Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет инженерно-экономический

Кафедра экономической информатики

Дисциплина: Распределенные системы обработки информации

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

АВТОМАТИЗИРОВАННАЯ СИСТЕМА ПРОКАТА АВТОМОБИЛЕЙ

Студентка: группа 074002 Нейжмак Д.В

Руководитель: Ассистент кафедры ЭИ А.П. Лыщик

Минск 2023

# 

Оглавление

[ВВЕДЕНИЕ 3](#_Toc133777735)

[1 АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОГРАММНОГО СРЕДСТВА 4](#_Toc133777736)

[1.1 Описание предметной области 4](#_Toc133777737)

[1.2 Разработка функциональной модели предметной области. 4](#_Toc133777738)

[1.3 Анализ требований к разрабатываемому программному 6](#_Toc133777739)

[средству. Спецификация функциональных требований. 6](#_Toc133777740)

[1.4 Разработка информационной модели предметной области. 8](#_Toc133777741)

[1.5 UML-модели представления программного средства и их 10](#_Toc133777742)

[описание. 10](#_Toc133777743)

[2 ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА. 16](#_Toc133777744)

[2.1. Постановка задачи. 16](#_Toc133777745)

[2.2. Обоснование выбора компонентов и технологий для реализации программного средства. 16](#_Toc133777746)

[2.3. Архитектурные решения. 18](#_Toc133777747)

[2.4. Описание алгоритмов, реализующих ключевую бизнес-логику 21](#_Toc133777748)

[разрабатываемого программного средства. 21](#_Toc133777749)

[2.5. Проектирование пользовательского интерфейса. 23](#_Toc133777750)

[3 РУКОВОДСТВО ПО РАЗВЕРТЫВАНИЮ И ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА. 30](#_Toc133777751)

[3.1 Руководство по установке (развертыванию) программного средства. 30](#_Toc133777752)

[3.2 Руководство пользователя 30](#_Toc133777753)

[ПРИЛОЖЕНИЕ А 32](#_Toc133777754)

[ПРИЛОЖЕНИЕ Б 36](#_Toc133777755)

[ПРИЛОЖЕНИЕ В 40](#_Toc133777756)

[ПРИЛОЖЕНИЕ Г 83](#_Toc133777757)

# ВВЕДЕНИЕ

В рамках данной курсовой работы рассматривается область «прокат автомобилей». Прокат автомобилей довольно распространенная услуга в наше время. Ее актуальность обоснована развитием коммерческих организаций по прокату автомобилей, в которых необходим компьютеризированный учет имеющихся машин. Эта информация обширна и разрознена. Чтобы вести учет всех автомобилей, которые можно взять в прокат, в организации имеется потребность в структурировании данных об автомобилях. Отсутствие такой возможности приводит к проблеме утери данных и большим временным затратам на выборку данных.

Современное техническое средство и услуга современного рынка: прокат автомобилей — это незаменимый помощник человека, без которых уже немыслимо само существование в нашем сложном и динамичном мире, только они помогают эффективно решать вопросы мобильности, динамичности и высокой производительности.

Целью данного проекта является повышение уровня автоматизации информационной системы проката автомобилей.

Для достижения данной цели был составлен и выполнен следующий список задач:

* изучение предметной области;
* оценка необходимости автоматизации процессов данной услуги;
* разработка схемы данных для хранения информации о машинах, а также

о клиентах, пользующихся услугой;

* написание программной части системы проката автомобилей.

В ходе курсовой работы будет разработано web-приложение с организацией взаимодействия с базой данных и многопоточным сервером на объектно-ориентированном языке программирования Java для автоматизации бизнес-процессов. На данный момент решение зaдaч в сфере проката автомобилей практически невозможно без использования системы управления баз данных. Система управления базами данных - комплекс программных и языковых средств, необходимых для создания и модификации базы данных, добавления, удаления, поиска и отбора информации, представления информации на экране и в печатном виде, разграничения прав доступа к информации, выполнения других операций с базой.

С целью облегчения проектирования программного комплекса будут разработаны стандартизированные способы моделирования и разработки систем, такие как IDEF0, IDEF1.x, UML диаграммы и блок-схемы.

# АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОГРАММНОГО СРЕДСТВА

## **1.1 Описание предметной области**

Прокат автомобилей (или аренда) автомобилей – сдача автомобиля без водителя на неопределенный период времени.

Прокат автомобилей сегодня — это доступный большинству мирового населения вид услуги, обеспечивающий быстрый, легкий и удобный способ арендовать нужный вам автомобиль в кратчайшие сроки и не требующий больших финансовых затрат. Это хороший бизнес и к тому же он очень удобен для тех, кто не купил себе машину или просто тем, кому просто необходим транспорт. В странах СНГ данный бизнес очень популярен и стремительно развивается.

Прокат автомобилей – это процесс разработки информационной системы, предназначенной для обеспечения учета автомобилей (как свободных, так и арендованных) в компании и исполнения следующих процессов: единый учет автомобилей в разрезе их характеристик (марка, пробег, свободен или арендован); поддержка учета поступления заявок; перемещение автомобиля от одного клиента к другому и учет по каждой аренде автомобиля; детализированный расчет стоимости каждого заказа.

## **1.2 Разработка функциональной модели предметной области.**

Функции автоматизированной системы проката автомобилей:

* Хранить данные о марках машин и автопарке, состоящем из марок

различных машин.

* Информация о сотрудниках и клиентах.
* Информация о выданных в прокат автомобилей.
* Автоматическое формирование договоров автопроката.
* Автоматическое формирование отчетов по результатам

деятельности: отчет по сотрудникам, клиентам, прибыли на марку и машину, информация по количеству выданный машин на данный момент.

* Возможность поиска и сортировке по базе данных, которая не занимает

много времени.

* Возможность использовать фотографии автомобилей.

Система проката автомобилей имеет:

* Авторизацию. Каждый сотрудник имеет свой пароль и логин для

доступа в систему.

* Разграничение доступа и возможность создавать группы для

пользователей. Сотрудники входят в группы и получают возможность на доступ к различным функциям программы: оформление договора, редактирование информации, возможность просматривать отчеты.

Данная информационная система прокат автомобилей поможет ускорить проверку занятости автомобилей (как свободных, так и арендованных).

Учитываются автомобили в разрезе их характеристик, по их личным данным, таким как регистрационный знак ТС, VIN ТС и индивидуальные технические характеристики.

Увеличивается экономия времени при заключении договора с клиентами, обратившимися в прокат автомобилей повторно, так как при первом обращении клиентов в любой филиал проката автомобилей, они в обязательной форме проходят регистрацию, при повторном обращении они уже будут зарегистрированы в базе данных.

Преимущества автопроката:

1. Клиенту не нужно покупать авто либо выплачивать всю его стоимость для решения своих дел. В таком случае гораздо выгоднее воспользоваться услугами автопроката, чем покупать машину.
2. Большой выбор автомобилей. Клиент может взять напрокат любое авто, которое представлено в автопарке каршеринга.
3. Все машины в хорошем состоянии, за это отвечает компания-арендодатель. Поэтому поездки доставят только удовольствие. Клиенту не придется ремонтировать взятое в aренду авто.
4. Достаточно простая и быстрая процедура оформления документов. Весь процесс может занять до получаса.
5. Различная ценовая политика. Взять машину в прокат могут все люди с любым достатком.
6. Прокат авто поможет людям, которые хотят купить автомобиль, но не могут выбрать модель. Они смогут поездить на разных машинах и самостоятельно понять все их плюсы и минусы.
7. При аренде авто клиент ничем не отягощен. Он может свободно распоряжаться личным временем.
8. Минимальные ограничения. Эта услуга доступна практически каждому. Достаточно иметь паспорт, водительское удостоверение и минимальный стаж.

Основные задачи автоматизации информационных процессов состоят в:

* увеличении скорости выполнения процессов обработки и

преобразования информации;

* хранение структурированной информации в больших объемах. Ведение

базы клиентов системы проката, их контактная информации. Ведение списка клиентов и их контактной информации. По каждому автомобилю можно посмотреть полную информацию об автомобиле, взятым им в прокат, а также о всех визитах клиента.

* обеспечении большей оперативности и качества обслуживания

клиентов.

## **1.3 Анализ требований к разрабатываемому программному**

## **средству. Спецификация функциональных требований.**

Для создания системы автоматизированного проката автомобилей необходимо подробное изучение бизнес-процессов занимающейся данным родом деятельности и последующее моделирование данных процессов согласно методологии IDEF0.

IDEF0 — методология функционального моделирования и графическая нотация, предназначенная для формализации и описания бизнес-процессов. Отличительной особенностью IDEF0 является её акцент на соподчиненность объектов. В IDEF0 рассматриваются логические отношения между работами, а не их временная последовательность (поток работ).

IDEF0 как стандарт был разработан в 1981 году департаментом Военно-воздушных сил США  в рамках программы автоматизации промышленных предприятий, которая носила обозначение ICAM (*Integrated Computer Aided Manufacturing*). Набор стaндартов IDEF унаследовал своё название от этой программы (IDEF расшифровывается как *ICAM Definition*). В процессе практической реализации, участники программы ICAM столкнулись с необходимостью разработки новых методов анализа процессов взаимодействия в промышленных системах. При этом кроме усовершенствованного набора функций для описания бизнес-процессов, одним из требований к новому стандарту было наличие эффективной методологии взаимодействия в рамках «аналитик-специалист». Другими словами, новый метод должен был обеспечить групповую работу над созданием модели, с непосредственным участием всех аналитиков и специалистов, занятых в рамках проекта.

Методология IDEF0 нашла широкое признание и применение, в первую очередь, благодаря простой графической нотации, используемой для построения модели. Главными компонентами модели являются диаграммы. На них отображаются функции системы в виде прямоугольников, а также изображаются связи между ними и внешней средой с помощью использования стрелок. Использование всего лишь двух графических примитивов (прямоугольник и стрелка) дают возможность быстро объяснить правила и принципы построения диаграмм IDEF0 людям, незнакомым с данной методологией. Это достоинство позволяет подключить и активизировать деятельность заказчика по описанию бизнес-процессов с использованием формального и наглядного графического языка (Рисунок 1.3.1).

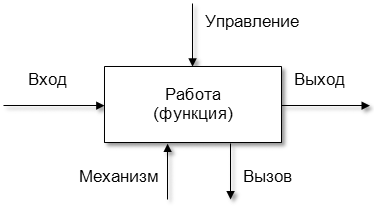


Рисунок 1.3.1 – Элементы графической нотации IDEF0

Прямоугольник представляет собой работу (процесс, деятельность, функцию или задачу), которая имеет фиксированную цель и приводит к некоторому конечному результату. Имя работы должны выражать действие

(например, «Изготовление детали», «Расчет допускаемых скоростей» и т.д.).

Взаимодействие работ между собой и внешним миром описывается в виде стрелок. В IDEF0 различают 5 видов стрелок:

– *вход* (англ. input) – материал или информация, которые используются и преобразуются работой для получения результата (выхода). Вход отвечает на вопрос «Что подлежит обработке?». В качестве входа может быть, как материальный объект (сырье, деталь, экзаменационный билет), так и не имеющий четких физических контуров (запрос к БД, вопрос преподавателя). Допускается, что работа может не иметь ни одной стрелки входа. Стрелки входа всегда рисуются входящими в левую грань работы;

– *управление* (англ. control) – управляющие, регламентирующие и нормативные данные, которыми руководствуется работа. Управление отвечает на вопрос «В соответствии с чем выполняется работа?». Управление влияет на работу, но не преобразуется ей, т.е. выступает в качестве ограничения. В качестве управления могут быть правила, стандарты, нормативы, расценки, устные указания. Стрелки управления рисуются входящими в верхнюю грань работы. Если при построении диаграммы возникает вопрос, как правильно нарисовать стрелку сверху или слева, то рекомендуется ее рисовать как вход (стрелка слева);

– *выход* (англ. output) – материал или информация, которые представляют результат выполнения работы. Выход отвечает на вопрос «Что является результатом работы?». В качестве выхода может быть как материальный объект (деталь, автомобиль, платежные документы, ведомость), так и нематериальный (выборка данных из БД, ответ на вопрос, устное указание). Стрелки выхода рисуются исходящими из правой грани работы;

– *механизм* (англ. mechanism) – ресурсы, которые выполняют работу. Механизм отвечает на вопрос «Кто выполнил работу или посредством чего?». В качестве механизма могут быть персонал предприятия, студент, станок, оборудование, программа. Стрелки механизма рисуются входящими в нижнюю грань работы:

– *вызов* (англ. call) – стрелка указывает, что некоторая часть работы выполняется за пределами рассматриваемого блока. Стрелки входа рисуются исходящими из нижней грани работы.

## **1.4 Разработка информационной модели предметной области.**

В соответствии с составленным описанием предметной области был составлен список информации необходимой для создания автоматизированной системы проката автомобилей. Эти данные представлены в виде таблиц, «Данные о клиентах», «Данные о заказах», «Данные о сотрудниках» и «Данные о машинах».

Таблица 1- «Данные о клиентах»

| Наименование поля | Тип поля | Обязательное поле(да, нет) |
| --- | --- | --- |
| Id | Числовой | Да |
| Фамилия | Текстовый | Да |
| Имя | Текстовый | Да |
| Отчество | Текстовый | Да |
| № паспорта | Текстовый (с маской) | Да |
| Телефон домашний | Текстовый (с маской) | Нет |
| Телефон мобильный | Текстовый (с маской) | Да |
| E-mail | Текстовый (с маской) | Нет |
| Номер дисконтной карты | Текстовый (с маской) | Нет |

Таблица 2 –«Данные о заказах»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип поля | Обязательное поле(да, нет) |
| Id | Числовой | Да |
| Id клиента | Числовой | Да |
| Id автомобиля | Числовой | Да |
| Дата начала проката | Дата | Да |
| Дата окончания проката | Дата | Да |

Таблица 3-«Данные о сотрудниках»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип поля | Обязательное поле(да, нет) |
| Id | Числовой | Да |
| Логин | Текстовый | Да |
| Пароль | Текстовый | Да |
| Имя сотрудника | Текстовый | Да |

Таблица 4 –«Данные о машинах»

| Наименование поля | Тип поля | Обязательное поле(да, нет) |
| --- | --- | --- |
| Id | Числовой | Да |
| Год выпуска | Числовой | Да |
| Марка авто | Список | Да |
| Модель авто | Текстовый | Да |
| Гос. номер | Текстовый (с маской) | Да |
| Пробег | Числовой | Да |
| Коробка передач | Перечисление | Да |
| Потребление топлива | Числовой | Да |
| Залог | Денежный | Да |
| Цена аренды за день | Денежный | Да |
| Цена аренды за день (при аренде на период больше 7 дней) | Денежный | Да |

Для хранения марки автомобиля был создан дополнительный справочник «Марки автомобилей», его структура представлена в таблице «Справочник «Марки автомобиля».

Таблица 5 –«Справочник «Марки автомобиля»»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип поля | Обязательное поле(да, нет) |
| Id | Числовой | Да |
| Наименование | Текстовый | Да |

Исходя из приведенных данных, представленных в таблицах, составим логическую и физическую модели базы данных.

Модель базы данных — то же, что и схема базы данных, то есть описания содержания, структуры и ограничений целостности, используемые для создания и поддержки базы данных.

Логическая модель данных является визуальным графическим представлением структур данных, их атрибутов и связей. Логическая модель представляет данные таким образом, чтобы они легко воспринимались бизнес-пользователями. Проектирование логической модели должно быть свободно от требований платформы и языка реализации или способа дальнейшего использования данных.

Физическая модель – логическая модель базы данных, выраженная в терминах языка описания данных конкретной СУБД. Физическая модель базы данных содержит все детали, необходимые конкретной СУБД для создания базы: наименования таблиц и столбцов, типы данных, определения первичных и внешних ключей и т. п.

Логическая модель представлена на рисунке Б.1, физическая модель представлена на рисунке Б.2 в приложение Б.

## **1.5 UML-модели представления программного средства и их**

## **описание.**

UML – унифицированный язык моделирования (Unified Modeling Language) – это система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования.

Его можно использовать для визуализации, спецификации, конструирования и документирования программных систем.

Словарь UML включает три вида строительных блоков:

– диаграммы;

– сущности;

– связи.

Сущности – это абстракции, являющиеся основными элементами модели, связи соединяют их между собой, а диаграммы группируют представляющие интерес наборы сущностей.

Диаграмма – это графическое представление набора элементов, чаще всего изображенного в виде связного графа вершин (сущностей) и путей (связей).

Большинство элементов UML имеют уникальную и прямую графическую нотацию, которая дает визуальное представление наиболее важных аспектов элемента.

Диаграмма взаимодействия используется для отображения интерактивного поведения системы. Поскольку визуализация взаимодействий в системе может быть трудоемкой задачей, используются различные типы диаграмм взаимодействия, чтобы охватить различные особенности и аспекты взаимодействия в системе.

Диаграмма последовaтельности (англ. sequence diagram) — диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл какого-либо определённого объекта (создание-деятельность-уничтожение некой сущности) и взаимодействие актеров (действующих лиц) ИС в рамках какого-либо определённого прецедента (отправка запросов и получение ответов).

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные «линии жизни» (англ. lifeline), которые отображают течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни»), и стрелки, показывающие обмен сигналами или сообщениями между объектами.

На данной диаграмме объекты располагаются слева направо.

Для описания процесса оформления заказа была добавлена диаграмма последовательности, которая находится на рисунке А.2 приложения А.

Главным объектом данной диаграммы является актер «Сотрудник». Для оформления нового заказа сотруднику требуется заполнить форму данными о заказе и нажать на кнопку «Оформить заказ». После нажатия на кнопку происходит проверка введенных данных на клиентской части приложении. Если проверка прошла успешно, то запрос с командой на оформления заказа отправляется на сервер, иначе выводится сообщение о неверно введенных данных. Как только запрос попадает в сервлет, с помощью метода IsValid класса OrderValidator, на серверной части приложения происходит процесс проверки данных на сервере, это дополнительно защищает от ввода неверных данных и от злонамеренного внедрения вредоносного SQL кода. В случае, когда проверка не пройдена успешно, сервлет возвращает сообщение об ошибке которое, в последствии, выводится в браузере, в противном случае, с помощью метода AddEnity класса SqlRepository, запрос на добавление заказа отправляется в базу данных.

Диаграмма состояний — это диаграмма состояний из теории автоматов со стандартизированными условными обозначениями, задача которой определять множество систем от компьютерных программ до бизнес-процессов. Используются следующие условные обозначения:

– Круг, который характеризует начальное состояние.

– Окружность с маленьким кругом внутри, обозначающая конечное состояние (если есть).

– Скруглённый прямоугольник, обозначающий состояние. Верхушка прямоугольника содержит название состояния. В середине может быть горизонтальная линия, под которой записываются активности, происходящие в данном состоянии.

– Стрелка, обозначающая переход. Название события (если есть), вызывающего переход, отмечается рядом со стрелкой. Охраняющее выражение может быть добавлено перед «/» и заключено в квадратные скобки (название события [охраняющее выражение]), что значит, что это выражение должно быть истинным, чтобы переход имел место. Если при переходе производится какое-то действие, то оно добавляется после «/» (название события [охраняющее выражение]/действие).

– Толстая горизонтальная линия с либо множеством входящих линий и одной выходящей, либо одной входящей линией и множеством выходящих. Это означает объединение и разветвление соответственно.

Для того, чтобы начать работу в системе сотруднику проката требуется авторизоваться в системе. Для этого этого необходимо заполнить форму с логином и паролем и нажать на кнопку «Войти». После этого система переходит в состояние проверки введенных данных на стороне клиента, данные, введенные сотрудником, проверяются на соответствие формату действительного логина и пароля. В случае прохождение проверки на сервер передается команда авторизации. После этого наступает этап проверки данных на сервере, данные также проверяются на соответствие нужному формату (с помощью метода IsValid класса EmployeeValidator), иначе выводится сообщение об ошибке. Если проверка прошла успешно, то с помощью класса SqlRepository данные о логине и пароле получаются и базы данных, в противном случае на клиентскую часть приложения возвращается сообщение об ошибке. После получения данных из базы данных происходит сопоставление введенного пароля и пароля, хранящегося в базе данных. В случае их совпадения пользователь, если же пароли не совпадают, то выводится сообщение о неверности логина или пароля. Диаграмма процесса приведена на рисунке А.4 приложения А.

Диаграмма компонентов (англ. Component diagram) — элемент языка моделирования UML, статическая структурная диаграмма, которая показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами.

С помощью диаграммы компонентов представляются инкапсулированные классы вместе с их интерфейсными оболочками, портами и внутренними структурами (которые тоже могут состоять из компонентов и коннекторов).

Компоненты связываются через зависимости, когда соединяется требуемый интерфейс одного компонента с имеющимся интерфейсом другого компонента. Таким образом иллюстрируются отношения клиент-источник между двумя компонентами.

Зависимость показывает, что один компонент предоставляет сервис, необходимый другому компоненту. Зависимость изображается стрелкой от интерфейса или порта клиента к импортируемому интерфейсу.

Когда диаграмма компонентов используется, чтобы показать внутреннюю структуру компонентов, предоставляемый и требуемый интерфейсы составного компонента, могут делегироваться в соответствующие интерфейсы внутренних компонентов.

Делегация показывает связь внешнего контракта компонента с внутренней реализацией этого поведения внутренними компонентами.

Диаграмма развёртывания (англ. Deployment diagram) в UML моделирует физическое развертывание артефактов на узлах. Например, чтобы описать веб-сайт, диаграмма развертывания должна показывать, какие аппаратные компоненты («узлы») существуют (например, веб-сервер, сервер базы данных, сервер приложения), какие программные компоненты («артефакты») работают на каждом узле (например, веб-приложение, база данных), и как различные части этого комплекса соединяются друг с другом (например, JDBC, REST, RMI).

Узлы представляются как прямоугольные параллелепипеды с артефактами, расположенными в них, изображенными в виде прямоугольников. Узлы могут иметь подузлы, которые представляются как вложенные прямоугольные параллелепипеды. Один узел диаграммы развертывания может концептуально представлять множество физических узлов, таких как кластер серверов баз данных.

Для описания процесса развертывания созданной системы в на рисунке А.3 приложения А приводится диаграмма развертывания. Диаграмма развертывания содержит два устройства: персональный компьютер с доступом к интернету и web-браузером для пользователя (сотрудника банка), и Web-сервер на котором развернуто созданное приложение. На Web-сервере развернут два Docker контейнера в которых развернуты Tomcat и сервер MySQL. Внутри Tomcat содержаться два артефакта: war файл собранного приложения и файлы Java Server Pages. MySQL сервер зависит от схемы данных разработанной в ходе выполнения данной курсовой работы.

Диаграмма классов UML — это граф, узлами которого являются элементы статической структуры проекта (классы, интерфейсы), а дугами - отношения между узлами (ассоциации, наследование, зависимости).

Класс — это группа сущностей (объектов), обладающих сходными свойствами, а именно, данными и поведением. Отдельный представитель некоторого класса называется объектом класса или просто объектом.

Диаграмма классов UML представлена на рисунке А.6 и А.7.

На диаграмме, представленной на рисунке А.6 существует класс Entity, от которого наследуются классы Car, Order, Client, Carmake, Employee. У класса Entity есть атрибут id и методы getid() и setid. У наследуемого класса Client атрибутами являются: name, surname, email и т. д., методы: getName(), getName(string), getSurname и др. Для класса Order атрибутами являются: StartDate, endDate и другие.

На диаграмме, представленной на рисунке А.6, существует пакет Package impl, от которого зависят классы ConnectionManager, Repository и RepositoryException. У класса ConnectionManager есть атрибуты connection, USER, PASSWORD, DATABASE и т.д., а также методы gerConnection(), values() и valueOf(String). У класса Repository нет атрибутов, есть методы: getEnrityId(T), getEntityByld(int). и др. И класс RepositoryException не имеет атрибутов и методов.

Диаграмма вариантов использования (англ. *use case diagram)* в UML — диаграмма, отражающая отношения между актерами и прецедентами и являющаяся составной частью *модели прецедентов*, позволяющей описать систему на концептуальном уровне.

Прецедент — возможность моделируемой системы (часть её функциональности), с помощью которой пользователь может получить конкретный, измеримый и нужный ему результат. Прецедент соответствует отдельному сервису системы, определяет один из вариантов ее использования и описывает типичный способ взаимодействия пользователя с системой. Варианты использования обычно применяются для спецификации внешних требований к системе [2].

