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](#_Toc131203913)

[1 АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОГРАММНОГО СРЕДСТВА 4](#_Toc131203914)

[1.1 Описание предметной области 4](#_Toc131203915)

[1.2 Разработка функциональной модели предметной области. 4](#_Toc131203916)

[1.3 Анализ требований к разрабатываемому программному 6](#_Toc131203917)

[средству. Спецификация функциональных требований. 6](#_Toc131203918)

[1.4 Разработка информационной модели предметной области. 8](#_Toc131203919)

[1.5 UML-модели представления программного средства и их 10](#_Toc131203920)

[описание. 10](#_Toc131203921)

[2 ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА. 12](#_Toc131203922)

[2.1. Постановка задачи. 12](#_Toc131203923)

[2.2. Обоснование выбора компонентов и технологий для реализации программного средства. 12](#_Toc131203924)

[2.3. Архитектурные решения. 14](#_Toc131203925)

[2.4. Описание алгоритмов, реализующих ключевую бизнес-логику 17](#_Toc131203926)

[разрабатываемого программного средства. 17](#_Toc131203927)

[2.5. Проектирование пользовательского интерфейса. 19](#_Toc131203928)

# 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:

1. единый учет автомобилей в разрезе их характеристик (марка, пробег, свободен или арендован);
2. поддержка учета поступления заявок;
3. перемещение автомобиля от одного клиента к другому и учет по каждой аренде автомобиля;
4. 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).

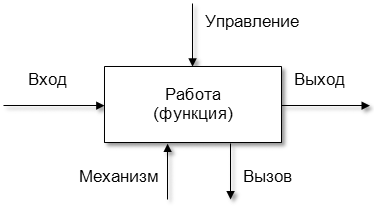


Рисунок 1.3.1 – Элементы графической нотации IDEF0

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.

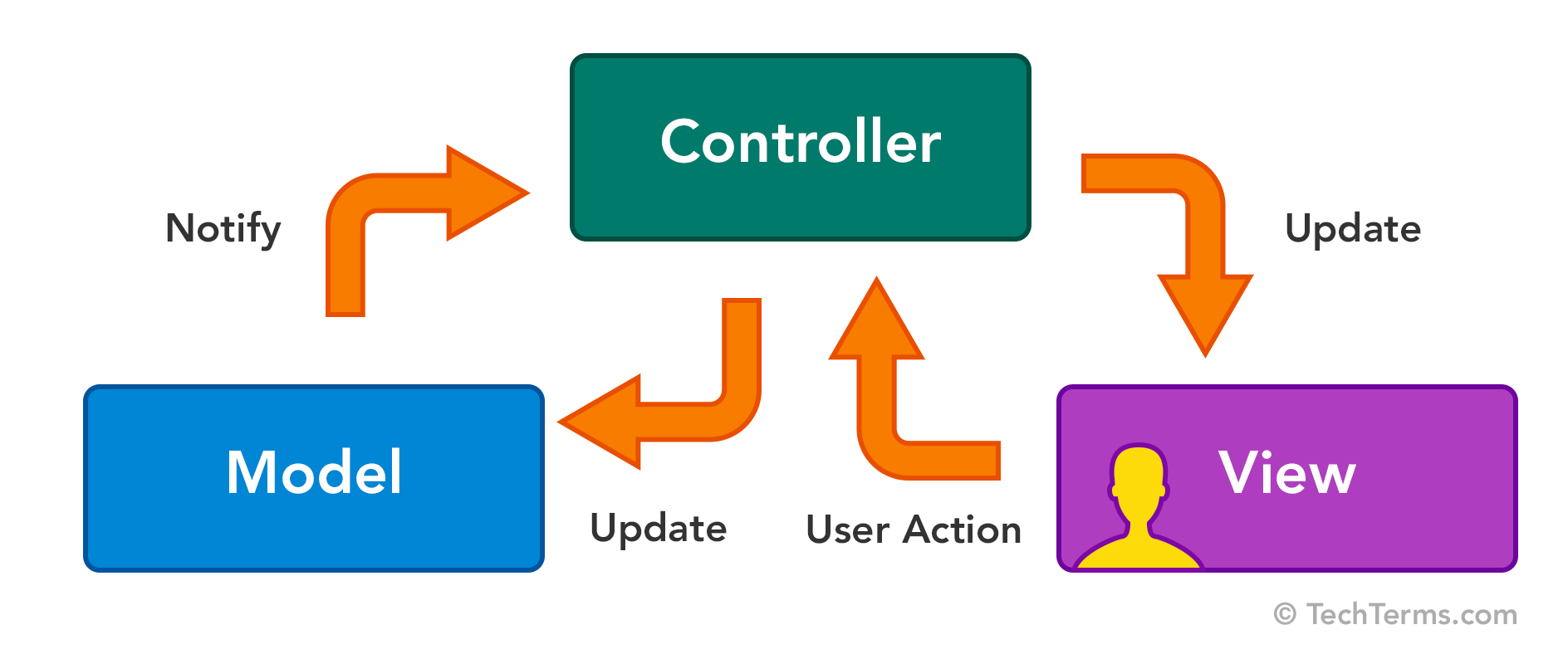


Рисунок 2.3.1 – Структура паттерна MVC

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].

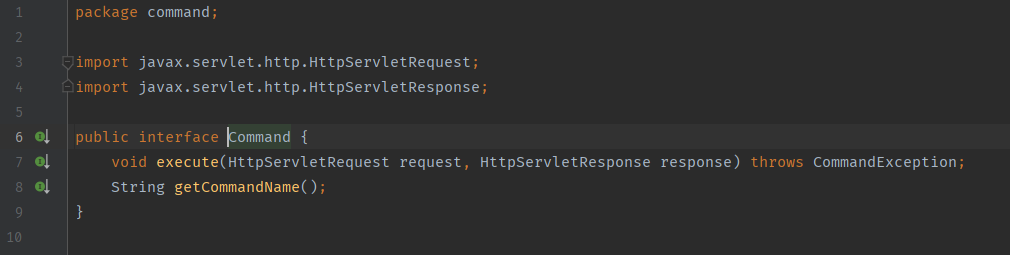


Рисунок 2.3.2 – Интерфейс «Command»

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).

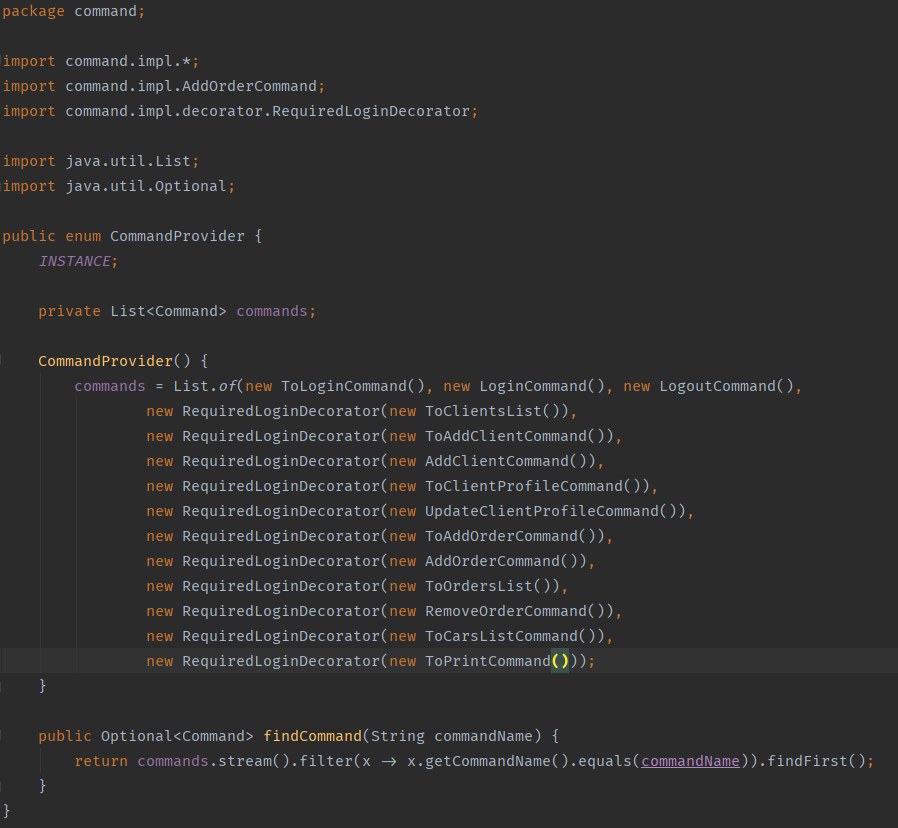


Рисунок 2.3.3 – Интерфейс «CommandProvider»

