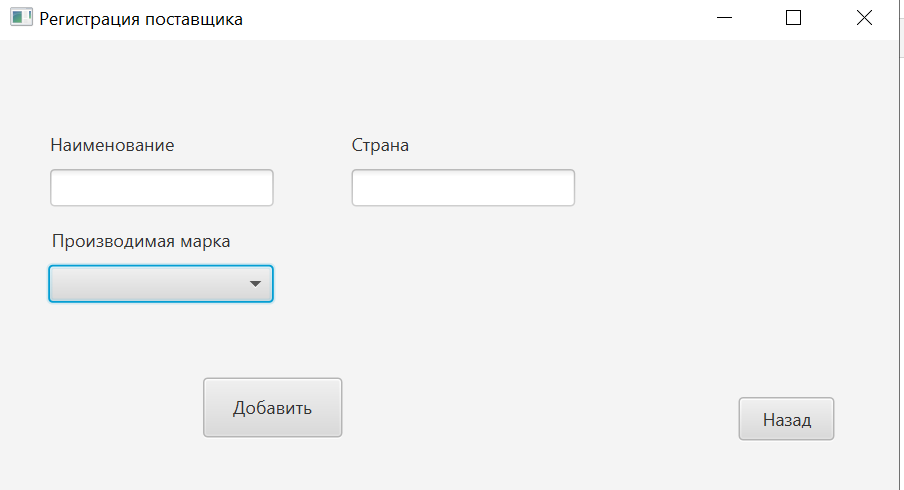


Рисунок 7.7 - Окно добавления поставщика

Для добавления поставщика необходимо заполнить все поля и выбрать пункт из выпадающего списка, после чего нажать кнопку «Добавить». Для возврата на предыдущее окно предусмотрена кнопка «Назад».

При нажатии на кнопку «Добавить страховку» откроется новое окно для добавления страховки, на котором есть две кнопки и два поля для ввода информации. Скриншот окна для добавления страховки приведен на рисунке 7.8.

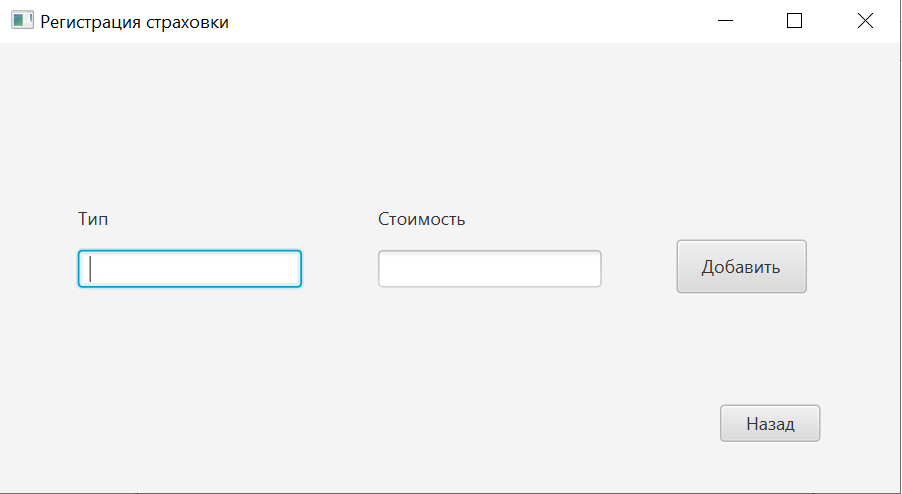


Рисунок 7.8 - Окно добавления страховки

Для добавления страховки необходимо заполнить все поля и нажать кнопку «Добавить». Для возврата на предыдущее окно предусмотрена кнопка «Назад».

При нажатии на кнопку «Отобразить список вип-клиентов» откроется новое окно для просмотра вип-клиентов. Скриншот окна приведен на рисунке 7.9.

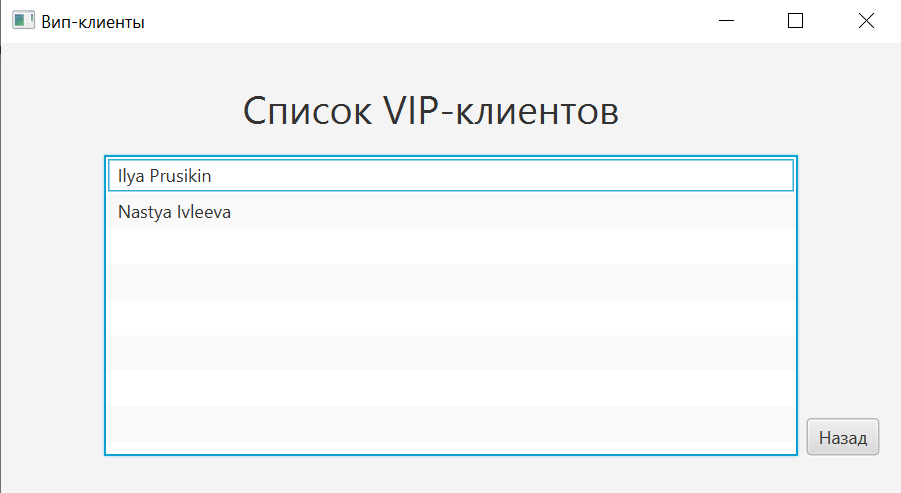


Рисунок 7.9-Вип-клиенты

При входе под учетной записью менеджера появляется окно выбора c 7 кнопками: «Добавить новую модель», «Просмотреть диаграмму заказов», «просмотреть график заказов», «Работать с заказами клиентов», «Сделать заказ поставщику», «Отобразить список вип-клиентов» и «Посмотреть заказы поставщикам» Также отображается имя и фамилия менеджера. Скриншот окна выбора учетной записи менеджера приведен на рисунке 7.10.

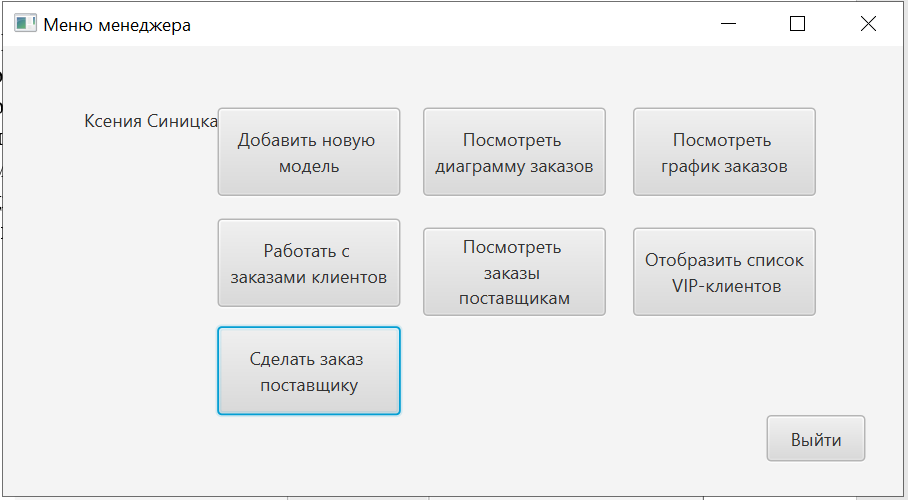


Рисунок 7.10 – Учетная запись менеджера

При нажатии на кнопку «выйти» происходит возврат на окно ввода учетной записи.

При нажатии на кнопку «Добавить заказ поставщику» откроется новое окно для добавления заказа поставщику, на котором есть две кнопки, четыре выпадающих списка и одно поле для ввода информации. Скриншот окна для добавления заказа поставщику приведен на рисунке 7.11.

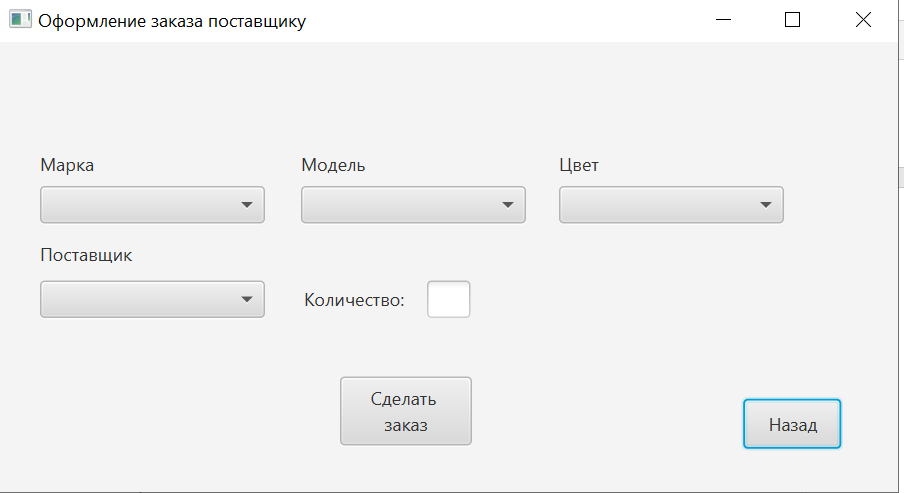


Рисунок 7.11 - Окно добавления заказа поставщику

При нажатии на кнопку «Список вип-клиентов» откроется новое окно для просмотра заказов к поставщикам, на котором находится одна таблица и кнопка «назад». Скриншот окна просмотра заказов к поставщикам приведен на рисунке 7.12.

