Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет инженерно-экономический

Кафедра экономической информатики

Дисциплина Программирование сетевых приложений

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

«Разработка системы автоматизации работы школы иностранных языков»

|  |  |
| --- | --- |
| Студент | Скачко Г.Д.  гр. 773601 |
| Руководитель | Богданова Е.А.  ассистент кафедры  экономической  информатики |

Минск 2019

**СОДЕРЖАНИЕ**

[Введение 5](#_Toc26646065)

[1 Описание автосалона 6](#_Toc26646066)

[2 Постановка задачи и обзор методов её решения 7](#_Toc26646067)

[2.1 Постановка задачи 7](#_Toc26646068)

[2.2 Обзор методов решения поставленных задач 8](#_Toc26646069)

[3 Функциональное моделирование 10](#_Toc26646070)

[4 Информационная модель системы и её описание 14](#_Toc26646071)

[5 Диаграммы представления и их описание 16](#_Toc26646073)

[5.1 Диаграмма вариантов использования 16](#_Toc26646074)

[5.2 Диаграмма последовательности 16](#_Toc26646075)

[5.3 Диаграмма состояний 17](#_Toc26646076)

[5.4 Диаграмма компонентов 17](#_Toc26646077)

[5.5 Диаграмма](#_Toc26646078)

[развертывания 17](#_Toc26646078)

[5.6 Диаграмма классов 17](#_Toc26646079)

[6 Описание алгоритмов реализующих бизнес-логику проектируемой системы 19](#_Toc26646080)

[6.1 Алгоритм добавления нового пользователя 19](#_Toc26646081)

[6.2 Алгоритм фильтрации автомобилей 19](#_Toc26646082)

[7 Руководство пользователя 20](#_Toc26646083)

[7.1 Вход под учетной записью пользователя 20](#_Toc26646084)

[7.2 Вход под учетной записью администратора 23](#_Toc26646085)

[8 Результаты тестирования разработанной системы 27](#_Toc26646086)

[Заключение 30](#_Toc26646087)

[Список использованных источников 31](#_Toc26646088)

[Приложение А UML-диаграммы 32](#_Toc26646089)

[Приложение Б Схема алгоритмов 36](#_Toc26646090)

[Приложение В Листинг исходного кода программного продукта 38](#_Toc26646091)

[Приложение Г Листинг скрипта генерации базы данных 48](#_Toc26646092)

# **ВВЕДЕНИЕ**

С каждым годом потребности в использовании иностранных языков возрастают. Знание иностранного языка становится жизненной необходимостью в профессиональной деятельности, в учебном процессе, на рынке труда, в повседневной жизни. Языковая культура всегда была и остается неотъемлемым компонентом общечеловеческой культуры. На Лиссабонской конференции 2000 года владение языками было включено в число пяти «ключевых компетенций» а на заседании Европарламента в декабре 2006 года владение языками включено в расширенный список из восьми «ключевых компетенций», которые необходимы человеку для полноценной жизни в обществе [1].

IT — это одна из самых глобализованных областей человеческой деятельности и общепризнанным языком IT является английский. Если вы не владеете языком на достаточном уровне, то вас ожидают множество разнообразных сложностей в карьере и профессиональном развитии. Помимо вполне очевидного ограничения на список работодателей, на которых вы сможете работать (только российские компании, ориентированные исключительно на российский рынок), что сразу сокращает ваши возможности зарплатного и карьерного роста, есть и менее очевидные ограничения. Главное из которых — те проекты и технологии, с которыми вы сможете работать [2].

Благодаря возрастающей потребности в изучении иностранных языков по всему миру появляется множество частных школ, которые предоставляют соответствующие услуги по преподаванию.

Цель курсового проекта – упростить и оптимизировать работу школ иностранных языков посредством частичной автоматизации системы учета и записи студентов.

Для достижения поставленной цели необходимо решить задачи:

– изучить систему работы языковых школ;

– проанализировать изученную информацию;

– спроектировать модули будущей программы;

– спроектировать и создать базу данных;

– разработать клиентскую и серверную части программы;

– протестировать полученный программный продукт.

# **1 ОПИСАНИЕ ШКОЛЫ ЯЗЫКОВ**

# **2 ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЁ РЕШЕНИЯ**

## **2.1 Постановка задачи**

Большинство частных языковых школ предлагают обширный выбор своим потенциальным студентам. Каждая из таких организаций может одновременно обучать сотни студентов десяткам различных языков. Всё это становится возможным благодаря подробному разделению групп по языкам и уровням сложности для каждой отдельной группы. Составлять расписание с таким большим количеством переменных становится намного проще при помощи специального программного средства, которое позволяет вести абсолютный учёт и статистику во всех аспектах работы школы.

Данное приложение будет иметь два режима доступа: администратора и пользователя. Целевой аудиторией пользовательского режима являются студенты, как действующие, так и потенциальные. Учащимся приложение поможет изучить расписание занятий их группы, найти преподавателя, отфильтровать занятия по датам и времени. Те же, кто ещё не участвует в процессе обучения имеют возможность исследовать различную информацию о языках и соответствующих программах обучения, чтобы впоследствии выбрать самый подходящий для себя вариант и записаться.

Режим администратора задуман как инструмент управления данными всей системы языковой школы. В его возможности входят всевозможные функции просмотра, записи, редактирования и удаления данных в системе. Помимо данных, касающихся непосредственно образовательного процесса, администратор уполномочен контролировать список всех пользователей системы и формировать содержательные отчёты о нынешнем состоянии школы.

Таким образом, приложение должно реализовывать следующие функции:

– авторизация пользователей и администраторов в системе;

– добавление новых пользователей и администраторов;

– добавление данных о новых занятиях, группах, языках и учителях;

– функции для записи студентов;

– составление и предоставление администратору отчета о данных системы;

– возможность простой настройки системы;

– возможность сохранять и просматривать ранее сохраненные данные и отчеты.

## **2.2 Обзор методов решения поставленных задач**

Объектно-ориентированный язык Java, выбранный для реализации данного курсового проекта, предоставляет огромное количество готовых программных решений для тех или иных задач. Так, для создания графического пользовательского интерфейса на стороне клиента используется платформа JavaFX на основе языка Java и призвана заменить собой использовавшуюся ранее и уже морально устаревшую библиотеку Swing. Она позволяет реализовывать пользовательские интерфейсы как для десктопных приложений, так для интернет-приложений. Исходный код интерфейса хранится в файле с расширением FXML. Сам интерфейс строится в программном приложении Scene Builder 2.0 от компании GLUON[4].

Реализация технологии «клиент-сервер» осуществляется встроенными средствами Java с использованием протокола TCP/IP, так как он является самым распространенным промышленным стандартом для большинства компьютерных сетей мира (в том числе самой большой сети – Интернет) и обеспечивает установление надежного соединения между клиентской и серверной частью приложения при наличии стабильного и быстрого соединения между компьютерами, при этом само число соединений никак не ограничивается.

На стороне сервера для хранения и обработки информации используется свободно распространяемая и бесплатная система управления базами данных MySQL версии 8.0. Она является одной из самых популярных систем и простых для работы с базами данных, особенно в клиент-серверной и веб-разработке, а также предоставляет множество средств для проектирования, обратного инжениринга и резервного копирования данных.

Основной средой разработки для проекта является Intellj IDEA 2019.3 от компании JetBrains. Данный продукт распространяется по ежемесячной платной подписке, однако она абсолютно и полностью бесплатна для студентов. Для создания UML-моделей в стандарте IDEF0 использовалось CASE-средство CA AllFusion Process Modeler r7. Для создания UML-диаграмм используется средство Enterprise Architect 15, которое позволяет создавать всевозможные диаграммы: классов, вариантов использования, развертывания, состояний, последовательности с помощью интуитивно понятного моделирования. Также имеется возможность создания логической модели реляционной базы данных с последующим созданием SQL-скрипта для генерации физической модели в самой СУБД.

# **3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ**

Для разработки функциональной модели программного приложения был выбран стандарт IDEF0.

На контекстной диаграмме (рисунок 3.1) представлена функциональная модель «Система автоматизации работы школы иностранных языков», а также определены потоки входных и выходных данных, механизмы ограничения и управления данными.

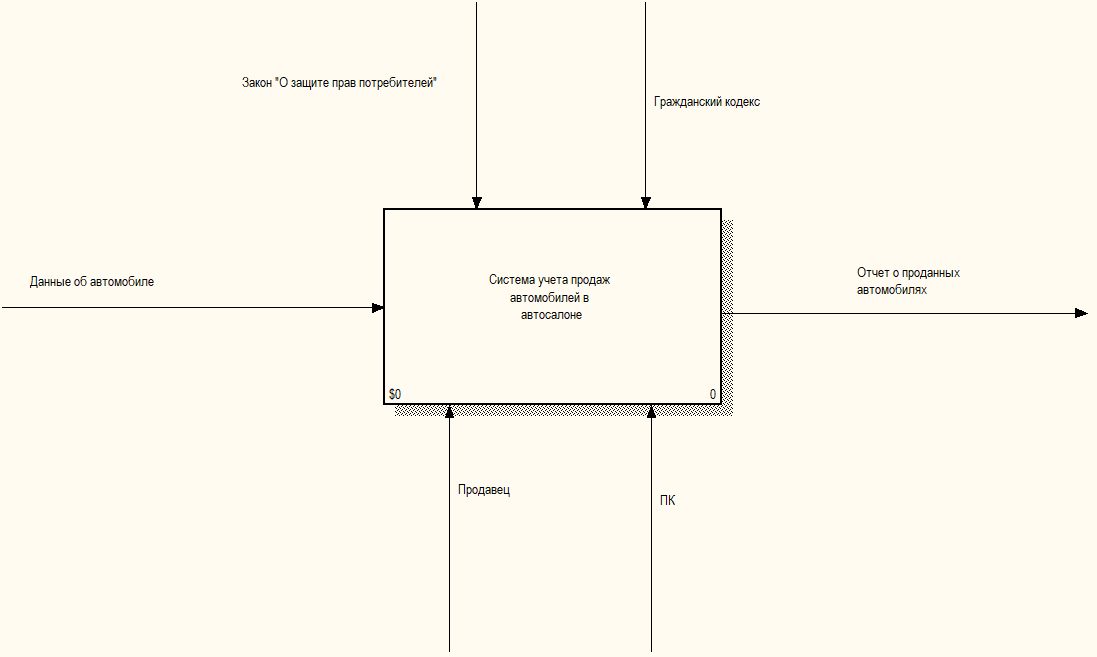


Рисунок 3.1 - Контекстная диаграмма

Входной поток включает данные об автомобиле для последующего внесения в систему учета. Основными исполнителями являются продавец, который использует ПК. В процессе своей работы они подчиняются таким управляющим механизмам, как закон «О защите прав потребителей», а также гражданскому кодексу РБ. В итоге в выходном потоке мы должны получить отчет о проданных автомобилях.

Рисунок 3.2 отображает декомпозицию контекстной диаграммы. В начале продавец добавляет данные в систему учета, после этого следует обработка и структурирование данных для последующей записи в систему учета и формирования отчета о проданных автомобилях.

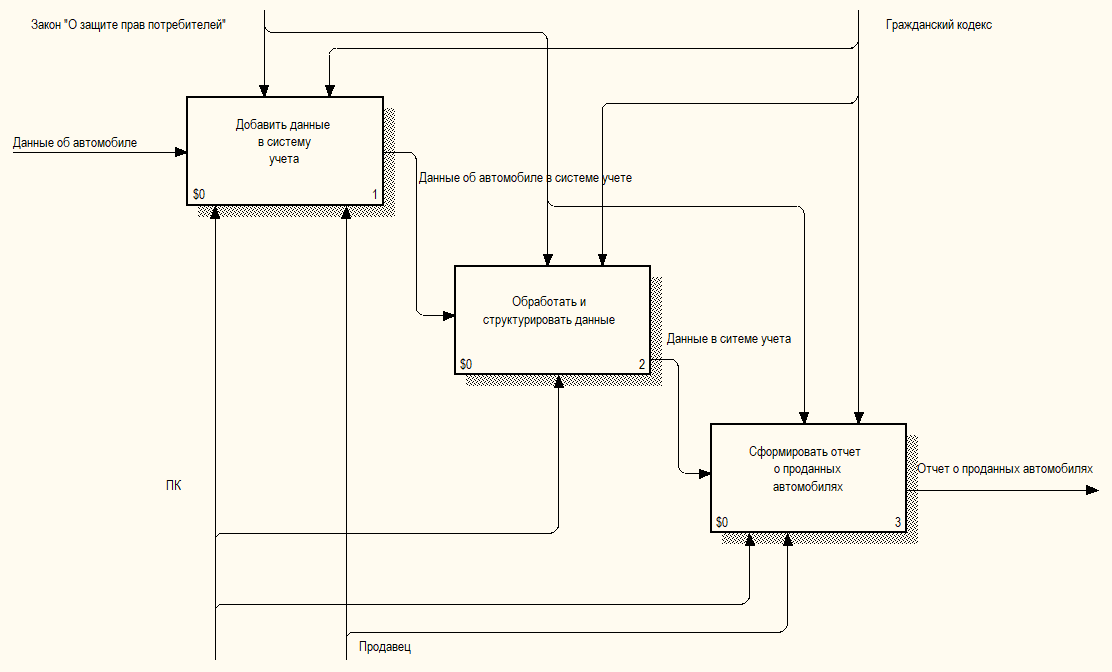


Рисунок 3.2 – Декомпозиция контекстной диаграммы

Подробное описание процесса добавления данных об автомобиле в систему учета представлено на рисунке 3.3:

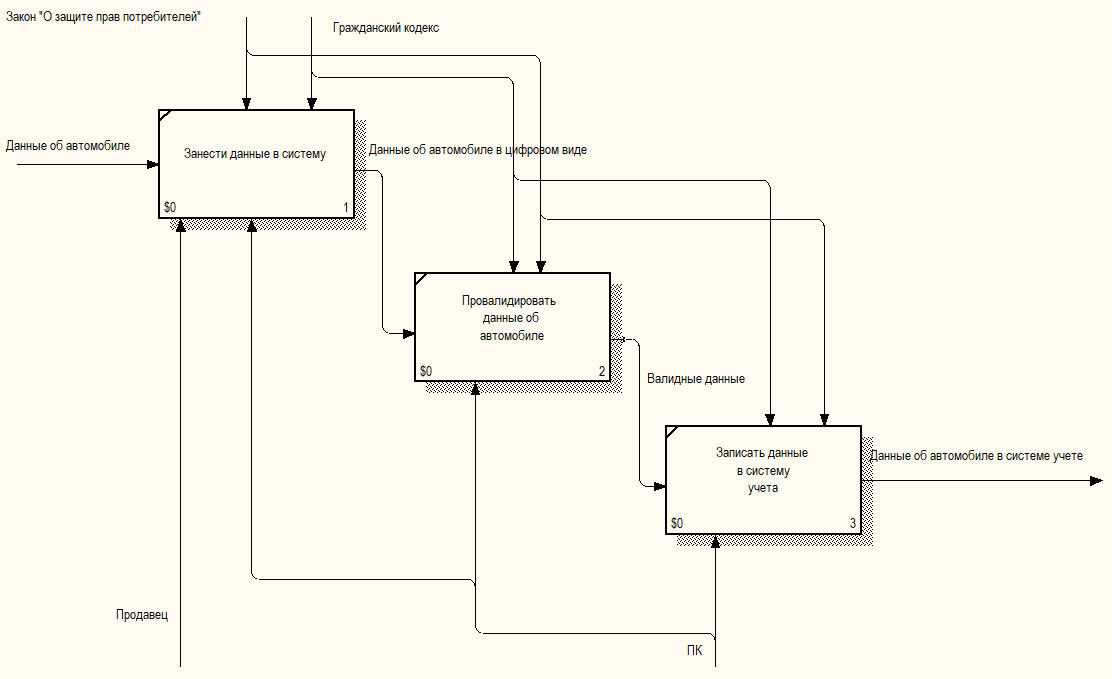


Рисунок 3.3 – Декомпозиция блока «Добавить данные в систему учета»

Данные об автомобилях заносятся в систему. После этого данные проходят валидацию, после этого данные записываются в базу системы учета.

Рисунок 3.4 представляет декомпозицию блока «Обработать и структурировать данные». На данном этапе происходит сортировка данных по дате продажи и стоимости с последующей записью отсортированных данных в систему.

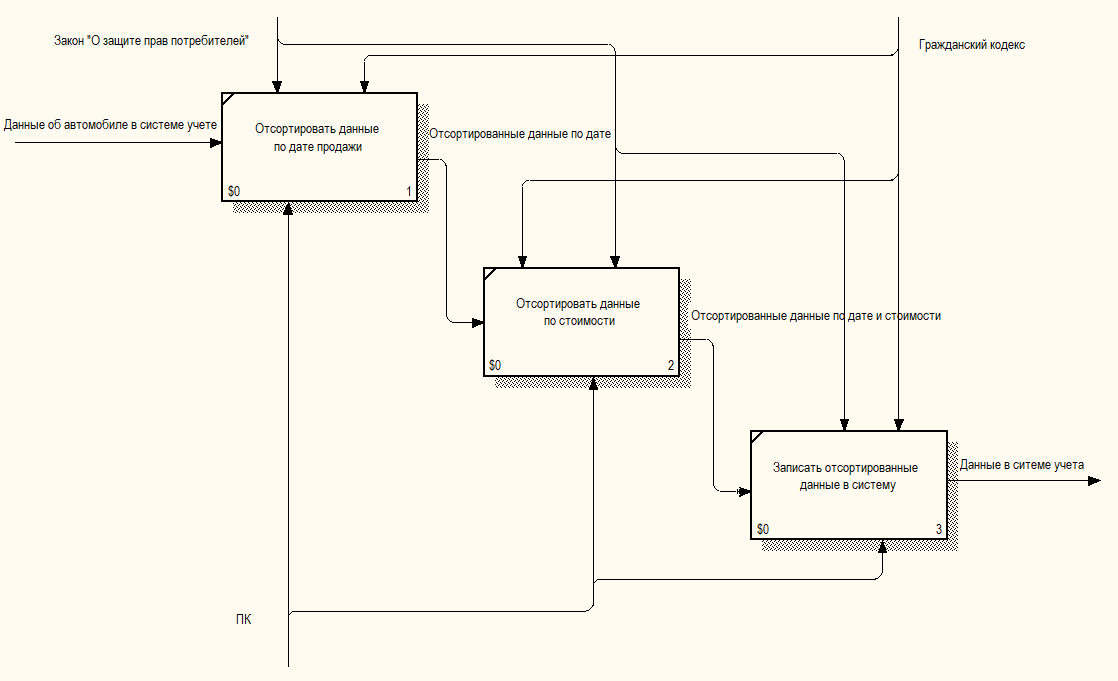


Рисунок 3.4 – Декомпозиция блока «Обратобать и структурировать данные»

Рисунок 3.5 представляет декомпозицию блока «Сформировать отчет о проданных автомобилях». На данном этапе происходит выборка данных, где берутся данные о проданных автомобилей. После происходит выборка по дате продаж. Затем все эти данные выгружаются в файл для формирования отчета.

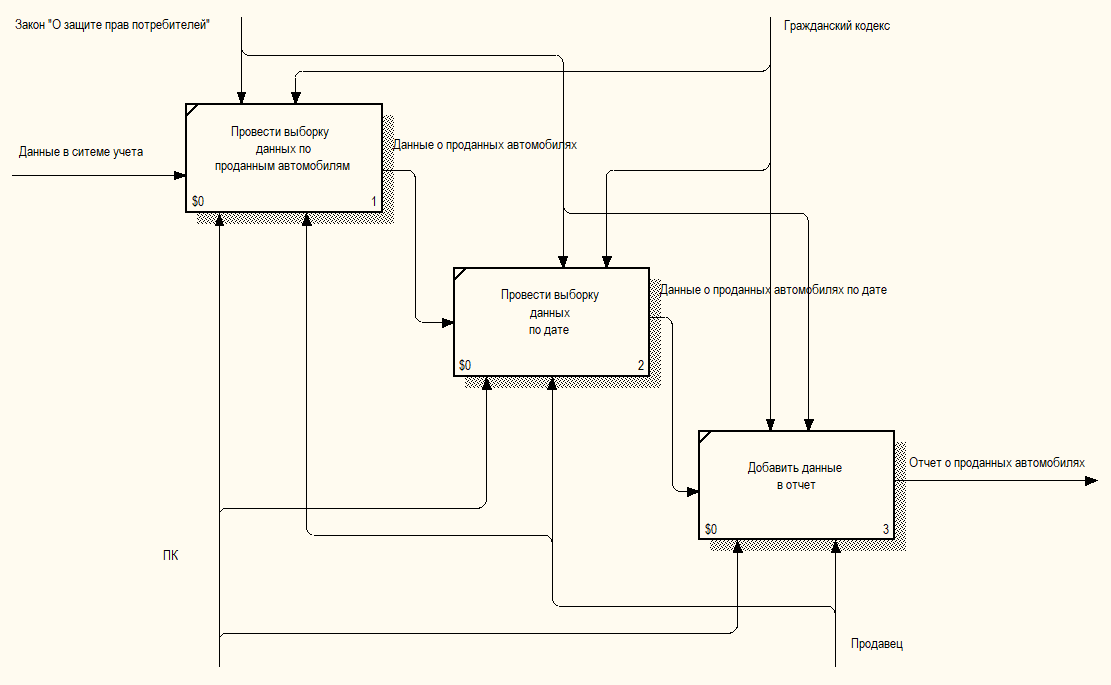


Рисунок 3.5 – Декомпозиция блока «Сформировать отчет о проданных автомобилях»

На основании построенной функциональной модели можно спроектировать и составить информационную модели системы для последующей разработки программного продукта.

# **4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЁ ОПИСАНИЕ**

В курсовом проекте используется база данных MySQL Server. В ней созданы 5 базовых сущностей для функционирования языковой школы:

– lesson;

– student;

– group;

– subject;

– teacher;  
 А также одна сущность для работы её системы авторизации:

– users;

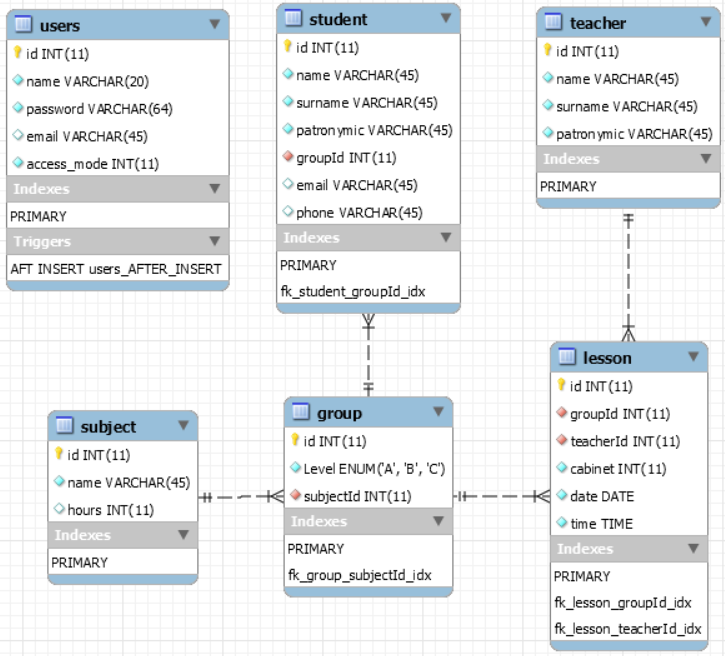


Рисунок 4.1 – Схема базы данных

Сущность users используется для авторизации пользователей и администратора.

Сущность lesson хранит данные о занятии, группе для которой оно организовано, преподавателе, который будет его проводить, а также о дате, месте и времени его начала.

Сущность student хранит данные о студенте, а именно его персональные данные, данные о принадлежности к группе и контактную информацию.

Сущность group хранит данные об отдельной группе, языке, который она изучает и об уровне сложности.

Сущность subject хранит данные о языке, такие как его название и предположительное количество часов, требуемое для его изучения.

Сущность teacher хранит персональные данные учителя.

# **5 ДИАГРАММЫ ПРЕДСТАВЛЕНИЯ И ИХ ОПИСАНИЕ**

## **5.1 Диаграмма вариантов использования**

UML-диаграмма «Use case» (диаграмма вариантов использования) описывает то, что система может делать в процессе своего функционирования, т.е. является своеобразной «инструкцией», которая помогает пользователям понять, как использовать данную систему [7].

В данном курсовом проекте есть два действующих лица (актёра) – администратор и пользователь.

Пользователь имеет функционал записи студентов, просмотра различной текстовой и графической информации о языках и школе и редактирования собственных данных.

Администратору доступны возможности по управлению всеми данными, такие как добавление, удаления и редактирование, а также формирование отчетов по работе школы.

Диаграмма вариантов использования показана на рисунке А.1 приложения А.

## **5.2 Диаграмма последовательности**

Диаграмма последовательности, представленная на рисунке А.2 приложения А, отображает процесс по предоставлению пользователю записей, о автомобилях хранящихся в системе. Вначале пользователь выбирает фильтр по которому будет проходить выборка данных. Это можно сделать с помощью графического интерфейса на клиентской стороне приложения. Получив запрос от пользователя, клиентское приложения обрабатывает его о передает серверу сообщение с фильтром для составления запроса. Сервер «парсит» данную информацию, генерирует SQL-скрипт для запроса информации. Далее сервер выполняет подключение к базе данных (в нашем случае к MySQL), извлекает из нее всю необходимую информации и генерирует массив JSON-объектов в качестве ответа. Далее массив отправляется на сторону клиента, где «парсится» в удобную для чтения форму в виде таблицы и демонстрируется в графическом интерфейсе.