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

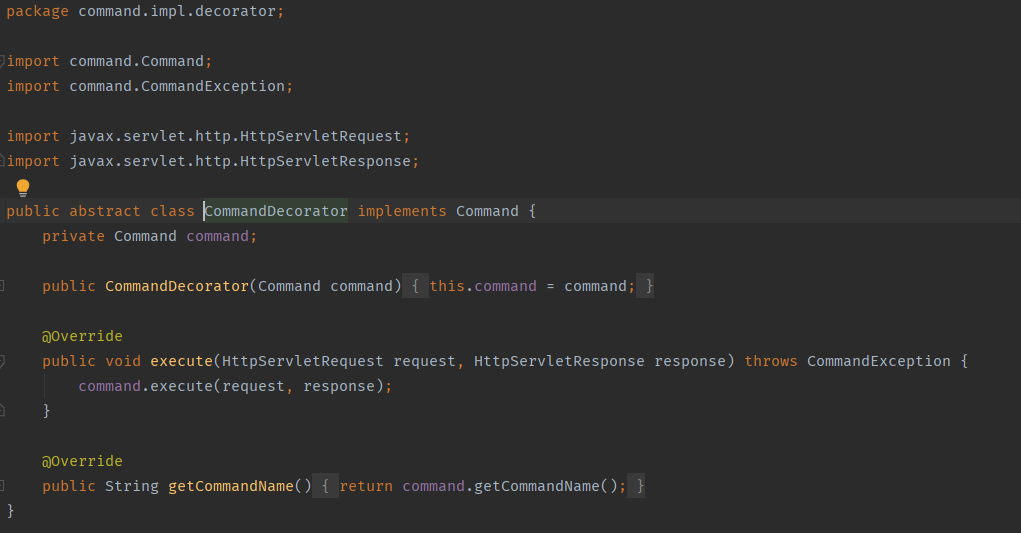


Рисунок 2.3.4 – Класс «CommandDecorator»

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.

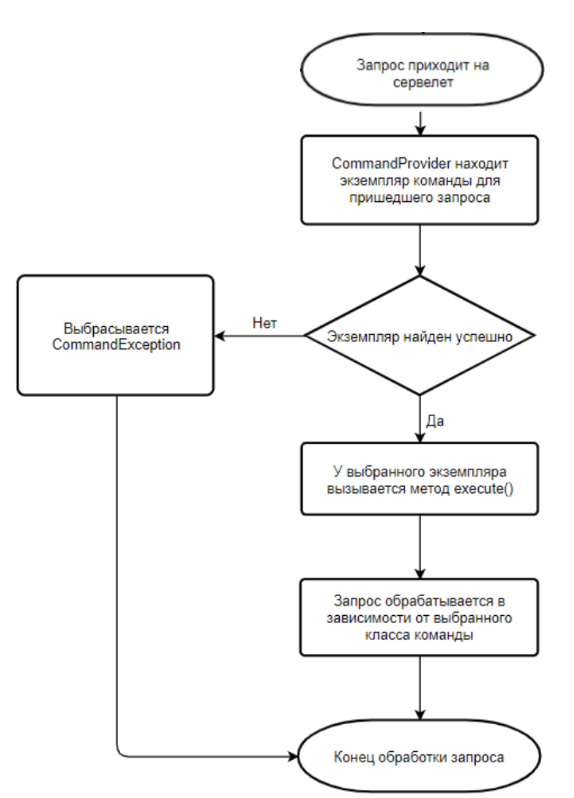


Рисунок 2.4.1 – Алгоритм обработки запроса

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).

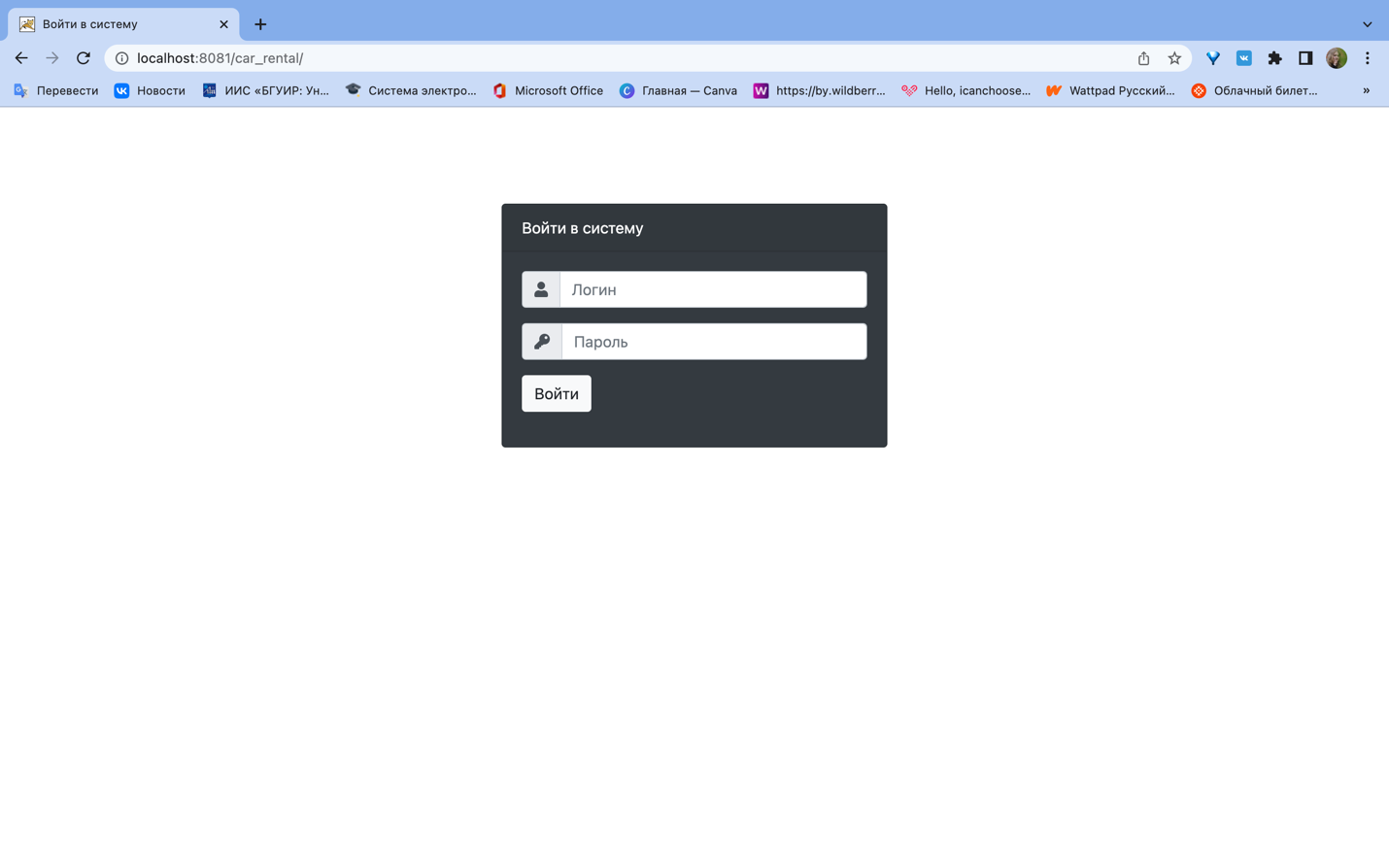


Рисунок 2.5.1 – Экран входа в систему.

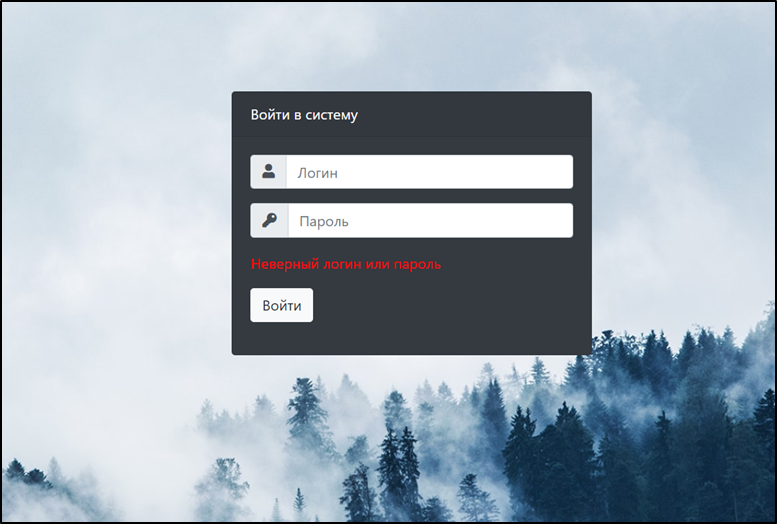


Рисунок 2.5.2 – Вывод сообщения об ошибке.