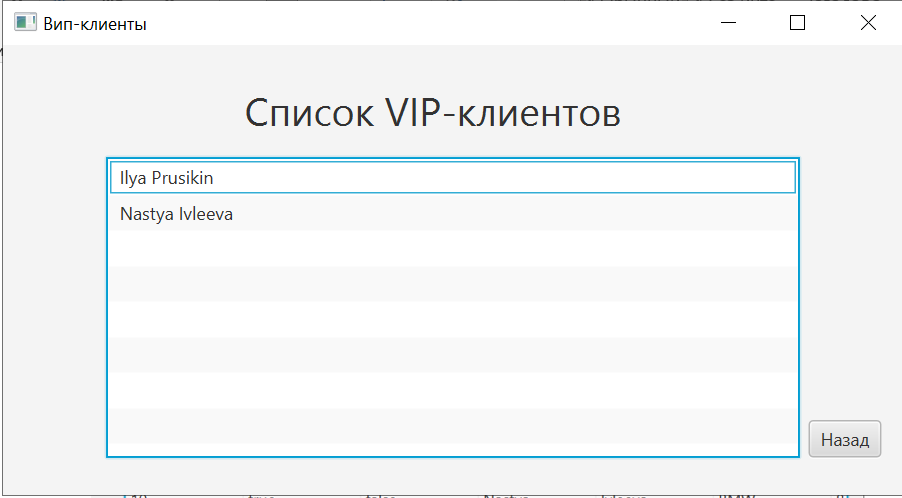


Рисунок 7 12 - Окно просмотра заказов к поставщикам

При нажатии на кнопку «Работа с заказами клиентов» появляется новое окно для просмотра заказов клиентов, на котором расположена таблица, четыре кнопки и одно поле для ввода информации. Скриншот окна просмотра заказов клиентов приведен на рисунке 7.13.

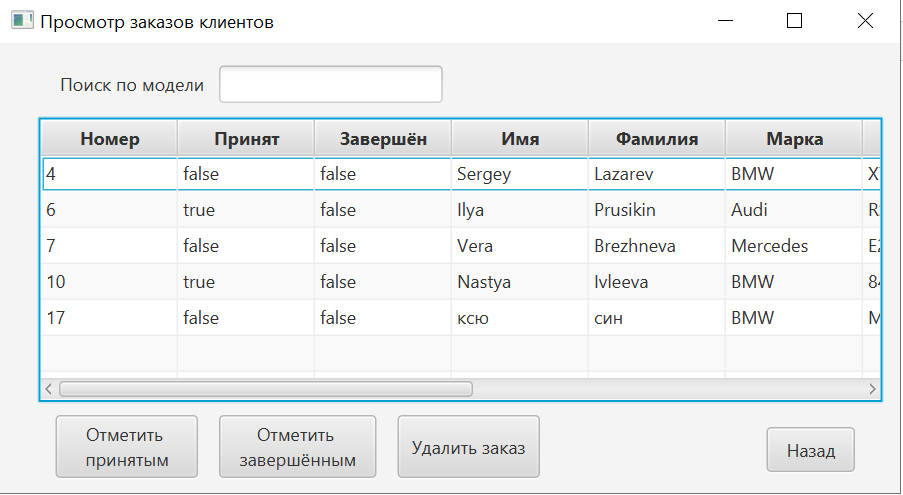


Рисунок 7.13 - Окно просмотра заказов клиентов

Для поиска заказов по марке автомобиля предусмотрено поле «Поиск». При вводе информации в него таблица обновляется динамически.

Для того, чтобы принять заказ нужно нажать на кнопку «Отметить принятым», изменения сразу вступят в силу и будут видны в таблице. То же касается и завершения заказа по нажатии на кнопку «Отметить завершенным».

Для удаления заказа нужно выбрать необходимый заказ и нажать на кнопку «Удалить заказ», изменения сразу вступят в силу и будут видны в таблице.

Для возврата на предыдущее окно предусмотрена кнопка «назад».

При нажатии на кнопку «Статистика заказов» появляется новое окно для просмотра статистики, на котором имеется одна диаграмма и кнопка «Назад». Скриншот окна просмотра статистики приведен на рисунке 7.14.

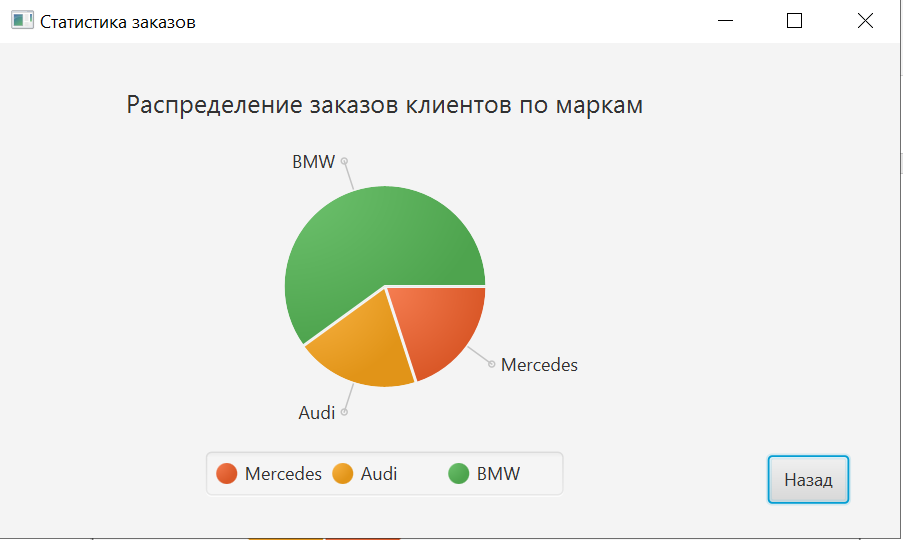


Рисунок 7.14 - Окно просмотра статистики

Для возврата на предыдущее окно предусмотрена кнопка «Назад».

При нажатии на кнопку «просмотр заказов поставщикам» появляется новое окно для просмотра заказов поставщикам, на котором имеется одна таблица и кнопка «Назад». Скриншот окна просмотра статистики приведен на рисунке 7.15.

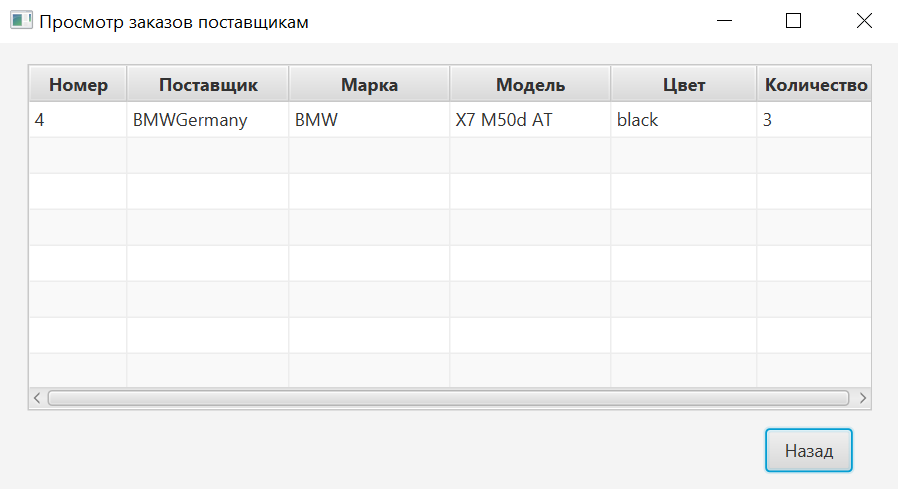


Рисунок 7.15-просмотр заказов поставщикам

При нажатии на кнопку «Добавить новую модель» появляется новое окно для добавления новой модели для заказа, на котором имеется три поля для заполнения и две кнопки «Добавить модель» и «Назад». Скриншот окна просмотра статистики приведен на рисунке 7.16.

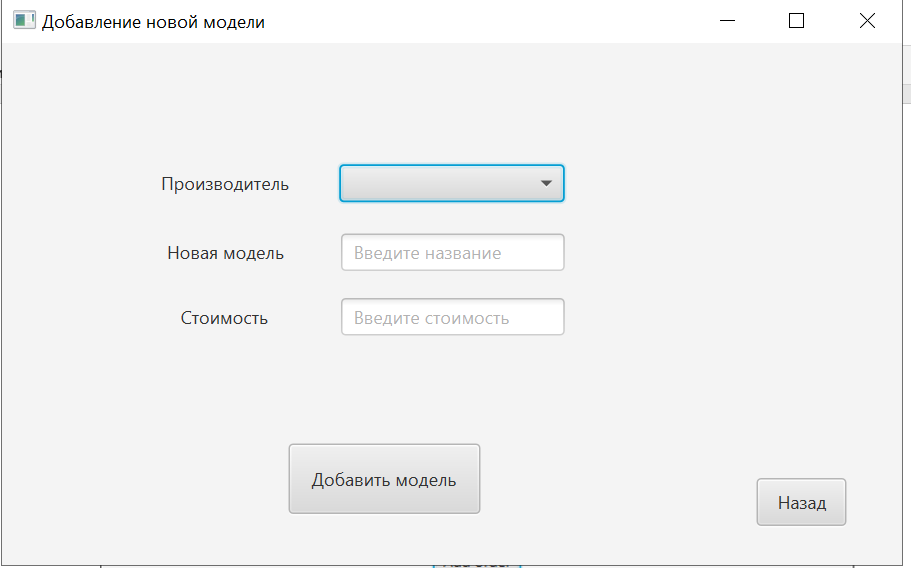


Рисунок 7.16- Добавление новой модели

При входе под учетной записью клиента появляется окно выбора с 5 кнопками: «Сделать заказ», «Просмотреть заказы», «Скоро в продаже», «Диаграмма популярных моделей», «Просмотр моделей» и «выйти». Также отображается имя и фамилия клиента. Скриншот окна выбора учетной записи клиента приведен на рисунке 7.17.

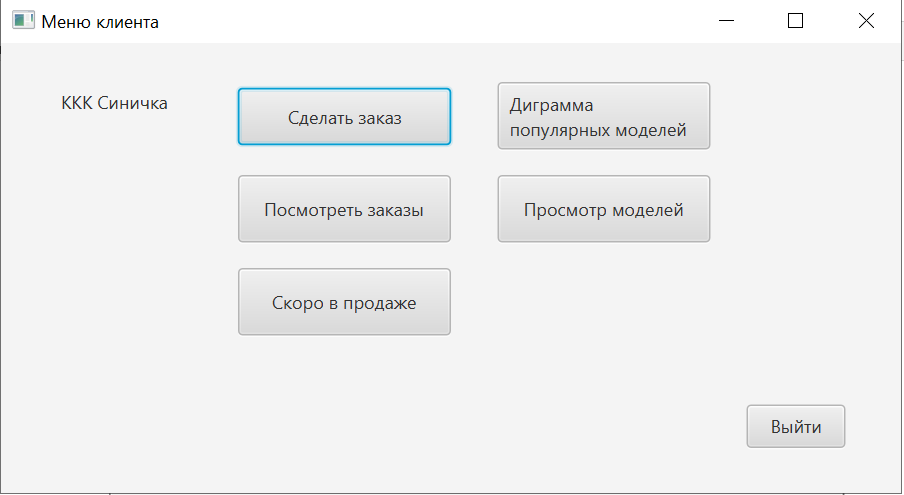


Рисунок 7.17 - Окно выбора учетной записи клиента

При нажатии на кнопку «Выйти» происходит возврат на окно ввода учетной записи.