Для отражения модели прецедентов на диаграмме используются:

* рамки системы (англ. *system boundary*) — прямоугольник с названием в верхней части и эллипсами (прецедентами) внутри. Часто может быть опущен без потери полезной информации,
* актёр (англ. *actor*) — стилизованный человек, представляющий собой набор ролей пользователя (понимается в широком смысле: человек, внешняя сущность, класс, другая система), который взаимодействует с некоторой сущностью (системой, подсистемой, классом). Актеры не могут быть связаны друг с другом (за исключением отношений обобщения/наследования),
* прецедент — эллипс с надписью, обозначающий выполняемые системой действия (могут включать возможные варианты), приводящие к наблюдаемым актерами результатам. Надпись может быть именем или описанием (с точки зрения актеров) того, «что» делает система. Имя прецедента связано с непрерывным (атомарным) сценарием — конкретной последовательностью действий, иллюстрирующей поведение. В ходе сценария актеры обмениваются с системой сообщениями. Сценарий может быть приведен на диаграмме прецедентов в виде UML-комментария. С одним прецедентом может быть связано несколько различных сценариев.

Отношения между прецедентами:

Часть дублирующейся информации в модели прецедентов можно устранить указанием связей между прецедентами:

- ***обобщение прецедента*** — стрелка с не закрашенным треугольником (треугольник ставится у более общего прецедента);

***- включение прецедента*** — пунктирная стрелка со стереотипом «include»;

***- расширение прецедента*** — пунктирная стрелка со стереотипом «extend» (стрелка входит в расширяемый прецедент, в дополнительном разделе которого может быть указана *точка расширения* и, возможно в виде комментария, условие расширения);

Разработка диаграммы вариантов использования преследует цели:

1. Помогает определить общие границы и контекст моделируемой предметной области на начальных этапах создания системы.
2. Формулирует общие требования к функциональному поведению проектируемой системы.
3. Дает возможность разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей.
4. Подготавливает исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых 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. В свою очередь, вариант использования (use case) служит для описания сервисов, с помощью которой она предоставляется актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой [3].

Составленная диаграмма вариантов использования находится в Приложении В.

Диаграмма вариантов использования содержит следующие варианты использования:

– Добавление нового заказа;

– Оплата заказа;

– Оплата заказа через банковский кредит;

– Закрытие заказа;

– Вход сотрудника в систему;

– Вывод сообщения об ошибке;

– Получение данных клиента;

– Добавление нового клиента;

– Добавление данных о заказе;

– Удаление данных о заказе.

Также в ней содержатся следующие актеры:

– Сотрудник проката автомобилей

Данная диаграмма представляет собой описание взаимодействия актера «Сотрудник проката автомобилей» с проектируемой системой. Этот актер взаимодействует с вариантами использования «Добавление нового заказа» и «Закрытие заказа». Оба этих варианта использования включают в себя вход сотрудника в создаваемую систему, что отражено на данной диаграмме. В случае неудачной попытки входа, система выводит сообщение об ошибке входа в систему. Вариант использования «Добавление нового заказа» также включает в себя «Добавление данных заказа», «Оплата заказа» и «Получение данных клиента». В случае, когда информация о данном клиенте еще не внесена в систему требуется произвести ее ввод. Оплата заказа может производиться непосредственно или через оплату с помощью банковского кредита. Вариант использование «Закрытие заказа» предусматривает удаление данных о заказе, что отражено на диаграмме соответствующей стрелкой.

# ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА.

## **2.1. Постановка задачи.**

Задача курсовой работы состоит в разработке web-сервиса на языке Java для онлайн-проката автомобилей. Разработанный web-сервис направлен на ускорение оформления клиента, а также управление процессом проката для компании, предоставляющей автомобили в аренду.

## **2.2. Обоснование выбора компонентов и технологий для реализации программного средства.**

Java - это популярный язык программирования высокого уровня, который был выпущен компанией Sun Microsystems в 1995 году. Он разработан для независимости от платформы, что означает, что код, написанный на Java, может работать на любом устройстве или операционной системе, на которых установлена виртуальная машина Java (JVM). Java - это мощный и универсальный язык программирования, который широко принят в программной индустрии. Он продолжает развиваться, с добавлением новых функций и улучшений в каждом новом выпуске.

Одной из ключевых особенностей Java является ее сильный упор на концепции объектно-ориентированного программирования (ООП). В Java все рассматривается как объект, и программы написаны с использованием классов и объектов.

Некоторые из ключевых особенностей Java включают автоматическое управление памятью (сбор мусора), обработку исключений и возможности многопоточности. Java также имеет обширную библиотеку предварительно созданных классов и API, которые упрощают выполнение общих задач, таких как сетевое взаимодействие, ввод-вывод файлов и доступ к базам данных.

IntelliJ IDEA - это интегрированная среда разработки (IDE) для языков программирования Java, Kotlin и других. Она была разработана компанией JetBrains и выпущена в 2001 году.

IntelliJ IDEA предоставляет широкий набор инструментов для разработки приложений на языках Java и Kotlin. Она включает в себя поддержку автодополнения, статический анализ кода, интеграцию с системами контроля версий, инструменты для рефакторинга кода, отладчик и многие другие функции.

Java Servlet - это технология Java, используемая для создания динамических веб-приложений. Она работает на серверной стороне и позволяет создавать обработчики запросов HTTP, которые могут генерировать динамический контент в формате HTML, XML, JSON и других форматах.

Java Servlet работает в рамках Java Servlet API, который предоставляет интерфейсы и классы для создания и обработки HTTP-запросов. Он также обеспечивает поддержку сессий и авторизации, что позволяет создавать более сложные веб-приложения.

Каждый Java Servlet представляет собой Java-класс, который реализует интерфейс Servlet. Сервер веб-приложений запускает каждый Servlet в отдельном потоке при обработке запросов.

JSP (JavaServer Pages) - это технология, используемая для создания динамических веб-страниц на языке Java. Она позволяет встраивать код Java в HTML-страницы, что позволяет генерировать HTML-код динамически на основе данных, полученных от пользователя или из базы данных.

Эта технология работает на серверной стороне, то есть код выполняется на сервере, а не на клиентском компьютере. Когда клиент запрашивает страницу, сервер обрабатывает JSP-файл, выполняет код Java и генерирует HTML-код, который отправляется обратно клиенту.

С использованием баз данных или сессий JSP имеет доступ к API Java Servlet, что позволяет создавать более сложные веб-приложения.

JSP, как и Java servlet, широко используется в веб-разработке и является частью Java EE (Enterprise Edition). Они позволяют разработчикам создавать динамические веб-приложения, которые могут быть адаптированы к изменяющимся потребностям пользователей.

MySQL - это свободная реляционная система управления базами данных (СУБД), которая использует язык SQL для управления данными. Она является одной из самых популярных СУБД в мире и широко используется в различных приложениях и веб-серверах.

Она поддерживает большое количество функций, включая поддержку транзакций, хранение процедур и функций, индексирование, шифрование данных, репликацию и многие другие. MySQL также легко масштабируется и может обрабатывать большие объемы данных.

Доступна в двух вариантах: Community Edition и Enterprise Edition. Community Edition является бесплатной и открытой для использования в любых проектах, в том числе в коммерческих. Enterprise Edition предлагает больше функций и инструментов для управления базами данных, и продается за плату.

MySQL поддерживает множество языков программирования и платформ, включая Java, PHP, Python, C ++ и другие. Она также интегрируется с различными средствами администрирования баз данных и совместима с большинством операционных систем.

Apache Maven - это инструмент управления проектами для сборки, тестирования и развертывания программного обеспечения, написанного на языке Java. Он позволяет автоматизировать процесс сборки проекта и управления зависимостями.

Maven использует файлы POM (Project Object Model), которые описывают проект и его зависимости. Он также предоставляет набор стандартных жизненных циклов сборки, которые можно настроить под нужды проекта.

Является открытым и свободным программным обеспечением, распространяемым под лицензией Apache. Maven интегрируется с различными средами разработки, такими как IntelliJ IDEA и Eclipse, и широко используется в проектах, основанных на Java.

Apache Tomcat, - это сервер приложений и контейнер сервлетов, который позволяет запускать веб-приложения, написанные на языке Java. Он разрабатывается Apache Software Foundation и является бесплатным и открытым программным обеспечением.

Tomcat предоставляет окружение для запуска Java-приложений, включая поддержку сервлетов, JSP и других технологий Java. Он также может использоваться в качестве веб-сервера для статических ресурсов, таких как HTML, CSS и JavaScript.

Tomcat имеет легковесную архитектуру, что делает его отличным выбором для небольших и средних проектов. Он также обладает хорошей масштабируемостью и может использоваться в крупных проектах в сочетании с балансировщиками нагрузки и другими инструментами.

## **2.3. Архитектурные решения.**

Web-сервис спроектирован по принципам монолитной архитектуры.

Монолитная архитектура - это традиционный подход к разработке приложений, в котором все компоненты приложения размещаются в одном кодовом базе и выполняются в одном процессе. В монолитной архитектуре, обычно, приложение состоит из нескольких слоев, таких как пользовательский интерфейс, бизнес-логика и доступ к данным, которые взаимодействуют друг с другом напрямую.

В монолитной архитектуре все функции и компоненты приложения находятся в одном месте, что может сделать разработку и развертывание приложения проще и быстрее. Однако, по мере роста размера и сложности приложения, монолитная архитектура может стать неудобной и трудно масштабируемой.

Шаблон проектирования Model-View-Controller (MVC) указывает, что приложение состоит из модели данных, представления и контроллера. Шаблон требует, чтобы все они были разделены на разные объекты.

MVC - это архитектурный паттерн, и он не описывает работу всего приложения. MVC в основном относится к пользовательскому интерфейсу и уровню взаимодействия с приложением. По-прежнему требуются классы с бизнес-логикой, а также, классы, отвечающие за обработку и доступ к данным Структура паттерна MVC представлена на рисунке 2.3.1.

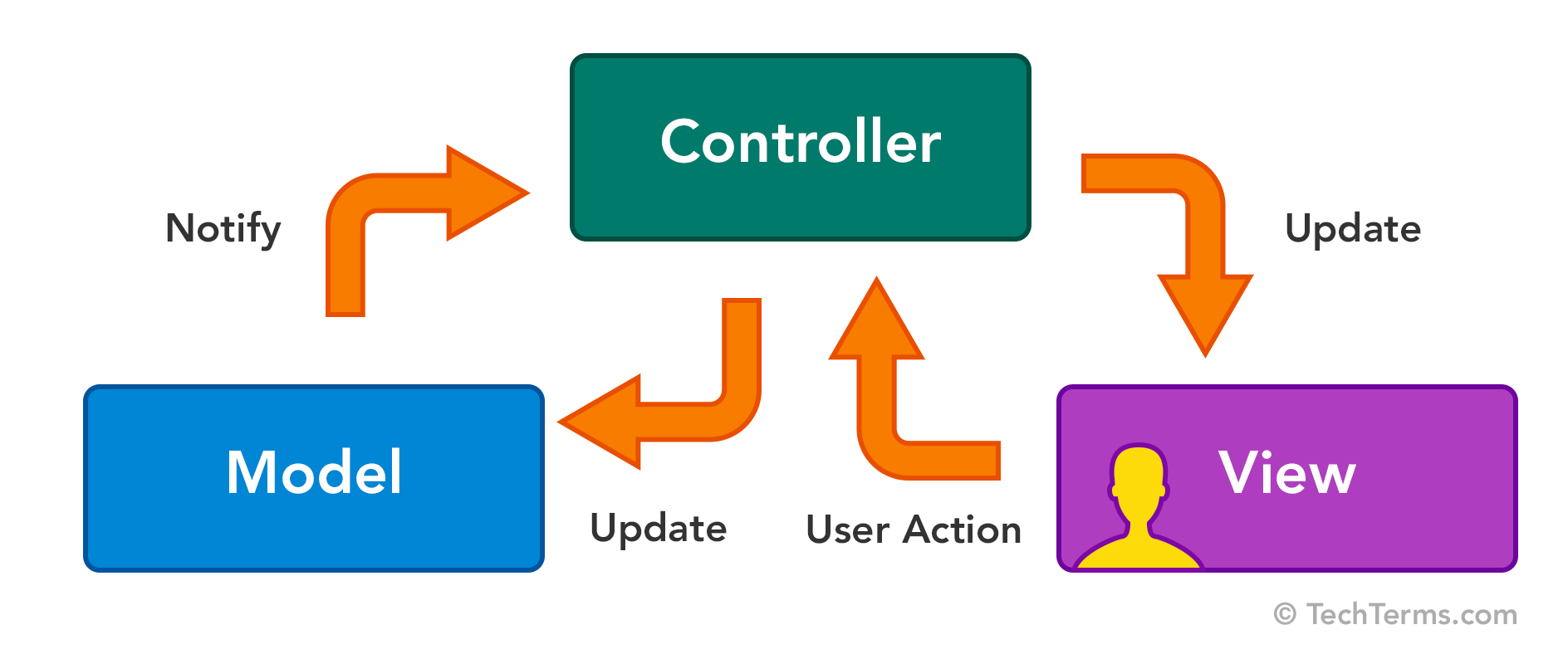


Рисунок 2.3.1 – Структура паттерна MVC

Модель содержит только данные приложения, в ней нет логики, описывающей, как следует представлять данные пользователю.

Представление показывает данные модели пользователю. Представление знает, как получить доступ к данным модели, но не знает, что означают эти данные или что пользователь может сделать, чтобы манипулировать ими.

Контроллер существует между представлением и моделью. Он слушает события, вызванные представлением (или другим внешним источником), и выполняет соответствующую реакцию на эти события. В большинстве случаев реакция заключается в вызове метода на модели.

В создаваемой системе контроллером является java сервлет обрабатывающий запросы, приходящие из клиентской части приложения, моделями являются классы «сущности», хранящие в себе необходимую информацию. А в роли представления выступает клиентская часть приложения, созданная с помощью Java Server Pages, с помощью взаимодействия с которой на сервлет отправляются запросы [5].

Паттерн «Команда» (Command) позволяет инкапсулировать запрос на выполнение определенного действия в виде отдельного объекта. Этот объект запроса на действие и называется командой (Рисунок 2.3.2). При этом объекты, инициирующие запросы на выполнение действия, отделяются от объектов, которые выполняют это действие [7].

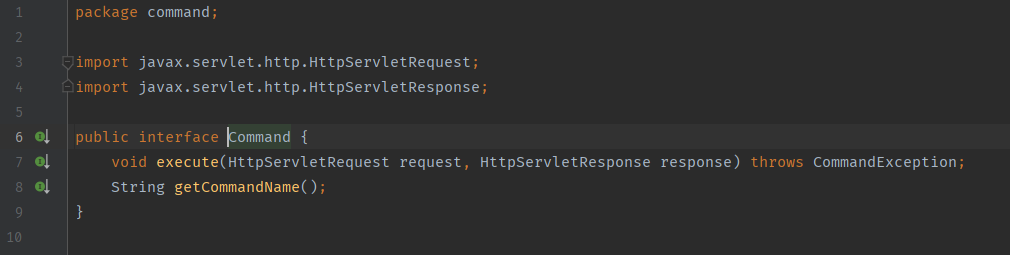


Рисунок 2.3.2 – Интерфейс «Command»

В данном проекте каждый запрос, приходящий на сервлет преобразуется в соответствующую команду. Для реализации этого каждый запрос содержит в себе параметр command, исходя из которого CommandProvider предоставляет экземпляр команды, который в дальнейшем выполняет операции по обработке пришедшего запроса (Рисунок 2.3.3).

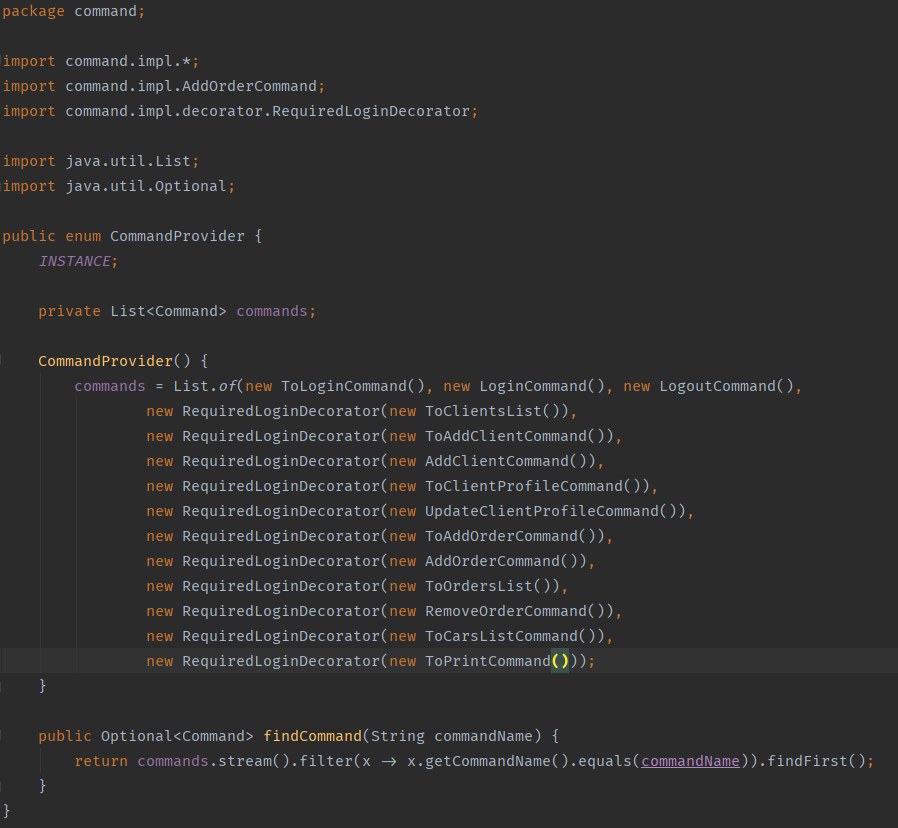


Рисунок 2.3.3 – Интерфейс «CommandProvider»

Декоратор (Decorator) представляет структурный шаблон проектирования, который позволяет динамически подключать к объекту дополнительную функциональность (Рисунок 2.3.4).

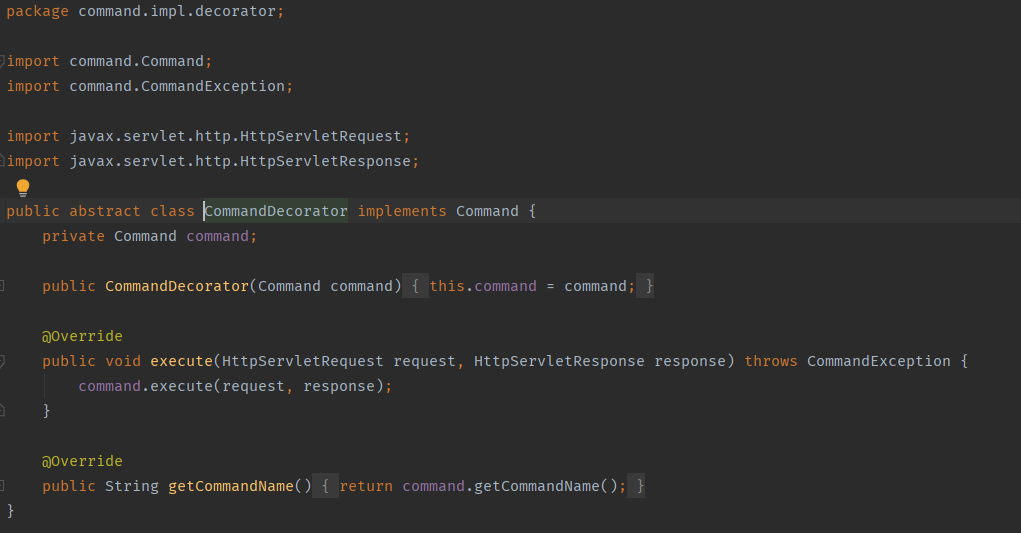


Рисунок 2.3.4 – Класс «CommandDecorator»

Для определения нового функционала в классах нередко используется наследование. Декораторы же предоставляет наследованию более гибкую альтернативу, поскольку позволяют динамически в процессе выполнения определять новые возможности у объектов.

В нашем случае декораторы служат для того, чтобы добавлять дополнительную функциональность классам команд. Например, дополнительные проверки перед запуском основной команды.

## **2.4. Описание алгоритмов, реализующих ключевую бизнес-логику**

## **разрабатываемого программного средства.**

Блок-схема алгоритма обработки запроса представлена на рисунке 2.4.1.

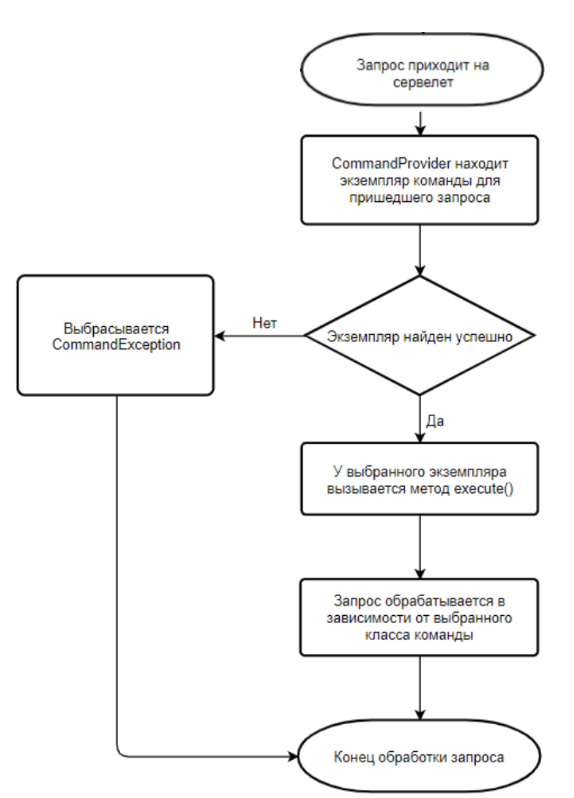


Рисунок 2.4.1 – Алгоритм обработки запроса

Для того, чтобы обработать запрос сервлету необходимо получить экземпляр класса команды. Для этого он обращается к классу CommandProvider, который в свою очередь ищет подходящий экземпляр команды исходя из данных, пришедших в запросе. В случае, если экземпляр не найден выбрасывается CommandException и на клиентскую часть приложения возвращается http код 500 (internal server error). Иначе полученному экземпляру передаются данные запроса посредством вызова метода execute. Тип класса экземпляра команды определяет логику обработки запроса. По завершении работы метода execute обработка запроса завершается.

Данный алгоритм позволяет эффективно обрабатывать запросы, поддерживая добавление новых обработчиков, что является основными критериями при разработке данного приложения.

## **2.5. Проектирование пользовательского интерфейса.**

Для начала работы пользователю необходимо войти в систему. Для этого на экране входа в систему требуется ввести логин и пароль, а затем нажать кнопку «Войти» (Рисунок 2.5.1). Если же введенные логин и пароль оказываются недействительными, то выводится сообщение об ошибке (Рисунок 2.5.2).

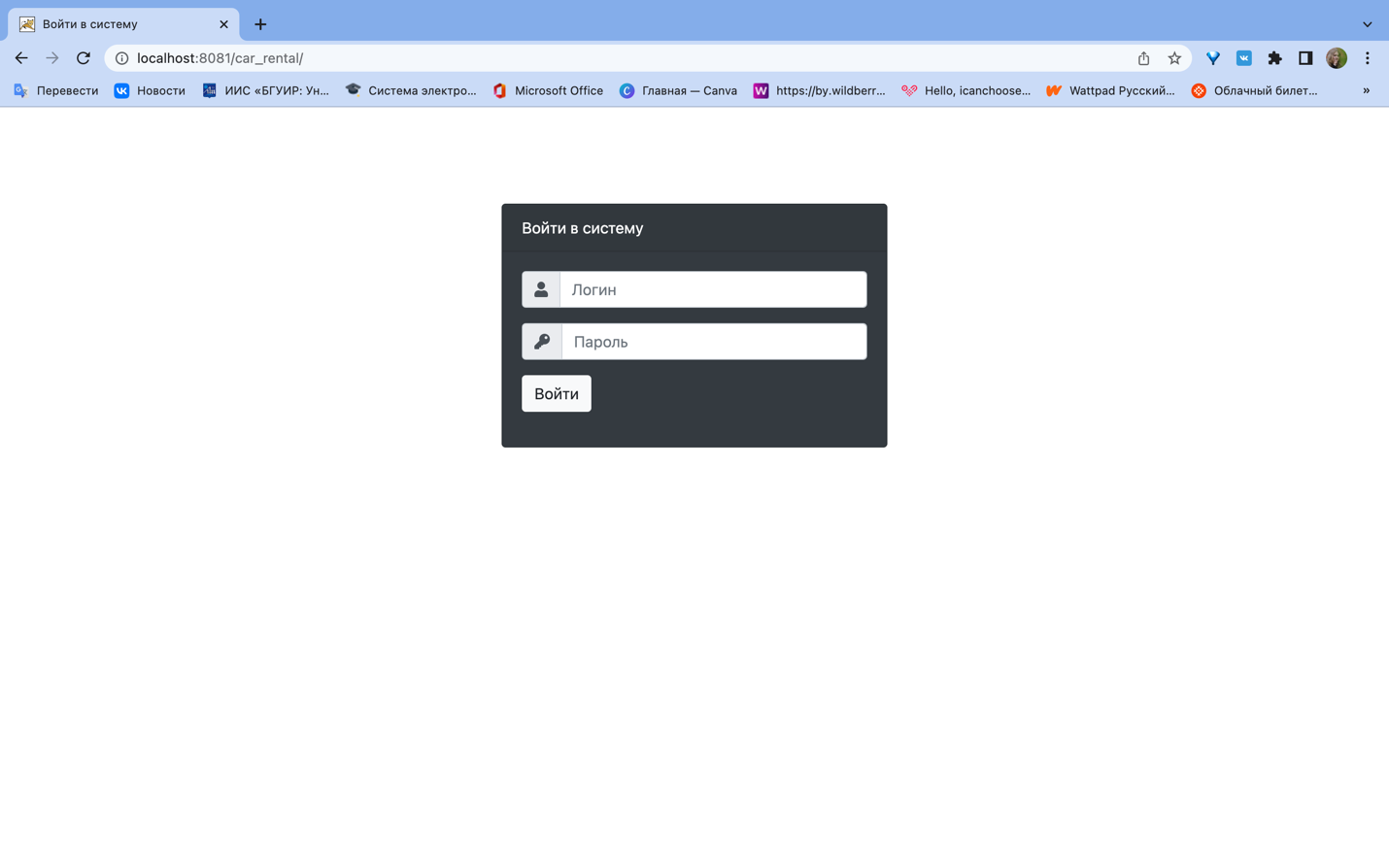


Рисунок 2.5.1 – Экран входа в систему.