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

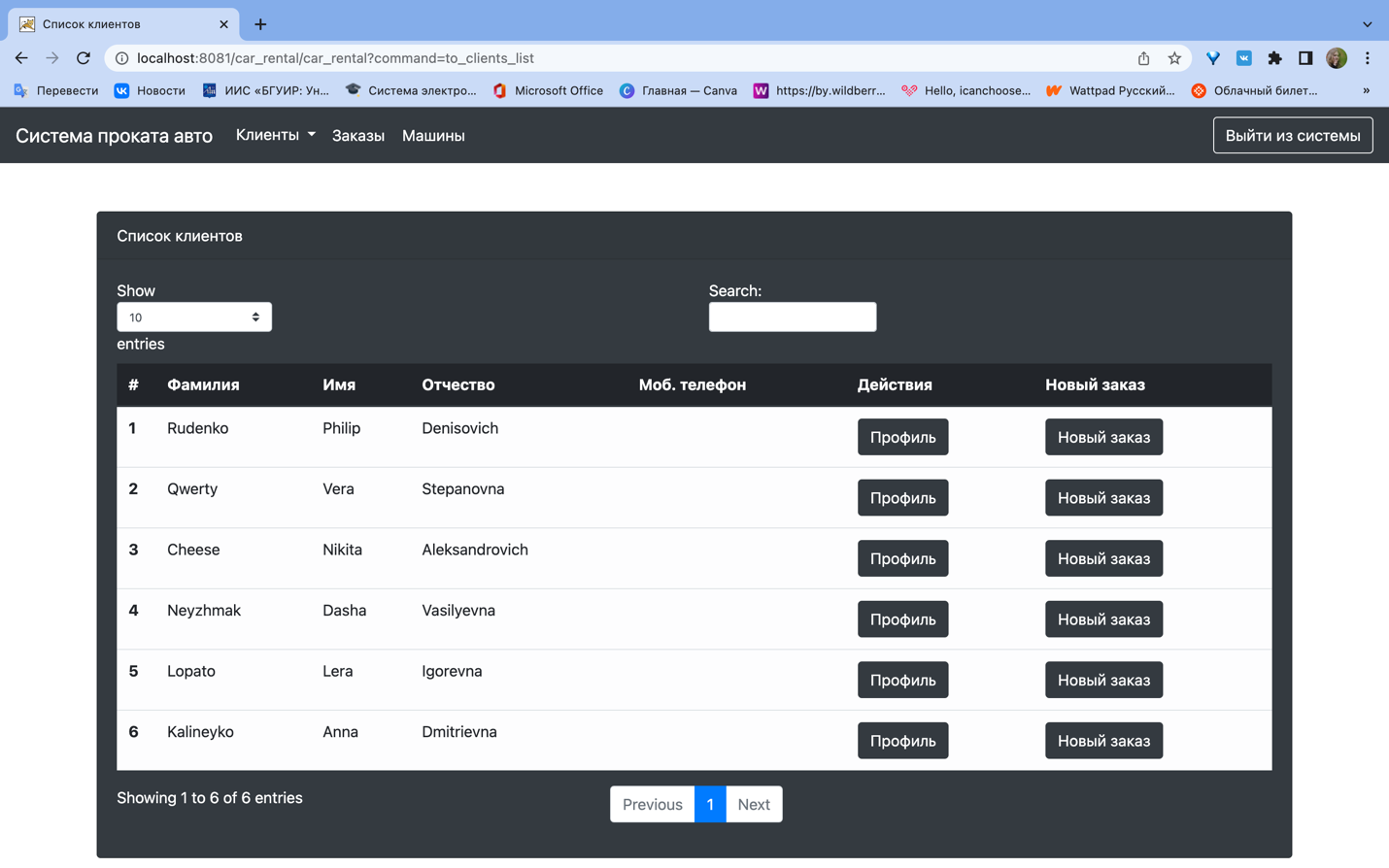


Рисунок 2.5.3 – Экран просмотра списка клиентов.

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).

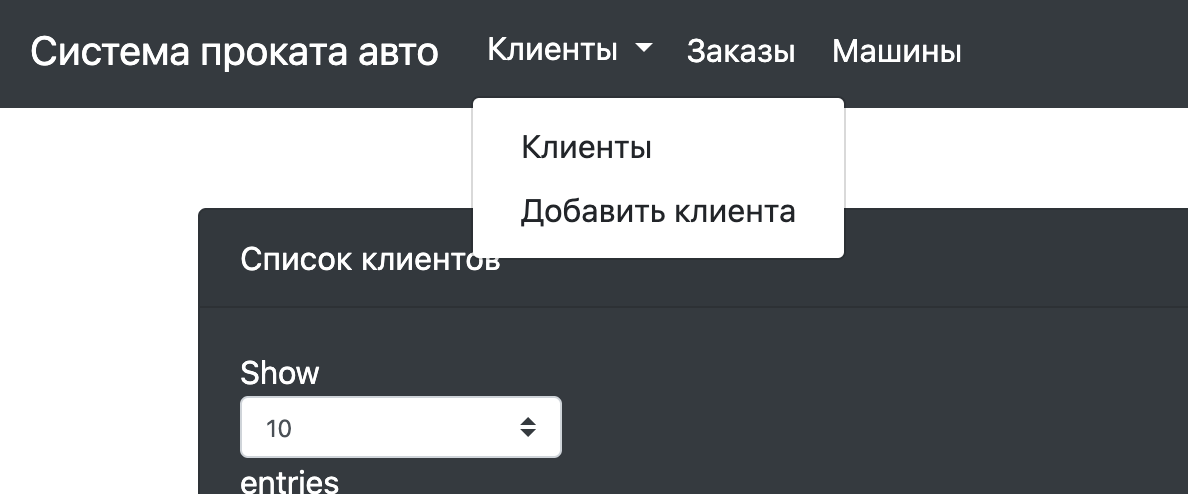


Рисунок 2.5.4 – Пункт меню «Клиенты».

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).