## **5.3 Диаграмма состояний**

Данный тип диаграмм служит для описания поведения объекта на основе его состояний и переходов между ними (рисунок А.3 приложение А). В нашем случае учетные данные пользователя после введения в форму добавления отправляются на валидацию. В случае ее успешного прохождения, объект с данными передается дальше на сервер, в случае ошибки – данные возвращаются в форму добавления пользователя, при этом пользователю указывается, что же он ввел не так. Оказавшись на сервере, данные записываются в БД, таки образом завершая очередной виток своего жизненного цикла.

## **5.4 Диаграмма компонентов**

Отображает архитектуру разрабатываемой системы с помощью установления зависимостей между ее компонентами. Пакеты в ее контексте – это пакеты приложения, содержащие классы и методы для работы с ними. Все классы содержащиеся в пакете находятся в логической зависимости и разделяют логику. На клиентской стороне большинство классов отвечает за предоставление пользователю графического интерфейса, на серверной – за бизнес-логику приложения. Представлена на рисунке А.4 приложения А.

## **5.5 Диаграмма развертывания**

Используется для отображения основных устройств и пакетов системы, которые используют те или иные компоненты. В данном случае клиент и сервер используют JDK версии 11, связь же между ними устанавливается по протоколу TCP/IP. Кроме того, сервер использует драйвер JBDC для соединения с базой данных и последующей работы с ней.

## **5.6 Диаграмма классов**

Данная диаграмма представляет классы из программы с их взаимосвязями, методами и атрибутами. В данном проекте она состоит из двух частей: классов клиентской части и классов серверной части.

Набор классов клиентской части представлен на рисунке А.7 приложения А. Так как основная логика программы находится на сервере, а классы пользовательского интерфейса отображать на диаграмме нецелесообразно, ввиду огромного количества методов, то она содержит всего шесть классов и один интерфейс: Car, OrderCar, User, Report, ServerConnection, TCPConnection, TCPListener.

Класс Car реализует весь функционал для работы с автомобилями на стороне клиента. OrderCar реализует функции для работы с заказанными автомобилями. User содержит функционал для авторизации и работы с пользователями системы. Report отвечает за генерацию Excel отчетов. Классы ServerConnection и TCPConnection и интерфейс TCPListener отвечают за соединение с сервером, отправку запросов на сервер и получение запросов с сервера.

На сервере реализуется бизнес-логика приложения, поэтому и классов там больше: Car, Report, OrderCar, Server, AdminEndpoint, UserEndpoint, User, TCPConnection, Database, Service и интерфейс TCPListener.

Класс Car реализует весь функционал для работы с автомобилями на стороне сервера и CRUD-операций в базу данных, а также фильтрации данных по полю «CAR\_STATUS». Класс Report и OrderCar отвечает за отправку и генерацию ответов для создания отчетов и просмотра заказанных автомобилей соответственно. Классы Server и TCPConnection, интерфейс TCPListener реализуют работу самого сервера. TCPConnection отвечает за создание серверного сокета и потоков каждому подключившемуся пользователю. TCPListener отвечает за выделение каждому пользователю методов для работы с серверными функциями. Server реализует модель многопоточного сервера, используя TCPListener и TCPConnection. Классы AdminEndpoint и UserEndpoint отвечают за разделение функционала сервера для администратора и пользователя соответственно. Service отвечает за начало работы сервиса для подключившегося пользователя. Класс User используется для авторизации пользователя на серверной части приложения. Класс Database отвечает за создание соединения к базе данных и работу с ней.

# **6 ОПИСАНИЕ АЛГОРИТМОВ РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ ПРОЕКТИРУЕМОЙ СИСТЕМЫ**

## **6.1 Алгоритм добавления нового пользователя**

На рисунке Б.1 приложения Б представлена схема алгоритма добавления нового пользователя. Если администратор хочет добавить нового пользователя, он выбирает соответствующий пункт в меню, после чего клиентская часть приложения устанавливает соединение с сервером и отправляет ему соответствующий запрос на добавление. Сервер принимает данный запрос, определяет, что ему нужно сделать и начинает принимать учетные данные пользователя. Далее осуществляется подключение к базе данных и запрос на добавление данных в соответствующую таблицу. Если данный пользователь уже существует, то база данных генерирует исключение о невозможности добавления, в противном случае запись добавляется в базу. Клиенту передается ответ о конечном результате операции добавления.

## **6.2 Алгоритм фильтрации автомобилей**

Одной из важнейших функци данного программного приложения является фильтрация автомобилей по их статусу. Алгоритм данного действия выглядит следующим образом (рисунок Б.2 приложение Б):

1. Клиент получает от пользователя команду на фильтрацию данных по определенному критерию и устанавливает соединение с сервером.
2. Сервер принимает запрос на фильтрацию и формирует соответствующий запрос.
3. Подключается БД, в ней запускается соответствующий запрос.
4. Если данные с соответствующим критерием были найдены – генерируется JSON-объект с этими данными. Если же данные не найдены – генерируется исключение.
5. Сервер передает ответ клиенту.

# **7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Чтобы начать работу в приложении, необходимо пройти авторизацию. Выбирать режим доступа не требуется, он будет определён автоматически в зависимости от уровня доступа вашего аккаунта.

Кнопка, находящаяся правее кнопки «Войти» с изображением знака «+» реализует функцию регистрации. Чтобы зарегистрироваться достаточно ввести данные авторизации и единожды нажать на кнопку «+». После этого по введённым данным будет открыт пользовательский доступ к приложению.

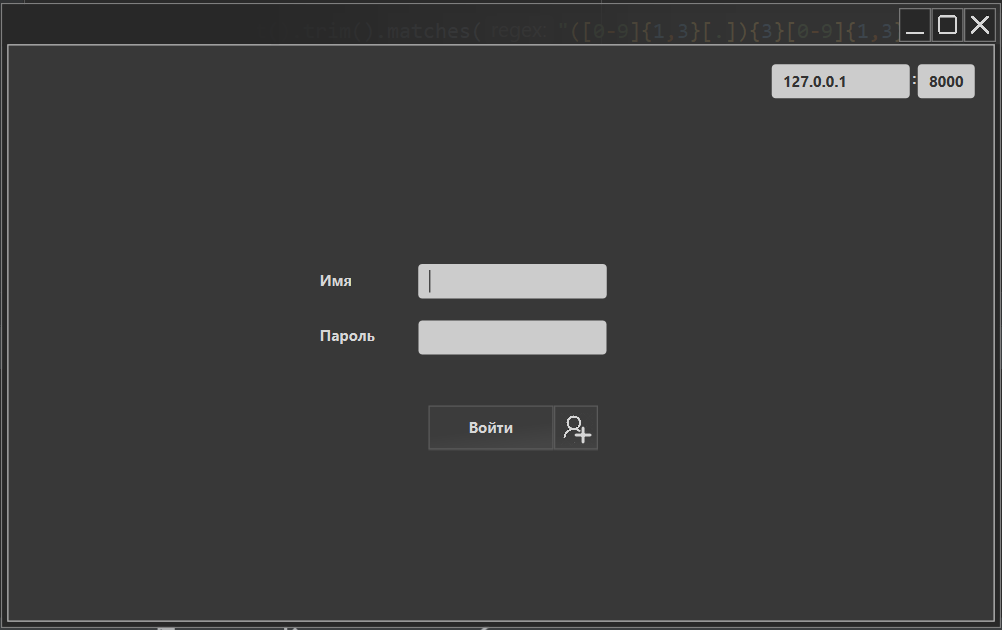


Рисунок 7.1 – Авторизация в приложении

Кроме функционала авторизации и регистрации на данном этапе реализован ручной ввод IP-адреса и порта для подключения к серверу, на случай если серверная часть приложения поменяет свою сетевую конфигурацию.

## **7.1 Вход под учетной записью пользователя**

В случае успешной авторизации в качестве пользователя, мы сразу попадаем на информационную страницу о языках, доступных для изучения, показанную рисунке 7.2.

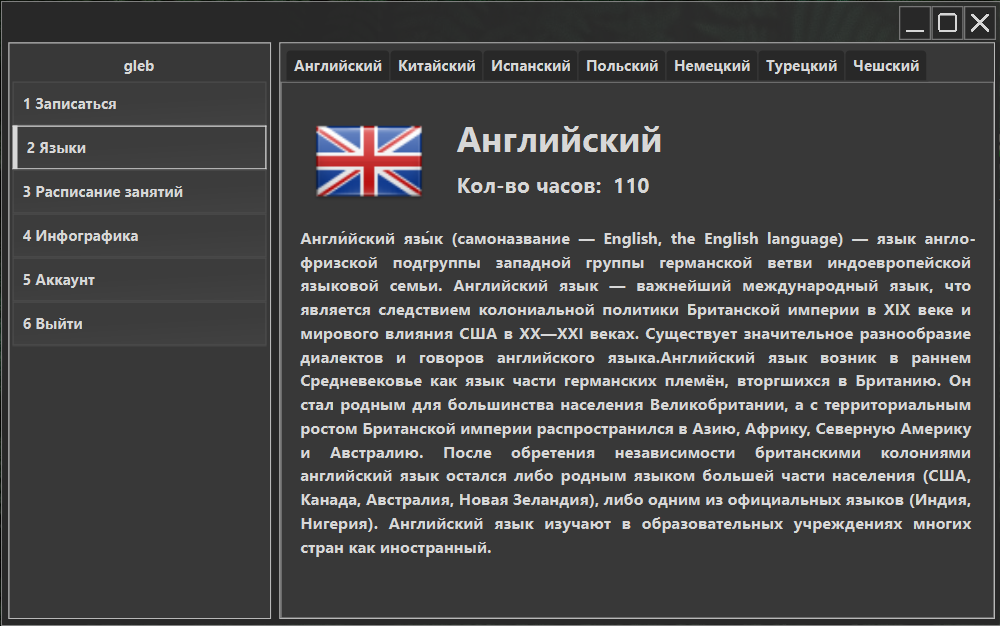


Рисунок 7.2 – Информационная страница о языках

На данной странице мы можем увидеть несколько вкладок с названиями различных языков, возможность изучения которых школа предлагает пользователю. На каждой из них присутствует краткое описания истории языка и предполагаемое количество часов, требуемое на обучение.

После нажатия на кнопку «Записаться», мы попадаем на страницу для записи новых студентов в группы, представленную на рисунке 7.3. На ней нам предлагается ввести свои персональные данные, контактную информацию (по желанию) и желаемый язык и уровень обучения. После ввода данных пользователь может подобрать себе требуемую группу нажав кнопку «Подобрать группу». После этого в таблице правее формы заполнения появится одна или более кнопок «Записаться», соответствующих группам, номера которых указаны в колонке правее.

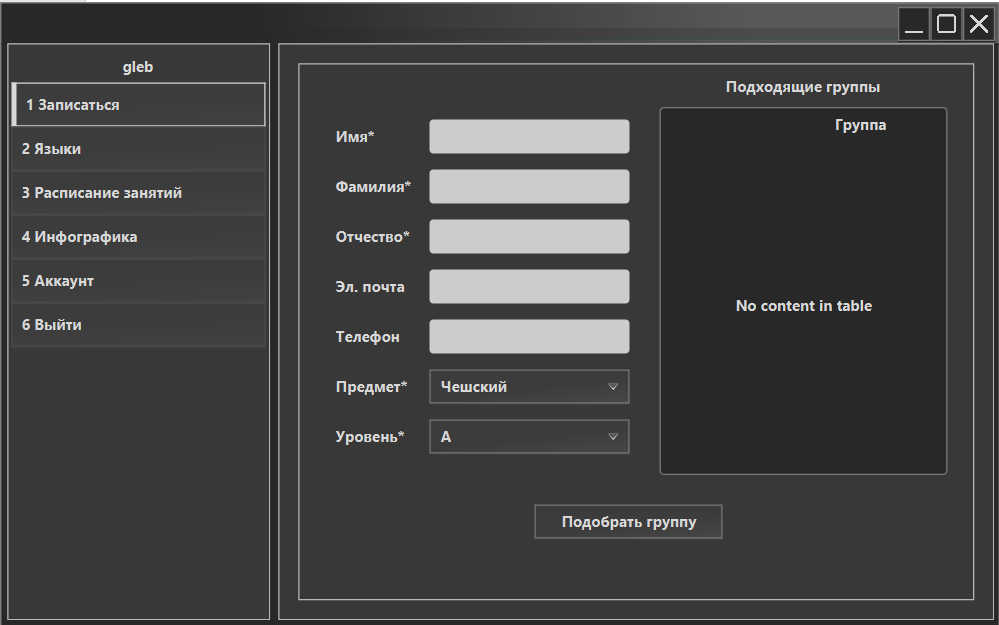


Рисунок 7.3 – Страница для записи студентов

На рисунке 7.4 представлена страница для работы с расписанием школы, которая предоставляет нам широкие возможности по поиску и фильтрации списка занятий. Она нам понадобится после записи в группу.