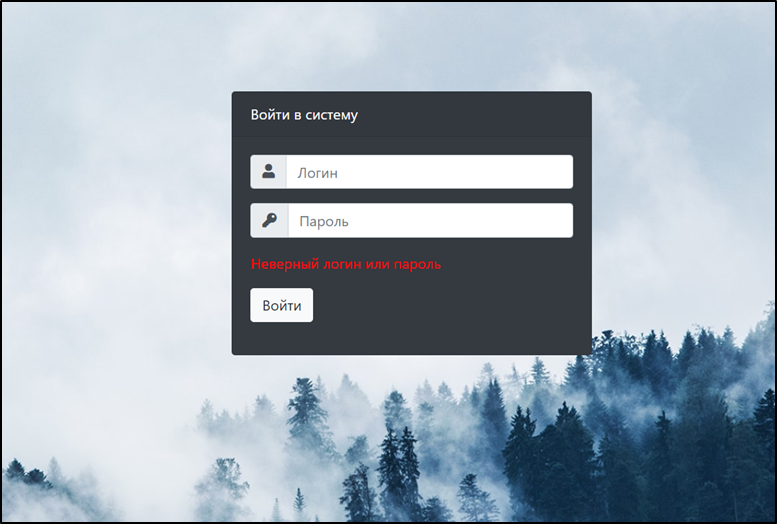


Рисунок 2.5.2 – Вывод сообщения об ошибке.

После выполнения процедуры авторизации пользователь попадает на экран просмотра списка клиентов (Рисунок 2.5.3).

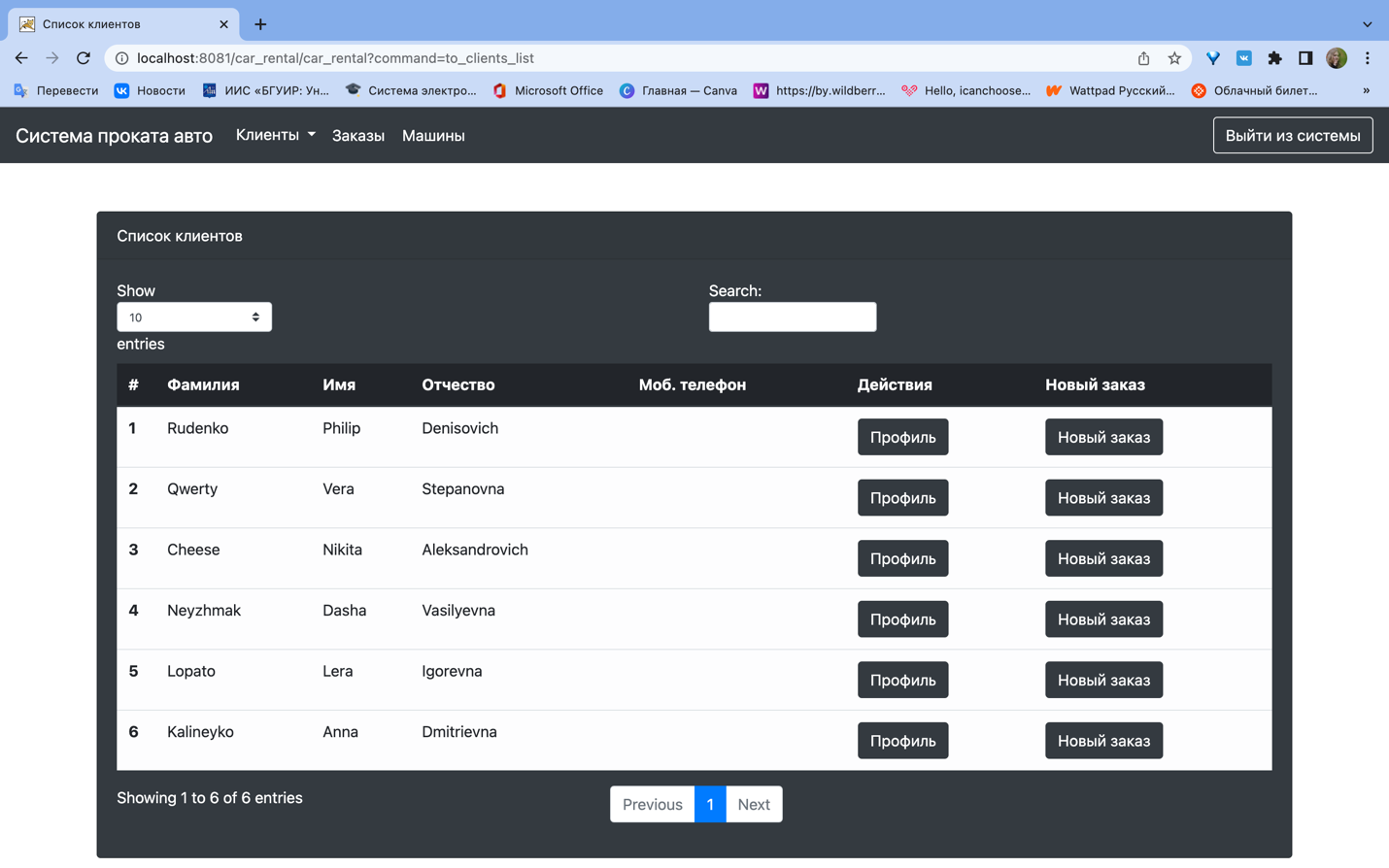


Рисунок 2.5.3 – Экран просмотра списка клиентов.

Нажав на пункт навигационного меню «Клиенты», и выбрав в появившемся выпадающем меню пункт «Добавить клиента» можно перейти на экран добавления нового клиента (Рисунок 2.5.4).

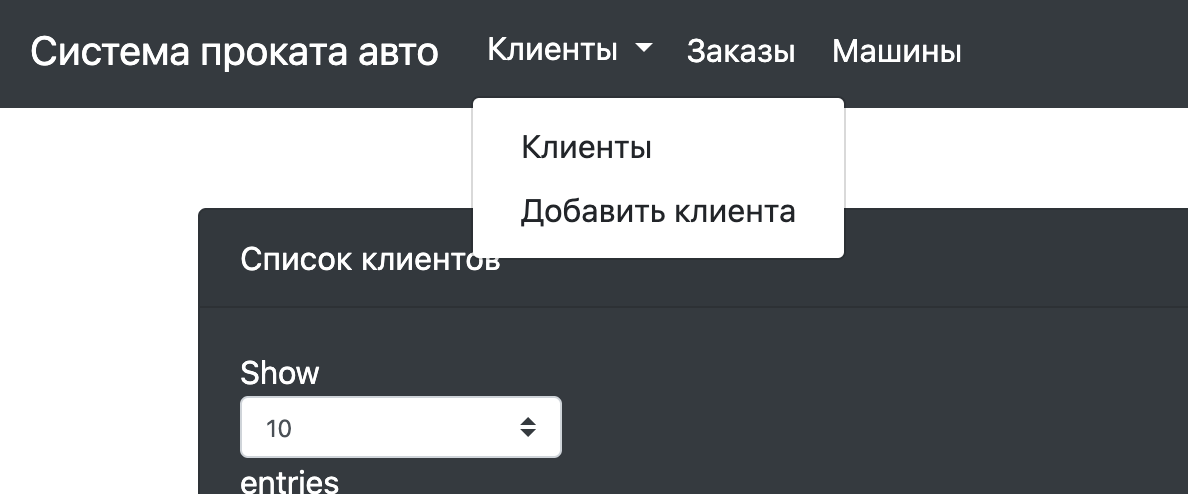


Рисунок 2.5.4 – Пункт меню «Клиенты».

Для добавления нового клиента необходимо заполнить всю требуемую информацию, и затем нажать на кнопку «Добавить клиента» (Рисунок 2.5.5). Если данные о пользователе будут иметь неверный формат, то выведется сообщение об ошибке (Рисунок 2.5.6).

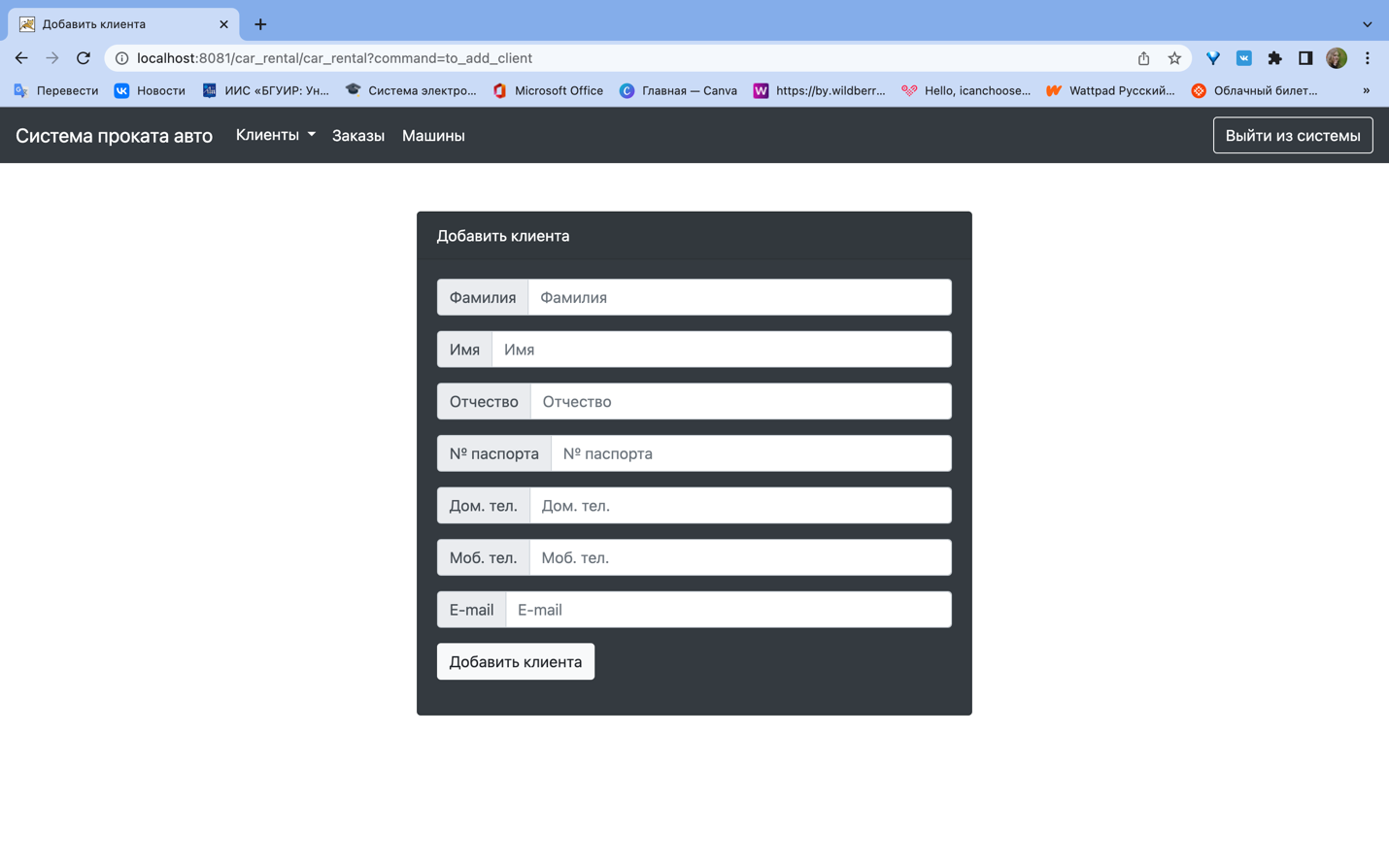


Рисунок 2.5.5 – Экран добавления нового клиента.

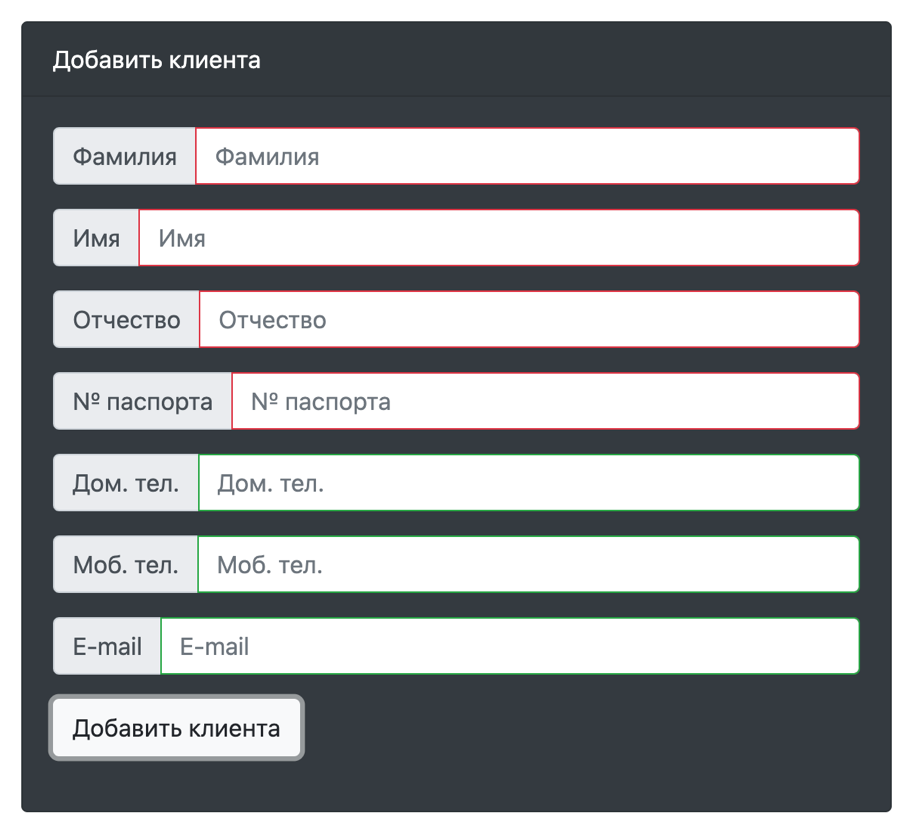


Рисунок 2.5.6 – Неудачная попытка заполнения

После удачного добавления клиента пользователь перенаправляется на экран со списком клиентов.

На экране списка клиентов также можно просмотреть либо изменить данные любого из клиентов. Для этого в таблице с клиентами для выбранного клиента необходимо нажать на кнопку «Профиль». Экран просмотра профиля клиента представлен на рисунке 2.5.7.

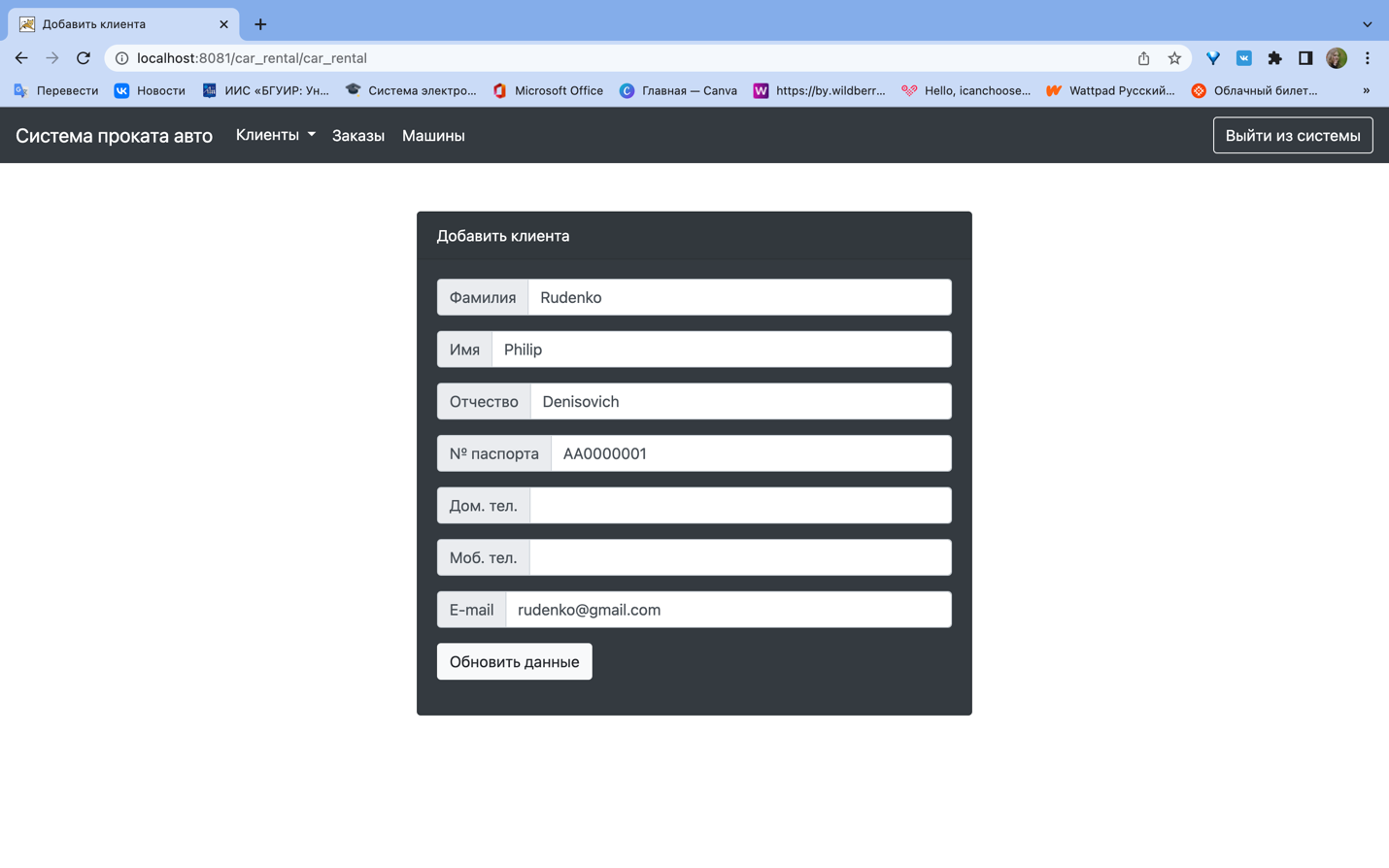


Рисунок 2.5.7 – Просмотр данных клиентов.

Из экрана списка клиентов также доступны списки заказов и доступных машин.

На экране заказов отображается список активных заказов с возможностью удалить их, а также напечатать информацию в формате pdf.

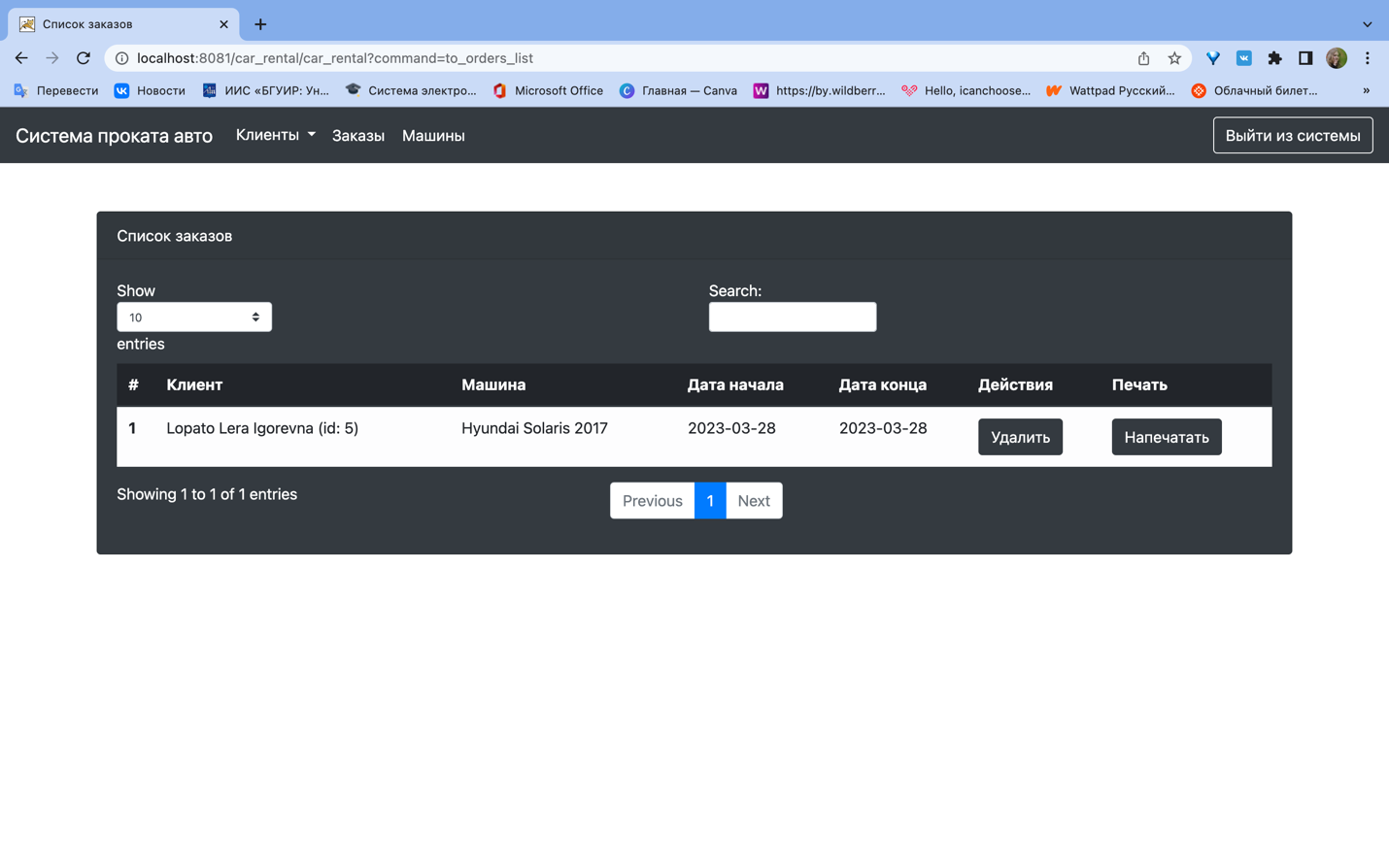


Рисунок 2.5.8 – Просмотр списка заказов.

Нажав на пункт навигационного меню «Машины», и выбрав в появившемся выпадающем меню пункт «Добавить машину» можно перейти на экран добавления новой машины (Рисунок 2.5.9).