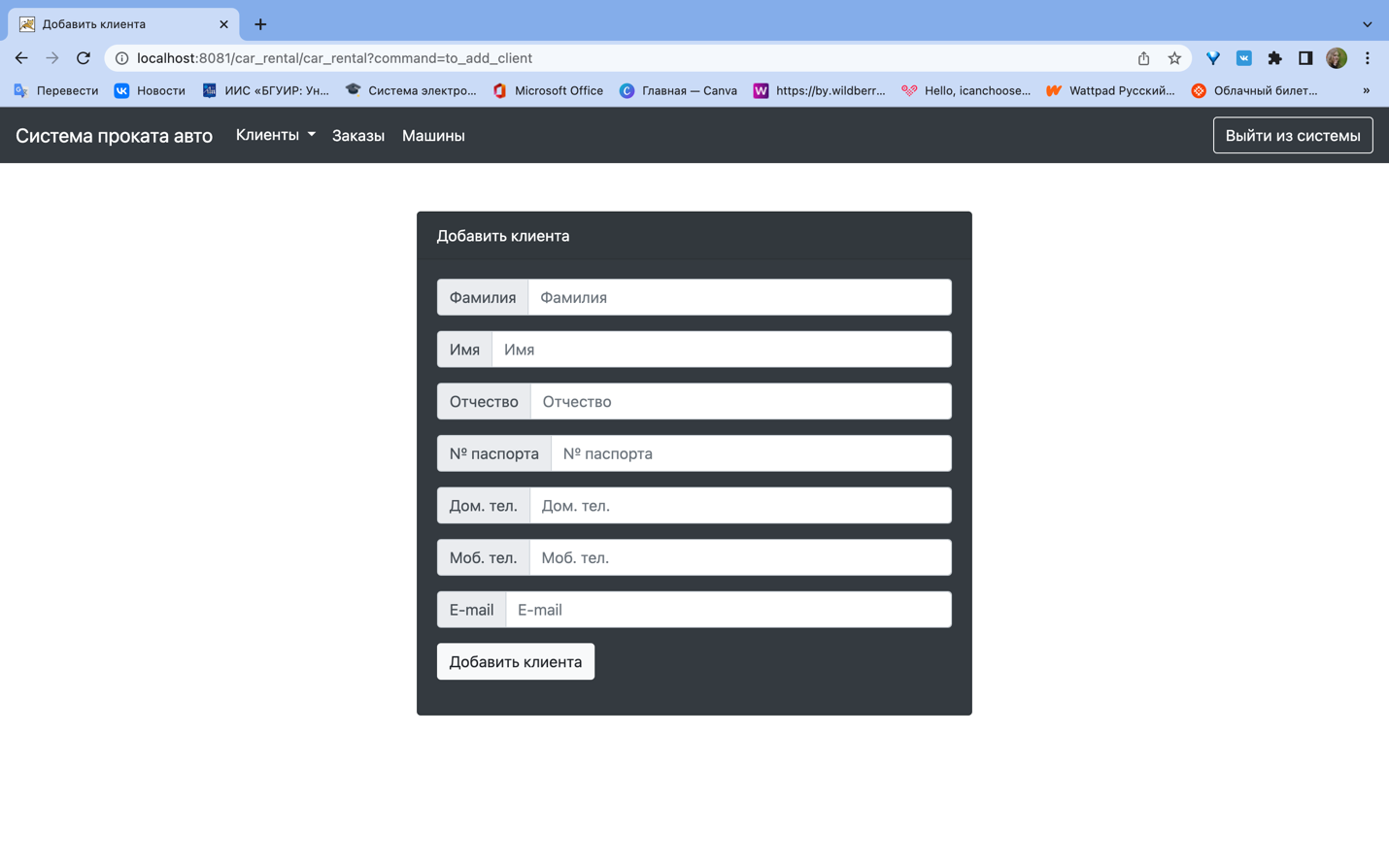


Рисунок 2.5.5 – Экран добавления нового клиента.

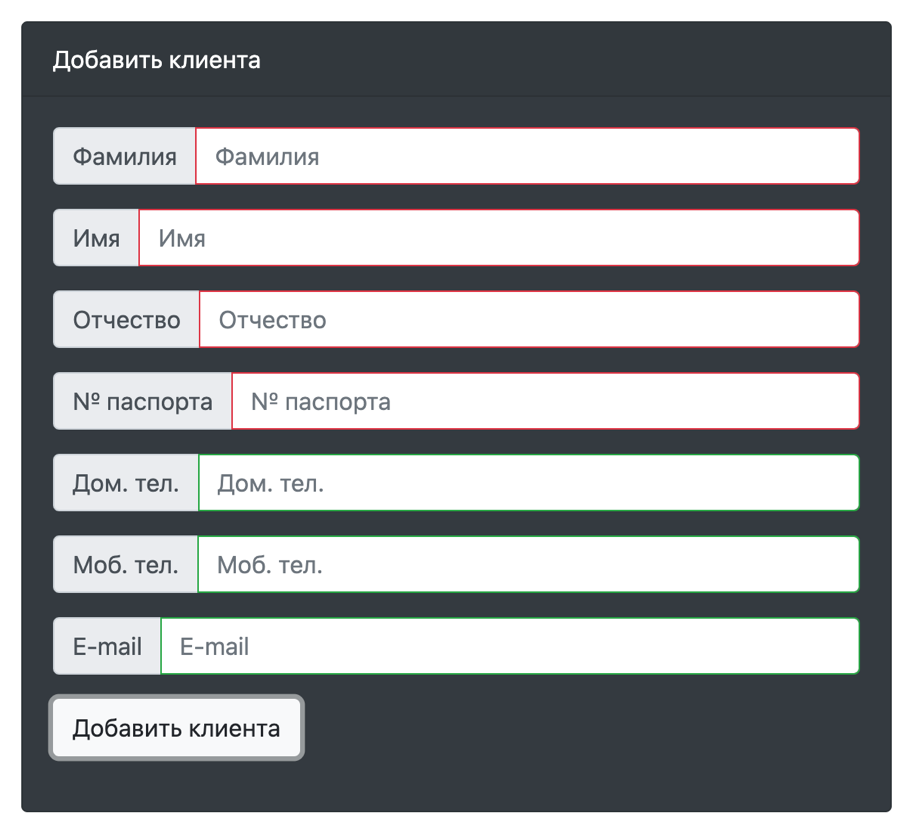


Рисунок 2.5.6 – Неудачная попытка заполнения

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.

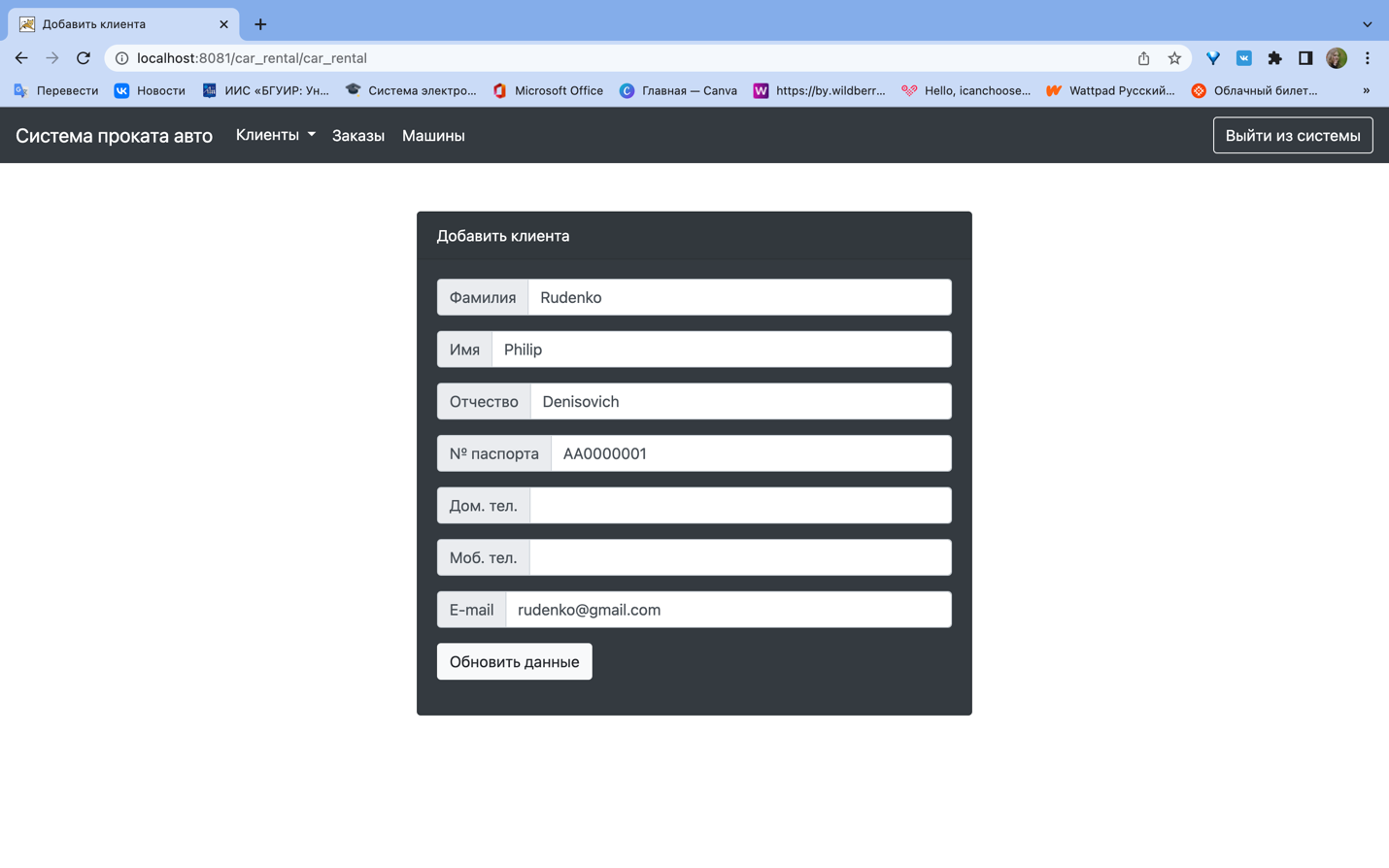


Рисунок 2.5.7 – Просмотр данных клиентов.

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.