При нажатии на кнопку «Сделать заказ» появляется окно добавления заказа, на котором есть две кнопки, шесть выпадающих меню и поле «Итого». Скриншот окна добавления заказа приведен на рисунке 7.18.

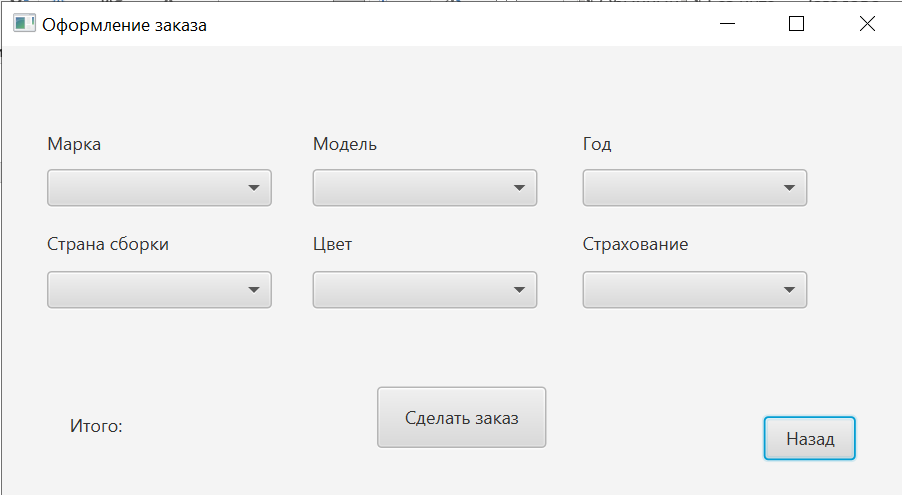


Рисунок 7.18 - Окно добавления заказа

Для добавления заказа необходимо выбрать пункты в каждом из выпадающих меню и нажать на кнопку «Сделать заказ». При изменении автомобиля и страховки изменяется цена заказа. Для возврата на предыдущее окно предусмотрена кнопка «выйти».

При нажатии на кнопку «Просмотреть заказы» появится окно просмотра заказов клиента, на котором расположена одна таблица и две кнопки: «Сохранить в файл» и «Назад». Скриншот окна просмотра заказов приведен на рисунке 7.19.

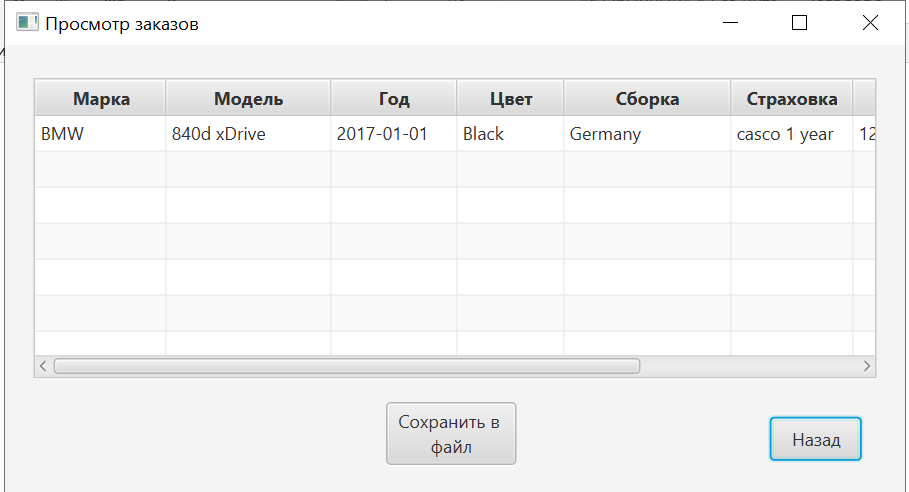


Рисунок 7.19 - Окно просмотра заказов

Для сохранения информации о заказах предусмотрена кнопка «Сохранить в файл». При ее нажатии появляется окно, в котором необходимо ввести путь и название файла, в который будет сохранена информация. Скриншот окна выбора пути и названия файла приведен на рисунке 7.20.

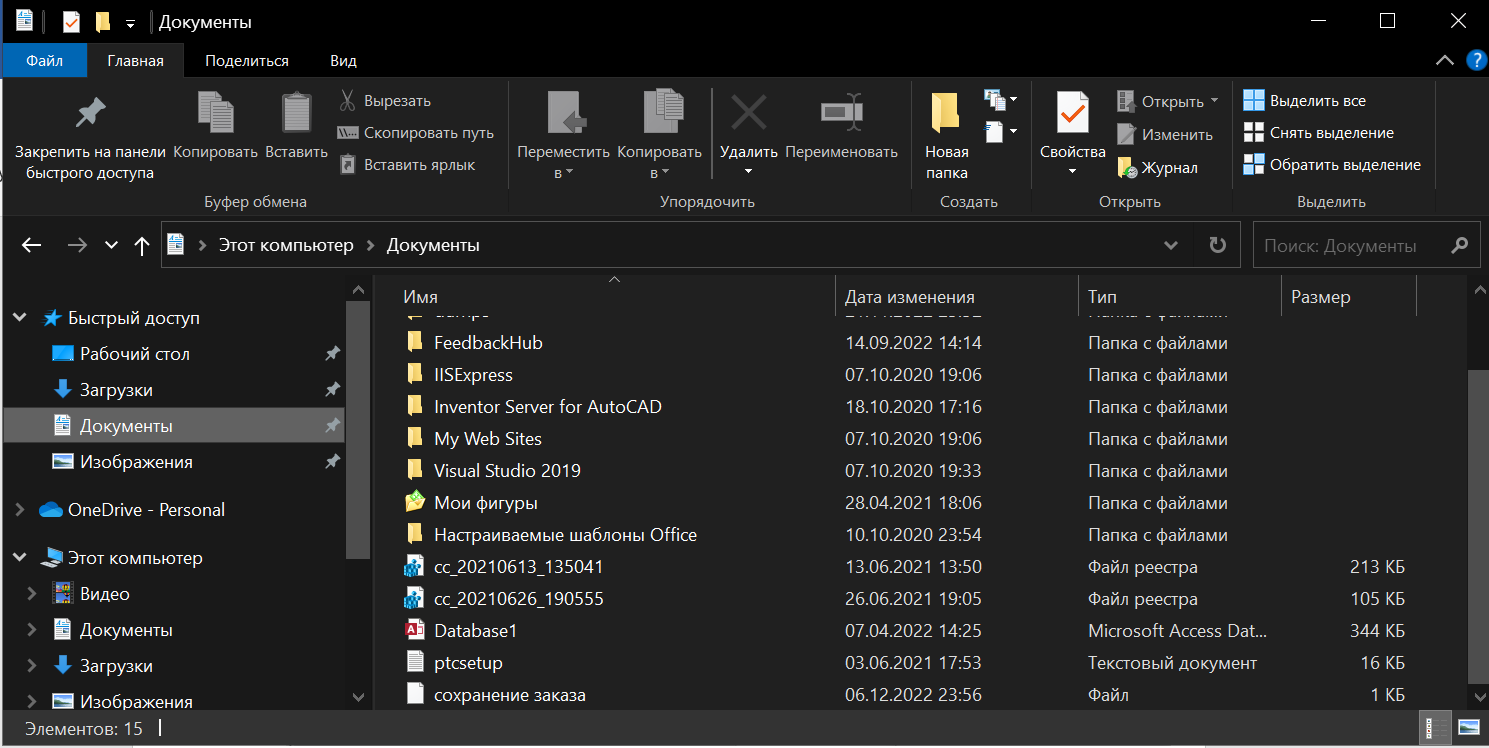


Рисунок 7.20 - Окно выбора пути и названия файла

При нажатии на кнопку «Скоро в продаже» появится окно просмотра моделей авто которые скоро будут в продаже, на котором расположена одна таблица и одна кнопка «Назад». Скриншот окна приведен на рисунке 7.21.

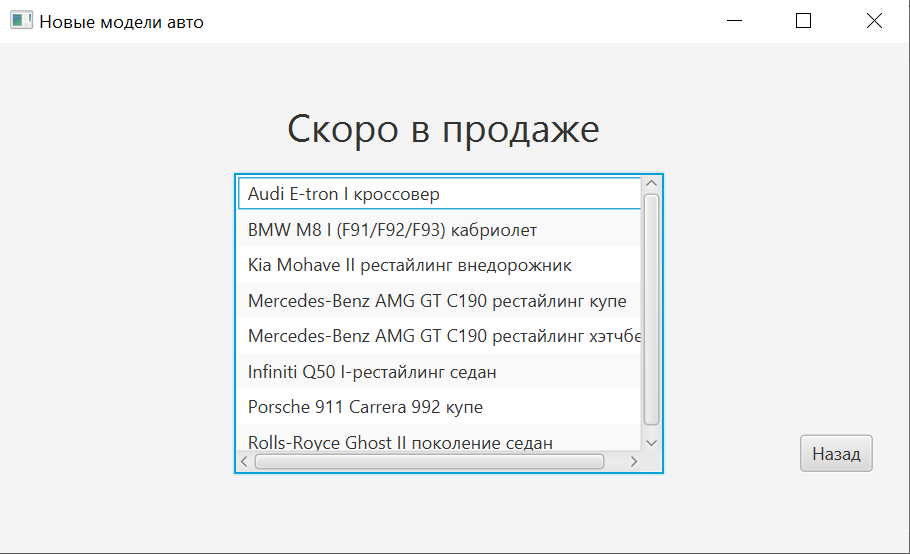


Рисунок 7.21-скоро в продаже

При нажатии на кнопку «просмотр популярных моделей» появляется новое окно для просмотра статистики, на котором имеется одна диаграмма и кнопка «Назад». Скриншот окна просмотра популярных моделей приведен на рисунке 7.22.