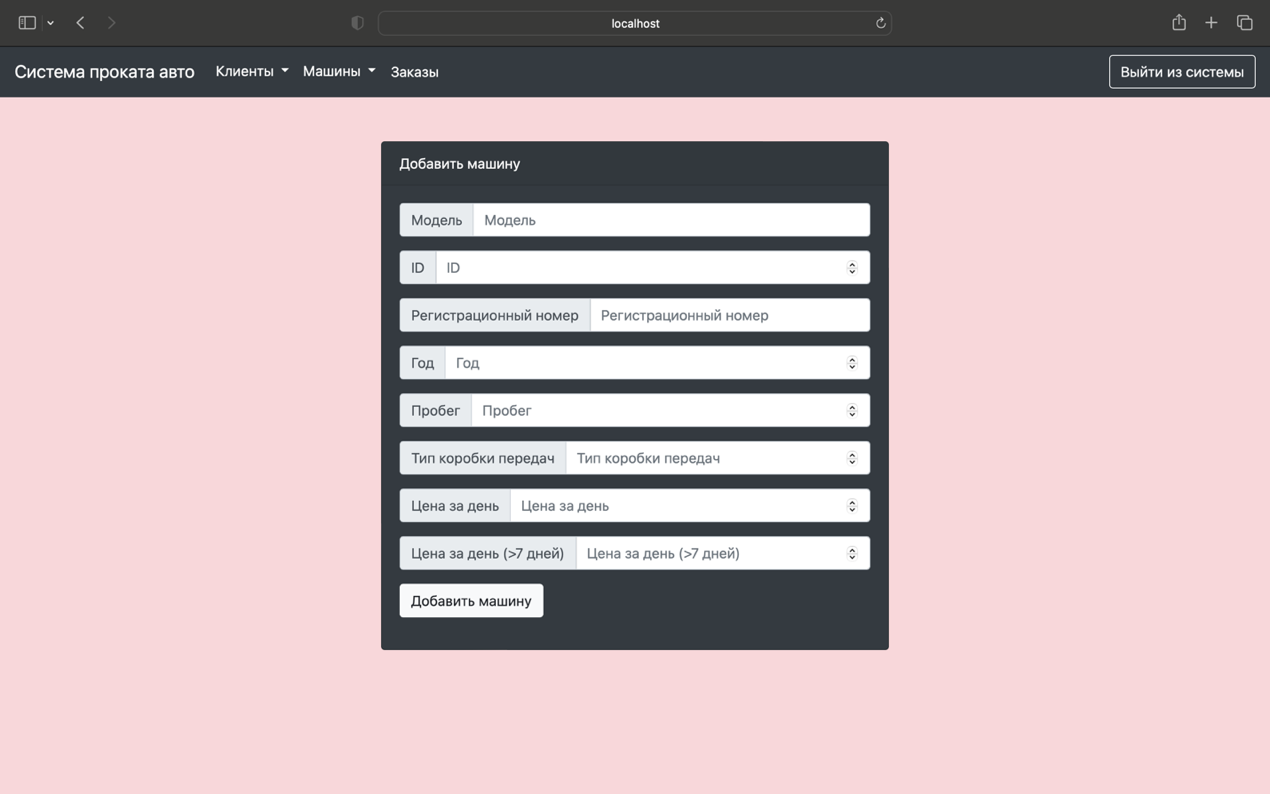


Рисунок 2.5.9 – Добавление машин

Если данные о пользователе будут иметь неверный формат, то поля отобразятся красным цветом (Рисунок 2.5.10).

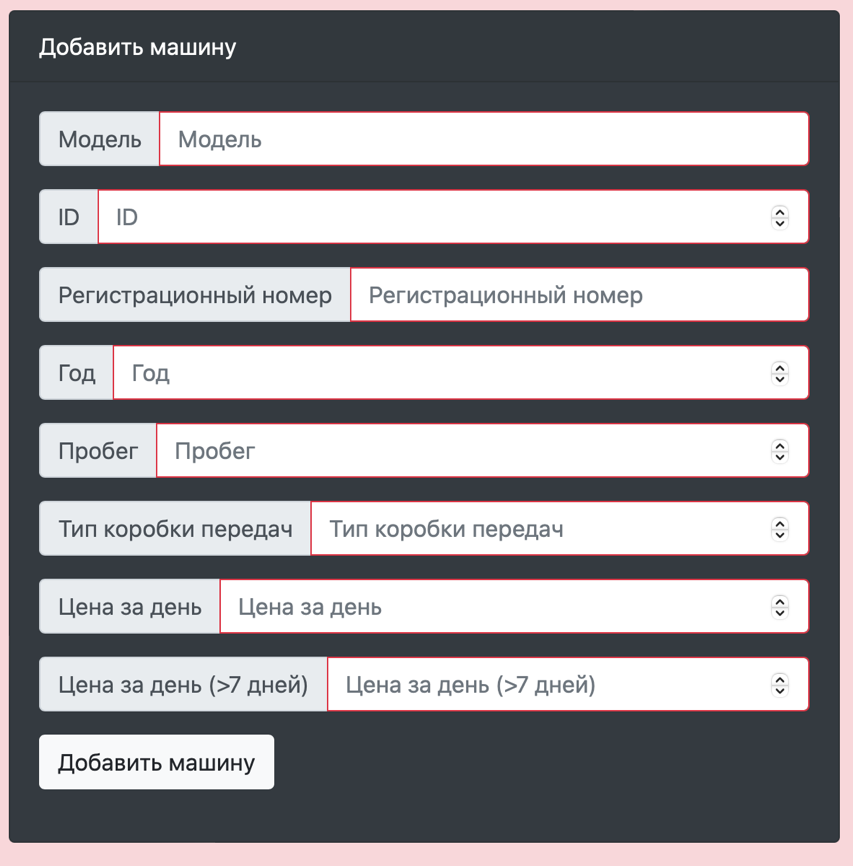


Рисунок 2.5.10 – Неудачная попытка заполнения

С помощью экрана «Машины» можно получить доступ к списку машин, с подробной информацией о каждой из них. Также есть возможность поиска необходимой модели в разделе «Search» (Рисунок 2.5.11).

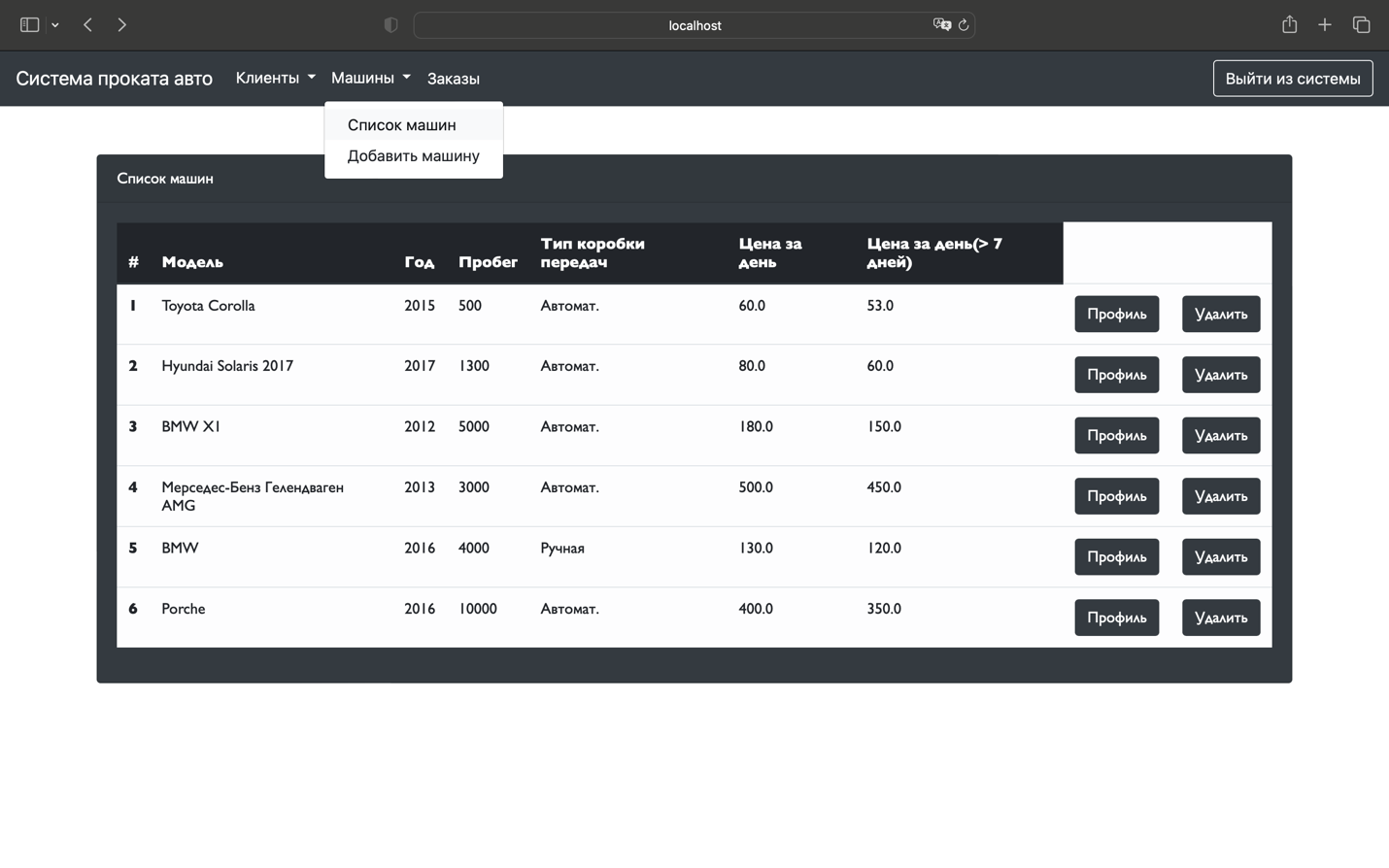


Рисунок 2.5.11 – Просмотр списка машин.

На экране списка машин также можно просмотреть либо изменить данные любой из машин. Для этого в таблице с машинами для выбранной машины необходимо нажать на кнопку «Профиль». Экран просмотра профиля машины представлен на рисунке 2.5.12. Также можно удалить любую машину, нажав на кнопку «Удалить».

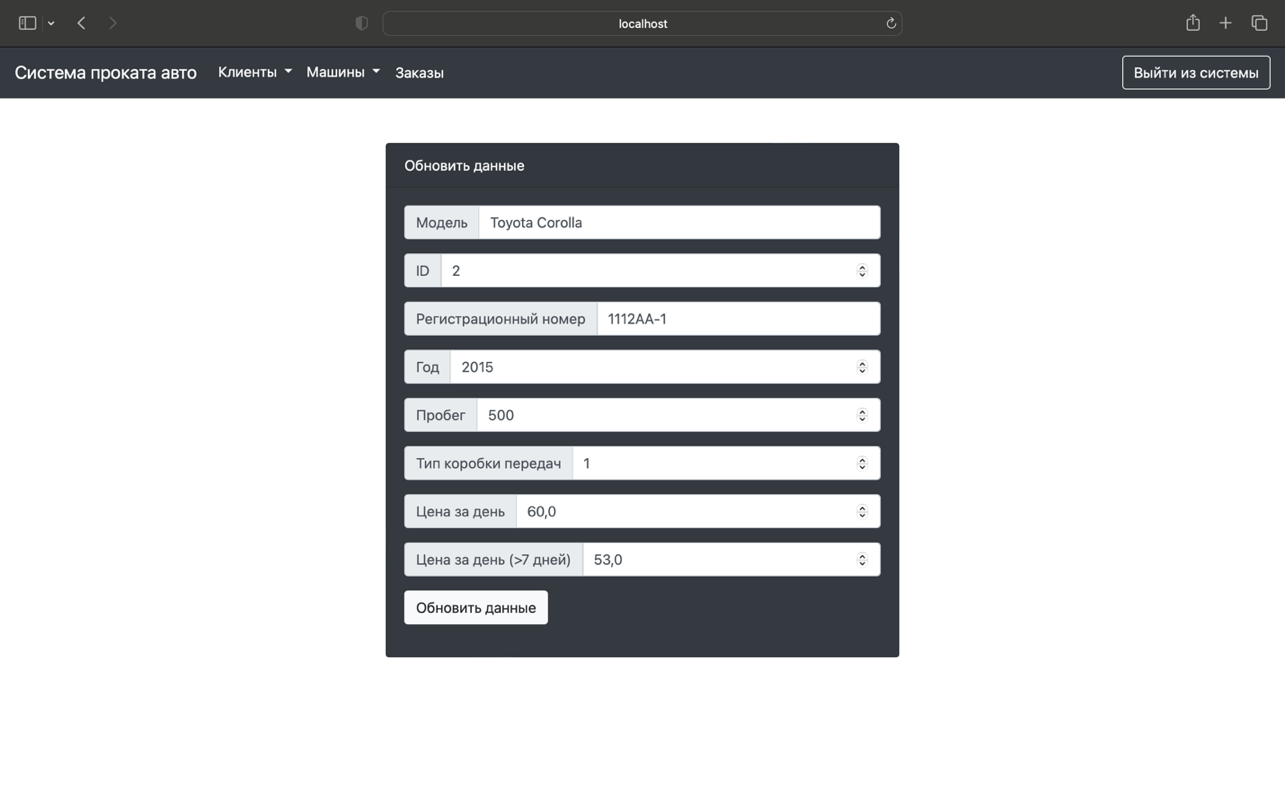


Рисунок 2.5.12 – Просмотр данных машин

Для того чтобы напечатать заказ, необходимо нажать кнопку «Напечатать». После нажатия, автоматически скачивается PDF файл.

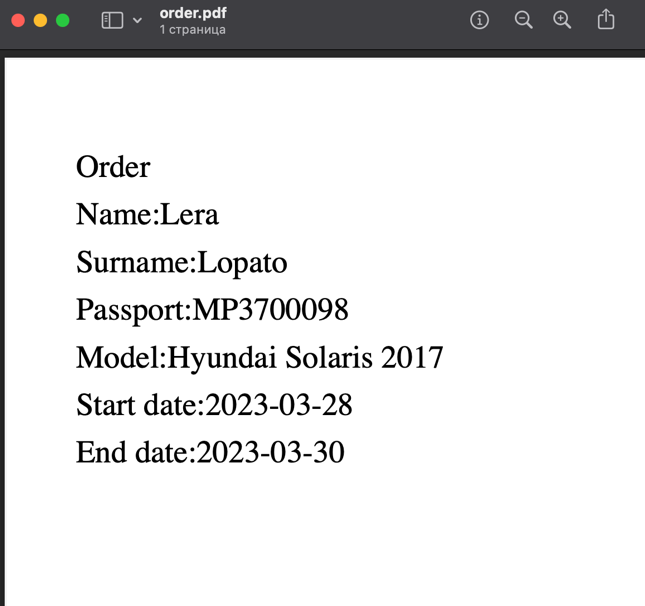


Рисунок 2.5.13 – Открытый PDF файл заказа.

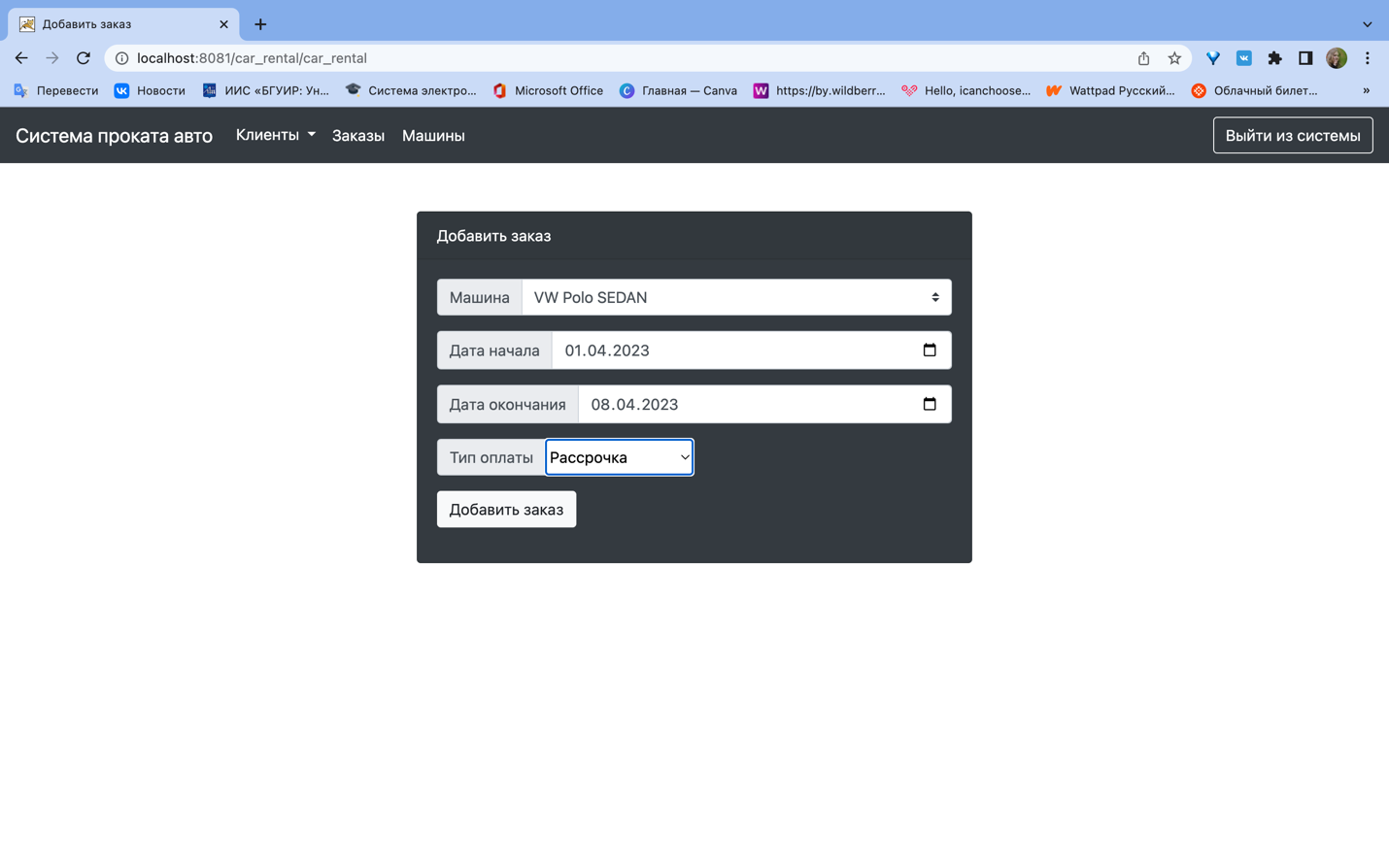


Рисунок 2.5.14 – Процесс заказа автомобиля.

При добавлении заказа можно выбрать тип оплаты: одним платежом или есть возможность взять рассрочку.

Завершение текущей сессии происходит с помощью нажатия кнопки «Выйти из системы», после чего пользователь будет снова перенаправлен на страницу входа.

# РУКОВОДСТВО ПО РАЗВЕРТЫВАНИЮ И ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА.

## **3.1 Руководство по установке (развертыванию) программного средства.**

Контейнеризация - это технология виртуализации, которая позволяет запускать приложения в изолированной среде, называемой контейнером. Контейнер является полностью изолированным окружением, которое включает в себя все необходимые зависимости, библиотеки и конфигурации, необходимые для запуска приложения. Контейнеризация позволяет разработчикам упаковывать приложения в контейнеры и переносить их между различными средами без необходимости настройки зависимостей и конфигураций.

Docker - это одна из самых популярных платформ для контейнеризации, которая использует стандарт контейнеризации Linux и предоставляет простой в использовании API для создания, управления и развертывания контейнеров. Docker позволяет разработчикам и IT-специалистам создавать, распространять и запускать приложения в контейнерах с помощью единого API и инструментария. Docker позволяет упаковывать приложения и их зависимости в контейнеры, что делает их переносимыми и изолированными от внешних факторов.

В разрабатываемом приложении содержится Dockerfile, который создает контейнер с Tomcat 9 и развертывает в нем приложение car\_rental.war.

FROM tomcat:9.0 - необходимо использовать официальный образ Tomcat 9 из Docker Hub в качестве базового образа.

RUN useradd tomcat – нужно создать пользователя tomcat внутри контейнера.

RUN chown -R tomcat:tomcat /usr/local/tomcat/\* - далее устанавливаем владельца и группу для всех файлов и папок в директории /usr/local/tomcat на пользователя tomcat.

COPY target/car\_rental.war /usr/local/tomcat/webapps/ROOT.war - копируем файл car\_rental.war из директории target (предполагается, что он существует в проекте) в директорию /usr/local/tomcat/webapps и переименовываем его в ROOT.war. Это позволяет развернуть приложение в корневом контексте.

CMD ["catalina.sh", "run"] - запускаем Tomcat, вызывая скрипт catalina.sh с параметром run.

Для сборки контейнера зайдем в репозиторий, где находится Dockerfile, используем докер-образ описанный выше и с помощью команды: docker build -t <ИМЯ ОБРАЗА> .

После того, как сборка образа завершится успешно, вы можете запустить контейнер, используя команду docker run: docker run -p 8080:8080 -d <ИМЯ ОБРАЗА>.

Итак, этот Dockerfile позволяет создать контейнер, который будет содержать Tomcat 9 и развернутое приложение car\_rental.war, однако наше приложение так же нуждается в базе данных, для ее развертывания также воспользуемся Докером. С помощью docker compose, инструмента для описания и записи многоконтейнерных приложений запустим контейнер собранный по нашем образу и контейнер в базой данных вместе.

В файле docker-compose.yml описано взаимодействие двух контейнеров.

Служба database использует официальный образ MySQL, задает параметры конфигурации и привязывает директорию хост-системы к директории контейнера MySQL для сохранения данных.

Служба web-server собирает образ из Dockerfile, который находится в родительском каталоге проекта.

Чтобы запустить эти контейнеры, вам нужно установить Docker Compose на свою машину и выполнить следующую команду в директории: docker-compose up –build.

Docker Compose автоматически создаст и запустит два контейнера: database и web-server.

При необходимости локального запуска разрабатываемого приложения на компьютере, необходимо установить и запустить Tomcat. Для того, чтобы установить Tomcat на компьютер, необходимо сначала установить Java Development Kit (JDK). После загрузки JDK, нужно запустить установочный. После установки JDK, следует скачать Tomcat с официального сайта Apache. Необходимо выбрать версию Tomcat, соответствующую операционной системе и архитектуре процессора на компьютере. После скачивания Tomcat, нужно распаковать архив в любую папку на компьютере. Для того, чтобы использовать Tomcat из командной строки, необходимо настроить переменные среды. После этого, нужно запустить Tomcat, перейдя в содержащуюся в нем папку bin и введя в командной строке «chmod +x \*.sh;», далее «./startup.sh». После этого Tomcat будет готов для дальнейшей работы.

После запуска проекта в любой установленной на компьютере IDE (Integrated Development Environment), необходимо собрать war-файл и поместить его в папку webapps, находящуюся в папке Tomcat.

В браузере необходимо задать путь «http://localhost:8081/car\_rental/car\_rental?command=to\_login», который выведет на страницу авторизации разрабатываемой программы. Пройдя ее, программа готова к использованию.

## **3.2 Руководство пользователя**

Разрабатываемое приложение для упрощения бронирования автомобилей на прокат. Пользователем приложения является работник проката автомобилей. Приложение создано для упрощения и автоматизации непосредственно его работы.

После прохождения авторизации, пользователь попадает на страницу «Клиенты», где отображается список клиентов уже внесенных в базу данных. Приложение дает доступ к профилю каждого клиента по кнопке «Профиль», а также возможность сразу оформить заказ для каждого клиента, перейдя по кнопке «Новый заказ». Также есть возможность добавить нового клиента или изменить данные о клиенте, уже внесенном в базу данных.

Приложение позволяет увидеть список автомобилей, уже внесенных в базу данных, а также изменить информацию о них, добавить новую машину в базу данных или вовсе удалить любую из машин.

На странице «Заказы», пользователь имеет доступ к выводу заказа в pdf-файл, а также к удалению любого заказа из базы данных.