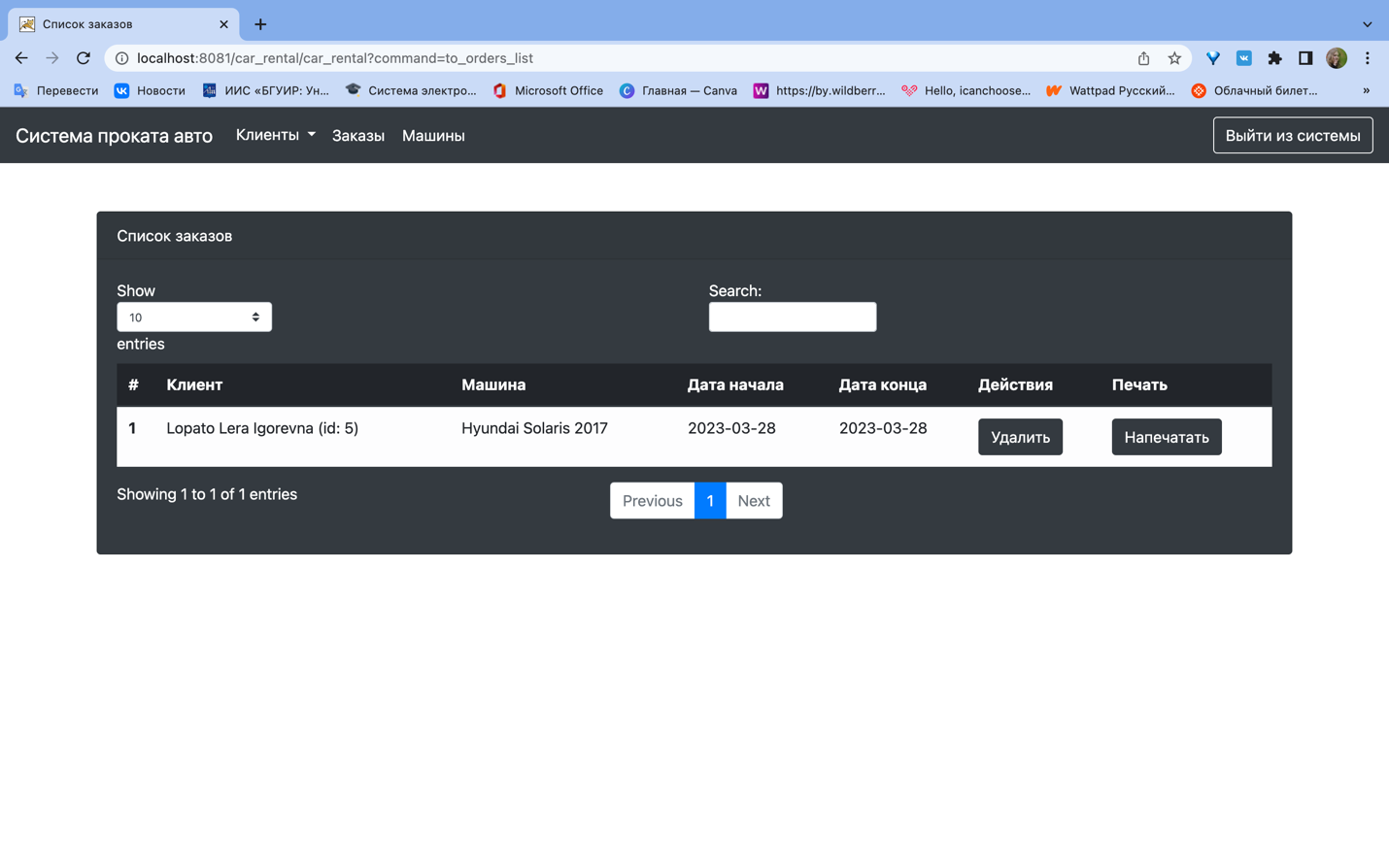


Рисунок 2.5.8 – Просмотр списка заказов.

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».

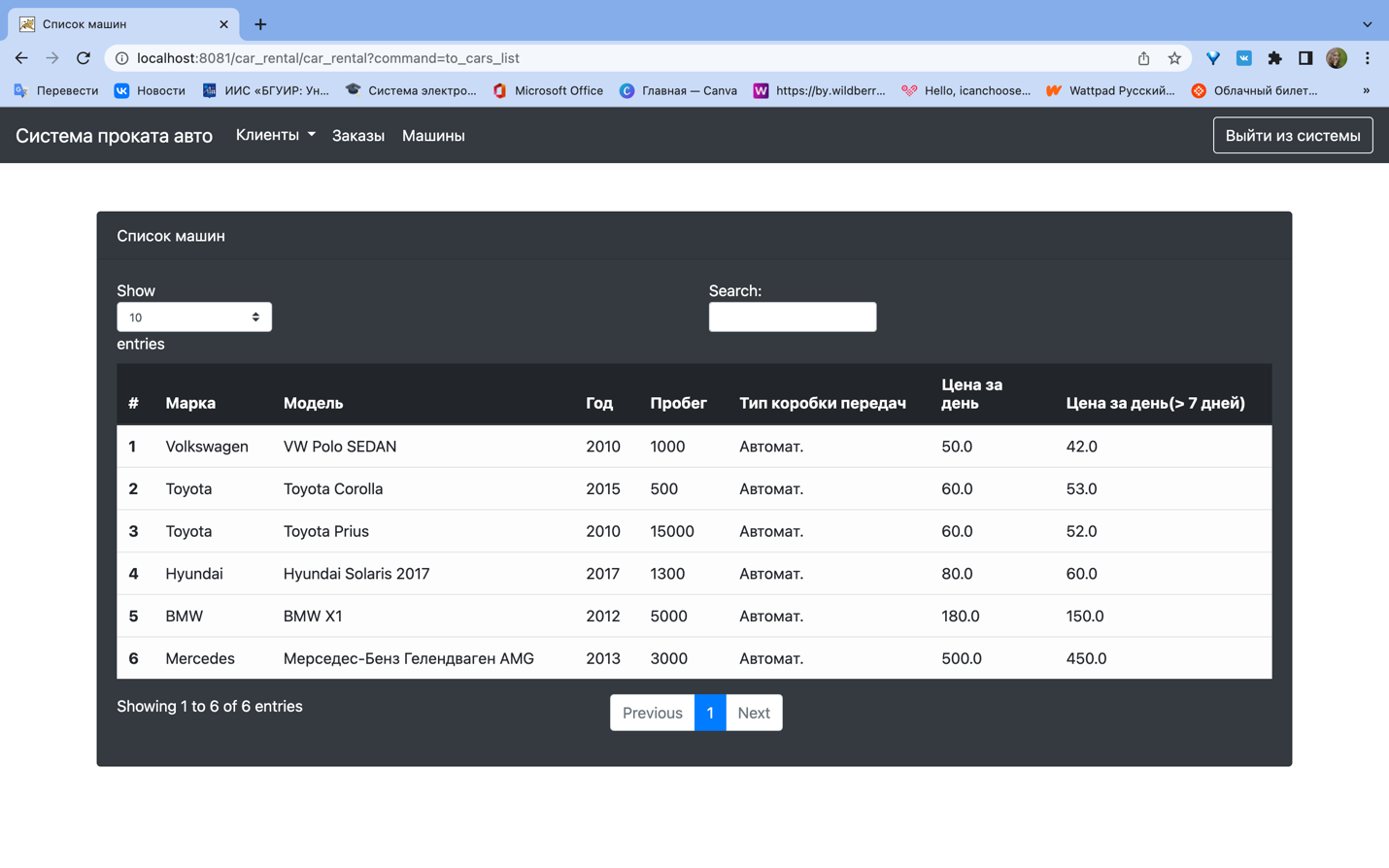


Рисунок 2.5.9 – Просмотр списка машин.

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

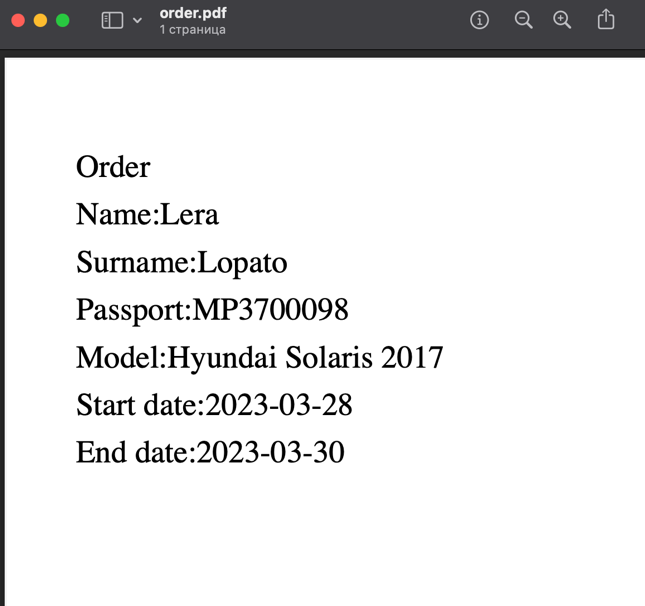


Рисунок 2.5.10 – Открытый PDF файл заказа.