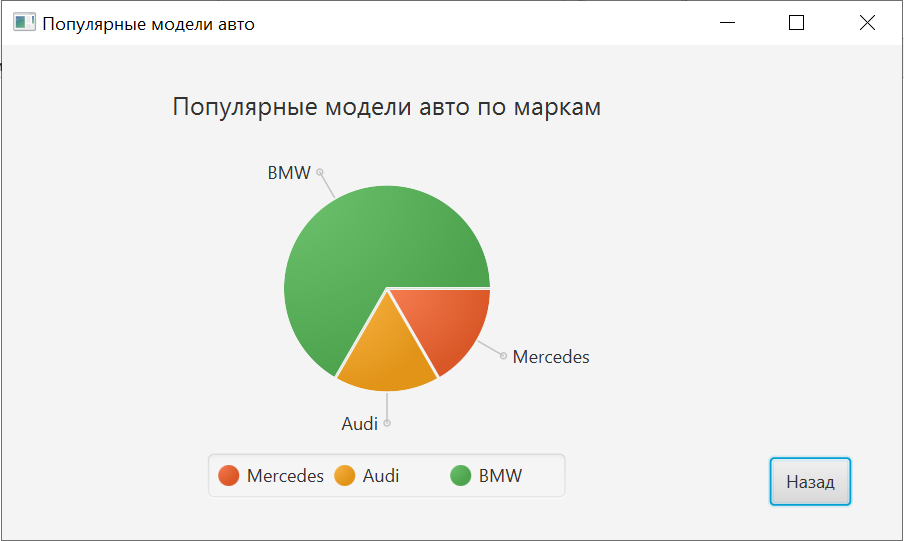


Рисунок 7.22- просмотр популярных моделей

При нажатии на кнопку «Просмотр моделей» появится окно просмотра моделей авто которые есть в оффлайн салоне, на котором расположена одна

таблица и одна кнопка «Назад». Скриншот окна приведен на рисунке 7.23.

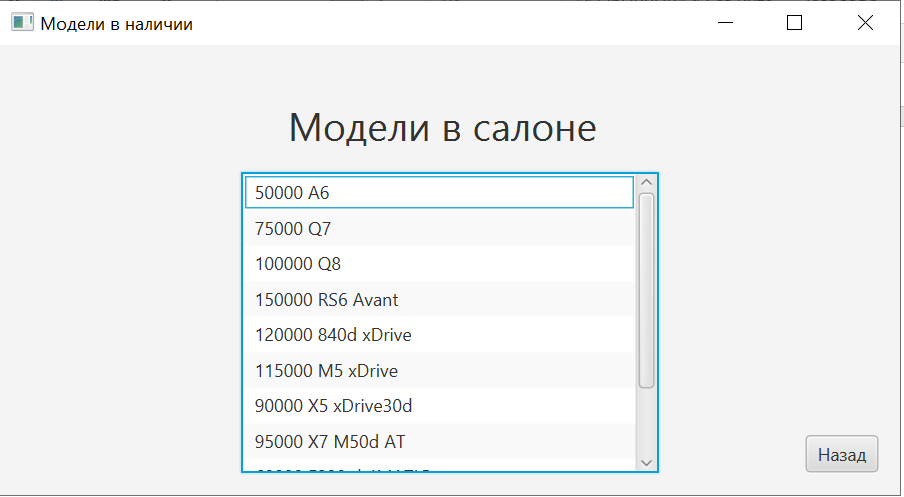


Рисунок 7.23- Модели в наличие

# 8 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ

В результате тестирования системы были выявлены и исправлены некоторые недочеты. При отправке данных на сервер, требуется проверка на правильность введенных данных и на наличие этих данных в полях для ввода данных, выпадающих меню и т.д. На рисунке 8.1 приведен пример сообщения о необходимости ввести данные.

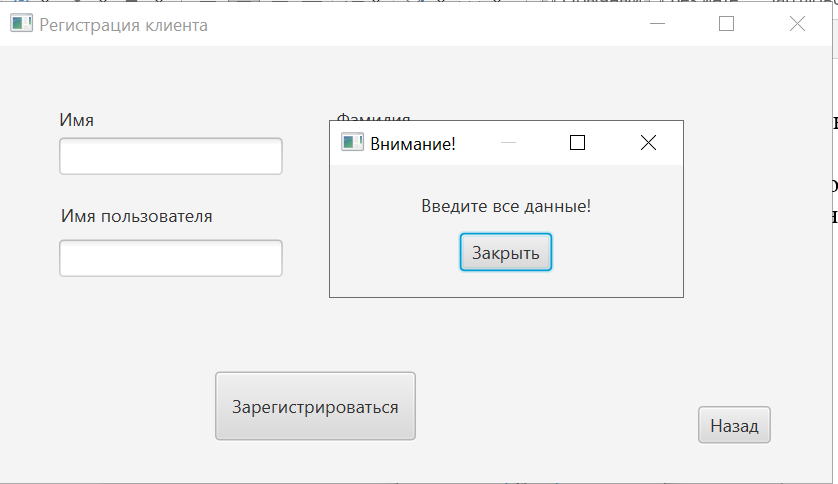


Рисунок 8.1 - Сообщение о необходимости ввести информацию

Как видно на рисунке выше, при регистрации не была введена информация ни в одно из полей. Клиент выдал предупреждение о необходимости ввести информацию и не отправлял пустые строки серверу, а тот, в свою очередь, не вносил информацию в базу данных.

# ЗАКЛЮЧЕНИЕ

В результате выполненной работы было разработано оконное приложение в архитектуре клиент-сервер для учета продаж автомобилей в автосалонах. Актуальность темы не вызывает вопросов, так как автомобили становятся все более и более доступными, их количество растет, рынок развивается.

Разработанное приложение может использоваться во всех видах автосалонов, размер и цена автомобиля не имеет значения.

В наше время практически невозможно существовать без использования всех благ, предлагаемых современными технологиями. Для того чтобы добиться наилучших результатов просто необходимо идти в ногу со временем. В данном курсовом проекте разработана программа способная улучшить результаты работы любого автосалона.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] 5 причин поменять старую машину на новую [Электронный ресурс]. – Режим доступа: <https://www.seb.lv/ru/info/avtomobil/5-prichin-dlya-togo-chtoby-pomenyat-staruyu-mashinu-na-novuyu>

[2] How does JDBC work? [Электронный ресурс] – Режим доступа: <https://www.progress.com/faqs/datadirect-jdbc-faqs/how-does-jdbc-work>

[3] JavaFX [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/JavaFX

[4] The YEAR Type [Электронный ресурс] – Режим доступа: https://dev.mysql.com/doc/refman/8.0/en/year.html

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Модели представления системы**