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Алгоритмы работы программы**

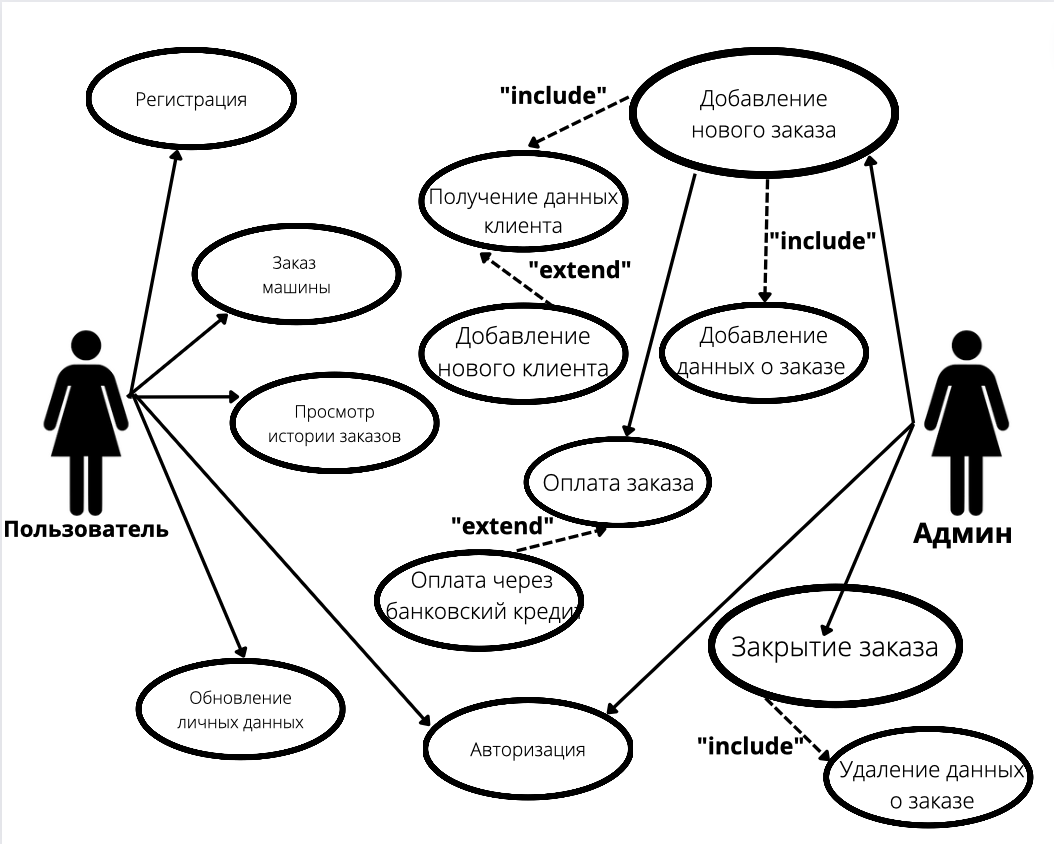


Рисунок А.1 – Система автоматизированного проката

Продолжение приложения А

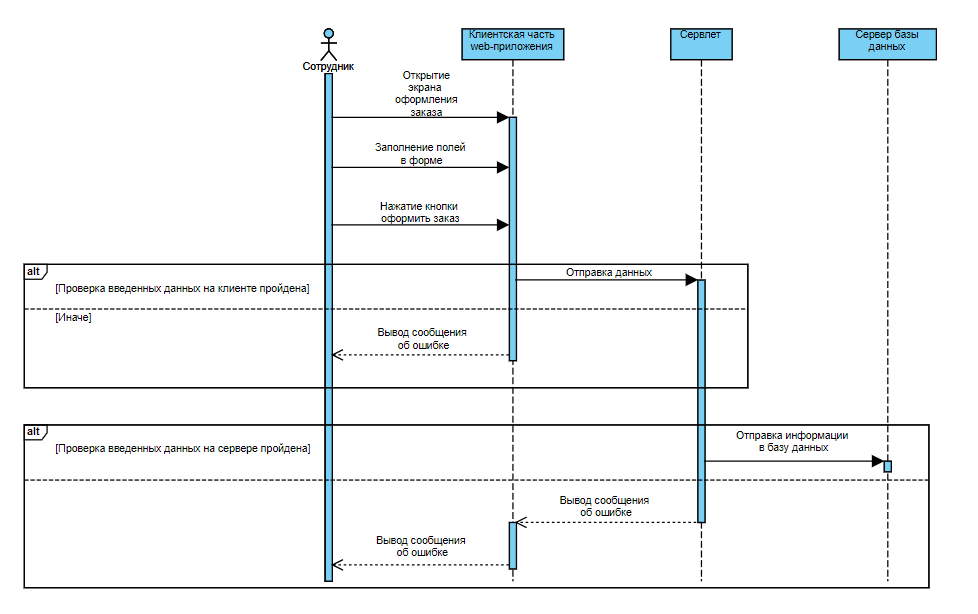


Рисунок А.2 – Диаграмма последовательности

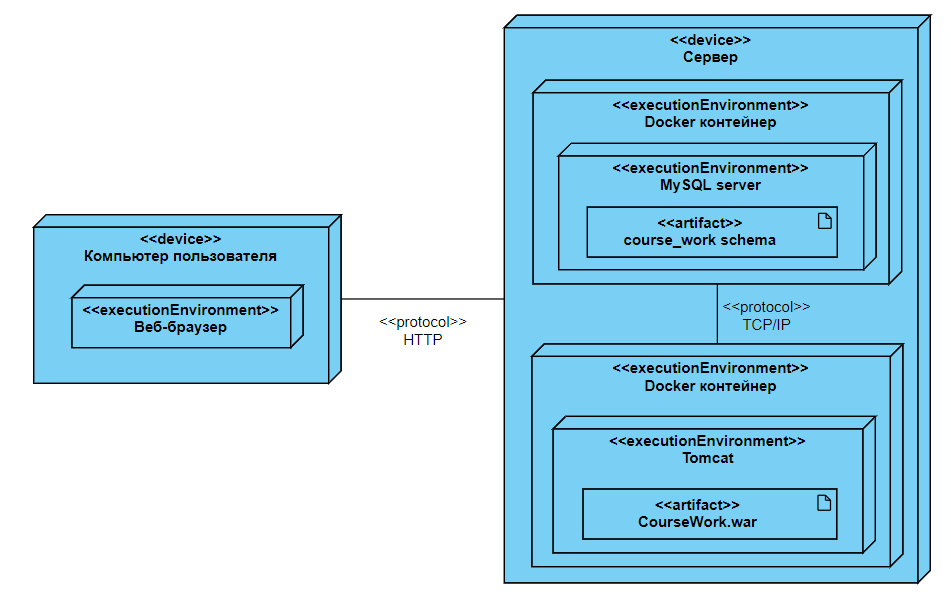


Рисунок А.3 – Диаграмма развертывания

Продолжение приложения А

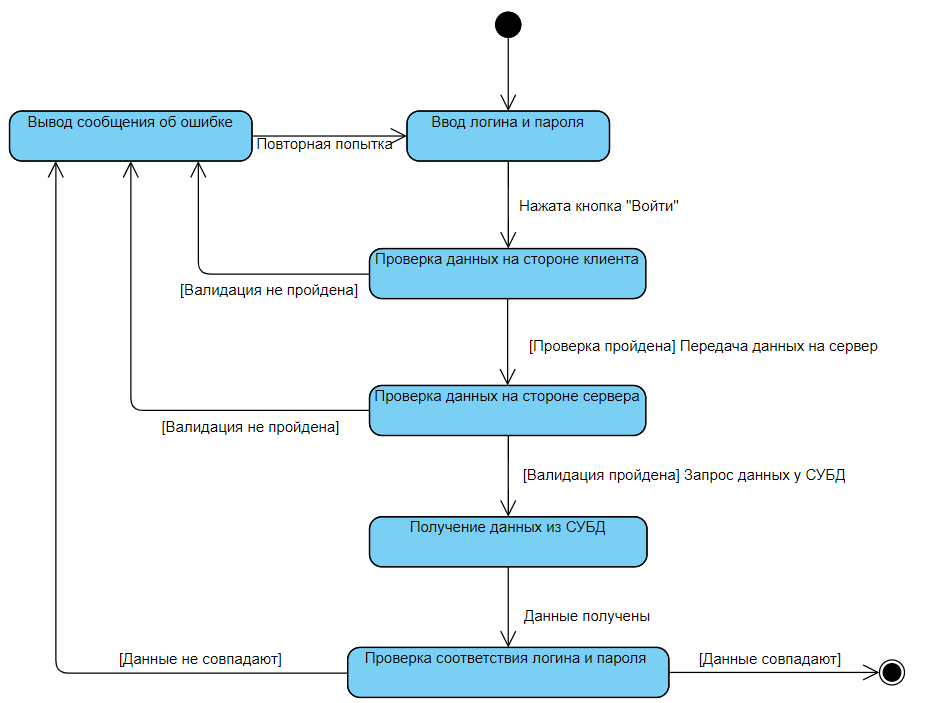


Рисунок А.4 – Диаграмма авторизации

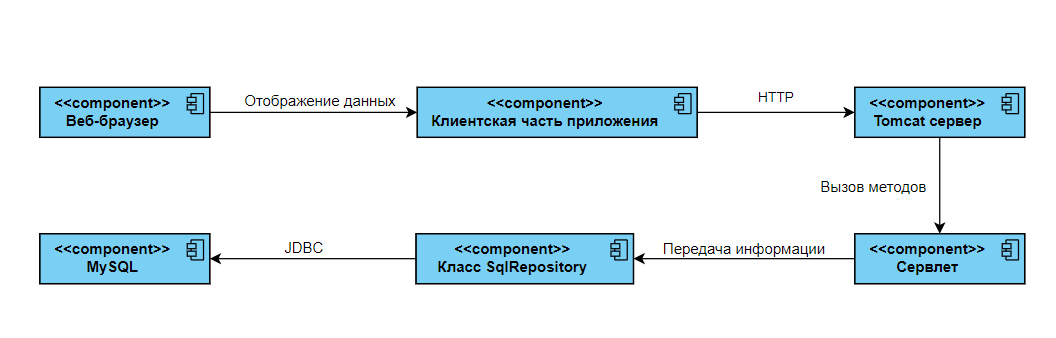


Рисунок А.5 – Диаграмма компонентов.

Продолжение приложения А

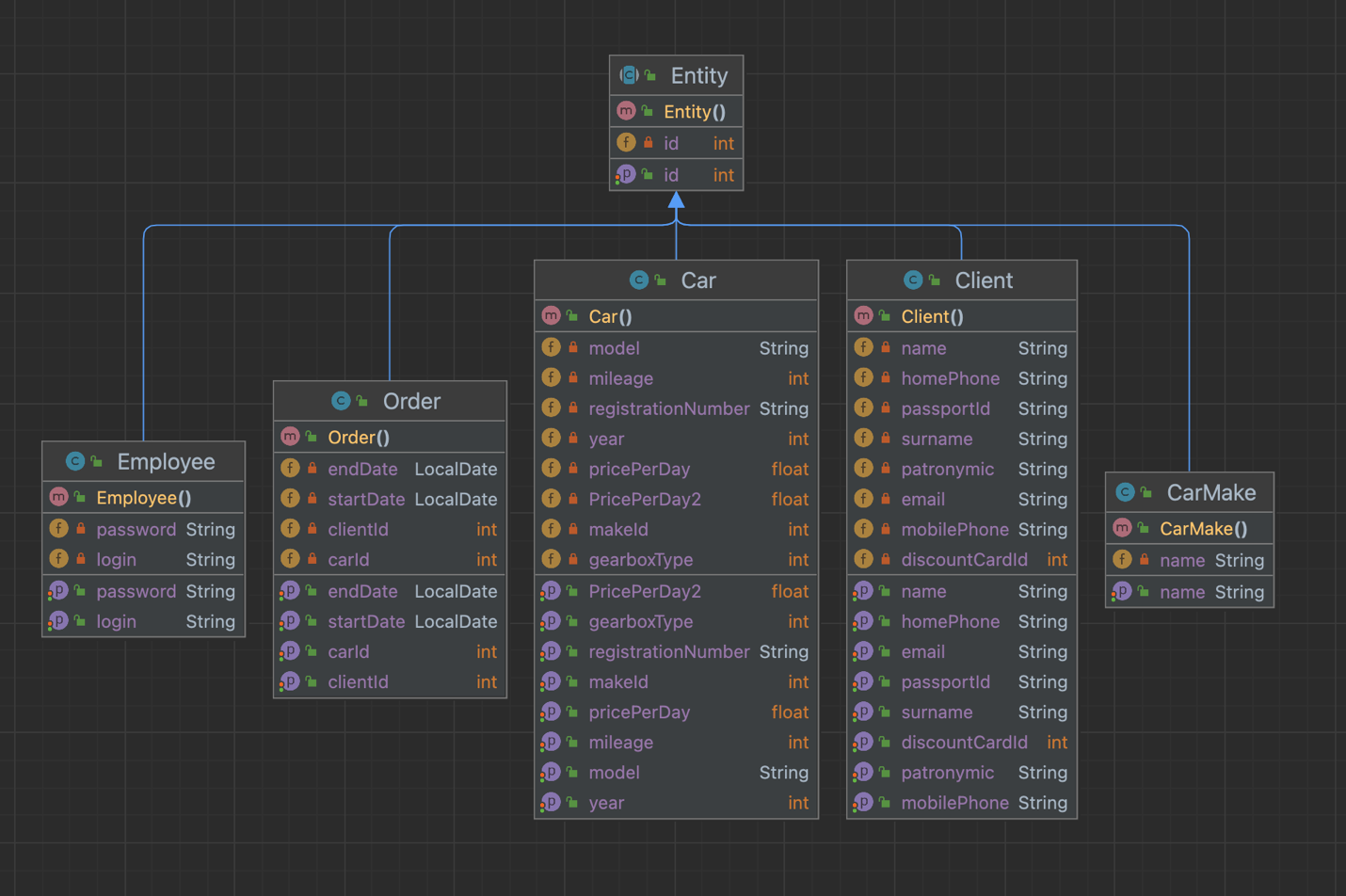


Рисунок А.6 – UML-диаграмма классов

Изображение выглядит как снимок экрана, монитор, черный, сидит

Автоматически созданное описание

Рисунок А.7 – UML-диаграмма классов

# 

# ПРИЛОЖЕНИЕ Б

**(обязательное)**

**Алгоритмы работы программы**

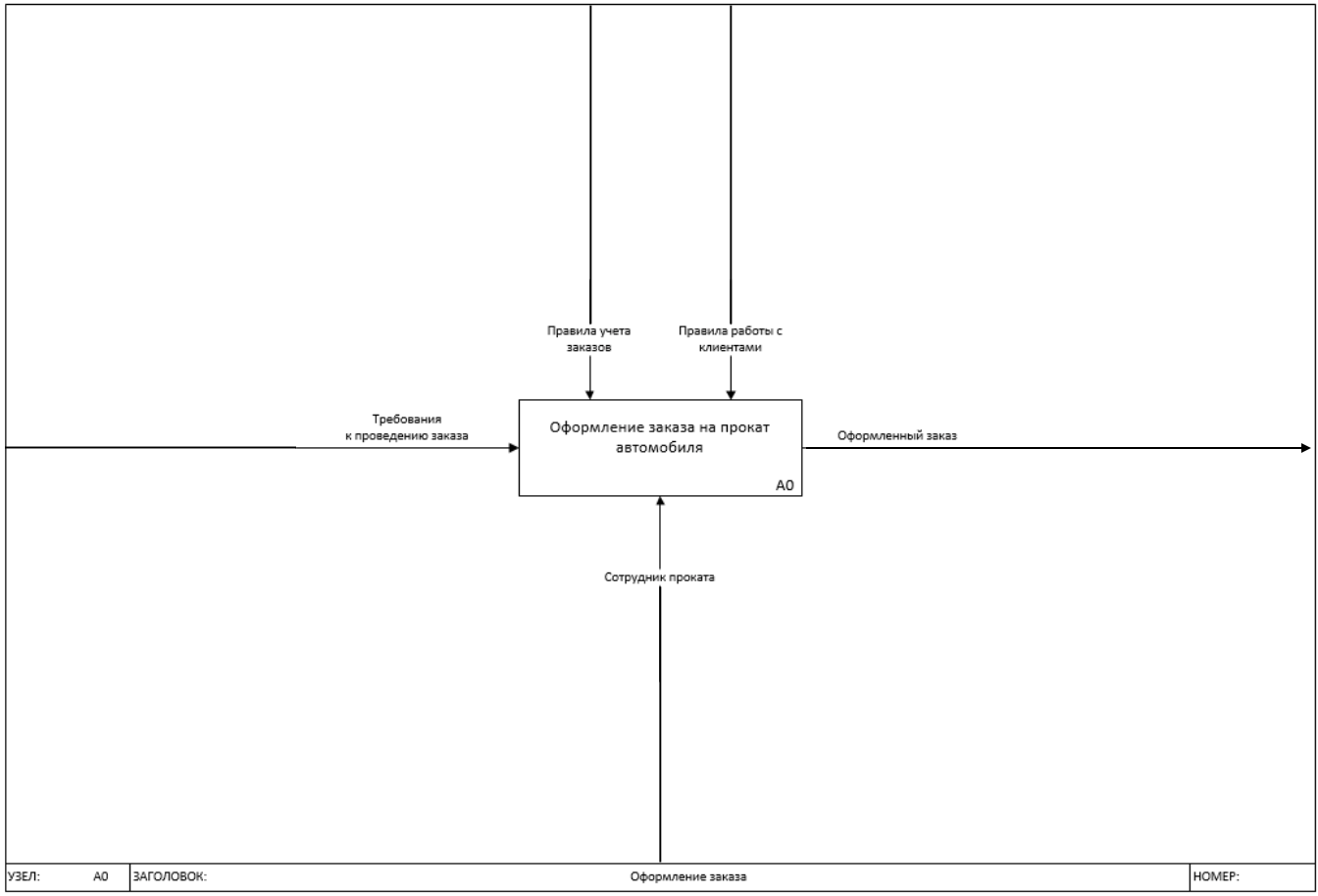


Рисунок Б.1 – Диаграмма оформления заказа

Продолжение приложения Б

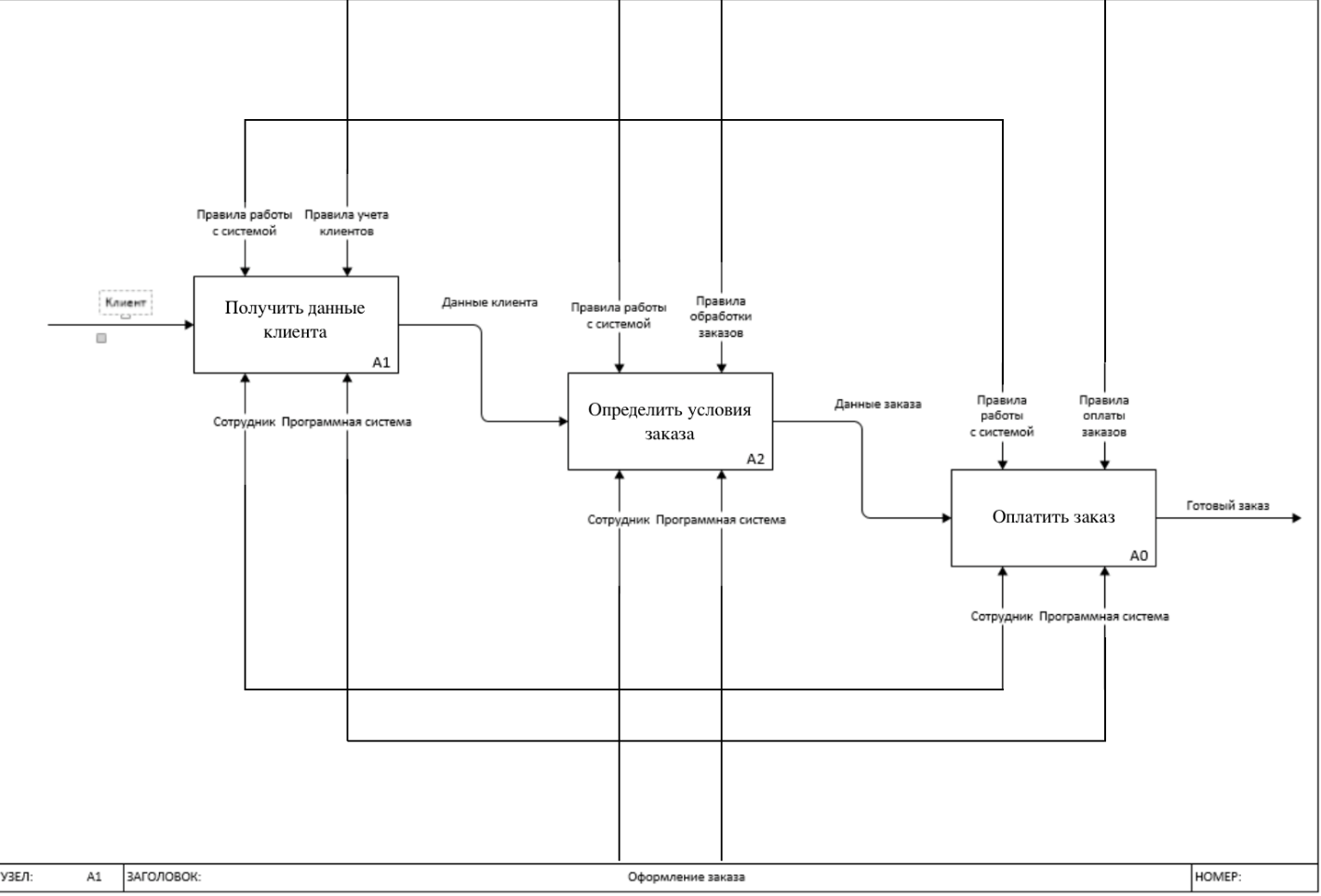
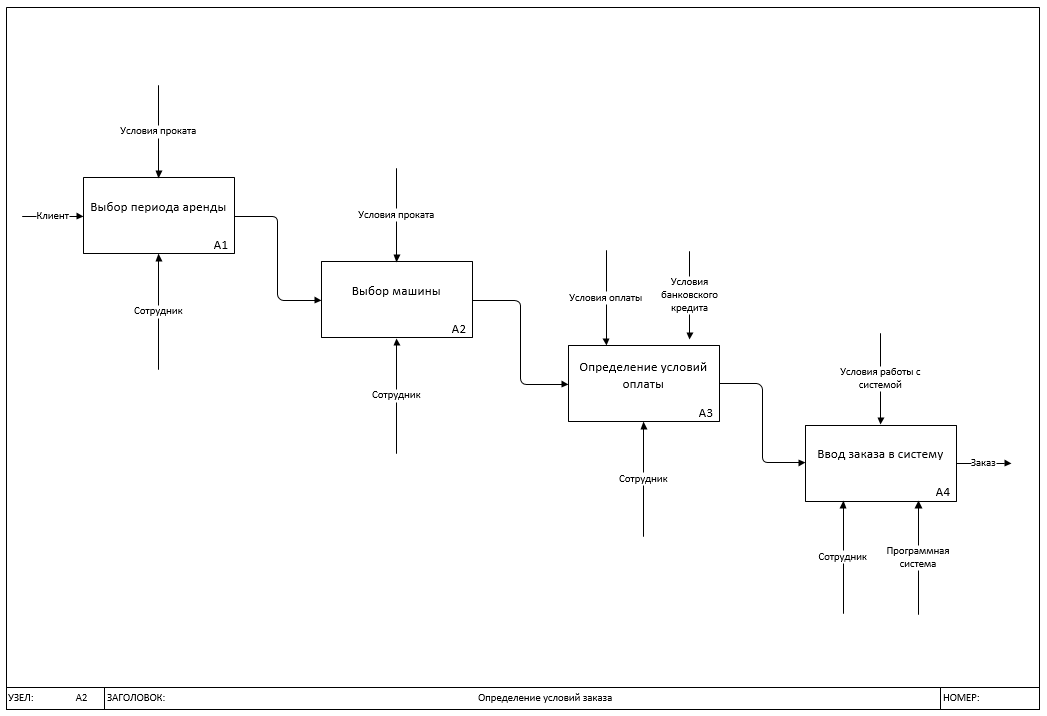


Рисунок Б.2 – Декомпозиция работы «Оформление заказа».