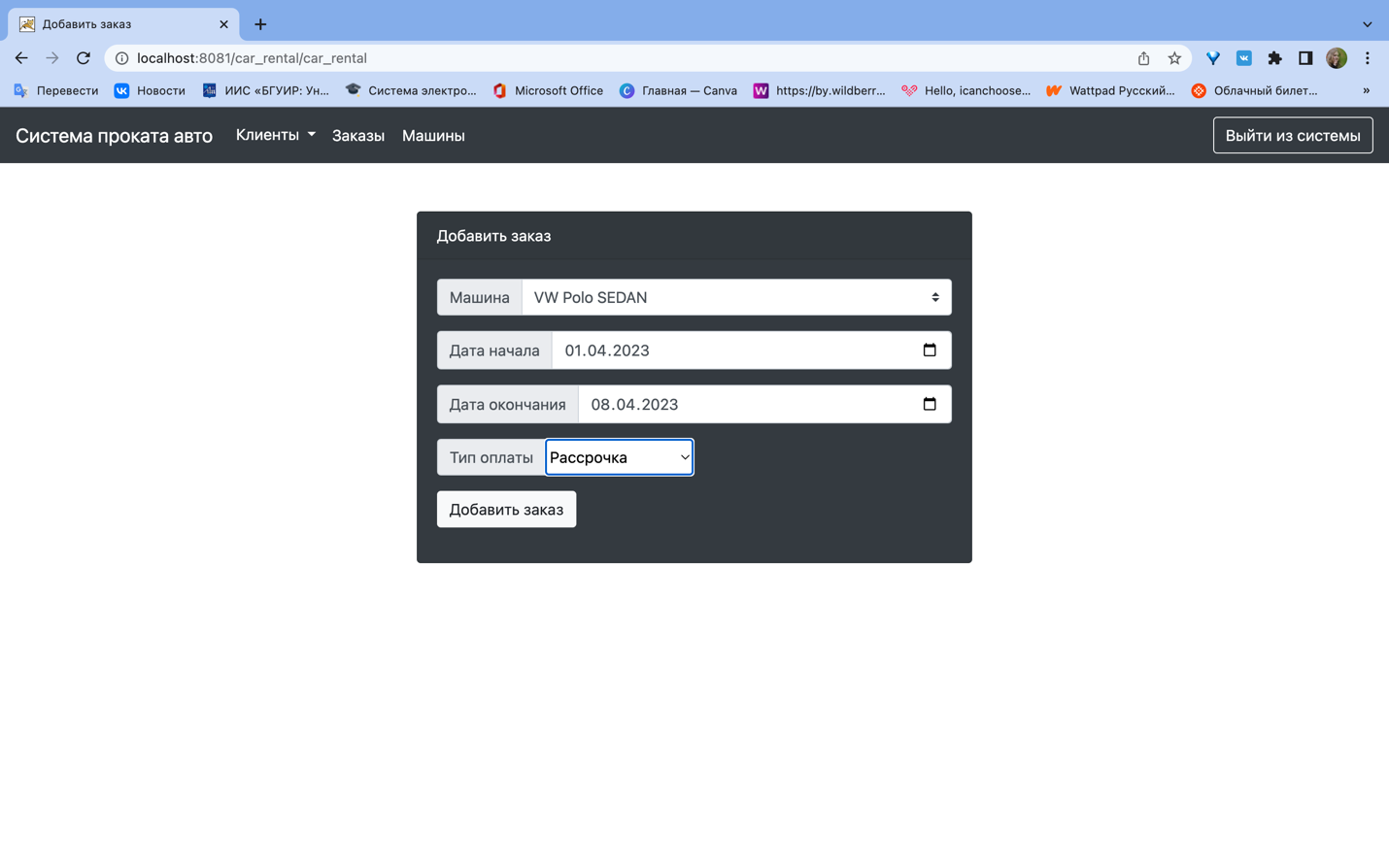


Рисунок 2.5.11 – Процесс заказа автомобиля.

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.

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Алгоритмы работы программы**