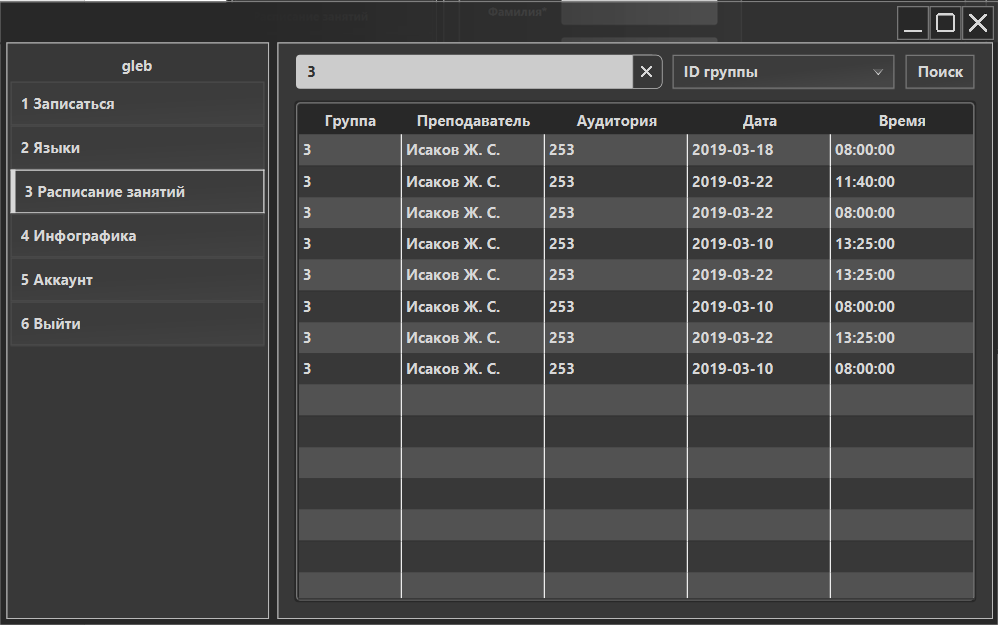


Рисунок 7.4 – Страница расписания занятий

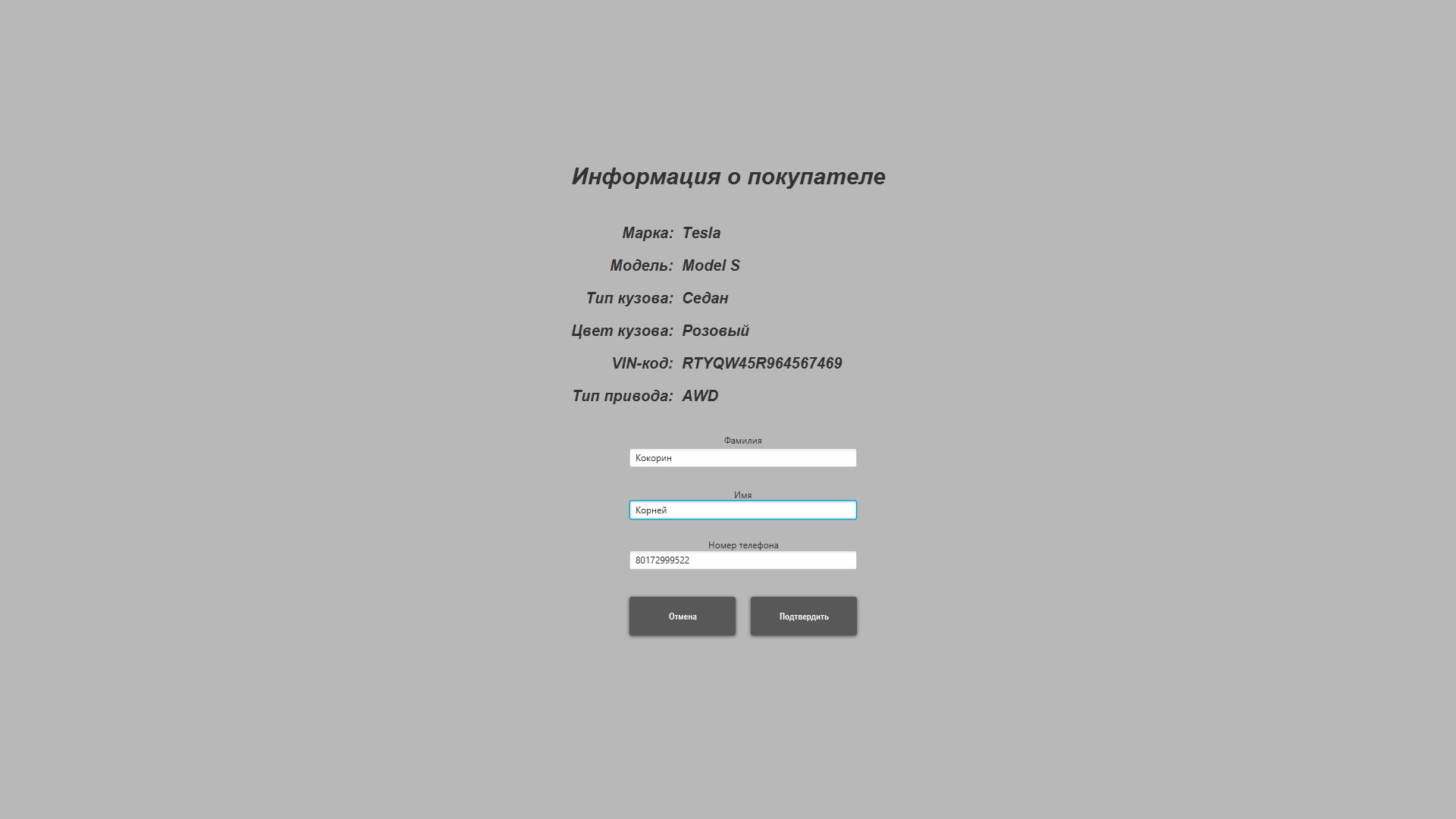


Рисунок 7.5 – Страница оформления заказа

## **7.2 Вход под учетной записью администратора**

На стартовой странице администратора присутствуют элементы для управления аккаунтами. В правом верхнем углу находится кнопка для получения информации о приложении. Слева размещены кнопки для получения отчетов за заданный день, либо отчет о всех автомобилях хранящихся в базе. Сформировать отчет по продажам и отчет по заказам можно лишь за выбранный день.

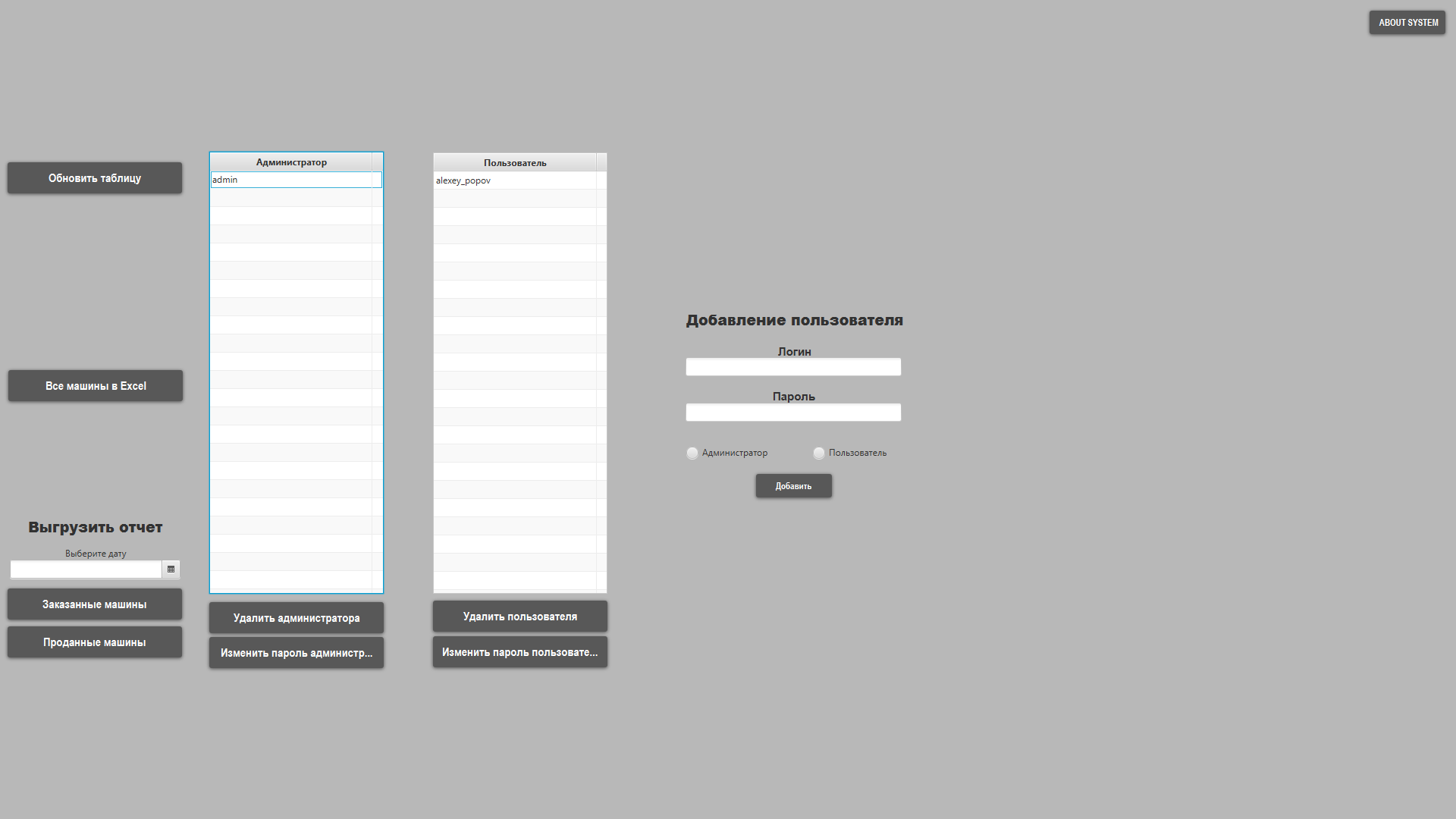


Рисунок 7.6 – Стартовая страница администратора

После нажатия на кнопку «About system» мы получаем вкладку с вресией программы, ее разработчиком, текущим портом соединения и текущей учетной записи пользователя(рисунок 7.7).

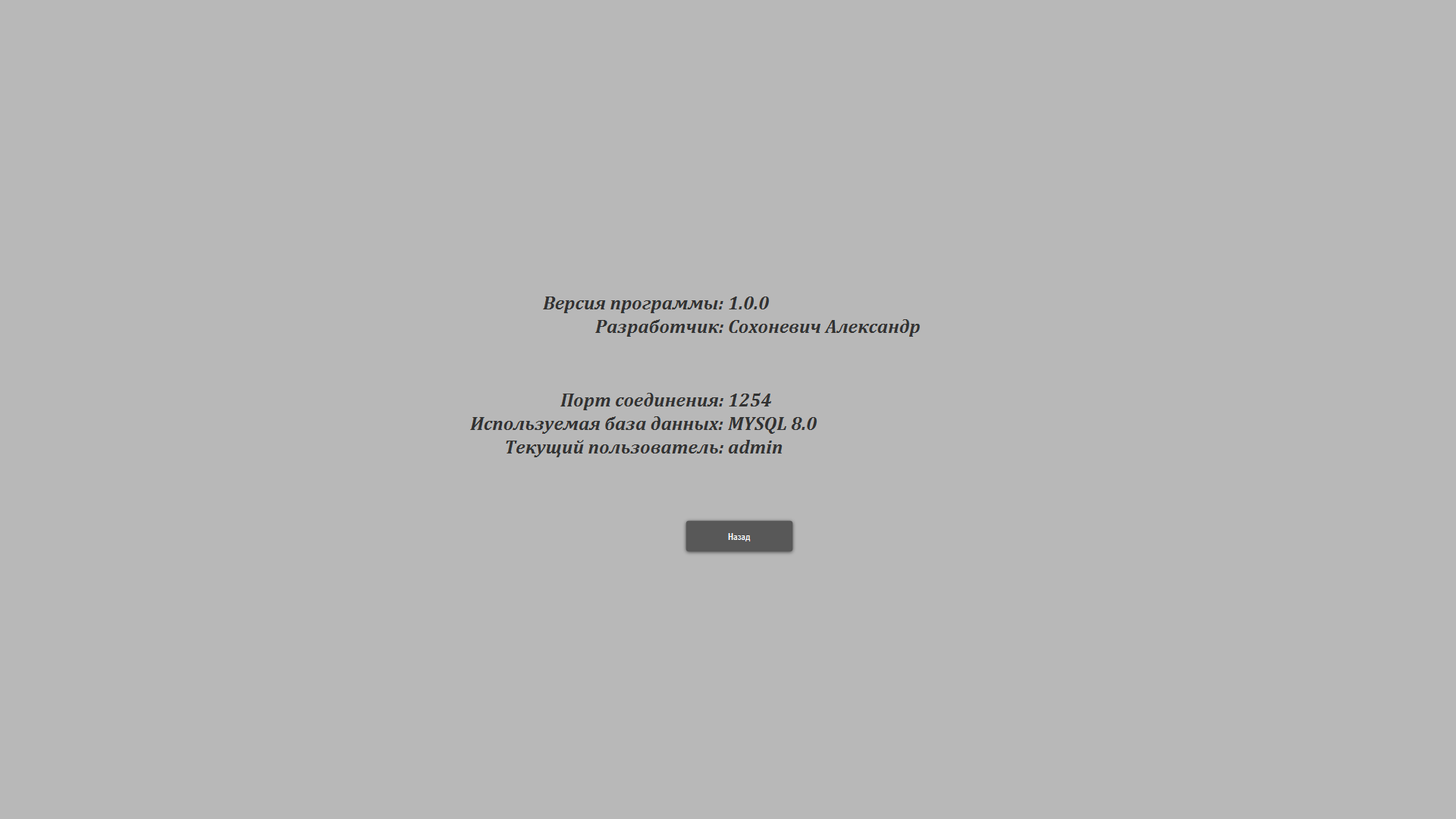


Рисунок 7.7 – Страница с текущей информацией о системе

Все сформированные отчеты сохраняются в отдельной папке Reports в директории Documents созданной в любом компьютере (рисунок 7.8).