Продолжение приложения Б

Рисунок Б.3 – Декомпозиция работы «Определение условий заказа»



Продолжение приложения Б

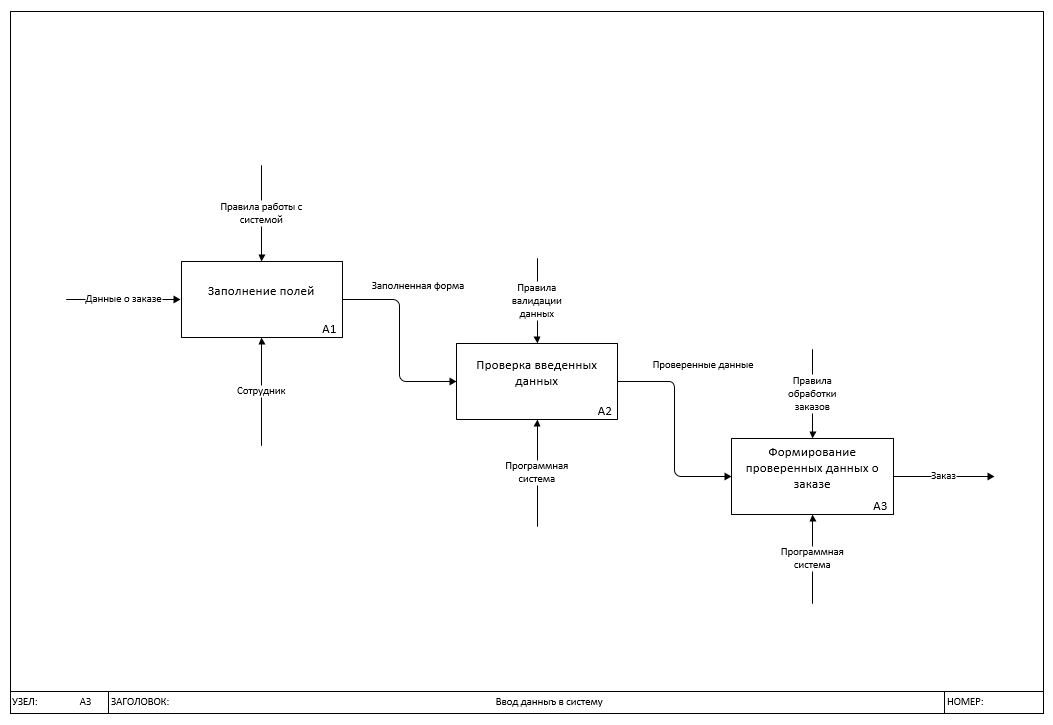


Рисунок Б.4 – Декомпозиция работы «Ввод данных в систему»

# ПРИЛОЖЕНИЕ В

**(обязательное)**

**Листинг кода программы**

package command.impl.decorator;  
  
import command.Command;  
import command.CommandException;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
  
public abstract class CommandDecorator implements Command {  
 private Command command;  
  
 public CommandDecorator(Command command) {  
 this.command = command;  
 }  
  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 command.execute(request, response);  
 }  
  
 @Override  
 public String getCommandName() {  
 return command.getCommandName();  
 }  
}

package command.impl.decorator;  
  
import command.Command;  
import command.CommandException;  
import entity.Employee;  
import service.EmployeeService;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.util.Optional;  
  
public class RequiredLoginDecorator extends CommandDecorator {  
 public RequiredLoginDecorator(Command command) {  
 super(command);  
 }  
  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 String employeeLogin = (String) request.getSession().getAttribute("login");  
 Optional<Employee> employeeOptional = EmployeeService.*INSTANCE*.getByLogin(employeeLogin);  
  
 if (!employeeOptional.isPresent()) {  
 try {  
 response.sendError(403); // forbidden  
 } catch (IOException e) {  
 throw new CommandException(e);  
 }  
 return;  
 }  
  
 super.execute(request, response);  
 }  
}

package command.impl;  
  
import command.CommandException;  
import command.EntityBuilder;  
import command.PagePath;  
import dbbinding.impl.CarBinding;  
import entity.Car;  
import repository.RepositoryException;  
import repository.impl.SqlRepository;  
import validation.impl.CarValidator;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.util.Optional;  
  
public class AddCarCommand implements CommandWithValidation {  
 @Override  
 public void onSuccess(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 try {  
 String params = "/car\_rental?command=to\_cars\_list";//стр со списком машин  
 response.sendRedirect(PagePath.*SERVLET* + params);  
 } catch (IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public void onValidationError(HttpServletRequest request, HttpServletResponse response, String feedback) throws CommandException {  
 try {  
 request.setAttribute("feedback", feedback);  
 request.getRequestDispatcher(PagePath.*SERVLET* + "/car\_rental?command=to\_add\_car").forward(request, response);  
 } catch (ServletException | IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 Optional<Car> optionalCar = EntityBuilder.*INSTANCE*.buildCar(request);  
 CarValidator validator = new CarValidator(); //проверка подлинности данных  
  
 if (!optionalCar.isPresent())  
 {  
 try {  
 response.sendError(400);  
 } catch (IOException e) {  
 throw new CommandException(e);  
 }  
 return;  
 }  
  
 Car car = optionalCar.get();  
  
 if (!validator.isValid(car)) {  
 onValidationError(request, response, validator.getValidationFeedback());  
 return;  
 }  
  
 try {  
 new SqlRepository<Car>(new CarBinding()).addEntity(car);//добавление машины в бд  
 } catch (RepositoryException e) {  
 onValidationError(request, response, "Ошибка добавления машины в базу данных");  
 return;  
 }  
  
 onSuccess(request, response);  
 }  
  
 @Override  
 public String getCommandName() {  
 return "add\_car";  
 }  
}

package command.impl;  
  
import command.CommandException;  
import command.EntityBuilder;  
import command.PagePath;  
import dbbinding.impl.ClientBinding;  
import entity.Client;  
import repository.RepositoryException;  
import repository.impl.SqlRepository;  
import validation.impl.ClientValidator;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.util.Optional;  
  
public class AddClientCommand implements CommandWithValidation {  
 @Override  
 public void onSuccess(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 try {  
 String params = "/car\_rental?command=to\_clients\_list";//стр со списком клиентов  
 response.sendRedirect(PagePath.*SERVLET* + params);  
 } catch (IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public void onValidationError(HttpServletRequest request, HttpServletResponse response, String feedback) throws CommandException {  
 try {  
 request.setAttribute("feedback", feedback);  
 request.getRequestDispatcher(PagePath.*SERVLET* + "/car\_rental?command=to\_add\_client").forward(request, response);  
 } catch (ServletException | IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 Optional<Client> optionalClient = EntityBuilder.*INSTANCE*.buildClient(request);  
 ClientValidator validator = new ClientValidator(); //проверка подлинности данных  
  
 if (!optionalClient.isPresent())  
 {  
 try {  
 response.sendError(400);  
 } catch (IOException e) {  
 throw new CommandException(e);  
 }  
 return;  
 }  
  
 Client client = optionalClient.get();  
  
 if (!validator.isValid(client)) {  
 onValidationError(request, response, validator.getValidationFeedback());  
 return;  
 }  
  
 try {  
 new SqlRepository<Client>(new ClientBinding()).addEntity(client);//добавление клиента в бд  
 } catch (RepositoryException e) {  
 onValidationError(request, response, "Ошибка добавления клиента в базу данных");  
 return;  
 }  
  
 onSuccess(request, response);  
 }  
  
 @Override  
 public String getCommandName() {  
 return "add\_client";  
 }  
}

package command.impl;  
  
import command.CommandException;  
import command.EntityBuilder;  
import command.PagePath;  
import command.impl.CommandWithValidation;  
import dbbinding.impl.OrderBinding;  
import entity.Order;  
import repository.RepositoryException;  
import repository.impl.SqlRepository;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.util.Optional;  
import java.util.zip.DataFormatException;  
  
public class AddOrderCommand implements CommandWithValidation {  
 @Override  
 public void onSuccess(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 try {  
 String params = "/car\_rental?command=to\_clients\_list";  
 response.sendRedirect(PagePath.*SERVLET* + params);  
 } catch (IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public void onValidationError(HttpServletRequest request, HttpServletResponse response, String feedback) throws CommandException {  
 try {  
 request.setAttribute("feedback", feedback);  
 request.getRequestDispatcher(PagePath.*SERVLET* + "/car\_rental?command=to\_add\_order").forward(request, response);  
 } catch (ServletException | IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 Optional<Order> optionalOrder = EntityBuilder.*INSTANCE*.buildOrder(request);  
  
 if (!optionalOrder.isPresent())  
 {  
 try {  
 response.sendError(400);  
 } catch (IOException e) {  
 throw new CommandException(e);  
 }  
 return;  
 }  
  
 Order order = optionalOrder.get();  
  
 try {  
 new SqlRepository<Order>(new OrderBinding()).addEntity(order);  
 } catch (RepositoryException e) {  
 onValidationError(request, response, "Ошибка добавления заказа в базу данных");  
 return;  
 }  
  
 onSuccess(request, response);  
 }  
  
 @Override  
 public String getCommandName() {  
 return "add\_order";  
 }  
}

package command.impl;  
  
import command.Command;  
import command.CommandException;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
  
public interface CommandWithValidation extends Command {  
 void onSuccess(HttpServletRequest request, HttpServletResponse response) throws CommandException;  
  
 void onValidationError(HttpServletRequest request, HttpServletResponse response, String feedback) throws CommandException;  
}

package command.impl;  
  
import command.CommandException;  
import command.EntityBuilder;  
import command.PagePath;  
import entity.Employee;  
import service.EmployeeService;  
import validation.impl.EmployeeValidator;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
public class LoginCommand implements CommandWithValidation {  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 Employee employee = EntityBuilder.*INSTANCE*.buildEmployee(request).orElseThrow(CommandException::new);  
 EmployeeValidator validator = new EmployeeValidator();  
  
 //добавляю сообщение об ошибке в запрос и перенаправляю на страницу логина  
 if (!validator.isValid(employee)) {  
 onValidationError(request, response, validator.getValidationFeedback());  
 return;  
 }  
  
 if (!EmployeeService.*INSTANCE*.checkCredentials(employee)) {  
 onValidationError(request, response, "Неверный логин или пароль");  
 return;  
 }  
 //в сессию добавляю "login" с логином сотрудника  
 request.getSession().setAttribute("login", employee.getLogin());  
 onSuccess(request, response);  
 }  
  
 @Override  
 public String getCommandName() {  
 return "login";  
 }  
  
 @Override  
 public void onSuccess(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 try {  
 String params = "car\_rental?command=to\_clients\_list";  
 response.sendRedirect(PagePath.*SERVLET* + params);  
 } catch (IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public void onValidationError(HttpServletRequest request, HttpServletResponse response, String feedback) throws CommandException {  
 try {  
 request.setAttribute("feedback", feedback);  
 request.getRequestDispatcher(PagePath.*SERVLET* + "car\_rental?command=to\_login").forward(request, response);  
 } catch (IOException | ServletException e) {  
 throw new CommandException(e);  
 }  
 }  
}

package command.impl;  
  
import command.Command;  
import command.CommandException;  
import command.PagePath;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
public class LogoutCommand implements Command {  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 try {  
 request.getSession().invalidate();  
 String param = "/car\_rental?command=to\_login";  
 response.sendRedirect(PagePath.*SERVLET* + param);  
 } catch (IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public String getCommandName() {  
 return "logout";  
 }  
}

package command.impl;  
import command.Command;  
import command.CommandException;  
import command.PagePath;  
import command.ParameterName;  
import dbbinding.impl.CarBinding;  
import dbbinding.impl.OrderBinding;  
import entity.Car;  
import entity.Entity;  
import entity.Order;  
import repository.RepositoryException;  
import repository.impl.SqlRepository;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
public class RemoveCarCommand implements Command {  
  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 int car\_id = Integer.*parseInt*(request.getParameter(ParameterName.*CAR\_ID*));  
  
 try {  
 SqlRepository<Car> repository = new SqlRepository<Car>(new CarBinding());  
 Car car = repository.getEntityById(car\_id);  
 repository.removeEntity(car);  
  
 response.sendRedirect(PagePath.*SERVLET* + "/car\_rental?command=to\_cars\_list");  
 } catch (RepositoryException | IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public String getCommandName() {  
 return "remove\_car";  
 }  
}

package command.impl;  
  
import command.Command;  
import command.CommandException;  
import command.PagePath;  
import command.ParameterName;  
import dbbinding.impl.OrderBinding;  
import entity.Entity;  
import entity.Order;  
import repository.RepositoryException;  
import repository.impl.SqlRepository;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
public class RemoveOrderCommand implements Command {  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 int order\_id = Integer.*parseInt*(request.getParameter(ParameterName.*ORDER\_ID*));  
  
 try {  
 SqlRepository<Order> repository = new SqlRepository<Order>(new OrderBinding());  
 Order order = repository.getEntityById(order\_id);  
 repository.removeEntity(order);  
  
 response.sendRedirect(PagePath.*SERVLET* + "/car\_rental?command=to\_orders\_list");  
 } catch (RepositoryException | IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public String getCommandName() {  
 return "remove\_order";  
 }  
}

package command.impl;  
  
import command.Command;  
import command.CommandException;  
import command.PagePath;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
public class ToAddCarCommand implements Command {  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 try {  
 request.getRequestDispatcher(PagePath.*ADD\_CAR*).forward(request, response);  
 } catch (ServletException | IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public String getCommandName() {  
 return "to\_add\_car";  
 }  
}

package command.impl;  
  
import command.Command;  
import command.CommandException;  
import command.PagePath;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
public class ToAddClientCommand implements Command {  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 try {  
 request.getRequestDispatcher(PagePath.*ADD\_CLIENT*).forward(request, response);  
 } catch (ServletException | IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public String getCommandName() {  
 return "to\_add\_client";  
 }  
}

package command.impl;  
  
import command.Command;  
import command.CommandException;  
import command.PagePath;  
import command.ParameterName;  
import dbbinding.impl.CarBinding;  
import entity.Car;  
import repository.RepositoryException;  
import repository.impl.SqlRepository;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
public class ToAddOrderCommand implements Command {  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 try {  
 request.setAttribute(ParameterName.*CLIENT\_ID*, request.getParameter(ParameterName.*CLIENT\_ID*));  
 request.setAttribute(ParameterName.*CARS\_LIST*,  
 new SqlRepository<Car>(new CarBinding()).getAllEntities());  
 request.getRequestDispatcher(PagePath.*ADD\_ORDER*).forward(request, response);  
 } catch (ServletException | IOException | RepositoryException e) {  
 throw new CommandException(e);  
 }  
  
 }  
  
 @Override  
 public String getCommandName() {  
 return "to\_add\_order";  
 }  
}

package command.impl;  
  
import command.Command;  
import command.CommandException;  
import command.PagePath;  
import command.ParameterName;  
import dbbinding.impl.CarBinding;  
import dbbinding.impl.ClientBinding;  
import entity.Car;  
import entity.Client;  
import repository.RepositoryException;  
import repository.impl.SqlRepository;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
public class ToCarProfileCommand implements Command{  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 String stringId = request.getParameter(ParameterName.*CAR\_ID*);  
 int id = Integer.*parseInt*(stringId);  
  
 try {  
 Car car = new SqlRepository<>(new CarBinding()).getEntityById(id);  
 request.setAttribute(ParameterName.*CAR*, car);  
 request.getRequestDispatcher(PagePath.*CAR\_PROFILE*).forward(request, response);  
 } catch (RepositoryException | ServletException | IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public String getCommandName() {  
 return "to\_car\_profile";  
 }  
}

package command.impl;  
  
import command.Command;  
import command.CommandException;  
import command.PagePath;  
import command.ParameterName;  
import dbbinding.impl.CarBinding;  
import dbbinding.impl.CarMakeBindingCommand;  
import entity.Car;  
import entity.CarMake;  
import repository.RepositoryException;  
import repository.impl.SqlRepository;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
public class ToCarsList implements Command {  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 try {  
 request.setAttribute(ParameterName.*CARS\_LIST*,  
 new SqlRepository<Car>(new CarBinding()).getAllEntities());  
 request.getRequestDispatcher(PagePath.*CARS\_LIST*).forward(request, response);  
 } catch (RepositoryException | IOException | ServletException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public String getCommandName() {  
 return "to\_cars\_list";  
 }  
}

package command.impl;  
  
import command.Command;  
import command.CommandException;  
import command.PagePath;  
import command.ParameterName;  
import dbbinding.impl.ClientBinding;  
import entity.Client;  
import repository.RepositoryException;  
import repository.impl.SqlRepository;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
public class ToClientProfileCommand implements Command {  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 String stringId = request.getParameter(ParameterName.*CLIENT\_ID*);  
 int id = Integer.*parseInt*(stringId);  
  
 try {  
 Client client = new SqlRepository<>(new ClientBinding()).getEntityById(id);  
 request.setAttribute(ParameterName.*CLIENT*, client);  
 request.getRequestDispatcher(PagePath.*CLIENT\_PROFILE*).forward(request, response);  
 } catch (RepositoryException | ServletException | IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public String getCommandName() {  
 return "to\_client\_profile";  
 }  
}

package command.impl;  
  
import command.Command;  
import command.CommandException;  
import command.PagePath;  
import command.ParameterName;  
import dbbinding.impl.ClientBinding;  
import entity.Client;  
import repository.RepositoryException;  
import repository.impl.SqlRepository;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
public class ToClientsList implements Command {  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 try {  
 request.setAttribute(ParameterName.*CLIENTS\_LIST*,  
 new SqlRepository<Client>(new ClientBinding()).getAllEntities());  
 request.getRequestDispatcher(PagePath.*CLIENTS\_LIST*).forward(request, response);  
 } catch (ServletException | IOException | RepositoryException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public String getCommandName() {  
 return "to\_clients\_list";  
 }  
}

package command.impl;  
  
import command.Command;  
import command.CommandException;  
import command.PagePath;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
public class ToLoginCommand implements Command {  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 try {  
 request.getRequestDispatcher(PagePath.*LOGIN*).forward(request, response);  
 } catch (ServletException | IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public String getCommandName() {  
 return "to\_login";  
 }  
}

package command.impl;  
  
import command.Command;  
import command.CommandException;  
import command.PagePath;  
import command.ParameterName;  
import dbbinding.impl.CarBinding;  
import dbbinding.impl.ClientBinding;  
import dbbinding.impl.OrderBinding;  
import entity.Car;  
import entity.Client;  
import entity.Order;  
import repository.RepositoryException;  
import repository.impl.SqlRepository;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
public class ToOrdersList implements Command {  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 try {  
 request.setAttribute(ParameterName.*CLIENTS\_LIST*,  
 new SqlRepository<Client>(new ClientBinding()).getAllEntities());  
 request.setAttribute(ParameterName.*ORDERS\_LIST*,  
 new SqlRepository<Order>(new OrderBinding()).getAllEntities());  
 request.setAttribute(ParameterName.*CARS\_LIST*,  
 new SqlRepository<Car>(new CarBinding()).getAllEntities());  
 request.getRequestDispatcher(PagePath.*ORDERS\_LIST*).forward(request, response);  
 } catch (ServletException | IOException | RepositoryException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public String getCommandName() {  
 return "to\_orders\_list";  
 }  
}

package command.impl;  
  
import com.itextpdf.text.Document;  
import com.itextpdf.text.DocumentException;  
import com.itextpdf.text.pdf.PdfWriter;  
import command.Command;  
import command.CommandException;  
import command.ParameterName;  
import dbbinding.impl.OrderBinding;  
import entity.Order;  
import repository.RepositoryException;  
import repository.impl.SqlRepository;  
import service.OrderService;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
public class ToPrintCommand implements Command {  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 try {  
 int id = Integer.*parseInt*(request.getParameter(ParameterName.*ORDER\_ID*));  
 Order client = new SqlRepository<Order>(new OrderBinding()).getEntityById(id);  
  
 Document document = new Document();  
 response.setContentType("application/pdf");  
 response.setCharacterEncoding("UTF-8");  
 response.setHeader("Content-disposition", "attachment; filename=order.pdf");  
 PdfWriter.*getInstance*(document, response.getOutputStream());  
 OrderService.*INSTANCE*.printPdf(document, client);  
 } catch (RepositoryException | DocumentException | IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public String getCommandName() {  
 return "to\_print";  
 }  
}

package command.impl;  
import command.CommandException;  
import command.EntityBuilder;  
import command.PagePath;  
import dbbinding.impl.CarBinding;  
import entity.Car;  
import repository.RepositoryException;  
import repository.impl.SqlRepository;  
import validation.impl.CarValidator;  
  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.util.Optional;  
public class UpdateCarProfileCommand implements CommandWithValidation{  
  
 @Override  
 public void onSuccess(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 try {  
 String params = "/car\_rental?command=to\_cars\_list";  
 response.sendRedirect(PagePath.*SERVLET* + params);  
 } catch (IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public void onValidationError(HttpServletRequest request, HttpServletResponse response, String feedback) throws CommandException {  
 try {  
 request.setAttribute("feedback", feedback);  
 request.getRequestDispatcher(PagePath.*SERVLET* + "/car\_rental?command=to\_car\_profile").forward(request, response);  
 } catch (ServletException | IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 Optional<Car> optionalCar = EntityBuilder.*INSTANCE*.buildCar(request);  
 CarValidator validator = new CarValidator();  
  
 if (!optionalCar.isPresent()) {  
 try {  
 response.sendError(400);  
 } catch (IOException e) {  
 throw new CommandException(e);  
 }  
 return;  
 }  
  
 Car car = optionalCar.get();  
  
 if (!validator.isValid(car)) {  
 onValidationError(request, response, validator.getValidationFeedback());  
 return;  
 }  
  
 int id = Integer.*parseInt*(request.getParameter("car\_id"));  
 car.setId(id);  
  
 try {  
 new SqlRepository<>(new CarBinding()).updateEntity(car);  
 } catch (RepositoryException e) {  
 onValidationError(request, response, "Ошибка обновления клиента");  
 }  
  
 onSuccess(request, response);  
 }  
  
 @Override  
 public String getCommandName() {  
 return "update\_car";  
 }  
}

package command.impl;  
  
import command.CommandException;  
import command.EntityBuilder;  
import command.PagePath;  
import dbbinding.impl.ClientBinding;  
import entity.Client;  
import repository.RepositoryException;  
import repository.impl.SqlRepository;  
import validation.impl.ClientValidator;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.util.Optional;  
  
public class UpdateClientProfileCommand implements CommandWithValidation {  
 @Override  
 public void onSuccess(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 try {  
 String params = "/car\_rental?command=to\_clients\_list";  
 response.sendRedirect(PagePath.*SERVLET* + params);  
 } catch (IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public void onValidationError(HttpServletRequest request, HttpServletResponse response, String feedback) throws CommandException {  
 try {  
 request.setAttribute("feedback", feedback);  
 request.getRequestDispatcher(PagePath.*SERVLET* + "/car\_rental?command=to\_client\_profile").forward(request, response);  
 } catch (ServletException | IOException e) {  
 throw new CommandException(e);  
 }  
 }  
  
 @Override  
 public void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException {  
 Optional<Client> optionalClient = EntityBuilder.*INSTANCE*.buildClient(request);  
 ClientValidator validator = new ClientValidator();  
  
 if (!optionalClient.isPresent()) {  
 try {  
 response.sendError(400);  
 } catch (IOException e) {  
 throw new CommandException(e);  
 }  
 return;  
 }  
  
 Client client = optionalClient.get();  
  
 if (!validator.isValid(client)) {  
 onValidationError(request, response, validator.getValidationFeedback());  
 return;  
 }  
  
 int id = Integer.*parseInt*(request.getParameter("client\_id"));  
 client.setId(id);  
  
 try {  
 new SqlRepository<>(new ClientBinding()).updateEntity(client);  
 } catch (RepositoryException e) {  
 onValidationError(request, response, "Ошибка обновления клиента");  
 }  
  
 onSuccess(request, response);  
 }  
  
 @Override  
 public String getCommandName() {  
 return "update\_client";  
 }  
}

package command;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
  
public interface Command {  
 void execute(HttpServletRequest request, HttpServletResponse response) throws CommandException;  
 String getCommandName();  
}

package command;  
  
public class CommandException extends Exception {  
 public CommandException() {  
 }  
  
 public CommandException(String message) {  
 super(message);  
 }  
  
 public CommandException(String message, Throwable cause) {  
 super(message, cause);  
 }  
  
 public CommandException(Throwable cause) {  
 super(cause);  
 }  
}

package command;  
  
import command.impl.\*;  
import command.impl.AddOrderCommand;  
import command.impl.decorator.RequiredLoginDecorator;  
  
import java.util.List;  
import java.util.Optional;  
  
public enum CommandProvider {  
 *INSTANCE*;  
  
 private List<Command> commands;  
  
 CommandProvider() {  
 commands = List.*of*(new ToLoginCommand(), new LoginCommand(), new LogoutCommand(),  
 new RequiredLoginDecorator(new ToClientsList()),  
 new RequiredLoginDecorator(new ToAddClientCommand()),  
 new RequiredLoginDecorator(new AddClientCommand()),  
 new RequiredLoginDecorator(new ToClientProfileCommand()),  
 new RequiredLoginDecorator(new UpdateClientProfileCommand()),  
  
 new RequiredLoginDecorator(new ToAddOrderCommand()),  
 new RequiredLoginDecorator(new AddOrderCommand()),  
 new RequiredLoginDecorator(new ToOrdersList()),  
 new RequiredLoginDecorator(new RemoveOrderCommand()),  
  
 new RequiredLoginDecorator(new ToPrintCommand()),  
  
 new RequiredLoginDecorator(new ToCarsList()),  
 new RequiredLoginDecorator(new ToAddCarCommand()),  
 new RequiredLoginDecorator(new AddCarCommand()),  
 new RequiredLoginDecorator(new ToCarProfileCommand()),  
 new RequiredLoginDecorator(new UpdateCarProfileCommand()),  
 new RequiredLoginDecorator(new RemoveCarCommand())  
 );  
 }  
  
 public Optional<Command> findCommand(String commandName) {  
 return commands.stream().filter(x -> x.getCommandName().equals(commandName)).findFirst();  
 }  
}

package command;  
  
import entity.\*;  
  
import javax.servlet.http.HttpServletRequest;  
import java.time.LocalDate;  
import java.time.format.DateTimeParseException;  
import java.util.Optional;  
  
public enum EntityBuilder {  
 *INSTANCE*;  
  
 public Optional<Employee> buildEmployee(HttpServletRequest request) {  
 Employee employee = new Employee();  
  
 employee.setLogin(request.getParameter(ParameterName.*LOGIN*));  
 employee.setPassword(request.getParameter(ParameterName.*PASSWORD*));  
  
 return Optional.*of*(employee);  
 }  
  
 public Optional<Client> buildClient(HttpServletRequest request) {  
 Client client = new Client();  
  
 try {  
 client.setName(request.getParameter(ParameterName.*NAME*));  
 client.setSurname(request.getParameter(ParameterName.*SURNAME*));  
 client.setPatronymic(request.getParameter(ParameterName.*PATRONYMIC*));  
 client.setPassportId(request.getParameter(ParameterName.*PASSPORT\_ID*));  
 client.setHomePhone(request.getParameter(ParameterName.*HOME\_PHONE*));  
 client.setMobilePhone(request.getParameter(ParameterName.*MOBILE\_PHONE*));  
 client.setEmail(request.getParameter(ParameterName.*EMAIL*));  
  