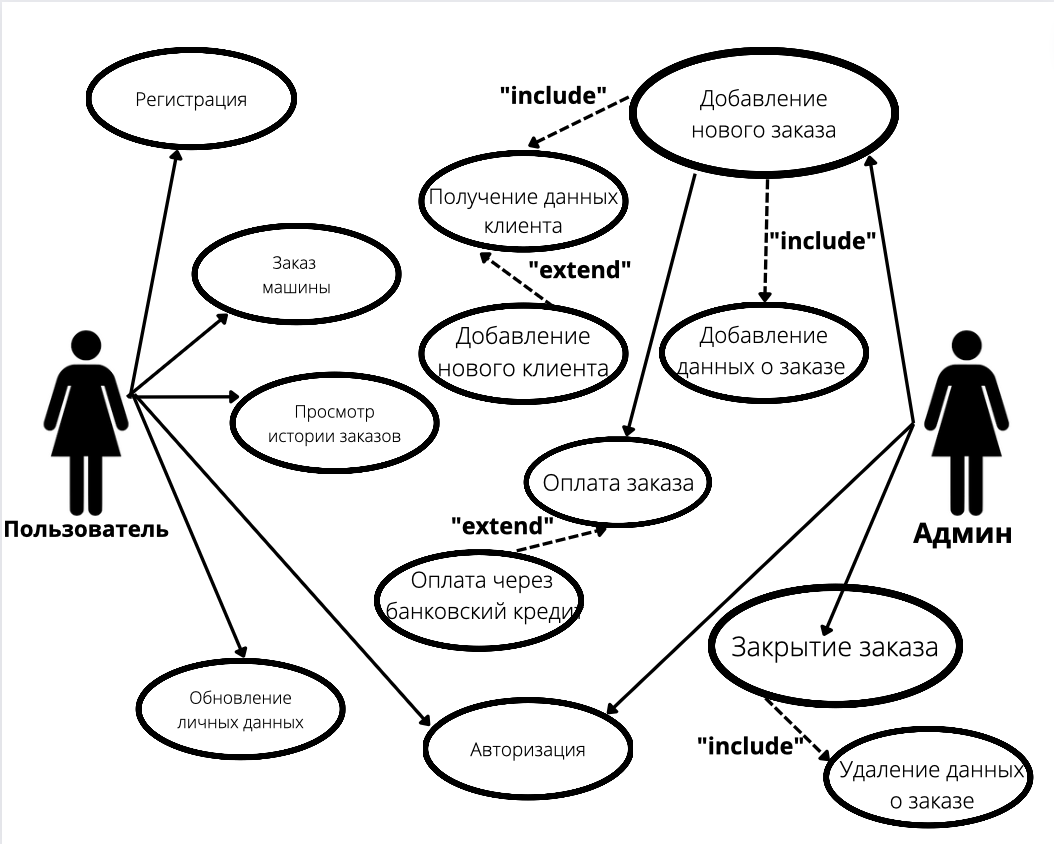


Рисунок А.1 – Система автоматизированного проката

Продолжение приложения А

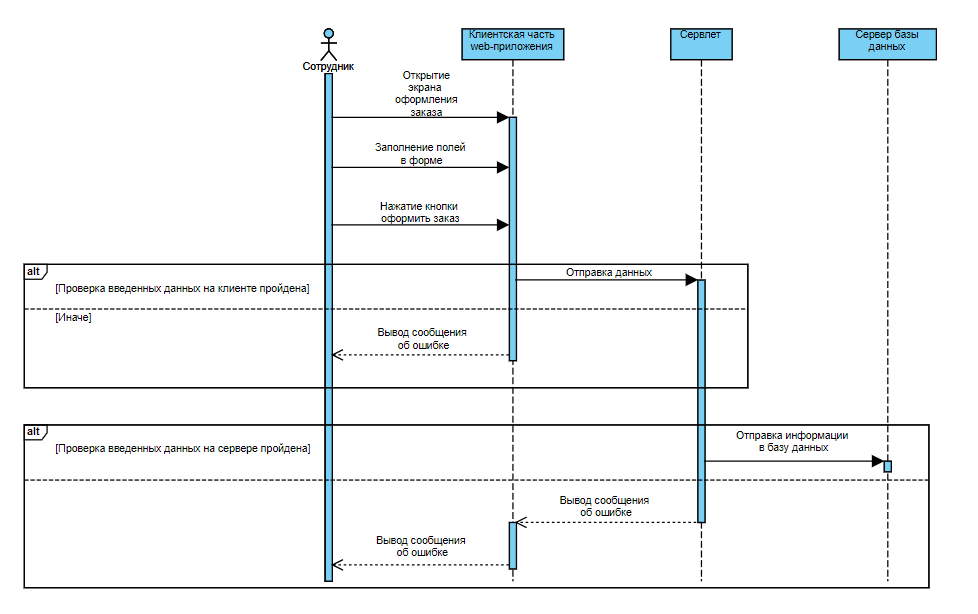


Рисунок А.2 – Диаграмма последовательности

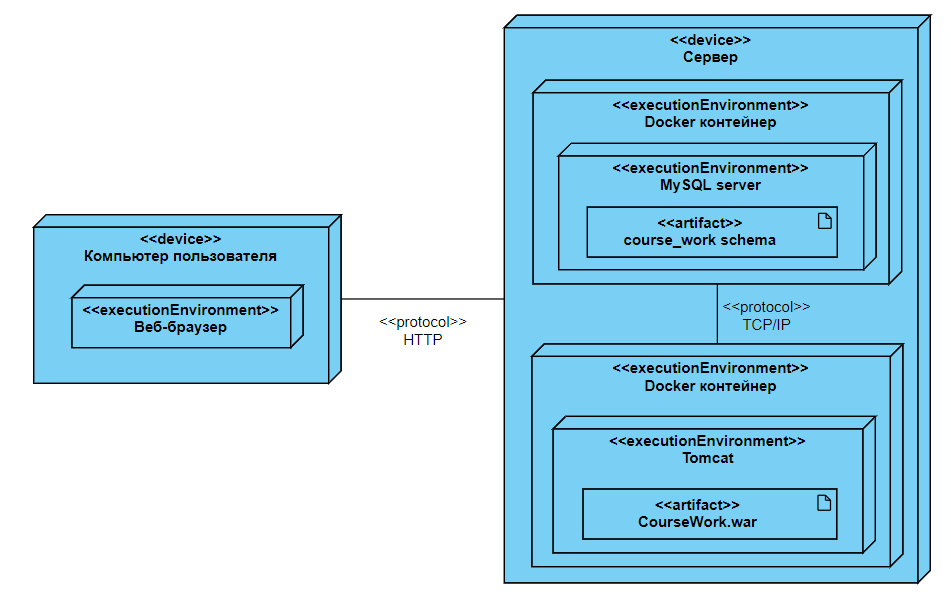


Рисунок А.3 – Диаграмма развертывания

Продолжение приложения А

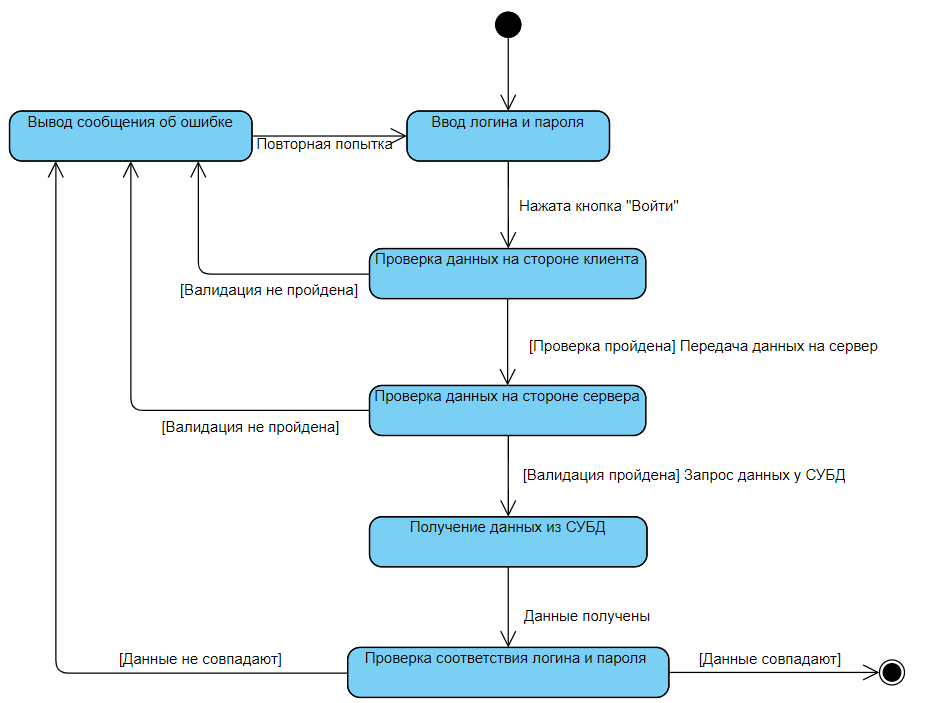


Рисунок А.4 – Диаграмма авторизации

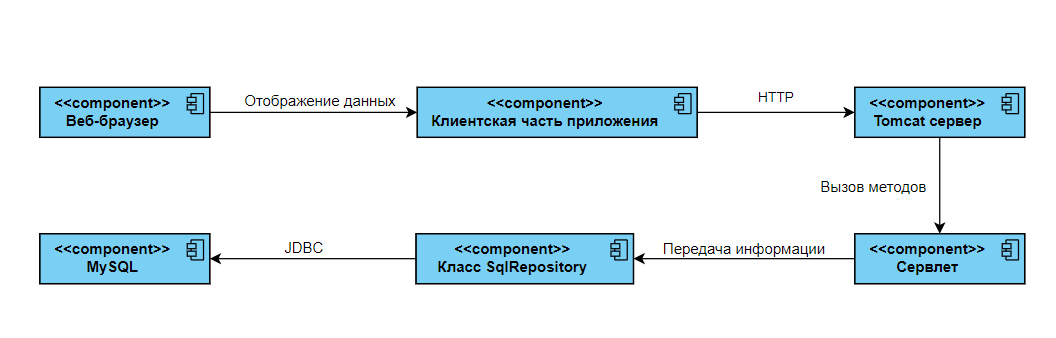


Рисунок А.5 – Диаграмма компонентов.