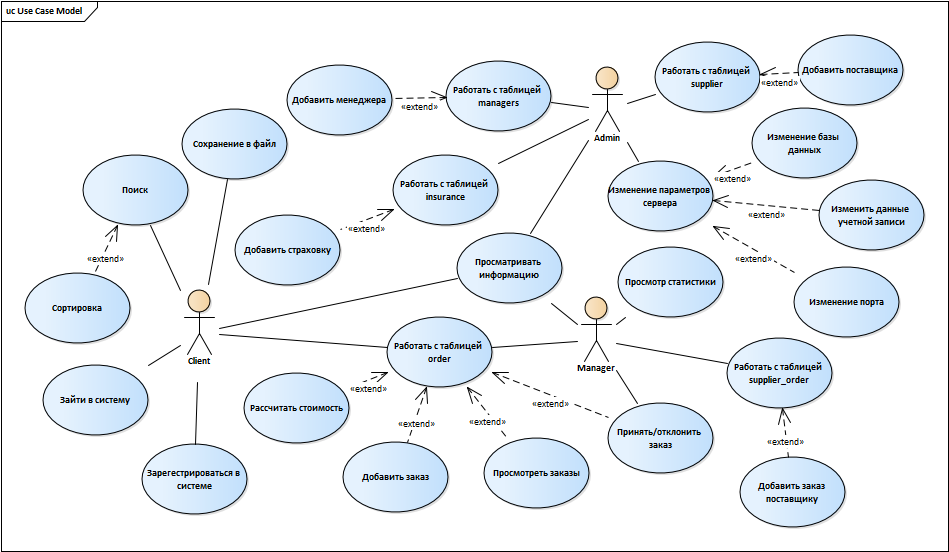
****

Рисунок А.1 - Диаграмма вариантов использования

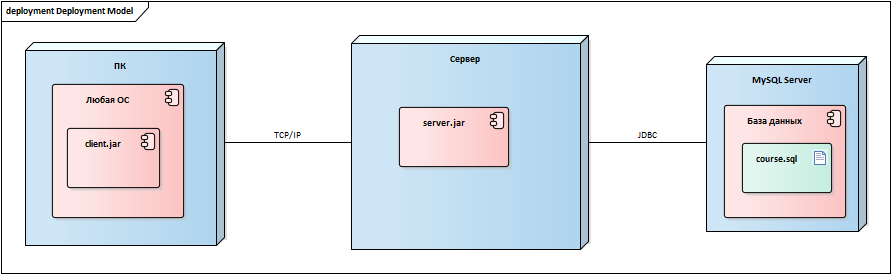


Рисунок А.2 - Диаграмма развертывания

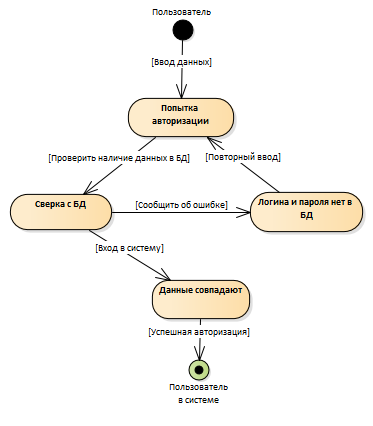


Рисунок А.3 - Диаграмма состояний

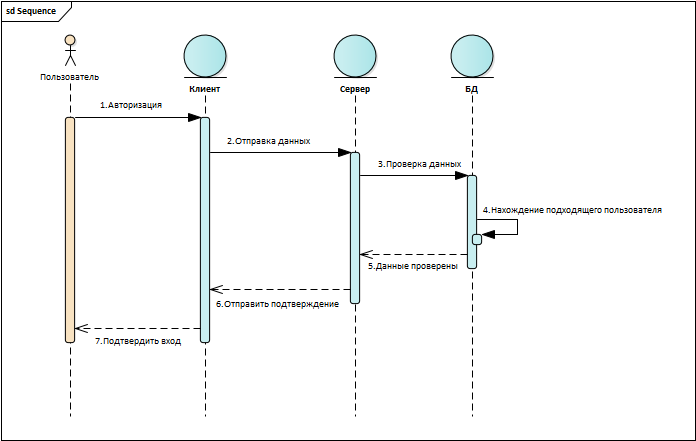


Рисунок А.4 - Диаграмма последовательностей

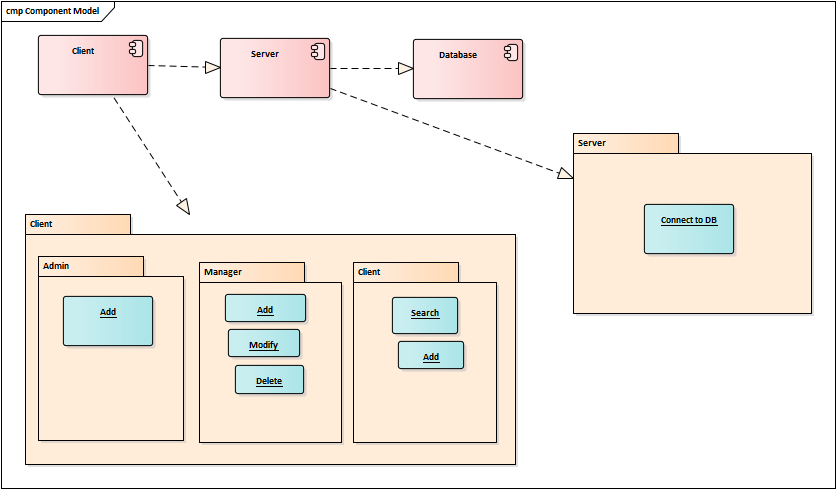


Рисунок А.5 - Диаграмма компонентов

# ПРИЛОЖЕНИЕ Б

**(обязательное)**

**Диаграммы классов**

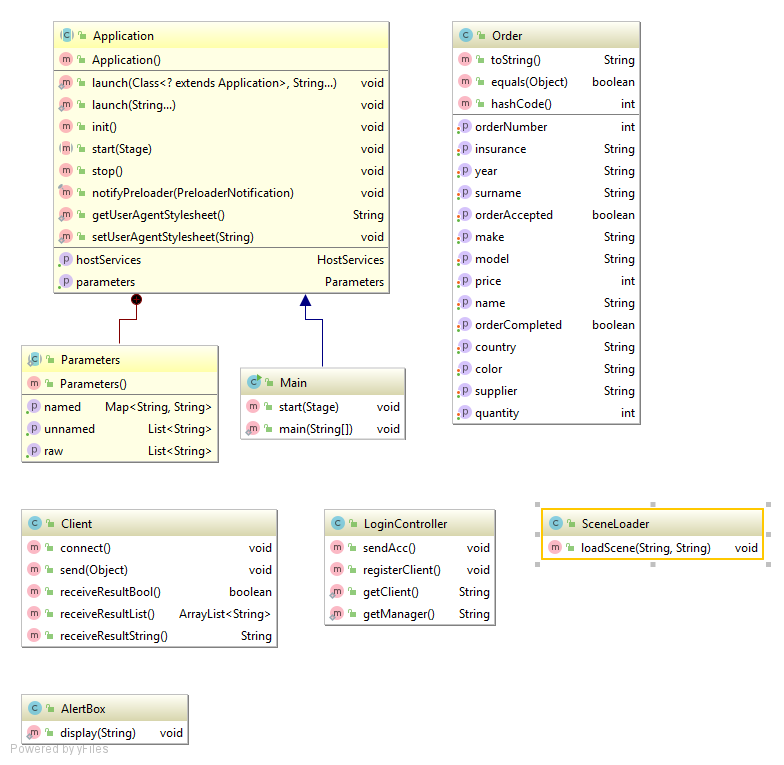


Рисунок Б.1 - Диаграмма классов клиентской части приложения

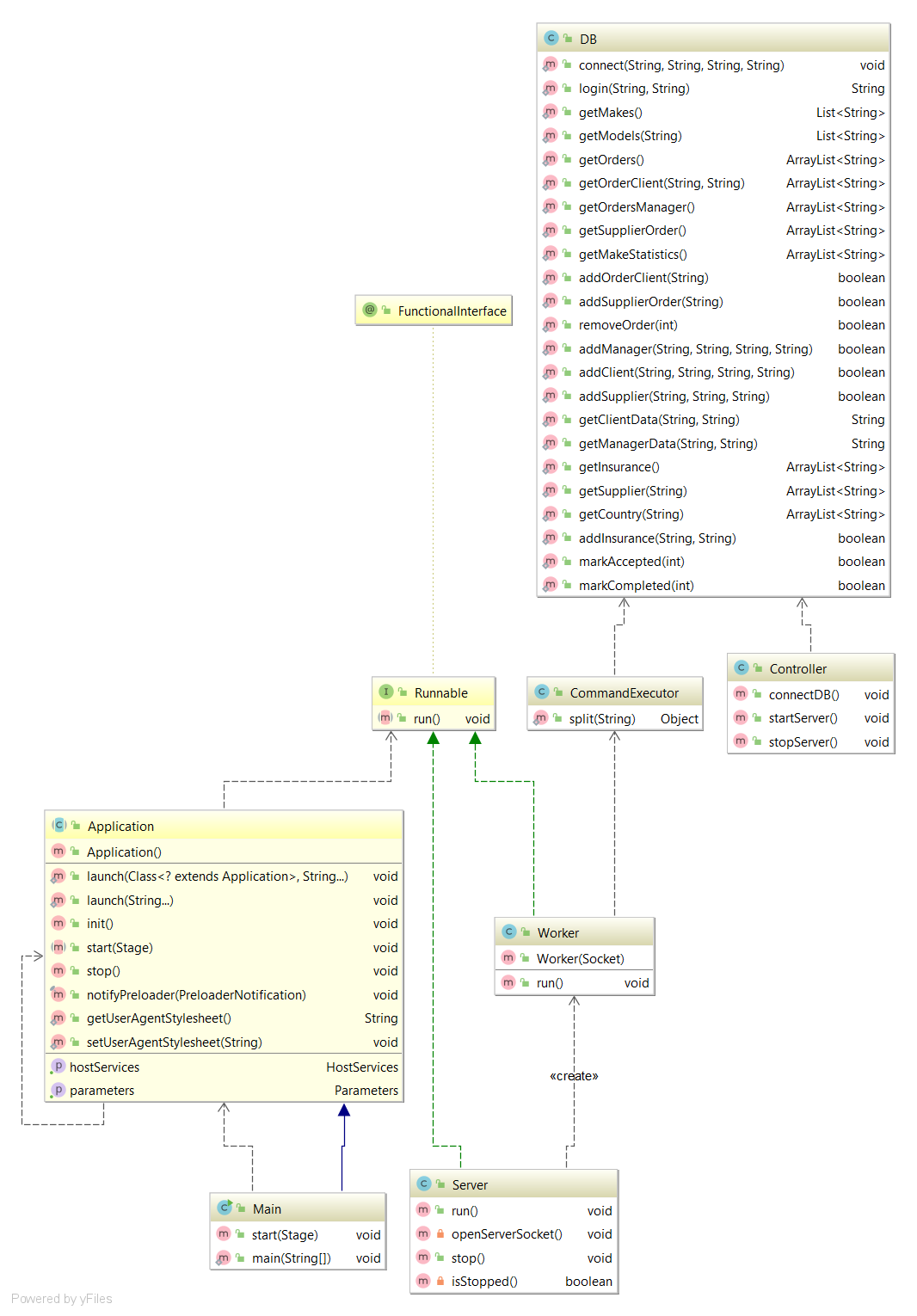


Рисунок Б.2 - Диаграмма классов серверной части приложения

# ПРИЛОЖЕНИЕ В

**(обязательное)**

**Листинг кода (выборочно)**

Класс CommandExecutor

**public class** CommandExecutor {  
  
 **public static** Object split(String command) {  
 String[] commandNumber = command.split(**" "**, 2);  
 String[] commands;  
 Object result = **true**;  
 **switch** (commandNumber[0]) {  
 **case "login"**:  
 commands = command.split(**" "**, 3);  
 result = DB.*login*(commands[1], commands[2]);  
 **break**;  
 **case "getmakes"**:  
 result = DB.*getMakes*();  
 **break**;  
 **case "getmodels"**:  
 commands = command.split(**" "**, 2);  
 result = DB.*getModels*(commands[1]);  
 **break**;  
 **case "vieworder"**:  
 result = DB.*getOrders*();  
 **break**;  
 **case "vieworderclient"**:  
 commands = command.split(**" "**, 3);  
 result = DB.*getOrderClient*(commands[1], commands[2]);  
 **break**;  
 **case "viewordersmanager"**:  
 result = DB.*getOrdersManager*();  
 **break**;  
 **case "viewsupplierorders"**:  
 result = DB.*getSupplierOrder*();  
 **break**;  
 **case "addorderclient"**:  
 result = DB.*addOrderClient*(command);  
 **break**;  
 **case "addsupplierorder"**:  
 result = DB.*addSupplierOrder*(command);  
 **break**;  
 **case "removeorder"**:  
 commands = command.split(**" "**, 2);  
 result = DB.*removeOrder*(Integer.*valueOf*(commands[1]));  
 **break**;  
 **case "addmanager"**:  
 commands = command.split(**" "**, 5);  
 result = DB.*addManager*(commands[1], commands[2], commands[3], commands[4]);  
 **break**;  
 **case "addsupplier"**:  
 commands = command.split(**" "**, 4);  
 result = DB.*addSupplier*(commands[1], commands[2], commands[3]);  
 **break**;  
 **case "getclientdata"**:  
 commands = command.split(**" "**, 3);  
 result = DB.*getClientData*(commands[1], commands[2]);