 String discount\_card = request.getParameter(ParameterName.*DISCOUNT\_CARD\_ID*);  
  
 if (discount\_card != null && !discount\_card.isEmpty()) {  
 client.setDiscountCardId(Integer.*parseInt*(discount\_card));  
 }  
 } catch (NumberFormatException e) {  
 return Optional.*empty*();  
 }  
  
 return Optional.*of*(client);  
 }  
  
 public Optional<Order> buildOrder(HttpServletRequest request) {  
 Order order = new Order();  
  
 try {  
 order.setClientId(Integer.*parseInt*(request.getParameter(ParameterName.*CLIENT\_ID*)));  
 order.setCarId(Integer.*parseInt*(request.getParameter(ParameterName.*CAR\_ID*)));  
 order.setStartDate(LocalDate.*parse*(request.getParameter(ParameterName.*START\_DATE*)));  
 order.setEndDate(LocalDate.*parse*(request.getParameter(ParameterName.*END\_DATE*)));  
 } catch (IllegalArgumentException | DateTimeParseException e) {  
 return Optional.*empty*();  
 }  
  
 return Optional.*of*(order);  
 }  
  
 public Optional<Car> buildCar(HttpServletRequest request) {  
 Car car = new Car();  
 try {  
 car.setYear(Integer.*parseInt*(request.getParameter(ParameterName.*YEAR*)));  
 car.setMakeId(Integer.*parseInt*(request.getParameter(ParameterName.*MAKE\_ID*)));  
 car.setModel(request.getParameter(ParameterName.*MODEL*));  
 car.setRegistrationNumber(request.getParameter(ParameterName.*REGISTRATION\_NUMBER*));  
 car.setMileage(Integer.*parseInt*(request.getParameter(ParameterName.*MILEAGE*)));  
 car.setGearboxType(Integer.*parseInt*(request.getParameter(ParameterName.*GEARBOX\_TYPE*)));  
 car.setPricePerDay(Float.*parseFloat*(request.getParameter(ParameterName.*PRICE\_PER\_DAY*)));  
 car.setPricePerDay2(Float.*parseFloat*(request.getParameter(ParameterName.*PRICE\_PER\_DAY\_2*)));  
 } catch (NumberFormatException e) {  
 return Optional.*empty*();  
 }  
  
 return Optional.*of*(car);  
 }  
  
}

package command;  
  
public class PagePath {  
 public final static String *SERVLET* = "/car\_rental";  
  
 public final static String *LOGIN* = "/login.jsp";  
 public static final String *CLIENTS\_LIST* = "/clients\_list.jsp";  
 public static final String *ADD\_CLIENT* = "/add\_client.jsp";  
 public static final String *CARS\_LIST* = "/cars\_list.jsp";  
 public static final String *ADD\_CAR* = "/add\_car.jsp";  
 public static final String *CLIENT\_PROFILE* = "/client\_profile.jsp";  
 public static final String *ORDERS\_LIST* = "/orders\_list.jsp";  
 public static final String *ADD\_ORDER* = "/add\_order.jsp";  
 public static final String *CAR\_PROFILE* = "/car\_profile.jsp";  
}

package command;  
  
public class ParameterName {  
 public final static String *COMMAND* = "command";  
  
 public static final String *CLIENTS\_LIST* = "clients";  
 public static final String *ORDERS\_LIST* = "orders";  
 public static final String *CARS\_LIST* = "cars";  
 public final static String *CLIENT* = "client";  
 public final static String *CLIENT\_ID* = "client\_id";  
 public final static String *ORDER\_ID* = "order\_id";  
  
  
 public final static String *LOGIN* = "login";  
 public final static String *PASSWORD* = "password";  
  
 public final static String *NAME* = "name";  
 public final static String *SURNAME* = "surname";  
 public final static String *PATRONYMIC* = "patronymic";  
 public final static String *PASSPORT\_ID* = "passport\_id";  
 public final static String *HOME\_PHONE* = "home\_phone";  
 public final static String *MOBILE\_PHONE* = "mobile\_phone";  
 public final static String *EMAIL* = "email";  
 public static final String *DISCOUNT\_CARD\_ID* = "discount\_card\_id";  
  
 public final static String *CAR\_ID* = "car\_id";  
 public final static String *START\_DATE* = "start\_date";  
 public final static String *END\_DATE* = "end\_date";  
  
  
 public static final String *YEAR* = "year";  
 public static final String *MAKE\_ID* = "make\_id";  
 public static final String *MODEL* = "model";  
 public static final String *REGISTRATION\_NUMBER* = "registration\_number";  
 public static final String *MILEAGE* = "mileage";  
 public static final String *GEARBOX\_TYPE* = "gearbox\_type";  
 public static final String *PRICE\_PER\_DAY* = "price\_per\_day";  
 public static final String *PRICE\_PER\_DAY\_2* = "price\_per\_day\_2";  
  
 public static final String *CAR* = "car";  
  
}

package controller.filter;  
  
import javax.servlet.\*;  
import javax.servlet.annotation.WebFilter;  
import javax.servlet.annotation.WebInitParam;  
import java.io.IOException;  
  
//аннотация для объявления фильтра для веб-приложения.  
@WebFilter(urlPatterns = {"/\*"},  
 initParams = {  
 @WebInitParam(name = "encoding", value = "utf-8", description = "Encoding Param")})  
  
public class EncodingFilter implements Filter {  
 private String code;  
  
 @Override  
 public void init(FilterConfig filterConfig) {  
 code = filterConfig.getInitParameter("encoding");  
 }  
  
 @Override  
 public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws ServletException, IOException {  
 String codeRequest = servletRequest.getCharacterEncoding();  
 if (code != null && !code.equalsIgnoreCase(codeRequest)) {  
 servletRequest.setCharacterEncoding(code);  
  
 servletResponse.setCharacterEncoding(code);  
 filterChain.doFilter(servletRequest, servletResponse);  
 }  
 }  
  
 @Override  
 public void destroy() {  
 code = null;  
 }  
}

package controller;  
  
import command.\*;  
  
import javax.servlet.ServletException;  
import javax.servlet.annotation.WebServlet;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.util.Optional;  
  
@WebServlet(PagePath.*SERVLET*)  
public class CarRentalServlet extends HttpServlet {  
  
 @Override //показывает, что метод предназначен для переопределения метода с той же сигнатурой  
 protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
 processRequest(req, resp);  
 }  
  
 @Override  
 protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
 processRequest(req, resp);  
 }  
  
 public void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException {  
 String commandName = request.getParameter(ParameterName.*COMMAND*);  
  
 try {  
 Optional<Command> optionalCommand = CommandProvider.*INSTANCE*.findCommand(commandName);  
 Command command = optionalCommand.orElseThrow(CommandException::new);  
  
 command.execute(request, response);  
 } catch (CommandException e) {  
 throw new ServletException(e);  
 }  
 }  
}

package dbbinding.impl;  
  
import dbbinding.DbAttributeName;  
import dbbinding.DbBindingException;  
import dbbinding.DbEntityBinding;  
import entity.Car;  
  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.List;  
  
public class CarBinding implements DbEntityBinding<Car> {  
 @Override  
 public String getTableName() {  
 return "cars";  
 }  
  
 @Override  
 public List<String> getColumns() {  
 return List.*of*(DbAttributeName.*YEAR*, DbAttributeName.*MAKE\_ID*, DbAttributeName.*MODEL*,  
 DbAttributeName.*REGISTRATION\_NUMBER*, DbAttributeName.*MILEAGE*, DbAttributeName.*GEARBOX\_TYPE*,  
 DbAttributeName.*PRICE\_PER\_DAY*, DbAttributeName.*PRICE\_PER\_DAY2*);  
 }  
  
 @Override  
 public Car build(ResultSet rs) throws SQLException, DbBindingException {  
 Car car = new Car();  
  
 car.setId(rs.getInt(DbAttributeName.*ID*));  
 car.setYear(rs.getInt(DbAttributeName.*YEAR*));  
 car.setMakeId(rs.getInt(DbAttributeName.*MAKE\_ID*));  
 car.setModel(rs.getString(DbAttributeName.*MODEL*));  
 car.setRegistrationNumber(rs.getString(DbAttributeName.*REGISTRATION\_NUMBER*));  
 car.setMileage(rs.getInt(DbAttributeName.*MILEAGE*));  
 car.setGearboxType(rs.getInt(DbAttributeName.*GEARBOX\_TYPE*));  
 car.setPricePerDay(rs.getInt(DbAttributeName.*PRICE\_PER\_DAY*));  
 car.setPricePerDay2(rs.getInt(DbAttributeName.*PRICE\_PER\_DAY2*));  
  
 return car;  
 }  
  
 @Override  
 public void bind(PreparedStatement statement, Car entity) throws SQLException, DbBindingException {  
 int index = 0;  
  
 statement.setInt(++index, entity.getYear());  
 statement.setInt(++index, entity.getMakeId());  
 statement.setString(++index, entity.getModel());  
 statement.setString(++index, entity.getRegistrationNumber());  
 statement.setInt(++index, entity.getMileage());  
 statement.setInt(++index, entity.getGearboxType());  
 statement.setFloat(++index, entity.getPricePerDay());  
 statement.setFloat(++index, entity.getPricePerDay2());  
  
 if (index != getColumns().size())  
 throw new DbBindingException("Неверное количество столбцов в таблице " + getTableName());  
 }  
  
 @Override  
 public String getIdentityColumn() {  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public void bindIdentityData(PreparedStatement statement, Car entity) throws SQLException {  
 throw new UnsupportedOperationException();  
 }  
}

package dbbinding.impl;  
  
import dbbinding.DbAttributeName;  
import dbbinding.DbBindingException;  
import dbbinding.DbEntityBinding;  
import entity.CarMake;  
  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.List;  
  
public class CarMakeBindingCommand implements DbEntityBinding<CarMake> {  
 @Override  
 public String getTableName() {  
 return "car\_makes";  
 }  
  
 @Override  
 public List<String> getColumns() {  
 return List.*of*(DbAttributeName.*CAR\_MAKE\_NAME*);  
 }  
  
 @Override  
 public CarMake build(ResultSet rs) throws SQLException, DbBindingException {  
 CarMake make = new CarMake();  
  
 make.setId(rs.getInt(DbAttributeName.*ID*));  
 make.setName(rs.getString(DbAttributeName.*CAR\_MAKE\_NAME*));  
  
 return make;  
 }  
  
 @Override  
 public void bind(PreparedStatement statement, CarMake entity) throws SQLException, DbBindingException {  
 statement.setString(1, entity.getName());  
 }  
  
 @Override  
 public String getIdentityColumn() {  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public void bindIdentityData(PreparedStatement statement, CarMake entity) throws SQLException {  
 throw new UnsupportedOperationException();  
 }  
}

package dbbinding.impl;  
  
import dbbinding.DbAttributeName;  
import dbbinding.DbBindingException;  
import dbbinding.DbEntityBinding;  
import entity.Client;  
  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.List;  
  
public class ClientBinding implements DbEntityBinding<Client> {  
 @Override  
 public String getTableName() {  
 return "clients";  
 }  
  
 @Override  
 public List<String> getColumns() {  
 return List.*of*(DbAttributeName.*NAME*, DbAttributeName.*SURNAME*, DbAttributeName.*PATRONYMIC*,  
 DbAttributeName.*PASSPORT\_ID*, DbAttributeName.*HOME\_PHONE*, DbAttributeName.*MOBILE\_PHONE*,  
 DbAttributeName.*EMAIL*, DbAttributeName.*DISCOUNT\_CARD\_ID*);  
 }  
  
 @Override  
 public Client build(ResultSet rs) throws SQLException, DbBindingException {  
 Client client = new Client();  
  
 client.setId(rs.getInt(DbAttributeName.*ID*));  
 client.setName(rs.getString(DbAttributeName.*NAME*));  
 client.setSurname(rs.getString(DbAttributeName.*SURNAME*));  
 client.setPatronymic(rs.getString(DbAttributeName.*PATRONYMIC*));  
 client.setPassportId(rs.getString(DbAttributeName.*PASSPORT\_ID*));  
 client.setHomePhone(rs.getString(DbAttributeName.*HOME\_PHONE*));  
 client.setMobilePhone(rs.getString(DbAttributeName.*MOBILE\_PHONE*));  
 client.setEmail(rs.getString(DbAttributeName.*EMAIL*));  
 client.setDiscountCardId(rs.getInt(DbAttributeName.*DISCOUNT\_CARD\_ID*));  
  
 return client;  
 }  
  
 @Override  
 public void bind(PreparedStatement statement, Client entity) throws SQLException, DbBindingException {  
 int index = 0;  
  
 statement.setString(++index, entity.getName());  
 statement.setString(++index, entity.getSurname());  
 statement.setString(++index, entity.getPatronymic());  
 statement.setString(++index, entity.getPassportId());  
 statement.setString(++index, entity.getHomePhone());  
 statement.setString(++index, entity.getMobilePhone());  
 statement.setString(++index, entity.getEmail());  
 statement.setInt(++index, entity.getDiscountCardId());  
  
 if (index != getColumns().size())  
 throw new DbBindingException("Неверное количество столбцов в таблице " + getTableName());  
 }  
  
 @Override  
 public String getIdentityColumn() {  
 return DbAttributeName.*PASSPORT\_ID*;  
 }  
  
 @Override  
 public void bindIdentityData(PreparedStatement statement, Client entity) throws SQLException {  
 statement.setString(1, entity.getPassportId());  
 }  
}

package dbbinding.impl;  
  
import dbbinding.DbAttributeName;  
import dbbinding.DbBindingException;  
import dbbinding.DbEntityBinding;  
import entity.Employee;  
  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.List;  
  
public class EmployeeBinding implements DbEntityBinding<Employee> {  
 @Override  
 public String getTableName() {  
 return "employees";  
 }  
  
 @Override  
 public List<String> getColumns() {  
 return List.*of*(DbAttributeName.*LOGIN*, DbAttributeName.*PASSWORD*);  
 }  
  
 @Override  
 public Employee build(ResultSet rs) throws SQLException, DbBindingException {  
 Employee employee = new Employee();  
  
 employee.setId(rs.getInt(DbAttributeName.*ID*));  
 employee.setLogin(rs.getString(DbAttributeName.*LOGIN*));  
 employee.setPassword(rs.getString(DbAttributeName.*PASSWORD*));  
  
 return employee;  
 }  
  
 @Override  
 public void bind(PreparedStatement statement, Employee entity) throws SQLException, DbBindingException {  
 int index = 0;  
  
 statement.setString(++index, entity.getLogin());  
 statement.setString(++index, entity.getPassword());  
  
 if (index != getColumns().size())  
 throw new DbBindingException("Неверное количество столбцов в таблице " + getTableName());  
 }  
  
 @Override  
 public String getIdentityColumn() {  
 return DbAttributeName.*LOGIN*;  
 }  
  
 @Override  
 public void bindIdentityData(PreparedStatement statement, Employee entity) throws SQLException {  
 statement.setString(1, entity.getLogin());  
 }  
}

package dbbinding.impl;  
  
import dbbinding.DbAttributeName;  
import dbbinding.DbBindingException;  
import dbbinding.DbEntityBinding;  
import entity.Order;  
  
import java.sql.Date;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.List;  
  
public class OrderBinding implements DbEntityBinding<Order> {  
 @Override  
 public String getTableName() {  
 return "orders";  
 }  
  
 @Override  
 public List<String> getColumns() {  
 return List.*of*(DbAttributeName.*CLIENT\_ID*, DbAttributeName.*CAR\_ID*,  
 DbAttributeName.*START\_DATE*, DbAttributeName.*END\_DATE*);  
 }  
  
 @Override  
 public Order build(ResultSet rs) throws SQLException, DbBindingException {  
 Order order = new Order();  
  
 order.setId(rs.getInt(DbAttributeName.*ID*));  
 order.setClientId(rs.getInt(DbAttributeName.*CLIENT\_ID*));  
 order.setCarId(rs.getInt(DbAttributeName.*CAR\_ID*));  
 order.setStartDate(rs.getDate(DbAttributeName.*START\_DATE*).toLocalDate());  
 order.setEndDate(rs.getDate(DbAttributeName.*END\_DATE*).toLocalDate());  
  
 return order;  
 }  
  
 @Override  
 public void bind(PreparedStatement statement, Order entity) throws SQLException, DbBindingException {  
 int index = 0;  
  
 statement.setInt(++index, entity.getClientId());  
 statement.setInt(++index, entity.getCarId());  
 statement.setDate(++index, Date.*valueOf*(entity.getStartDate()));  
 statement.setDate(++index, Date.*valueOf*(entity.getEndDate()));  
  
 if (index != getColumns().size())  
 throw new DbBindingException("Неверное количество столбцов в таблице " + getTableName());  
 }  
  
 @Override  
 public String getIdentityColumn() {  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public void bindIdentityData(PreparedStatement statement, Order entity) throws SQLException {  
 throw new UnsupportedOperationException();  
 }  
}

package dbbinding;  
  
public class DbAttributeName {  
 public final static String *ID* = "id";  
  
 public final static String *NAME* = "name";  
 public final static String *SURNAME* = "surname";  
 public final static String *PATRONYMIC* = "patronymic";  
 public final static String *PASSPORT\_ID* = "passport\_id";  
 public final static String *HOME\_PHONE* = "home\_phone";  
 public final static String *MOBILE\_PHONE* = "mobile\_phone";  
 public final static String *EMAIL* = "email";  
 public final static String *DISCOUNT\_CARD\_ID* = "discount\_card\_id";  
  
 public final static String *CLIENT\_ID* = "client\_id";  
 public final static String *CAR\_ID* = "car\_id";  
 public final static String *START\_DATE* = "start\_date";  
 public final static String *END\_DATE* = "end\_date";  
  
 public final static String *LOGIN* = "login";  
 public final static String *PASSWORD* = "password";  
  
 public final static String *YEAR* = "year";  
 public final static String *MAKE\_ID* = "make\_id";  
 public final static String *MODEL* = "model";  
 public final static String *REGISTRATION\_NUMBER* = "registration\_number";  
 public final static String *MILEAGE* = "mileage";  
 public final static String *GEARBOX\_TYPE* = "gearbox\_type";  
 public final static String *PRICE\_PER\_DAY* = "price\_per\_day";  
 public final static String *PRICE\_PER\_DAY2* = "price\_per\_day\_2";  
  
 public final static String *CAR\_MAKE\_NAME* = "name";  
}

package dbbinding;  
  
public class DbBindingException extends Exception{  
 public DbBindingException() {  
 }  
  
 public DbBindingException(String message) {  
 super(message);  
 }  
  
 public DbBindingException(String message, Throwable cause) {  
 super(message, cause);  
 }  
  
 public DbBindingException(Throwable cause) {  
 super(cause);  
 }  
}

package dbbinding;  
  
import entity.Entity;  
  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.List;  
  
public interface DbEntityBinding<T extends Entity> {  
 String getTableName();  
 List<String> getColumns();  
  
 T build(ResultSet rs) throws SQLException, DbBindingException;  
 void bind(PreparedStatement statement, T entity) throws SQLException, DbBindingException;  
  
 String getIdentityColumn();  
 void bindIdentityData(PreparedStatement statement, T entity) throws SQLException;  
}

package entity;  
  
public class Car extends Entity {  
 private int year;  
 private int makeId;  
 private String model;  
 private String registrationNumber;  
 private int mileage;  
 private int gearboxType;  
 private float pricePerDay;  
 private float PricePerDay2;  
  
 public int getYear() {  
 return year;  
 }  
  
 public void setYear(int year) {  
 this.year = year;  
 }  
  
 public int getMakeId() {  
 return makeId;  
 }  
  
 public void setMakeId(int makeId) {  
 this.makeId = makeId;  
 }  
  
 public String getModel() {  
 return model;  
 }  
  
 public void setModel(String model) {  
 this.model = model;  
 }  
  
 public String getRegistrationNumber() {  
 return registrationNumber;  
 }  
  
 public void setRegistrationNumber(String registrationNumber) {  
 this.registrationNumber = registrationNumber;  
 }  
  
 public int getMileage() {  
 return mileage;  
 }  
  
 public void setMileage(int mileage) {  
 this.mileage = mileage;  
 }  
  
 public int getGearboxType() {  
 return gearboxType;  
 }  
  
 public void setGearboxType(int gearboxType) {  
 this.gearboxType = gearboxType;  
 }  
  
 public float getPricePerDay() {  
 return pricePerDay;  
 }  
  
 public void setPricePerDay(float pricePerDay) {  
 this.pricePerDay = pricePerDay;  
 }  
  
 public float getPricePerDay2() {  
 return PricePerDay2;  
 }  
  
 public void setPricePerDay2(float pricePerDay2) {  
 PricePerDay2 = pricePerDay2;  
 }  
}

package entity;  
  
public class CarMake extends Entity {  
 private String name;  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

package entity;  
  
public class Client extends Entity {  
 private String name;  
 private String surname;  
 private String patronymic;  
 private String passportId;  
 private String homePhone;  
 private String mobilePhone;  
 private String email;  
 private int discountCardId;  
  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getSurname() {  
 return surname;  
 }  
  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
  
 public String getPatronymic() {  
 return patronymic;  
 }  
  
 public void setPatronymic(String patronymic) {  
 this.patronymic = patronymic;  
 }  
  
 public String getPassportId() {  
 return passportId;  
 }  
  
 public void setPassportId(String passportId) {  
 this.passportId = passportId;  
 }  
  
 public String getHomePhone() {  
 return homePhone;  
 }  
  
 public void setHomePhone(String homePhone) {  
 this.homePhone = homePhone;  
 }  
  
 public String getMobilePhone() {  
 return mobilePhone;  
 }  
  
 public void setMobilePhone(String mobilePhone) {  
 this.mobilePhone = mobilePhone;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 public int getDiscountCardId() {  
 return discountCardId;  
 }  
  
 public void setDiscountCardId(int discountCardId) {  
 this.discountCardId = discountCardId;  
 }  
}

package entity;  
  
public class Employee extends Entity {  
 private String login;  
 private String password;  
  
 public String getLogin() {  
 return login;  
 }  
  
 public void setLogin(String login) {  
 this.login = login;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
}

package entity;  
  
public abstract class Entity {  
 private int id;  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
}

package entity;  
  
import java.time.LocalDate;  
  
public class Order extends Entity {  
 private int clientId;  
 private int carId;  
 private LocalDate startDate;  
 private LocalDate endDate;  
  
 public int getClientId() {  
 return clientId;  
 }  
  
 public void setClientId(int clientId) {  
 this.clientId = clientId;  
 }  
  
 public int getCarId() {  
 return carId;  
 }  
  
 public void setCarId(int carId) {  
 this.carId = carId;  
 }  
  
 public LocalDate getStartDate() {  
 return startDate;  
 }  
  
 public void setStartDate(LocalDate startDate) {  
 this.startDate = startDate;  
 }  
  
 public LocalDate getEndDate() {  
 return endDate;  
 }  
  
 public void setEndDate(LocalDate endDate) {  
 this.endDate = endDate;  
 }  
}

package repository.impl;  
  
import dbbinding.DbBindingException;  
import dbbinding.DbEntityBinding;  
import entity.Entity;  
import repository.ConnectionManager;  
import repository.Repository;  
import repository.RepositoryException;  
  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.ArrayList;  
import java.util.List;  
  
public class SqlRepository<T extends Entity> implements Repository<T> {  
 private Connection connection;  
 private DbEntityBinding<T> binding;  
  
 public SqlRepository(DbEntityBinding<T> binding) {  
 this.binding = binding;  
 connection = ConnectionManager.*INSTANCE*.getConnection();  
 }  
  
 @Override  
 public int getEntityId(T entity) throws RepositoryException {  
 try {  
 PreparedStatement statement = connection.prepareStatement(getEntityIdQuery());  
 binding.bindIdentityData(statement, entity);  
  