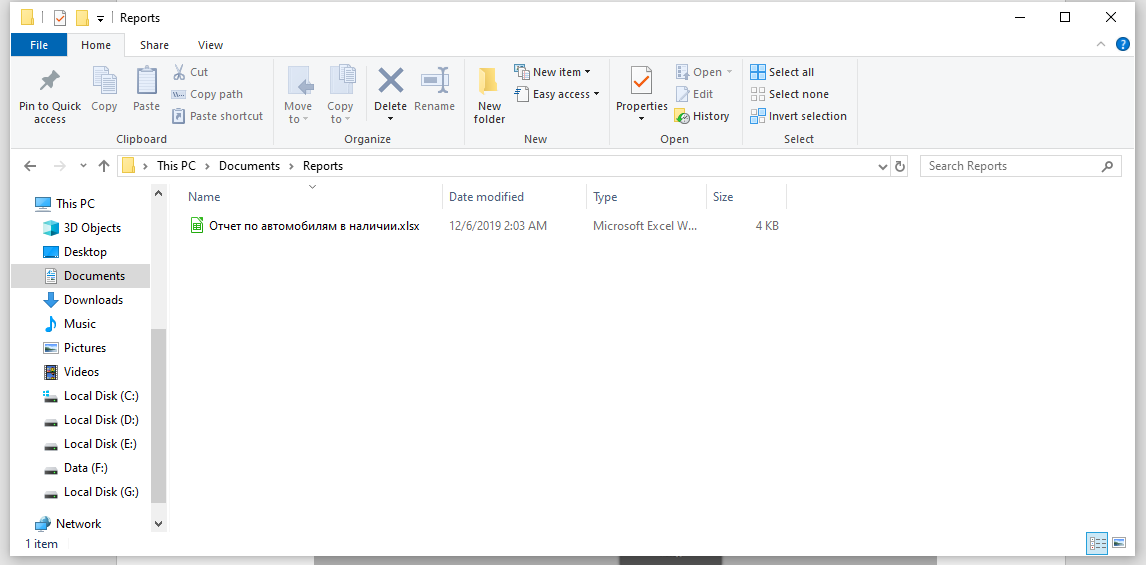


Рисунок 7.8 – Сформированный отчет

Пользовательский интерфейс на стороне сервера представлен в виде консольного приложения (рисунок 7.9).

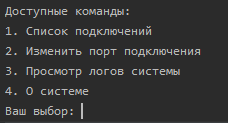


Рисунок 7.9 – Пользовательский интерфейс сервера

На рисунке 7.10 представлена работа функции «Список подключений». Она позволяет просмотреть всех текущих пользователей подключенных к данному серверу.

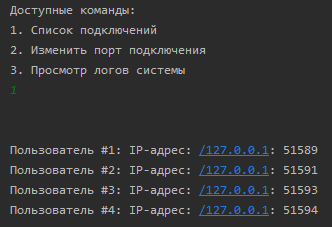


Рисунок 7.10 – Просмотр списка подключений

На рисунке 7.11 представлена работа функции «Изменить порт подключения». Она позволяет изменить порт сервера, по которому клиентская часть приложения сможет подключаться к серверу.

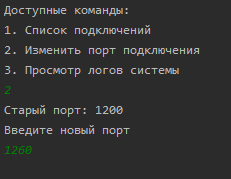


Рисунок 7.11 – Изменение порта подключения

На рисунке 7.12 представлена работа функции «Просмотр логов системы». Там отображается дата, время и сама операция в системе.

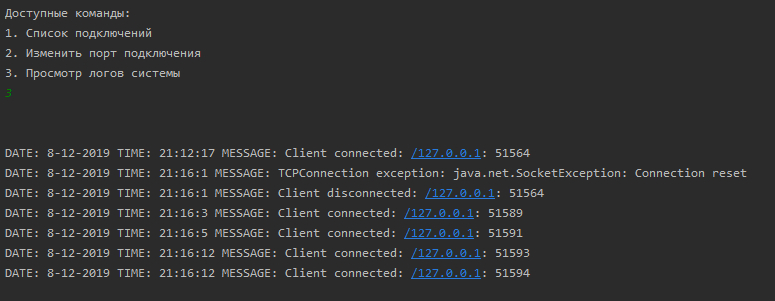


Рисунок 7.12 – Просмотр логов работы системы

На рисунке 7.13 изображена работа функции «О системе». Здесь выводится информация о порте подключения, строке подключения к базе данных, пользователе базы данных и его пароле.

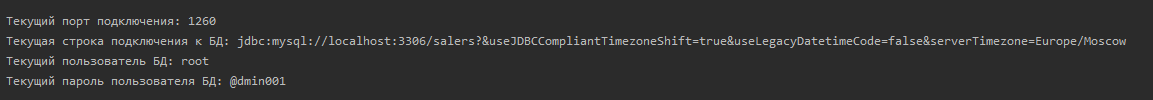


Рисунок 7.13 – Просмотр информации о системе

# **8 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ** **СИСТЕМЫ**

Программа обладает функциями оповещения и обработки ошибок, а также генерация и обработка исключительный ситуаций.

На рисунке 8.1 отображено оповещение системы о том, что отсутствует соединение с сервером и дальнейшая работа приложения невозможна.



Рисунок 8.1 – Ошибка об отсутствии соединения

На рисунке 8.2 показано оповещение системы о невалидности данных авторизации пользователя.

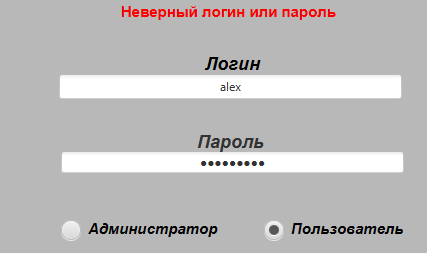


Рисунок 8.2 – Ошибка авторизации

На рисунках 8.3 и 8.4 показана попытка изменить значение строки на не валидное значение с последующим откатом изменений к первоначальным.

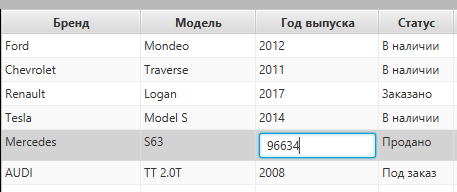


Рисунок 8.3 – Попытка изменить содержимое строки

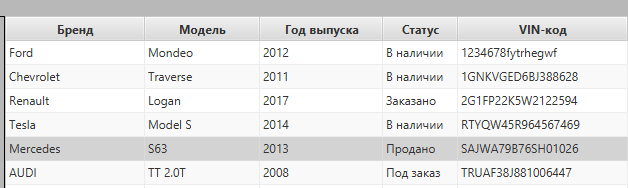


Рисунок 8.4 – Откат не валидных изменений назад

На рисунках 8.5 и 8.6 представлена работа системы оповещения, при попытке добавления нового пользователя в систему.