**ПРИЛОЖЕНИЕ В (продолжение)**

**break**;  
 **case "getmanagerdata"**:  
 commands = command.split(**" "**, 3);  
 result = DB.*getManagerData*(commands[1], commands[2]);  
 **break**;  
 **case "addclient"**:  
 System.***out***.println(command);  
 commands = command.split(**" "**, 5);  
 result = DB.*addClient*(commands[1], commands[2], commands[3], commands[4]);  
 **break**;  
 **case "getinsurance"**:  
 result = DB.*getInsurance*();  
 **break**;  
 **case "getcountry"**:  
 commands = command.split(**" "**, 2);  
 result = DB.*getCountry*(commands[1]);  
 **break**;  
 **case "getsupplier"**:  
 commands = command.split(**" "**, 2);  
 result = DB.*getSupplier*(commands[1]);  
 **break**;  
 **case "addinsurance"**:  
 commands = command.split(**" "**, 3);  
 result = DB.*addInsurance*(commands[1], commands[2]);  
 **break**;  
 **case "markaccepted"**:  
 commands = command.split(**" "**, 2);  
 result = DB.*markAccepted*(Integer.*valueOf*(commands[1]));  
 **break**;  
 **case "markcompleted"**:  
 commands = command.split(**" "**, 2);  
 result = DB.*markCompleted*(Integer.*valueOf*(commands[1]));  
 **break**;  
 **case "getmakesstatistics"**:  
 result = DB.*getMakeStatistics*();  
 **break**;  
 **default**:  
 result = **false**;  
 **break**;  
 }  
 **return** result;  
 }  
}

Класс DB

**public class** DB {  
  
 **private static** Connection *connection*;  
  
 **public static void** connect(String database, String user, String password, String port) {  
 **try** {  
 *connection* = DriverManager.*getConnection*((**"jdbc:mysql://localhost:"** + port + **"/"** +  
 database + **"?serverTimezone=UTC"**), user, password);  
 } **catch** (SQLException sqlexc) {  
 **try** {

**ПРИЛОЖЕНИЕ В (продолжение)**

*connection*.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 sqlexc.printStackTrace();  
 }  
 }  
  
 **public static** String login(String login, String password) {  
 ResultSet resultSet;  
 String accountType = **""**;  
  
 String select = **"SELECT** *\** **FROM users WHERE login=? and password=?"**;  
  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(select);  
 preparedStatement.setString(1, login);  
 preparedStatement.setString(2, password);  
 resultSet = preparedStatement.executeQuery();  
  
 **if** (resultSet.next()) {  
 accountType = resultSet.getString(**"account\_type"**);  
 } **else** {  
 accountType = **""**;  
 }  
 } **catch** (SQLException e) {  
 System.***out***.println(**"login func exception"**);  
 e.printStackTrace();  
 }  
 **return** accountType;  
 }  
  
 **public static** List<String> getMakes() {  
 ResultSet resultSet;  
 List<String> makes = **new** ArrayList<>(0);  
  
 String select = **"SELECT** *\** **FROM make"**;  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(select);  
 resultSet = preparedStatement.executeQuery();  
 **while** (resultSet.next()) {  
 makes.add(resultSet.getString(**"make"**));  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 **return** makes;  
 }  
  
 **public static** List<String> getModels(String make) {  
 ResultSet resultSet;  
 List<String> models = **new** ArrayList<>(0);  
  
 String select = **"SELECT** *\** **FROM "** + make + **"\_models"**;  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(select);  
 resultSet = preparedStatement.executeQuery();

**ПРИЛОЖЕНИЕ В (продолжение)**

**while** (resultSet.next()) {  
 StringBuilder temp = **new** StringBuilder();  
 temp.append(resultSet.getString(**"price"**)).append(**" "**).  
 append(resultSet.getString(**"model"**));  
 models.add(temp.toString());  
 }  
 } **catch** (SQLException e) {  
 }  
 **return** models;  
 }  
  
 **public static** ArrayList<String> getOrders() {  
 ResultSet resultSet;  
 ArrayList<String> ordersList = **new** ArrayList<>(0);  
  
 String select = **"SELECT** *\** **FROM course.order"**;  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(select);  
 resultSet = preparedStatement.executeQuery();  
 **while** (resultSet.next()) {  
 StringBuilder order = **new** StringBuilder();  
 order.append(resultSet.getInt(1) + **" "** + resultSet.getString(2) +  
 **" "** + resultSet.getString(3) + **" "** + resultSet.getInt(5) + **" "** +  
 resultSet.getString(6) + **" "** + resultSet.getString(7) +  
 **" "** + resultSet.getString(4));  
 ordersList.add(order.toString());  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 **return** ordersList;  
 }  
  
 **public static** ArrayList<String> getOrderClient(String name, String surname) {  
 ResultSet resultSet;  
 ArrayList<String> ordersList = **new** ArrayList<>(0);  
  
 String select = **"SELECT** *\** **FROM course.order WHERE name=? and surname=?"**;  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(select);  
 preparedStatement.setString(1, name);  
 preparedStatement.setString(2, surname);  
 resultSet = preparedStatement.executeQuery();  
 **while** (resultSet.next()) {  
 StringBuilder order = **new** StringBuilder();  
 order.append(resultSet.getString(**"make"**)).append(**" "**).  
 append(resultSet.getString(**"year"**)).append(**" "**).  
 append(resultSet.getString(**"price"**)).append(**" "**).  
 append(resultSet.getBoolean(**"order\_accepted"**)).append(**" "**).  
 append(resultSet.getBoolean(**"order\_completed"**)).append(**" "**).  
 append(resultSet.getString(**"color"**)).append(**" "**).

**ПРИЛОЖЕНИЕ В (продолжение)**

append(resultSet.getString(**"country"**)).append(**" "**).  
 append(resultSet.getString(**"insurance"**)).append(**" "**).  
 append(resultSet.getString(**"model"**));  
 ordersList.add(order.toString());  
 }  
 } **catch** (SQLException e) {  
  
 }  
 **return** ordersList;  
 }  
  
 **public static** ArrayList<String> getOrdersManager() {  
 ResultSet resultSet;  
 ArrayList<String> list = **new** ArrayList<>(0);  
 String select = **"SELECT** *\** **FROM course.order"**;  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(select);  
 resultSet = preparedStatement.executeQuery();  
 **while** (resultSet.next()) {  
 StringBuilder order = **new** StringBuilder();  
 order.append(resultSet.getInt(**"number"**)).append(**" "**).  
 append(resultSet.getString(**"name"**)).append(**" "**).  
 append(resultSet.getString(**"surname"**)).append(**" "**).  
 append(resultSet.getString(**"make"**)).append(**" "**).  
 append(resultSet.getString(**"year"**)).append(**" "**).  
 append(resultSet.getString(**"price"**)).append(**" "**).  
 append(resultSet.getBoolean(**"order\_accepted"**)).append(**" "**).  
 append(resultSet.getBoolean(**"order\_completed"**)).append(**" "**).  
 append(resultSet.getString(**"color"**)).append(**" "**).  
 append(resultSet.getString(**"country"**)).append(**" "**).  
 append(resultSet.getString(**"insurance"**)).append(**" "**).  
 append(resultSet.getString(**"model"**));  
 list.add(order.toString());  
 }  
 } **catch** (SQLException e) {  
  
 }  
 **return** list;  
 }  
  
 **public static** ArrayList<String> getSupplierOrder() {  
 ResultSet resultSet;  
 ArrayList<String> list = **new** ArrayList<>(0);  
 String select = **"SELECT** *\** **FROM course.supplier\_order"**;  
  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(select);  
 resultSet = preparedStatement.executeQuery();  
 **while** (resultSet.next()) {  
 StringBuilder order = **new** StringBuilder();  
 order.append(resultSet.getInt(**"id"**)).append(**" "**).  
 append(resultSet.getString(**"supplier"**)).append(**" "**).  
 append(resultSet.getString(**"make"**)).append(**" "**).  
 append(resultSet.getString(**"color"**)).append(**" "**).  
 append(resultSet.getInt(**"quantity"**)).append(**" "**).  
 append(resultSet.getString(**"model"**));

**ПРИЛОЖЕНИЕ В (продолжение)**

list.add(order.toString());  
 }  
 } **catch** (SQLException e) {  
  
 }  
 **return** list;  
 }  
  
 **public static** ArrayList<String> getMakeStatistics() {  
 ResultSet resultSet;  
 ArrayList<String> list = **new** ArrayList<>(0);  
 String select = **"SELECT make FROM course.order"**;  
 **double** ravon = 0;  
 **double** geely = 0;  
 **double** haval = 0;  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(select);  
 resultSet = preparedStatement.executeQuery();  
 **while** (resultSet.next()) {  
 **if** (resultSet.getString(**"make"**).equalsIgnoreCase(**"haval"**)) haval++;  
 **if** (resultSet.getString(**"make"**).equalsIgnoreCase(**"geely"**)) geely++;  
 **if** (resultSet.getString(**"make"**).equalsIgnoreCase(**"ravon"**)) ravon++;  
 }  
 **double** ravonPart = ravon / (haval + ravon + geely);  
 **double** geelyPart = geely / (haval + ravon + geely);  
 **double** havalPart = haval / (haval + ravon + geely);  
 list.add(Double.*toString*(ravonPart));  
 list.add(Double.*toString*(geelyPart));  
 list.add(Double.*toString*(havalPart));  
 } **catch** (SQLException e) {  
 }  
 **return** list;  
 }  
  