 ResultSet rs = statement.executeQuery();  
  
 if (rs.next()) {  
 return rs.getInt("id");  
 } else {  
 throw new RepositoryException();  
 }  
 } catch (SQLException e) {  
 throw new RepositoryException(e);  
 }  
 }  
  
 @Override  
 public T getEntityById(int id) throws RepositoryException {  
 try {  
 PreparedStatement statement = connection.prepareStatement(getGetEntityQuery());  
 statement.setInt(1, id);  
 ResultSet rs = statement.executeQuery();  
  
 if (rs.next()) {  
 return binding.build(rs);  
 } else {  
 throw new RepositoryException();  
 }  
 } catch (SQLException | DbBindingException e) {  
 throw new RepositoryException();  
 }  
 }  
  
 @Override  
 public List<T> getAllEntities() throws RepositoryException {  
 List<T> entities = new ArrayList<>();  
  
 try {  
 PreparedStatement statement = connection.prepareStatement(getGetAllEntitiesQuery());  
 ResultSet rs = statement.executeQuery();  
  
 while (rs.next()) {  
 entities.add(binding.build(rs));  
 }  
 } catch (SQLException | DbBindingException e) {  
 throw new RepositoryException(e);  
 }  
  
 return entities;  
 }  
  
 @Override  
 public void addEntity(T entity) throws RepositoryException {  
 try {  
 PreparedStatement statement = connection.prepareStatement(getAddEntityQuery());  
 binding.bind(statement, entity);  
 statement.execute();  
 } catch (SQLException | DbBindingException e) {  
 throw new RepositoryException(e);  
 }  
 }  
  
 @Override  
 public void updateEntity(T entity) throws RepositoryException {  
 try {  
 PreparedStatement statement = connection.prepareStatement(getUpdateEntityQuery());  
 binding.bind(statement, entity);  
 statement.setInt(binding.getColumns().size() + 1, entity.getId());  
 statement.execute();  
 } catch (SQLException | DbBindingException e) {  
 throw new RepositoryException(e);  
 }  
 }  
  
 @Override  
 public void removeEntity(T entity) throws RepositoryException {  
 try {  
 PreparedStatement statement = connection.prepareStatement(getRemoveEntityQuery());  
 statement.setInt(1, entity.getId());  
 statement.execute();  
 } catch (SQLException e) {  
 throw new RepositoryException(e);  
 }  
 }  
  
 private String getEntityIdQuery() {  
 return "SELECT id FROM " + binding.getTableName() + " WHERE " + binding.getIdentityColumn() + "=?";  
 }  
  
 private String getGetEntityQuery() {  
 return "SELECT \* FROM " + binding.getTableName() + " WHERE id=?";  
 }  
  
 private String getGetAllEntitiesQuery() {  
 return "SELECT \* FROM " + binding.getTableName() + " ORDER BY id ASC";  
 }  
  
 private String getRemoveEntityQuery() {  
 return "DELETE FROM " + binding.getTableName() + " WHERE id=?";  
 }  
  
 private String getAddEntityQuery() {  
 StringBuilder builder = new StringBuilder("INSERT INTO ")  
 .append(binding.getTableName())  
 .append(" VALUES(NULL, ");  
  
 for (int i = 0; i < binding.getColumns().size(); i++) {  
 if (i == binding.getColumns().size() - 1) {  
 builder.append("?)");  
 } else {  
 builder.append("?, ");  
 }  
 }  
  
 return builder.toString();  
 }  
  
 private String getUpdateEntityQuery() {  
 StringBuilder builder = new StringBuilder("UPDATE ")  
 .append(binding.getTableName())  
 .append(" SET ");  
  
 for (int i = 0; i < binding.getColumns().size(); i++) {  
 builder.append(binding.getColumns().get(i)).append("=?");  
  
 if (i != binding.getColumns().size() - 1) {  
 builder.append(", ");  
 }  
 }  
  
 builder.append(" WHERE id=?");  
  
 return builder.toString();  
 }  
}

package repository;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
  
public enum ConnectionManager {  
 *INSTANCE*;  
  
 private Connection connection;  
  
 private final static String *DATABASE\_SERVICE* = "localhost";  
 private final static String *DATABASE* = "car\_rental";  
  
// private final static String USER = "student";  
// private final static String PASSWORD = "student";  
 private final static String *USER* = "root";  
 //private final static String PASSWORD = "root";  
  
 private final static String *PASSWORD* = "new\_password";  
  
 ConnectionManager()  
 {  
 try {  
 java.sql.DriverManager.*registerDriver*(new com.mysql.cj.jdbc.Driver());  
 String url = "jdbc:mysql://"+ *DATABASE\_SERVICE* + ":3306/" + *DATABASE* + "?serverTimezone=Europe/Minsk";  
  
 connection = DriverManager.*getConnection*(url, *USER*, *PASSWORD*);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 //соединение с базой данных для всех классов  
 public Connection getConnection()  
 {  
 return connection;  
 }  
}

package repository;  
  
import java.util.List;  
  
public interface Repository<T> {  
 int getEntityId(T Entity) throws RepositoryException;  
 T getEntityById(int id) throws RepositoryException;  
 List<T> getAllEntities() throws RepositoryException;  
 void addEntity(T entity) throws RepositoryException;  
 void updateEntity(T entity) throws RepositoryException;  
 void removeEntity(T entity) throws RepositoryException;  
}

package repository;  
  
public class RepositoryException extends Exception{  
 public RepositoryException() {  
 }  
  
 public RepositoryException(String message) {  
 super(message);  
 }  
  
 public RepositoryException(String message, Throwable cause) {  
 super(message, cause);  
 }  
  
 public RepositoryException(Throwable cause) {  
 super(cause);  
 }  
}

package service;  
  
import dbbinding.impl.EmployeeBinding;  
import entity.Employee;  
import repository.RepositoryException;  
import repository.impl.SqlRepository;  
  
import java.util.Objects;  
import java.util.Optional;  
  
public enum EmployeeService {  
 *INSTANCE*;  
  
 public boolean checkCredentials(Employee employee)  
 {  
 SqlRepository<Employee> repository = new SqlRepository<Employee>(new EmployeeBinding());  
  
 try {  
 int employeeId = repository.getEntityId(employee);  
 String password = repository.getEntityById(employeeId).getPassword();  
  
 return Objects.*equals*(password, employee.getPassword());  
 } catch (RepositoryException e) {  
 return false;  
 }  
 }  
  
 public Optional<Employee> getByLogin(String login)  
 {  
 Employee employee = new Employee();  
 employee.setLogin(login);  
  
 try {  
 SqlRepository<Employee> repository = new SqlRepository<Employee>(new EmployeeBinding());  
 int employeeId = repository.getEntityId(employee);  
 return Optional.*of*(repository.getEntityById(employeeId));  
 } catch (RepositoryException e) {  
 e.printStackTrace();  
 }  
  
 return Optional.*empty*();  
 }  
}

package service;  
  
import com.ibm.icu.text.Transliterator;  
import com.itextpdf.text.Document;  
import com.itextpdf.text.DocumentException;  
import com.itextpdf.text.Font;  
import com.itextpdf.text.Paragraph;  
import com.itextpdf.text.pdf.BaseFont;  
import dbbinding.impl.CarBinding;  
import dbbinding.impl.ClientBinding;  
import entity.Car;  
import entity.Client;  
import entity.Order;  
import repository.RepositoryException;  
import repository.impl.SqlRepository;  
  
import java.io.IOException;  
  
public enum OrderService {  
 *INSTANCE*;  
  
 public void printPdf(Document document, Order order) throws IOException, DocumentException, RepositoryException {  
 Client client = new SqlRepository<Client>(new ClientBinding()).getEntityById(order.getClientId());  
 Car car = new SqlRepository<Car>(new CarBinding()).getEntityById(order.getCarId());  
 BaseFont bf=BaseFont.*createFont*(BaseFont.*TIMES\_ROMAN*, "UTF-8", BaseFont.*EMBEDDED*);  
 Transliterator trans = Transliterator.*getInstance*("Cyrillic-Latin");  
 Font font=new Font(bf,16, Font.*NORMAL*);  
 document.open();  
 document.add(new Paragraph("Order", font));  
 document.add(new Paragraph("Name:" + trans.transliterate(client.getName()), font));  
 document.add(new Paragraph("Surname:" + trans.transliterate(client.getSurname()), font));  
 document.add(new Paragraph("Passport:" + trans.transliterate(client.getPassportId()), font));  
 document.add(new Paragraph("Model:" + trans.transliterate(car.getModel()), font));  
 document.add(new Paragraph("Start date:" + order.getStartDate().toString(), font));  
 document.add(new Paragraph("End date:" + order.getEndDate().toString(), font));  
  
 document.close();  
 }  
}

package validation.impl;  
  
import entity.Car;  
import validation.RegExProvider;  
import validation.Validator;  
  
public class CarValidator extends Validator<Car> {  
 private final static int *MIN\_SHORT\_TEXT\_LENGTH* = 2;  
 private final static int *MAX\_SHORT\_TEXT\_LENGTH* = 50;  
 private final static int *MIN\_REG\_NUMBER\_LENGTH* = 6;  
 private final static int *MAX\_REG\_NUMBER\_LENGTH* = 10;  
 private final static int *MIN\_MILEAGE* = 0;  
 private final static int *MAX\_MILEAGE* = 999999;  
 private final static float *MIN\_PRICE* = 0f;  
 private final static float *MAX\_PRICE* = 9999.99f;  
  
 @Override  
 public boolean isValid(Car entity) {  
  
 String model = entity.getModel();  
 String registrationNumber = entity.getRegistrationNumber();  
 int year = entity.getYear();  
 int makeId = entity.getMakeId();  
 int mileage = entity.getMileage();  
 int gearboxType = entity.getGearboxType();  
 float pricePerDay = entity.getPricePerDay();  
 float pricePerDay2 = entity.getPricePerDay2();  
  
 if (!validateString(model, *MIN\_SHORT\_TEXT\_LENGTH*, *MAX\_SHORT\_TEXT\_LENGTH*, RegExProvider.*LETTERS*)) {  
 setValidationFeedback("Неверный формат модели автомобиля");  
 return false;  
 }  
  
 if (!validateString(registrationNumber, *MIN\_REG\_NUMBER\_LENGTH*, *MAX\_REG\_NUMBER\_LENGTH*, RegExProvider.*LETTERS*)) {  
 setValidationFeedback("Неверный формат номера регистрации автомобиля");  
 return false;  
 }  
  
 if (year < 1900) {  
 setValidationFeedback("Неверный год выпуска автомобиля");  
 return false;  
 }  
  
 if (makeId <= 0) {  
 setValidationFeedback("Неверный id производителя автомобиля");  
 return false;  
 }  
  
 if (mileage < *MIN\_MILEAGE* || mileage > *MAX\_MILEAGE*) {  
 setValidationFeedback("Неверный пробег автомобиля");  
 return false;  
 }  
  
 if (gearboxType < 1 || gearboxType > 2) {  
 setValidationFeedback("Неверный формат коробки передач");  
 return false;  
 }  
  
 if (pricePerDay < *MIN\_PRICE* || pricePerDay > *MAX\_PRICE*) {  
 setValidationFeedback("Неверная цена аренды автомобиля в будние дни");  
 return false;  
 }  
  
 if (pricePerDay2 < *MIN\_PRICE* || pricePerDay2 > *MAX\_PRICE*) {  
 setValidationFeedback("Неверная цена аренды автомобиля в выходные и праздничные дни");  
 return false;  
 }  
  
 return true;  
 }  
}

package validation.impl;  
  
import entity.Client;  
import validation.RegExProvider;  
import validation.Validator;  
  
public class ClientValidator extends Validator<Client> {  
 private final static int *MIN\_SHORT\_TEXT\_LENGTH* = 3;  
 private final static int *MAX\_SHORT\_TEXT\_LENGTH* = 50;  
 private final static int *MIN\_PASSPORT\_ID\_LENGTH* = 9;  
 private final static int *MAX\_PASSPORT\_ID\_LENGTH* = 9;  
 private final static int *MIN\_PHONE\_LENGTH* = 6;  
 private final static int *MAX\_PHONE\_LENGTH* = 20;  
  
 @Override  
 public boolean isValid(Client entity) {  
 String name = entity.getName();  
 String surname = entity.getSurname();  
 String patronymic = entity.getPatronymic();  
 String passportId = entity.getPassportId();  
 String homePhone = entity.getHomePhone();  
 String mobilePhone = entity.getMobilePhone();  
 String email = entity.getEmail();  
  
 if (!validateString(name, *MIN\_SHORT\_TEXT\_LENGTH*, *MAX\_SHORT\_TEXT\_LENGTH*, RegExProvider.*LETTERS*)) {  
 setValidationFeedback("Неверный формат имени");  
 return false;  
 }  
  
 if (!validateString(surname, *MIN\_SHORT\_TEXT\_LENGTH*, *MAX\_SHORT\_TEXT\_LENGTH*, RegExProvider.*LETTERS*)) {  
 setValidationFeedback("Неверный формат фамилии");  
 return false;  
 }  
  
 if (!validateString(patronymic, *MIN\_SHORT\_TEXT\_LENGTH*, *MAX\_SHORT\_TEXT\_LENGTH*, RegExProvider.*LETTERS*)) {  
 setValidationFeedback("Неверный формат отчества");  
 return false;  
 }  
  
 if (!validateString(passportId, *MIN\_PASSPORT\_ID\_LENGTH*, *MAX\_PASSPORT\_ID\_LENGTH*, RegExProvider.*PASSPORT\_ID*)) {  
 setValidationFeedback("Неверный формат номера паспорта");  
 return false;  
 }  
  
 if (homePhone != null && !homePhone.isEmpty() &&  
 !validateString(homePhone, *MIN\_PHONE\_LENGTH*, *MAX\_PHONE\_LENGTH*, RegExProvider.*NUMBERS\_AND\_SYMBOLS*)) {  
 setValidationFeedback("Неверный формат домашнего телефона");  
 return false;  
 }  
  
 if (mobilePhone != null && !mobilePhone.isEmpty() &&  
 !validateString(mobilePhone, *MIN\_PHONE\_LENGTH*, *MAX\_PHONE\_LENGTH*, RegExProvider.*NUMBERS\_AND\_SYMBOLS*)) {  
 setValidationFeedback("Неверный формат мобильного телефона");  
 return false;  
 }  
  
 if (email != null && !email.isEmpty() &&  
 !validateString(email, *MIN\_SHORT\_TEXT\_LENGTH*, *MAX\_SHORT\_TEXT\_LENGTH*, RegExProvider.*LATIN\_LETTERS\_WITH\_NUMBERS\_AND\_SYMBOLS*)) {  
 setValidationFeedback("Неверный формат email");  
 return false;  
 }  
  
 return true;  
 }  
}

package validation.impl;  
  
import entity.Employee;  
import validation.RegExProvider;  
import validation.Validator;  
  
public class EmployeeValidator extends Validator<Employee> {  
 private final static int *MIN\_PASSWORD\_LENGTH* = 4;  
 private final static int *MAX\_PASSWORD\_LENGTH* = 50;  
 private final static int *MIN\_LOGIN\_LENGTH* = 5;  
 private final static int *MAX\_LOGIN\_LENGTH* = 50;  
  
 @Override  
 public boolean isValid(Employee entity) {  
 String login = entity.getLogin();  
 String password = entity.getPassword();  
  
 if (!validateString(login, *MIN\_LOGIN\_LENGTH*, *MAX\_LOGIN\_LENGTH*, RegExProvider.*LATIN\_LETTERS\_WITH\_NUMBERS\_AND\_SYMBOLS*)) {  
 setValidationFeedback("Неверный формат логина");  
 return false;  
 }  
  
 if (!validateString(password, *MIN\_PASSWORD\_LENGTH*, *MAX\_PASSWORD\_LENGTH*, RegExProvider.*LATIN\_LETTERS\_WITH\_NUMBERS\_AND\_SYMBOLS*)) {  
 setValidationFeedback("Неверный формат пароля");  
 return false;  
 }  
  
 return true;  
 }  
}

package validation;  
  
//регулярные выражения для проверки полей ввода  
public class RegExProvider {  
 public final static String *LETTERS* = "[а-яА-Яa-zA-Z]+";  
 public final static String *LATIN\_LETTERS\_WITH\_NUMBERS\_AND\_SYMBOLS* = "[a-zA-Z0-9/\*-+\_@.]+";  
 public final static String *PASSPORT\_ID* = "[A-Z]{2}[0-9]{7}";  
 public final static String *NUMBERS\_AND\_SYMBOLS* = "[0-9+-\*/#()]";  
}

package validation;  
  
public abstract class Validator<T> {  
 private String validationFeedback;  
  
 public abstract boolean isValid(T entity);  
  
 //исп для устан сообщения обратн связи при проверке, если проверка не пройдена  
 protected void setValidationFeedback(String feedback)  
 {  
 validationFeedback = feedback;  
 }  
  
 //метод возвращает логическое значение, указывающее, является ли строка допустимой или нет  
 protected boolean validateString(String str, int minLength, int maxLength, String regex) {  
 return str != null && str.length() <= maxLength && str.length() >= minLength &&  
 str.matches(regex);  
 }  
  
 public String getValidationFeedback()  
 {  
 return validationFeedback;  
 }  
}

# ПРИЛОЖЕНИЕ Г

**(обязательное)**

**Листинг баз данных**

CREATE DATABASE IF NOT EXISTS `car\_rental` /\*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci \*/ /\*!80016 DEFAULT ENCRYPTION='N' \*/;

USE `car\_rental`;

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!50503 SET NAMES utf8 \*/;

/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;

/\*!40103 SET TIME\_ZONE='+00:00' \*/;

/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;

--

-- Table structure for table `car\_makes`

--

DROP TABLE IF EXISTS `car\_makes`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `car\_makes` (

`id` int NOT NULL AUTO\_INCREMENT,

`name` varchar(50) NOT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `car\_makes\_id\_uindex` (`id`),

UNIQUE KEY `car\_makes\_name\_uindex` (`name`)

) ENGINE=InnoDB AUTO\_INCREMENT=9 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `car\_makes`

--

LOCK TABLES `car\_makes` WRITE;

/\*!40000 ALTER TABLE `car\_makes` DISABLE KEYS \*/;

INSERT INTO `car\_makes` VALUES (8,'Audi'),(1,'BMW'),(6,'Ford'),(4,'Honda'),(7,'Hyundai'),(3,'Mercedes'),(2,'Toyota'),(5,'Volkswagen');

/\*!40000 ALTER TABLE `car\_makes` ENABLE KEYS \*/;

UNLOCK TABLES;

/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;

/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;

/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;

CREATE DATABASE IF NOT EXISTS `car\_rental` /\*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci \*/ /\*!80016 DEFAULT ENCRYPTION='N' \*/;

USE `car\_rental`;

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!50503 SET NAMES utf8 \*/;

/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;

/\*!40103 SET TIME\_ZONE='+00:00' \*/;

/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;

--

-- Table structure for table `cars`

--

DROP TABLE IF EXISTS `cars`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `cars` (

`id` int NOT NULL AUTO\_INCREMENT,

`year` int NOT NULL,

`make\_id` int NOT NULL,

`model` varchar(50) NOT NULL,

`registration\_number` varchar(10) NOT NULL,

`mileage` int NOT NULL,

`gearbox\_type` int NOT NULL,

`price\_per\_day` float NOT NULL,

`price\_per\_day\_2` float NOT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `cars\_id\_uindex` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=7 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `cars`

--

LOCK TABLES `cars` WRITE;

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

INSERT INTO `cars` VALUES (1,2010,5,'VW Polo SEDAN','1111AA-1',1000,1,50,42),(2,2015,2,'Toyota Corolla','1112AA-1',500,1,60,53),(3,2010,2,'Toyota Prius','1113AA-1',15000,1,60,52),(4,2017,7,'Hyundai Solaris 2017','1114AA-1',1300,1,80,60),(5,2012,1,'BMW Х1','1115AA-1',5000,1,180,150),(6,2013,3,'Мерседес-Бенз Гелендваген AMG','1116AA-1',3000,1,500,450);

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

UNLOCK TABLES;

/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;

/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;

/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;

CREATE DATABASE IF NOT EXISTS `car\_rental` /\*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci \*/ /\*!80016 DEFAULT ENCRYPTION='N' \*/;

USE `car\_rental`;

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!50503 SET NAMES utf8 \*/;

/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;

/\*!40103 SET TIME\_ZONE='+00:00' \*/;

/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;

--

-- Table structure for table `cars`

--

DROP TABLE IF EXISTS `cars`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `cars` (

`id` int NOT NULL AUTO\_INCREMENT,

`year` int NOT NULL,

`make\_id` int NOT NULL,

`model` varchar(50) NOT NULL,

`registration\_number` varchar(10) NOT NULL,

`mileage` int NOT NULL,

`gearbox\_type` int NOT NULL,

`price\_per\_day` float NOT NULL,

`price\_per\_day\_2` float NOT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `cars\_id\_uindex` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=7 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `cars`

--

LOCK TABLES `cars` WRITE;

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

INSERT INTO `cars` VALUES (1,2010,5,'VW Polo SEDAN','1111AA-1',1000,1,50,42),(2,2015,2,'Toyota Corolla','1112AA-1',500,1,60,53),(3,2010,2,'Toyota Prius','1113AA-1',15000,1,60,52),(4,2017,7,'Hyundai Solaris 2017','1114AA-1',1300,1,80,60),(5,2012,1,'BMW Х1','1115AA-1',5000,1,180,150),(6,2013,3,'Мерседес-Бенз Гелендваген AMG','1116AA-1',3000,1,500,450);

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

UNLOCK TABLES;

/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;

/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;

/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;

CREATE DATABASE IF NOT EXISTS `car\_rental` /\*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci \*/ /\*!80016 DEFAULT ENCRYPTION='N' \*/;

USE `car\_rental`;

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!50503 SET NAMES utf8 \*/;

/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;

/\*!40103 SET TIME\_ZONE='+00:00' \*/;

/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;

--

-- Table structure for table `cars`

--

DROP TABLE IF EXISTS `cars`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `cars` (

`id` int NOT NULL AUTO\_INCREMENT,

`year` int NOT NULL,

`make\_id` int NOT NULL,

`model` varchar(50) NOT NULL,

`registration\_number` varchar(10) NOT NULL,

`mileage` int NOT NULL,

`gearbox\_type` int NOT NULL,

`price\_per\_day` float NOT NULL,

`price\_per\_day\_2` float NOT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `cars\_id\_uindex` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=7 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `cars`

--

LOCK TABLES `cars` WRITE;

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

INSERT INTO `cars` VALUES (1,2010,5,'VW Polo SEDAN','1111AA-1',1000,1,50,42),(2,2015,2,'Toyota Corolla','1112AA-1',500,1,60,53),(3,2010,2,'Toyota Prius','1113AA-1',15000,1,60,52),(4,2017,7,'Hyundai Solaris 2017','1114AA-1',1300,1,80,60),(5,2012,1,'BMW Х1','1115AA-1',5000,1,180,150),(6,2013,3,'Мерседес-Бенз Гелендваген AMG','1116AA-1',3000,1,500,450);

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

UNLOCK TABLES;

/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;

/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;

/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;

CREATE DATABASE IF NOT EXISTS `car\_rental` /\*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci \*/ /\*!80016 DEFAULT ENCRYPTION='N' \*/;

USE `car\_rental`;

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!50503 SET NAMES utf8 \*/;

/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;

/\*!40103 SET TIME\_ZONE='+00:00' \*/;

/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;

--

-- Table structure for table `cars`

--

DROP TABLE IF EXISTS `cars`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `cars` (

`id` int NOT NULL AUTO\_INCREMENT,

`year` int NOT NULL,

`make\_id` int NOT NULL,

`model` varchar(50) NOT NULL,

`registration\_number` varchar(10) NOT NULL,

`mileage` int NOT NULL,

`gearbox\_type` int NOT NULL,

`price\_per\_day` float NOT NULL,

`price\_per\_day\_2` float NOT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `cars\_id\_uindex` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=7 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `cars`

--

LOCK TABLES `cars` WRITE;

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

INSERT INTO `cars` VALUES (1,2010,5,'VW Polo SEDAN','1111AA-1',1000,1,50,42),(2,2015,2,'Toyota Corolla','1112AA-1',500,1,60,53),(3,2010,2,'Toyota Prius','1113AA-1',15000,1,60,52),(4,2017,7,'Hyundai Solaris 2017','1114AA-1',1300,1,80,60),(5,2012,1,'BMW Х1','1115AA-1',5000,1,180,150),(6,2013,3,'Мерседес-Бенз Гелендваген AMG','1116AA-1',3000,1,500,450);

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

UNLOCK TABLES;

/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;

/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;

/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;