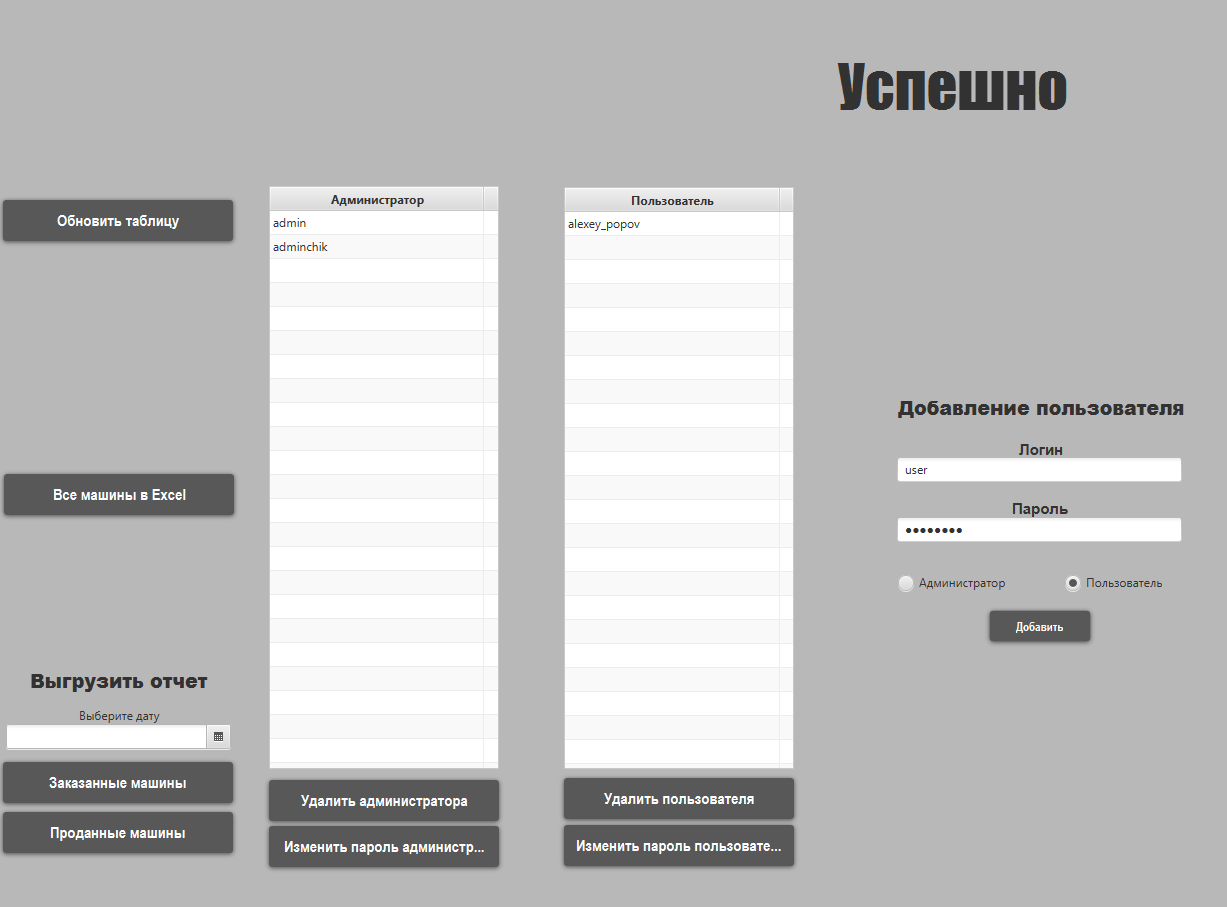


Рисунок 8.5 – Добавление пользователя в систему

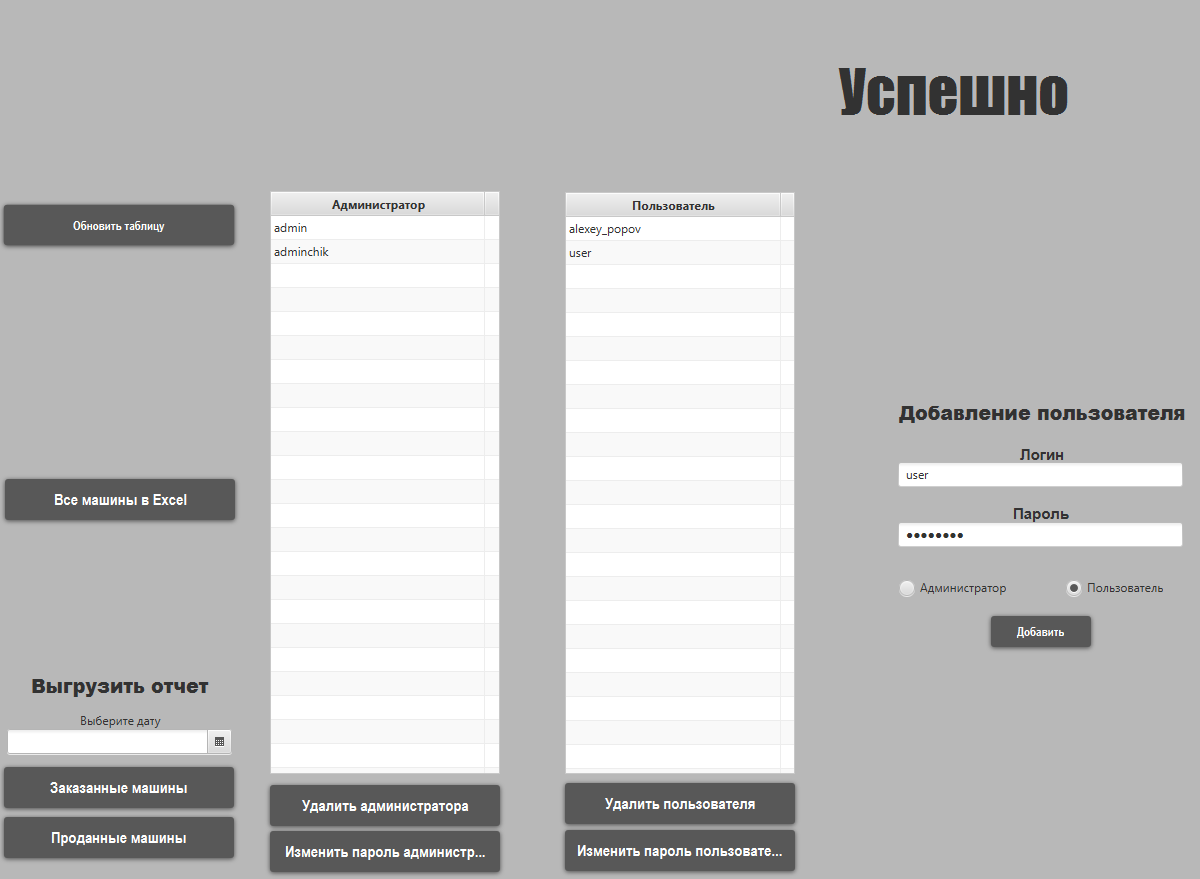


Рисунок 8.6 – Обновление таблицы с данными о пользователях

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения курсового проекта получено программное приложение, позволяющее автоматизировать систему учета продажи автомобилей в автосалоне или автохаусе. Оно еще требует улучшений и доработок, однако даже в таком виде значительно упрощает работу учета автомобилей. Все задачи, поставленные в начале проекта, достигнуты:

– изучена система работы автосалонов и система продажи автомобилей;

– в процессе разработки информация была разбита на отдельные модули, каждый из которых программировался отдельно и работает независимо от других;

– были спроектированы модули приложения и базы данных

– была разработана и реализована клиент-серверная архитектура приложения, которая позволяет подключаться нескольки пользователям одновременно;

– программный продукт успешно прошел тестирование.

Кроме того, был создан интуитивно понятный пользовательский интерфейс, который заметно упростит освоение программы.

Программу можно применять на компаниях, которые продают автомобили, для упрощения учета ее ведения.

Таким образом, в итоге можно сказать, что все первоначальные задачи решены и цель курсового успешно достигнута.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Компьютерные статьи [Электронный ресурс], - Изучение языков в рамках образовательной политики евросоюза – Режим доступа: https://cyberleninka.ru/article/n/izuchenie-yazykov-v-ramkah-obrazovatelnoy-politiki-evrosoyuza/viewer

[2] Компьютерные статьи [Электронный ресурс], - Причины продолжить изучать английский после уровня Intermediate – Режим доступа: https://habr.com/ru/post/479646/

[3] Компьютерные статьи [Электронный ресурс], - Финансовый риск – Режим доступа: http://www.grandars.ru/

[4] Компьютерные статьи [Электронный ресурс], - Документация по технологии JavaFX – Режим доступа: https://openjfx.io/openjfx-docs/

[5] Компьютерные статьи [Электронный ресурс], - Введение в JSON – Режим доступа: https://www.json.org/json-ru.html

[6] Компьютерные статьи [Электронный ресурс], - – Режим доступа: https://openjfx.io/openjfx-docs/

[7] Грэди Буч. Язык UML: руководство пользователя. / Г. Буч, Д. Рамбо, А. Джекобсон. - ДМК Пресс, 2007. – 496 с