 **public static boolean** addOrderClient(String order) {  
 String[] orderDetails = order.split(**" "**, 12);  
 String insert = **"INSERT INTO course.order (make, year, name, surname, price, color, country, insurance, model)"** +  
 **"VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)"**;  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(insert);  
 preparedStatement.setString(1, orderDetails[1]);  
 preparedStatement.setString(2, orderDetails[2]);  
 preparedStatement.setString(3, orderDetails[3]);  
 preparedStatement.setString(4, orderDetails[4]);  
 preparedStatement.setString(5, orderDetails[5]);  
 preparedStatement.setString(6, orderDetails[6]);  
 preparedStatement.setString(7, orderDetails[7]);  
 preparedStatement.setString(8, orderDetails[8] + **" "** +  
 orderDetails[9] + **" "** + orderDetails[10]);  
 preparedStatement.setString(9, orderDetails[11]);  
 preparedStatement.executeUpdate();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 **return false**;

**ПРИЛОЖЕНИЕ В (продолжение)**

}  
 **return true**;  
 }  
  
 **public static boolean** addSupplierOrder(String order) {  
 String[] orderDetails = order.split(**" "**, 6);  
 System.***out***.println(order);  
 String insert = **"INSERT INTO course.supplier\_order (make, supplier, model, quantity, color) "** +  
 **"VALUES (?, ?, ?, ?, ?)"**;  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(insert);  
 preparedStatement.setString(1, orderDetails[1]);  
 preparedStatement.setString(2, orderDetails[3]);  
 preparedStatement.setString(3, orderDetails[5]);  
 preparedStatement.setString(4, orderDetails[4]);  
 preparedStatement.setString(5, orderDetails[2]);  
 preparedStatement.executeUpdate();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 **return true**;  
 }  
  
 **public static boolean** removeOrder(**int** orderNumber) {  
  
 String delete = **"DELETE FROM course.order WHERE number=?"**;  
  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(delete);  
 preparedStatement.setInt(1, orderNumber);  
 preparedStatement.executeUpdate();  
 } **catch** (SQLException e) {  
 **return false**;  
 }  
 **return true**;  
 }  
  
 **public static boolean** addManager(String name, String surname, String login, String password) {  
 String insertUser = **"INSERT INTO course.users (account\_type, login, password) VALUES (?, ?, ?)"**;  
 String insertManager = **"INSERT INTO course.managers (name, surname, login, password) VALUES (?, ?, ?, ?)"**;  
  
 **try** {  
 PreparedStatement preparedStatementUser = *connection*.prepareStatement(insertUser);  
 PreparedStatement preparedStatementManager = *connection*.prepareStatement(insertManager);  
 preparedStatementUser.setString(1, **"manager"**);  
 preparedStatementUser.setString(2, login);  
 preparedStatementUser.setString(3, password);  
 preparedStatementManager.setString(1, name);  
 preparedStatementManager.setString(2, surname);  
 preparedStatementManager.setString(3, login);  
 preparedStatementManager.setString(4, password);

**ПРИЛОЖЕНИЕ В (продолжение)**

preparedStatementUser.executeUpdate();  
 preparedStatementManager.executeUpdate();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 **return true**;  
 }  
  
 **public static boolean** addClient(String name, String surname, String login, String password) {  
 String insertUser = **"INSERT INTO course.users (account\_type, login, password) VALUES (?, ?, ?)"**;  
 String insertClient = **"INSERT INTO course.clients (name, surname, login, password) VALUES (?, ?, ?, ?)"**;  
  
 **try** {  
 PreparedStatement preparedStatementUser = *connection*.prepareStatement(insertUser);  
 PreparedStatement preparedStatementClient = *connection*.prepareStatement(insertClient);  
 preparedStatementUser.setString(1, **"client"**);  
 preparedStatementUser.setString(2, login);  
 preparedStatementUser.setString(3, password);  
 preparedStatementClient.setString(1, name);  
 preparedStatementClient.setString(2, surname);  
 preparedStatementClient.setString(3, login);  
 preparedStatementClient.setString(4, password);  
 preparedStatementUser.executeUpdate();  
 preparedStatementClient.executeUpdate();  
 } **catch** (SQLException e) {  
 **return false**;  
 }  
 **return true**;  
 }  
  
 **public static boolean** addSupplier(String name, String coutry, String make) {  
 String insert = **"INSERT INTO course.supplier (country, make, name) VALUES (?, ?, ?)"**;  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(insert);  
 preparedStatement.setString(1, coutry);  
 preparedStatement.setString(2, make);  
 preparedStatement.setString(3, name);  
 preparedStatement.executeUpdate();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 **return true**;  
 }  
  
 **public static** String getClientData(String login, String password) {  
 ResultSet resultSet;  
 String list = **new** String();  
 String select = **"SELECT** *\** **FROM course.clients WHERE login=? and password=?"**;  
 **try** {

**ПРИЛОЖЕНИЕ В (продолжение)**

PreparedStatement preparedStatement = *connection*.prepareStatement(select);  
 preparedStatement.setString(1, login);  
 preparedStatement.setString(2, password);  
 resultSet = preparedStatement.executeQuery();  
 **while** (resultSet.next()) {  
 list = resultSet.getString(**"name"**) + **" "** +  
 resultSet.getString(**"surname"**);  
 }  
 } **catch** (Exception e) {  
 }  
 **return** list;  
 }  
  
 **public static** String getManagerData(String login, String password) {  
 ResultSet resultSet;  
 String list = **new** String();  
 String select = **"SELECT** *\** **FROM course.managers WHERE login=? and password=?"**;  
  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(select);  
 preparedStatement.setString(1, login);  
 preparedStatement.setString(2, password);  
 resultSet = preparedStatement.executeQuery();  
 **while** (resultSet.next()) {  
 list = resultSet.getString(**"name"**) + **" "** +  
 resultSet.getString(**"surname"**);  
 }  
 } **catch** (Exception e) {  
 }  
 **return** list;  
 }  
  
 **public static** ArrayList<String> getInsurance() {  
 ResultSet resultSet;  
 ArrayList<String> list = **new** ArrayList<>(0);  
 String select = **"SELECT** *\** **FROM course.insurance"**;  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(select);  
 resultSet = preparedStatement.executeQuery();  
 **while** (resultSet.next()) {  
 list.add(resultSet.getString(**"price"**) + **" "** +  
 resultSet.getString(**"type"**));  
 }  
 } **catch** (SQLException e) {  
  
 }  
 **return** list;  
 }  
  
 **public static** ArrayList<String> getSupplier(String make) {  
 ResultSet resultSet;  
 ArrayList<String> list = **new** ArrayList<>(0);  
 String select = **"SELECT** *\** **FROM course.supplier WHERE make=?"**;  
  
 **try** {  
 PreparedStatement preparedStatement =

**ПРИЛОЖЕНИЕ В (продолжение)**

*connection*.prepareStatement(select);  
 preparedStatement.setString(1, make);  
 resultSet = preparedStatement.executeQuery();  
 **while** (resultSet.next()) {  
 list.add(resultSet.getString(**"name"**));  
 }  
 } **catch** (SQLException e) {  
  
 }  
 **return** list;  
 }  
  
 **public static** ArrayList<String> getCountry(String make) {  
 ResultSet resultSet;  
 ArrayList<String> list = **new** ArrayList<>(0);  
  
 String select = **"SELECT** *\** **FROM course.supplier WHERE make=?"**;  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(select);  
 preparedStatement.setString(1, make);  
 resultSet = preparedStatement.executeQuery();  
 **while** (resultSet.next()) {  
 list.add(resultSet.getString(**"country"**));  
 }  
  
 } **catch** (SQLException e) {  
 }  
 **return** list;  
 }  
  
 **public static boolean** addInsurance(String price, String type) {  
 String insert = **"INSERT INTO course.insurance (type, price) VALUES (?, ?)"**;  
  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(insert);  
 preparedStatement.setString(1, type);  
 preparedStatement.setInt(2, Integer.*valueOf*(price));  
 preparedStatement.executeUpdate();  
 } **catch** (SQLException e) {  
 **return false**;  
 }  
 **return true**;  
 }  
  
 **public static boolean** markAccepted(**int** orderNumber) {  
 String update = **"UPDATE course.order SET order\_accepted=1 where number=?"**;  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(update);  
 preparedStatement.setInt(1, orderNumber);  
 preparedStatement.executeUpdate();  
 } **catch** (SQLException e) {  
 **return false**;  
 }  
 **return true**;  
 }

**ПРИЛОЖЕНИЕ В (продолжение)**

**public static boolean** markCompleted(**int** orderNumber) {  
 String update = **"UPDATE course.order SET order\_completed=1 where number=?"**;  
 **try** {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(update);  
 preparedStatement.setInt(1, orderNumber);  
 preparedStatement.executeUpdate();  
 } **catch** (SQLException e) {  
 **return false**;  
 }  
 **return true**;  
  
 }  
}

# ПРИЛОЖЕНИЕ Г

**(обязательное)**

**Скрипт генерации базы данных**

CREATE DATABASE IF NOT EXISTS `course` /\*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci \*/;

USE `course`;

CREATE TABLE IF NOT EXISTS `clients` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`name` varchar(20) NOT NULL,

`surname` varchar(20) NOT NULL,

`login` varchar(20) NOT NULL,

`password` varchar(20) NOT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `id` (`id`),

KEY `password` (`password`),

KEY `FK\_clients\_users` (`login`),

KEY `name` (`name`),

KEY `surname` (`surname`),

CONSTRAINT `FK\_clients\_users` FOREIGN KEY (`id`) REFERENCES `users` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40000 ALTER TABLE `clients` DISABLE KEYS \*/;

INSERT INTO `clients` (`id`, `name`, `surname`, `login`, `password`) VALUES

(1, 'Konstantin', 'Zarutskiy', 'user1', 'user1'),

(2, 'Vasya', 'Bogomolov', 'user2', 'user2'),

(3, 'Masha', 'Vasilieva', 'user3', 'user3'),

(4, 'Fedor', 'Bondarchyk', 'user4', 'user4');

/\*!40000 ALTER TABLE `clients` ENABLE KEYS \*/;