# **ПРИЛОЖЕНИЕ А**

**(обязательное)**

**UML диаграммы**

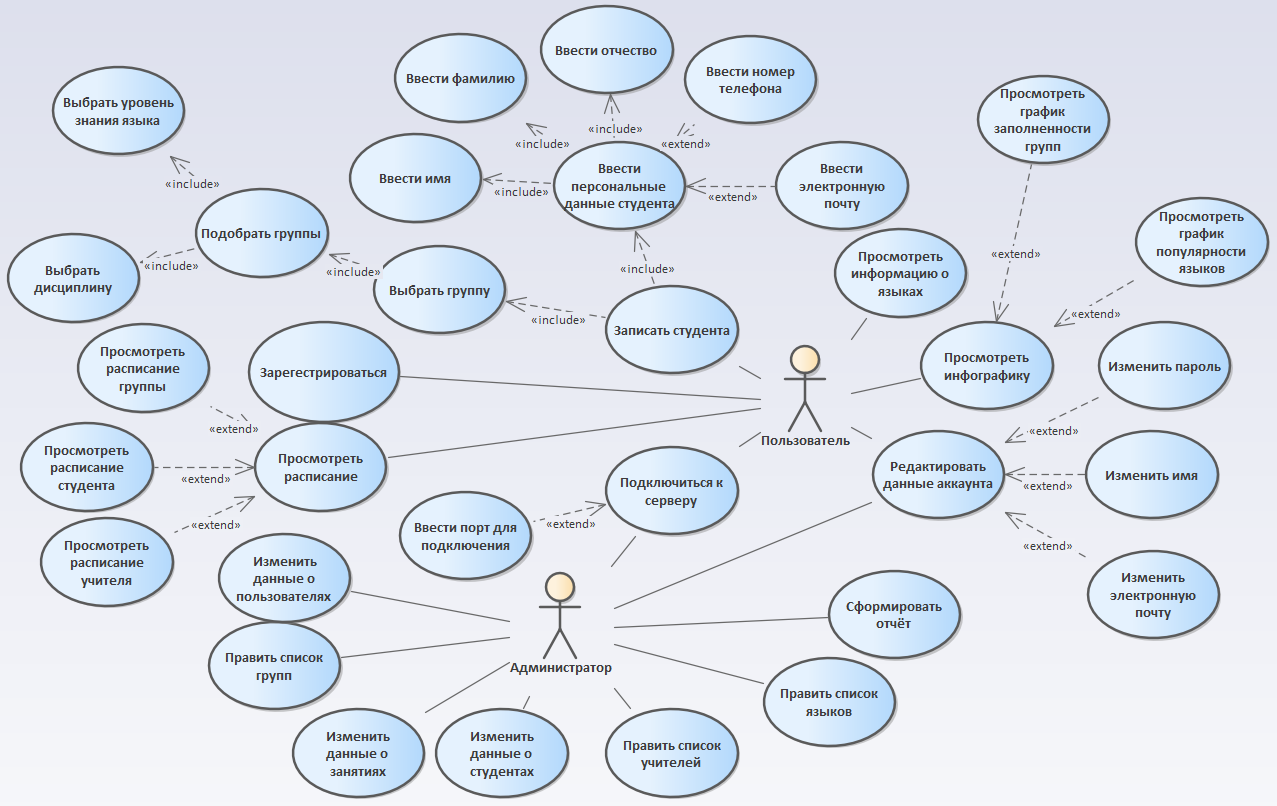


Рисунок А.1 – Диаграмма вариантов использования

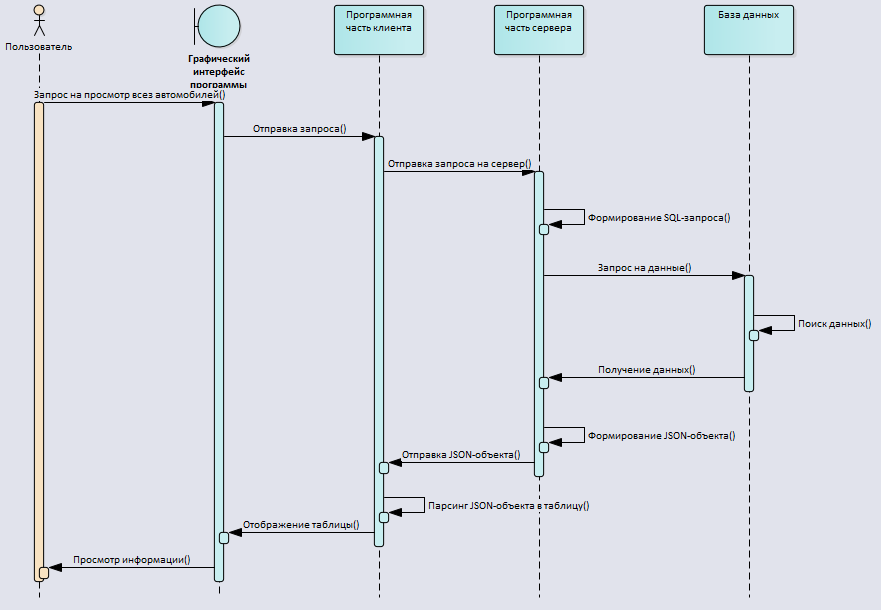


Рисунок А.2 – Диаграмма последовательности

Продолжение приложения А

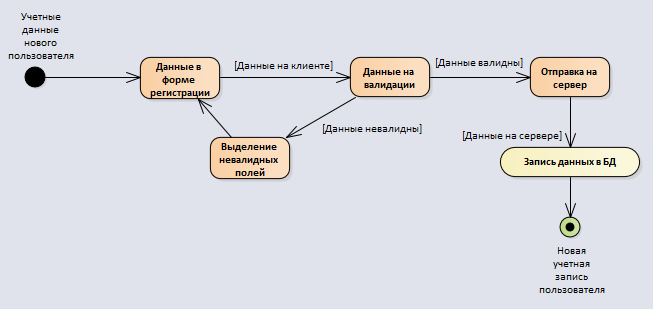


Рисунок А.3 - Диаграмма состояний

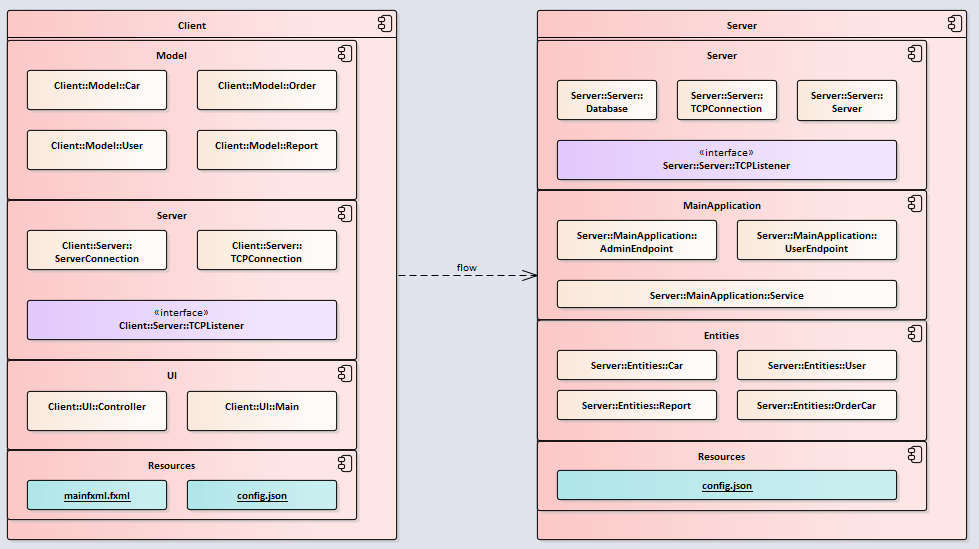


Рисунок А.4 – Диаграмма компонентов

Продолжение приложения А

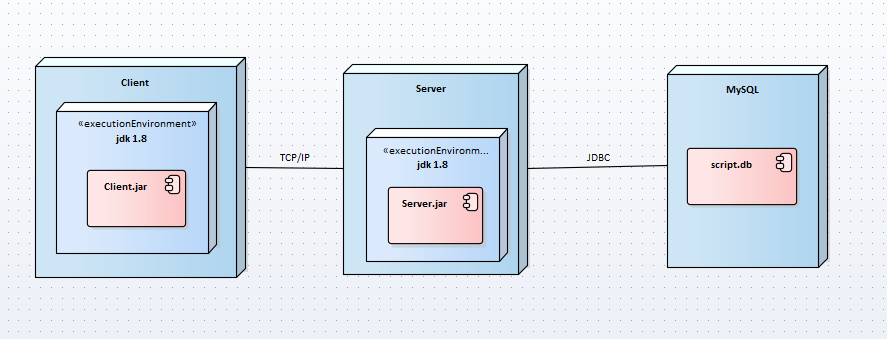


Рисунок А.5 – Диаграмма развертывания

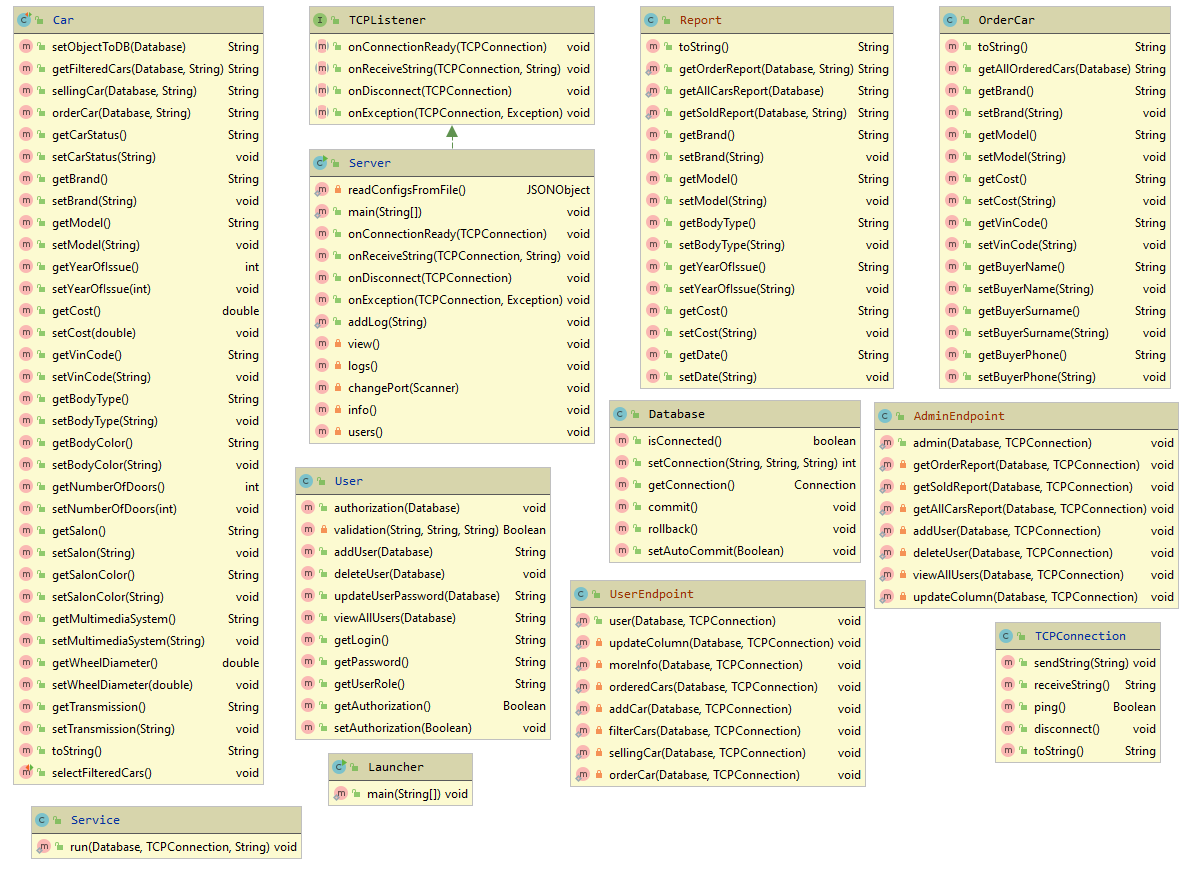


Рисунок А.6 – Диаграмма классов (сервер)

Продолжение приложения А

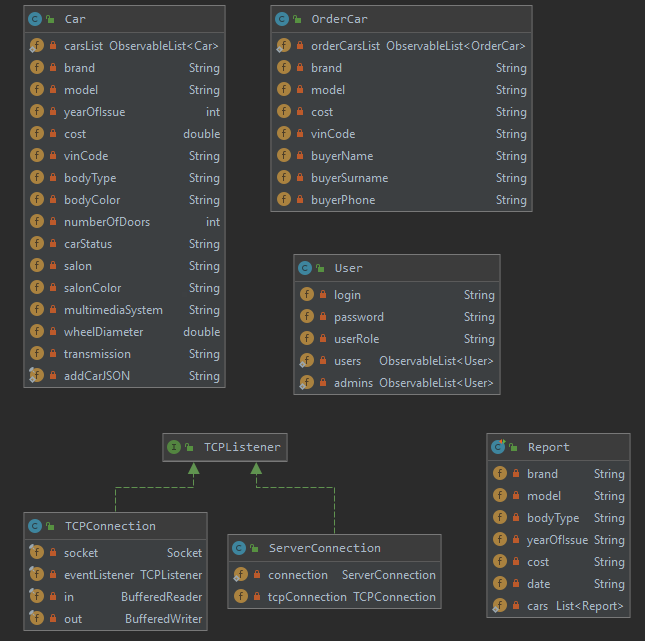


Рисунок А.7 – Диаграмма классов(клиент)

# **ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Схемы алгоритмов(к разделу 6)**

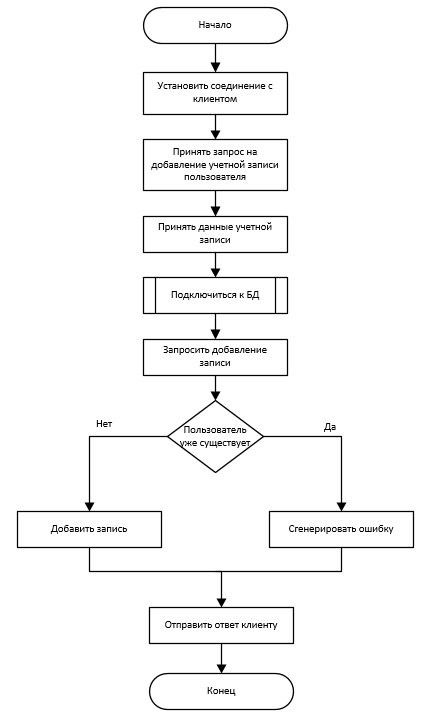


Рисунок Б.1 – Блок-схема алгоритма добавления пользователя

Продолжение приложения Б

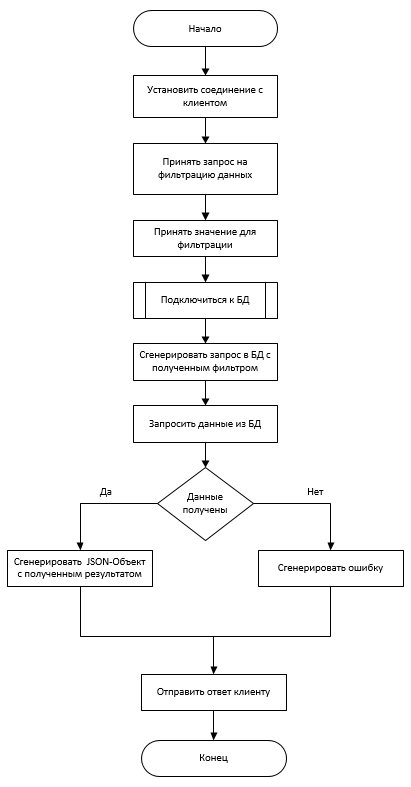


Рисунок Б.2 – Блок-схема алгоритма получения отчета

# **ПРИЛОЖЕНИЕ В**

**(обязательное)**

**Листинг исходного кода программного продукта**

**Интерфейс TCPListener**

package Server;

public interface TCPListener {

void onConnectionReady(TCPConnection tcpConnection);

void onReceiveString(TCPConnection tcpConnection, String value);

void onDisconnect(TCPConnection tcpConnection);

void onException(TCPConnection tcpConnection, Exception e);

}

**Класс TCPConnection (сервер)**

public class TCPConnection {

private final Socket socket;

private final Thread rxThread;

private final TCPListener eventListener;

private final BufferedReader in;

private final BufferedWriter out;

public TCPConnection(TCPListener eventListener, String IP, int port) throws IOException {

this(eventListener, new Socket(IP, port));

}

public TCPConnection(TCPListener eventListener, Socket socket) throws IOException {

this.eventListener = eventListener;

this.socket = socket;

in = new BufferedReader(new InputStreamReader(socket.getInputStream(), StandardCharsets.UTF\_8));

out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream(), StandardCharsets.UTF\_8));

rxThread = new Thread(new Runnable() {

@Override

public void run() {

try {

eventListener.onConnectionReady(TCPConnection.this);

while (!rxThread.isInterrupted()) {

eventListener.onReceiveString(TCPConnection.this, in.readLine());

}

} catch (IOException e) {

eventListener.onException(TCPConnection.this, e);

} finally {

eventListener.onDisconnect(TCPConnection.this);

Продолжение приложения В

}

}

});

rxThread.start();

}

public synchronized void sendString(String value) {

Server.addLog(this.toString() + ": Sending string: " + value);

try {

out.write(value + "\r\n");

out.flush();

} catch (IOException e) {

eventListener.onException(TCPConnection.this, e);

disconnect();

}

}

public synchronized String receiveString() {

try {

String receivedString = in.readLine();

Server.addLog(this.toString() + ": Received string: " + receivedString);

return receivedString;

} catch (IOException e) {

e.printStackTrace();

}

return null;

}

public synchronized Boolean ping() {

if (socket.isConnected()) {

return true;

}

return false;

}

public synchronized void disconnect() {

rxThread.interrupt();

try {

socket.close();

} catch (IOException e) {

eventListener.onException(TCPConnection.this, e);

}

}

@Override

public String toString() {

return socket.getInetAddress() + ": " + socket.getPort();

}

}

Продолжение приложения В

**Класс Server (сервер)**

public class Server implements TCPListener {

private static final ArrayList<String> logs = new ArrayList<>();

private final ObservableList<TCPConnection> connections = FXCollections.observableArrayList();

private final Database conn;

public Server(int port, String db\_url, String db\_user, String db\_password) {

System.out.println("Server's running...");

conn = new Database(db\_url, db\_user, db\_password);

new Thread(this::view).start();

try (ServerSocket serverSocket = new ServerSocket(port)) {

while (true) {

try {

new TCPConnection(this, serverSocket.accept());

} catch (IOException e) {

System.out.println("TCPConnection exception: " + e);

}

}

} catch (IOException e) {

throw new RuntimeException();

}

}

private static JSONObject readConfigsFromFile() {

JSONObject json;

JFileChooser fr = new JFileChooser();

FileSystemView fw = fr.getFileSystemView();

String fileName = fw.getDefaultDirectory() + "\\AppConfig\\configServer.json";

try (FileReader file = new FileReader(fileName)) {

StringBuffer stringBuffer = new StringBuffer();

int num;

char[] charArray = new char[1024];

while ((num = file.read(charArray)) > 0) {

stringBuffer.append(charArray, 0, num);

}

json = new JSONObject(stringBuffer.toString());

} catch (FileNotFoundException e) {

json = new JSONObject();

json.put("port", 1000);

Продолжение приложения В

json.put("db\_connection\_string", "jdbc:mysql://localhost:3306/salers?&useJDBCCompliantTimezoneShift=true" +

"&useLegacyDatetimeCode=false&serverTimezone=Europe/Moscow");

json.put("db\_user", "root");

json.put("db\_password", "@dmin001");

} catch (IOException e) {

json = new JSONObject();

json.put("port", 1000);

json.put("db\_connection\_string", "jdbc:mysql://localhost:3306/salers?&useJDBCCompliantTimezoneShift=true" +

"&useLegacyDatetimeCode=false&serverTimezone=Europe/Moscow");

json.put("db\_user", "root");

json.put("db\_password", "@dmin001");

}

return json;

}

public static void main(String[] args) {

JSONObject configs = readConfigsFromFile();

new Server(configs.getInt("port"), configs.getString("db\_connection\_string"),

configs.getString("db\_user"), configs.getString("db\_password"));

}

@Override

public synchronized void onConnectionReady(TCPConnection tcpConnection) {

connections.add(tcpConnection);

addLog("Client connected: " + tcpConnection);

}

@Override

public synchronized void onReceiveString(TCPConnection tcpConnection, String value) {

Service.run(conn, tcpConnection, value);

}

@Override

public synchronized void onDisconnect(TCPConnection tcpConnection) {

connections.remove(tcpConnection);

addLog("Client disconnected: " + tcpConnection);

}

@Override

public synchronized void onException(TCPConnection tcpConnection, Exception e) {

Продолжение приложения В

addLog("TCPConnection exception: " + e);

}

public static void addLog(String s) {

LocalDateTime time = LocalDateTime.now();

logs.add("DATE: "

+ time.getDayOfMonth() + "-"

+ time.getMonthValue() + "-" +

time.getYear() +

" TIME: " +

time.getHour() + ":" +

time.getMinute() + ":" +

time.getSecond()

+ " MESSAGE: " + s);

}

private void view() {

while (true) {

System.out.print("\n\n\nДоступные команды: \n" +

"1. Список подключений\n" +

"2. Изменить порт подключения\n" +

"3. Просмотр логов системы\n" +

"4. О системе\n" +

"5. Выход\n" +

"Ваш выбор: ");

Scanner in = new Scanner(System.in);

try {

int command = in.nextInt();

switch (command) {

case 1:

users();

break;

case 2:

changePort(in);

break;

case 3:

logs();

break;

case 4:

info();

break;

case 5:

System.exit(0);

break;

default:

System.out.println("Ошибка! Команда не найдена");

}

} catch (NumberFormatException n) {

addLog("EXCEPTION: " + n.toString());

Продолжение приложения В

} catch (InputMismatchException e) {

addLog("EXCEPTION: " + e.toString());

}

}

}

**Класс ServerConnection (клиент)**

public class ServerConnection implements TCPListener {

private static ServerConnection connection;

private TCPConnection tcpConnection;

public static ServerConnection takeConnection() {

if (connection == null) {

connection = new ServerConnection();

}

return connection;

}

private ServerConnection() {

JSONObject config = readConfigsFromFile();

try {

tcpConnection = new TCPConnection(this,"localhost", config.getInt("port"));

} catch (

IOException ex) {

ex.printStackTrace();

}

}

}

**Класс TCPConnection (клиент)**

public class TCPConnection implements TCPListener {

private final Socket socket;

private final TCPListener eventListener;

private final BufferedReader in;

private final BufferedWriter out;

public TCPConnection(TCPListener eventListener, String IP, int port) throws IOException {

this(eventListener, new Socket(IP, port));

}

private TCPConnection(TCPListener eventListener, Socket socket) throws IOException {

this.eventListener = eventListener;

this.socket = socket;

Продолжение приложения В

in = new BufferedReader(new InputStreamReader(socket.getInputStream(), StandardCharsets.UTF\_8));

out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream(), StandardCharsets.UTF\_8));

}

public synchronized void sendString(String value) {

try {

out.write(value + "\r\n");

out.flush();

} catch (IOException e) {

eventListener.onException(TCPConnection.this, e);

disconnect();

}

}

public synchronized Boolean ping() {

if (socket==null) {

return false;

}

return socket.isConnected();

}

public synchronized String receiveString() {

try {

return in.readLine();

} catch (IOException e) {

eventListener.onException(TCPConnection.this, e);

disconnect();

}

return null;

}

public synchronized void disconnect() {

try {

socket.close();

} catch (IOException e) {

eventListener.onException(TCPConnection.this, e);

}

}

@Override

public String toString() {

return "Connectivity.TCPConnection: " + socket.getInetAddress() + ": " + socket.getPort();

}

@Override

public void onException(TCPConnection tcpConnection, Exception e) {

Продолжение приложения В

}

}

**Класс Report (клиент)**

public class Report {

private String brand;

private String model;

private String bodyType;

private String yearOfIssue;

private String cost;

private String date;

private static List<Report> cars = new ArrayList<>();

public static void parseJSONReport(String result) {

Gson gson = new Gson();

Report[] reports = gson.fromJson(result, Report[].class);

cars.clear();

for(Report report: reports) {

cars.add(report);

}

}

public static void createAllCarsReport(String reportName) {

Workbook workbook = new XSSFWorkbook();

Sheet sheet = workbook.createSheet("Report");

Font headerFont = workbook.createFont();

headerFont.setBold(true);

headerFont.setFontHeightInPoints((short) 25);

headerFont.setColor(IndexedColors.RED.getIndex());

CellStyle headerCellStyle = workbook.createCellStyle();

headerCellStyle.setFont(headerFont);

Row rowHead = sheet.createRow(0);

rowHead.createCell(0).setCellValue("БРЕНД");

rowHead.createCell(1).setCellValue("МОДЕЛЬ");

rowHead.createCell(2).setCellValue("ТИП КУЗОВА");

rowHead.createCell(3).setCellValue("ГОД ВЫПУСКА");

rowHead.createCell(4).setCellValue("ЦЕНА");

int rowNum = 1;

for (Report report : cars) {

Row row = sheet.createRow(rowNum++);

row.createCell(0).setCellValue(report.brand);

row.createCell(1).setCellValue(report.model);

row.createCell(2).setCellValue(report.bodyType);

Продолжение приложения В

row.createCell(3).setCellValue(report.yearOfIssue);

row.createCell(4).setCellValue(report.cost);

}

for (int i = 0; i < 5; i++) {

sheet.autoSizeColumn(i);

}

JFileChooser fr = new JFileChooser();

FileSystemView fw = fr.getFileSystemView();

String fileName = fw.getDefaultDirectory() + "\\Reports\\" + reportName + ".xlsx";

Boolean checkFile = true;

while (checkFile) {

if (new File(fileName).exists()) {

reportName = reportName + "1";

fileName = fw.getDefaultDirectory() + "\\Reports\\"+ reportName + ".xlsx";

} else {

checkFile = false;

}

}

try (FileOutputStream fileOut = new FileOutputStream(fileName)) {

workbook.write(fileOut);

fileOut.flush();

} catch (IOException e) {

System.err.println(e);

}

}

public static void createReport(String reportName) {

Workbook workbook = new XSSFWorkbook();

Sheet sheet = workbook.createSheet("Report");

Font headerFont = workbook.createFont();

headerFont.setBold(true);

headerFont.setFontHeightInPoints((short) 20);

headerFont.setColor(IndexedColors.RED.getIndex());

CellStyle headerCellStyle = workbook.createCellStyle();

headerCellStyle.setFont(headerFont);

Row rowHead = sheet.createRow(0);

rowHead.createCell(0).setCellValue("БРЕНД");

rowHead.createCell(1).setCellValue("МОДЕЛЬ");

rowHead.createCell(2).setCellValue("ТИП КУЗОВА");

Продолжение приложения В

rowHead.createCell(3).setCellValue("ГОД ВЫПУСКА");

rowHead.createCell(4).setCellValue("ЦЕНА");

rowHead.createCell(5).setCellValue("ДАТА");

int rowNum = 1;

for (Report report : cars) {

Row row = sheet.createRow(rowNum++);

row.createCell(0).setCellValue(report.brand);

row.createCell(1).setCellValue(report.model);

row.createCell(2).setCellValue(report.bodyType);

row.createCell(3).setCellValue(report.yearOfIssue);

row.createCell(4).setCellValue(report.cost);

row.createCell(5).setCellValue(report.date);

}

for (int i = 0; i < 6; i++) {

sheet.autoSizeColumn(i);

}

JFileChooser fr = new JFileChooser();

FileSystemView fw = fr.getFileSystemView();

String fileName = fw.getDefaultDirectory() + "\\Reports\\" + reportName + ".xlsx";

Boolean checkFile = true;

while (checkFile) {

if (new File(fileName).exists()) {

reportName = reportName + "1";

fileName = fw.getDefaultDirectory() + "\\Reports\\"+ reportName + ".xlsx";

} else {

checkFile = false;

}

}

try (FileOutputStream fileOut = new FileOutputStream(fileName)) {

workbook.write(fileOut);

fileOut.flush();

} catch (IOException e) {

System.err.println(e);

}

}

# **ПРИЛОЖЕНИЕ Г**

**(обязательное)**

**Листинг скрипта генерации базы данных**

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

DROP SCHEMA IF EXISTS `salers` ;

CREATE SCHEMA IF NOT EXISTS `salers` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci ;

USE `salers` ;

DROP TABLE IF EXISTS `salers`.`auto` ;

CREATE TABLE IF NOT EXISTS `salers`.`auto` (

`BRAND` VARCHAR(50) NOT NULL,

`MODEL` VARCHAR(50) NOT NULL,

`YEAR\_OF\_ISSUE` VARCHAR(50) NOT NULL,

`COST` DECIMAL(15,2) NOT NULL,

`VIN\_CODE` VARCHAR(18) NOT NULL,

PRIMARY KEY (`VIN\_CODE`),

UNIQUE INDEX `VIN\_CODE` (`VIN\_CODE` ASC) VISIBLE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

DROP TABLE IF EXISTS `salers`.`body` ;

CREATE TABLE IF NOT EXISTS `salers`.`body` (

`VIN\_CODE` VARCHAR(18) NOT NULL,

`BODY\_TYPE` VARCHAR(50) NOT NULL,

`BODY\_COLOR` VARCHAR(50) NOT NULL,

`NUMBER\_OF\_DOORS` INT(11) NOT NULL,

UNIQUE INDEX `VIN\_CODE` (`VIN\_CODE` ASC) VISIBLE,

INDEX `IXFK\_BODY\_AUTO` (`VIN\_CODE` ASC) VISIBLE,

CONSTRAINT `FK\_BODY\_AUTO`

FOREIGN KEY (`VIN\_CODE`)

REFERENCES `salers`.`auto` (`VIN\_CODE`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

DROP TABLE IF EXISTS `salers`.`car\_status` ;

Продолжение приложения Г

CREATE TABLE IF NOT EXISTS `salers`.`car\_status` (

`VIN\_CODE` VARCHAR(18) NOT NULL,

`CAR\_STATUS` VARCHAR(50) NOT NULL,

UNIQUE INDEX `VIN\_CODE` (`VIN\_CODE` ASC) VISIBLE,

INDEX `IXFK\_CAR\_STATUS\_AUTO` (`VIN\_CODE` ASC) VISIBLE,

CONSTRAINT `FK\_CAR\_STATUS\_AUTO`

FOREIGN KEY (`VIN\_CODE`)

REFERENCES `salers`.`auto` (`VIN\_CODE`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

DROP TABLE IF EXISTS `salers`.`ordered\_cars` ;

CREATE TABLE IF NOT EXISTS `salers`.`ordered\_cars` (

`VIN\_CODE` VARCHAR(18) NOT NULL,

`BUYER\_NAME` VARCHAR(50) NOT NULL,

`BUYER\_SURNAME` VARCHAR(50) NOT NULL,

`BUYER\_PHONE` VARCHAR(15) NOT NULL,

`ORDER\_DATE` VARCHAR(50) NULL DEFAULT NULL,

INDEX `IXFK\_ORDERED\_CARS\_AUTO` (`VIN\_CODE` ASC) VISIBLE,

CONSTRAINT `FK\_ORDERED\_CARS\_AUTO`

FOREIGN KEY (`VIN\_CODE`)

REFERENCES `salers`.`auto` (`VIN\_CODE`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

DROP TABLE IF EXISTS `salers`.`sold\_cars` ;

CREATE TABLE IF NOT EXISTS `salers`.`sold\_cars` (

`VIN\_CODE` VARCHAR(18) NOT NULL,

`BUYER\_NAME` VARCHAR(50) NOT NULL,

`BUYER\_SURNAME` VARCHAR(50) NOT NULL,

`BUYER\_ID` VARCHAR(50) NOT NULL,

`SOLD\_DATE` VARCHAR(50) NULL DEFAULT NULL,

UNIQUE INDEX `VIN\_CODE` (`VIN\_CODE` ASC) VISIBLE,

UNIQUE INDEX `BUYER\_ID` (`BUYER\_ID` ASC) VISIBLE,

INDEX `IXFK\_SOLD\_CARS\_AUTO` (`VIN\_CODE` ASC) VISIBLE,

CONSTRAINT `FK\_SOLD\_CARS\_AUTO`

FOREIGN KEY (`VIN\_CODE`)

REFERENCES `salers`.`auto` (`VIN\_CODE`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB

Продолжение приложения Г

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

DROP TABLE IF EXISTS `salers`.`systems` ;

CREATE TABLE IF NOT EXISTS `salers`.`systems` (

`VIN\_CODE` VARCHAR(18) NOT NULL,

`SALON` VARCHAR(50) NOT NULL,

`SALON\_COLOR` VARCHAR(50) NOT NULL,

`MUTLIMEDIA\_SYSTEM` VARCHAR(50) NOT NULL,

UNIQUE INDEX `VIN\_CODE` (`VIN\_CODE` ASC) VISIBLE,

INDEX `IXFK\_SYSTEMS\_AUTO` (`VIN\_CODE` ASC) VISIBLE,

CONSTRAINT `FK\_SYSTEMS\_AUTO`

FOREIGN KEY (`VIN\_CODE`)

REFERENCES `salers`.`auto` (`VIN\_CODE`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

DROP TABLE IF EXISTS `salers`.`transmission` ;

CREATE TABLE IF NOT EXISTS `salers`.`transmission` (

`VIN\_CODE` VARCHAR(18) NOT NULL,

`TRANSMISSION` VARCHAR(50) NOT NULL,

`WHEEL\_DIAMETER` DECIMAL(5,3) NULL DEFAULT NULL,

UNIQUE INDEX `VIN\_CODE` (`VIN\_CODE` ASC) VISIBLE,

INDEX `IXFK\_TRANSMISSION\_AUTO` (`VIN\_CODE` ASC) VISIBLE,

CONSTRAINT `FK\_TRANSMISSION\_AUTO`

FOREIGN KEY (`VIN\_CODE`)

REFERENCES `salers`.`auto` (`VIN\_CODE`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

DROP TABLE IF EXISTS `salers`.`users` ;

CREATE TABLE IF NOT EXISTS `salers`.`users` (

`LOGIN` VARCHAR(50) NOT NULL,

`PASSWORD` VARCHAR(65) NOT NULL,

`USER\_ROLE` VARCHAR(15) NOT NULL,

UNIQUE INDEX `LOGIN` (`LOGIN` ASC))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;