CREATE TABLE IF NOT EXISTS `geely\_models` (

`model` varchar(20) NOT NULL,

`price` int(11) NOT NULL,

PRIMARY KEY (`model`),

UNIQUE KEY `geely\_models\_model\_uindex` (`model`),

KEY `model` (`model`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40000 ALTER TABLE `geely\_models` DISABLE KEYS \*/;

INSERT INTO `geely\_models` (`model`, `price`) VALUES

('ATLAS', 25000),

('EMGRAND 7', 12000),

('EMGRAND GT', 24000),

('EMGRAND X7', 15000);

/\*!40000 ALTER TABLE `geely\_models` ENABLE KEYS \*/;

CREATE TABLE IF NOT EXISTS `haval\_models` (

`model` varchar(20) NOT NULL,

`price` int(11) NOT NULL,

PRIMARY KEY (`model`),

UNIQUE KEY `haval\_models\_model\_uindex` (`model`),

**ПРИЛОЖЕНИЕ Г (продолжение)**

KEY `model` (`model`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40000 ALTER TABLE `haval\_models` DISABLE KEYS \*/;

INSERT INTO `haval\_models` (`model`, `price`) VALUES

('6 COUPE', 30000),

('H2', 20000),

('H6', 25000),

('H9', 27000);

/\*!40000 ALTER TABLE `haval\_models` ENABLE KEYS \*/;

CREATE TABLE IF NOT EXISTS `insurance` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`type` varchar(20) NOT NULL,

`price` int(11) NOT NULL,

PRIMARY KEY (`id`),

KEY `type` (`type`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40000 ALTER TABLE `insurance` DISABLE KEYS \*/;

INSERT INTO `insurance` (`id`, `type`, `price`) VALUES

(1, 'standard 15 day', 30),

(2, 'casco 1 year', 1000),

(3, 'standard 30 days', 50);

/\*!40000 ALTER TABLE `insurance` ENABLE KEYS \*/;

CREATE TABLE IF NOT EXISTS `make` (

`make` varchar(20) NOT NULL,

PRIMARY KEY (`make`),

UNIQUE KEY `make` (`make`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40000 ALTER TABLE `make` DISABLE KEYS \*/;

INSERT INTO `make` (`make`) VALUES

('GEELY'),

('HAVAL'),

('RAVON');

/\*!40000 ALTER TABLE `make` ENABLE KEYS \*/;

CREATE TABLE IF NOT EXISTS `managers` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`name` varchar(20) NOT NULL,

`surname` varchar(20) NOT NULL,

`login` varchar(20) NOT NULL,

`password` varchar(20) NOT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `id` (`id`),

KEY `login` (`login`),

KEY `password` (`password`),

CONSTRAINT `FK\_managers\_users` FOREIGN KEY (`id`) REFERENCES `users` (`id`),

CONSTRAINT `FK\_managers\_users\_2` FOREIGN KEY (`login`) REFERENCES `users` (`login`),

**ПРИЛОЖЕНИЕ Г (продолжение)**

CONSTRAINT `FK\_managers\_users\_3` FOREIGN KEY (`password`) REFERENCES `users` (`password`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40000 ALTER TABLE `managers` DISABLE KEYS \*/;

INSERT INTO `managers` (`id`, `name`, `surname`, `login`, `password`) VALUES

(1, 'Ivan', 'Ivanov', 'man1', 'man1'),

(2, 'Petr', 'Petrov', 'man2', 'man2');

/\*!40000 ALTER TABLE `managers` ENABLE KEYS \*/;

CREATE TABLE IF NOT EXISTS `order` (

`number` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`order\_accepted` tinyint(3) unsigned NOT NULL DEFAULT '0',

`insurance` varchar(20) NOT NULL DEFAULT '0',

`country` varchar(20) NOT NULL DEFAULT '0',

`color` varchar(20) NOT NULL DEFAULT '0',

`make` varchar(20) NOT NULL,

`year` year(4) NOT NULL,

`order\_completed` tinyint(3) unsigned NOT NULL DEFAULT '0',

`model` varchar(20) NOT NULL,

`price` int(11) unsigned NOT NULL,

`name` varchar(20) NOT NULL,

`surname` varchar(20) NOT NULL,

PRIMARY KEY (`number`),

KEY `FK\_order\_make` (`make`),

KEY `model` (`model`),

KEY `name` (`name`),

KEY `surname` (`surname`),

KEY `insurance` (`insurance`),

CONSTRAINT `FK\_order\_clients` FOREIGN KEY (`name`) REFERENCES `clients` (`name`),

CONSTRAINT `FK\_order\_clients\_2` FOREIGN KEY (`surname`) REFERENCES `clients` (`surname`),

CONSTRAINT `FK\_order\_insurance` FOREIGN KEY (`insurance`) REFERENCES `insurance` (`type`) ON UPDATE CASCADE,

CONSTRAINT `FK\_order\_make` FOREIGN KEY (`make`) REFERENCES `make` (`make`)

) ENGINE=InnoDB AUTO\_INCREMENT=17 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40000 ALTER TABLE `order` DISABLE KEYS \*/;

INSERT INTO `order` (`number`, `order\_accepted`, `insurance`, `country`, `color`, `make`, `year`, `order\_completed`, `model`, `price`, `name`, `surname`) VALUES

(9, 0, 'standard 15 day', 'China', 'white', 'RAVON', '2016', 0, 'R2', 4800, 'Konstantin', 'Zarutskiy'),

(11, 0, 'standard 15 day', 'Belarus', 'red', 'GEELY', '2016', 0, 'EMGRAND X7', 12000, 'Vasya', 'Bogomolov'),

(13, 0, 'casco 1 year', 'Belarus', 'black', 'GEELY', '2018', 0, 'EMGRAND GT', 24000, 'Konstantin', 'Zarutskiy'),

(15, 1, 'casco 1 year', 'China', 'RED', 'HAVAL', '2019', 0, '6 COUPE', 30500, 'Masha', 'Vasilieva');

**ПРИЛОЖЕНИЕ Г (продолжение)**

/\*!40000 ALTER TABLE `order` ENABLE KEYS \*/;

CREATE TABLE IF NOT EXISTS `ravon\_models` (

`model` varchar(20) NOT NULL,

`price` int(11) NOT NULL,

PRIMARY KEY (`model`),

UNIQUE KEY `ravon\_models\_model\_uindex` (`model`),

KEY `model` (`model`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40000 ALTER TABLE `ravon\_models` DISABLE KEYS \*/;

INSERT INTO `ravon\_models` (`model`, `price`) VALUES

('R2', 4800),

('R3', 5000),

('R4', 5200);

/\*!40000 ALTER TABLE `ravon\_models` ENABLE KEYS \*/;

CREATE TABLE IF NOT EXISTS `supplier` (

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

`country` varchar(20) NOT NULL DEFAULT '0',

`make` varchar(20) NOT NULL DEFAULT '0',

`name` varchar(20) NOT NULL DEFAULT '0',

PRIMARY KEY (`id`),

UNIQUE KEY `id` (`id`),

KEY `make` (`make`),

KEY `name` (`name`),

CONSTRAINT `FK\_supplier\_make` FOREIGN KEY (`make`) REFERENCES `make` (`make`)

) ENGINE=InnoDB AUTO\_INCREMENT=8 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40000 ALTER TABLE `supplier` DISABLE KEYS \*/;

INSERT INTO `supplier` (`id`, `country`, `make`, `name`) VALUES

(1, 'Russia', 'HAVAL', 'HavalRussia'),

(3, 'China', 'HAVAL', 'HavalChina'),

(4, 'Belarus', 'GEELY', 'BelGeely'),

(5, 'China', 'GEELY', 'GeelyChina'),

(6, 'China', 'RAVON', 'RavonChina'),

(7, 'Russia', 'RAVON', 'RavonRussia');

/\*!40000 ALTER TABLE `supplier` ENABLE KEYS \*/;

CREATE TABLE IF NOT EXISTS `supplier\_order` (

`id` int(11) unsigned NOT NULL AUTO\_INCREMENT,

`make` varchar(20) NOT NULL DEFAULT '0',

`supplier` varchar(20) NOT NULL DEFAULT '0',

`model` varchar(20) NOT NULL DEFAULT '0',

`quantity` int(10) unsigned NOT NULL DEFAULT '0',

`color` varchar(20) NOT NULL DEFAULT '0',

PRIMARY KEY (`id`),

KEY `FK\_supplier\_order\_make` (`make`),

KEY `FK\_supplier\_order\_ravon\_models` (`model`),

KEY `FK\_supplier\_order\_supplier` (`supplier`),

CONSTRAINT `FK\_supplier\_order\_make` FOREIGN KEY (`make`) REFERENCES `make` (`make`),

**ПРИЛОЖЕНИЕ Г (продолжение)**

CONSTRAINT `FK\_supplier\_order\_supplier` FOREIGN KEY (`supplier`) REFERENCES `supplier` (`name`)

) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40000 ALTER TABLE `supplier\_order` DISABLE KEYS \*/;

INSERT INTO `supplier\_order` (`id`, `make`, `supplier`, `model`, `quantity`, `color`) VALUES

(4, 'HAVAL', 'HavalRussia', '6 COUPE', 1, 'BLACK');

/\*!40000 ALTER TABLE `supplier\_order` ENABLE KEYS \*/;

CREATE TABLE IF NOT EXISTS `users` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`account\_type` varchar(10) NOT NULL,

`login` varchar(20) NOT NULL,

`password` varchar(20) NOT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `login` (`login`),

KEY `password` (`password`)

) ENGINE=InnoDB AUTO\_INCREMENT=11 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40000 ALTER TABLE `users` DISABLE KEYS \*/;

INSERT INTO `users` (`id`, `account\_type`, `login`, `password`) VALUES

(1, 'admin', 'admin', 'admin'),

(2, 'client', 'user1', 'user1'),

(3, 'manager', 'man1', 'man1'),

(4, 'manager', 'man2', 'man2'),

(6, 'client', 'user2', 'user2'),

(7, 'client', 'user3', 'user3'),

(8, 'client', 'user4', 'user4'),

(9, 'client', 'user5', 'user5');