Продолжение приложения А

Изображение выглядит как черный, компьютер, ноутбук, экран

Автоматически созданное описание

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**

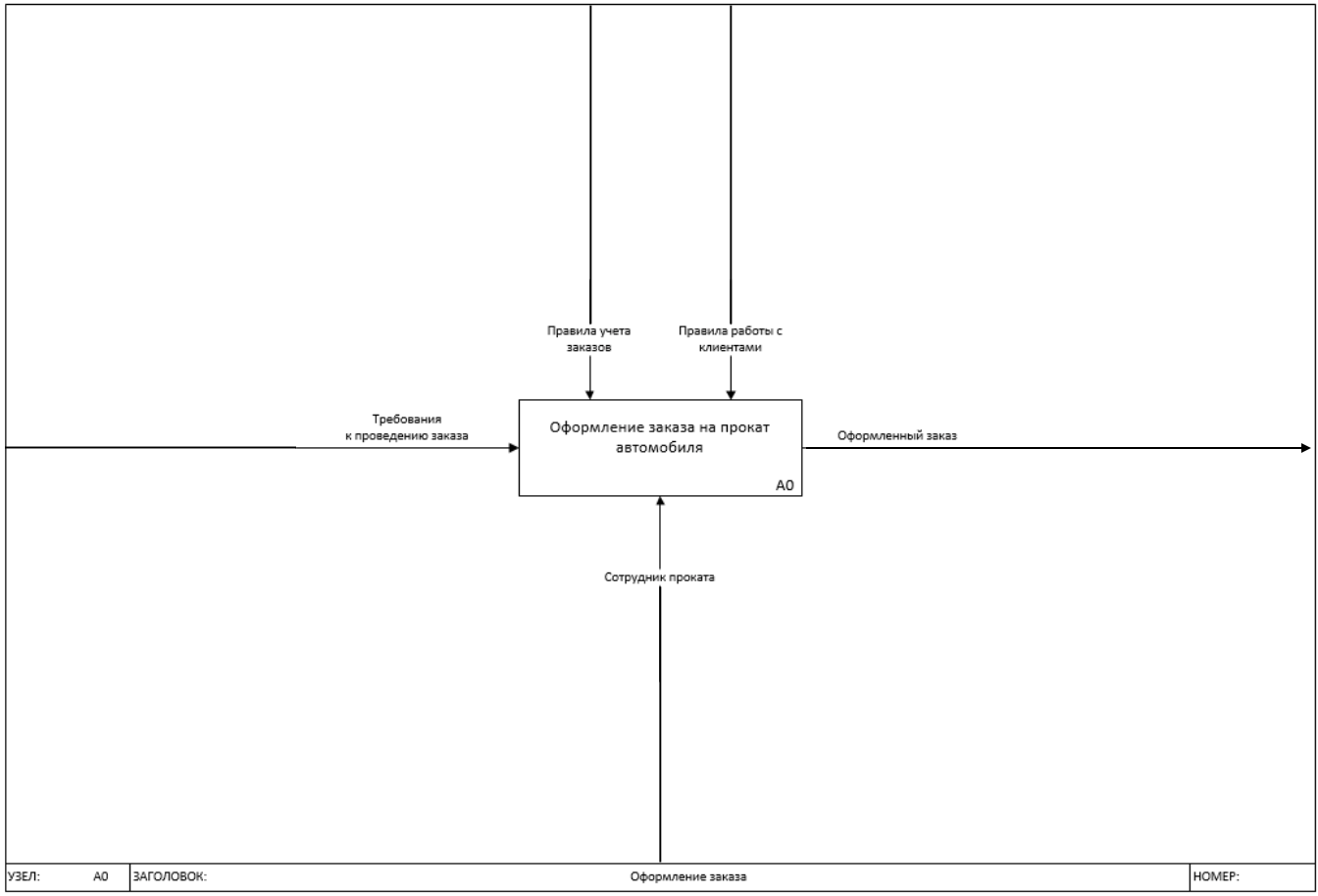


Рисунок Б.1 – Диаграмма оформления заказа

Продолжение приложения Б

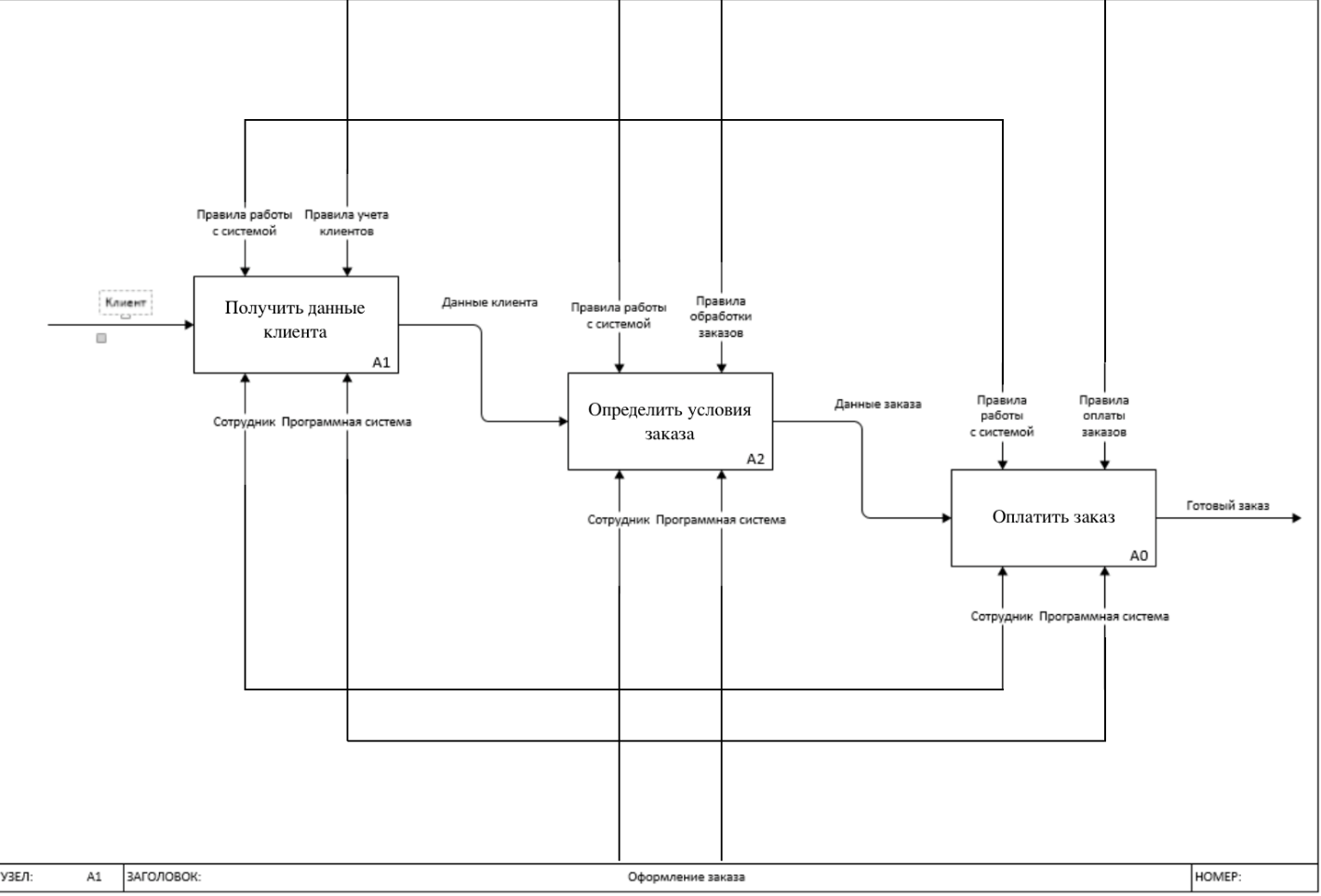
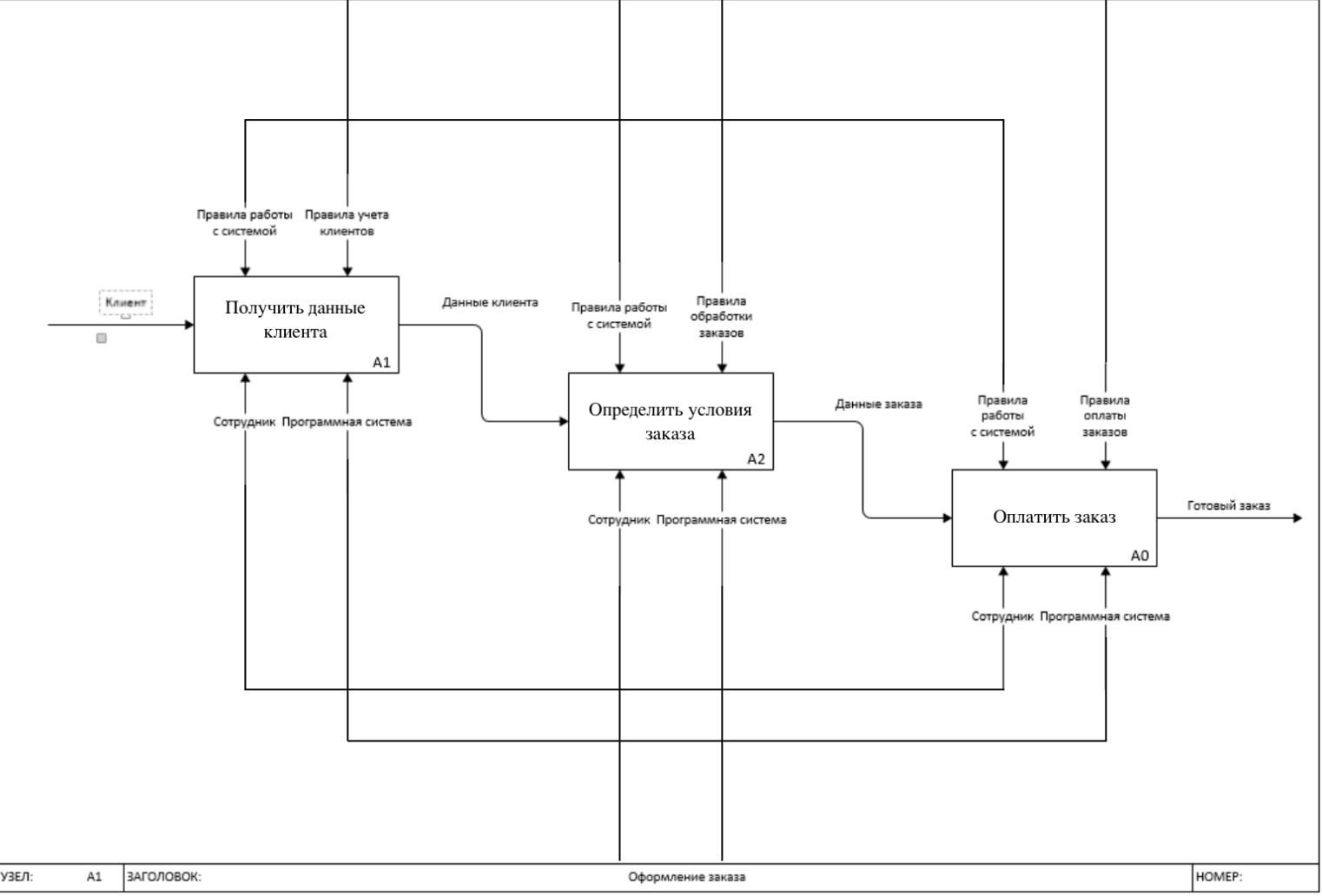


Рисунок Б.2 – Декомпозиция работы «Оформление заказа».

Продолжение приложения Б

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



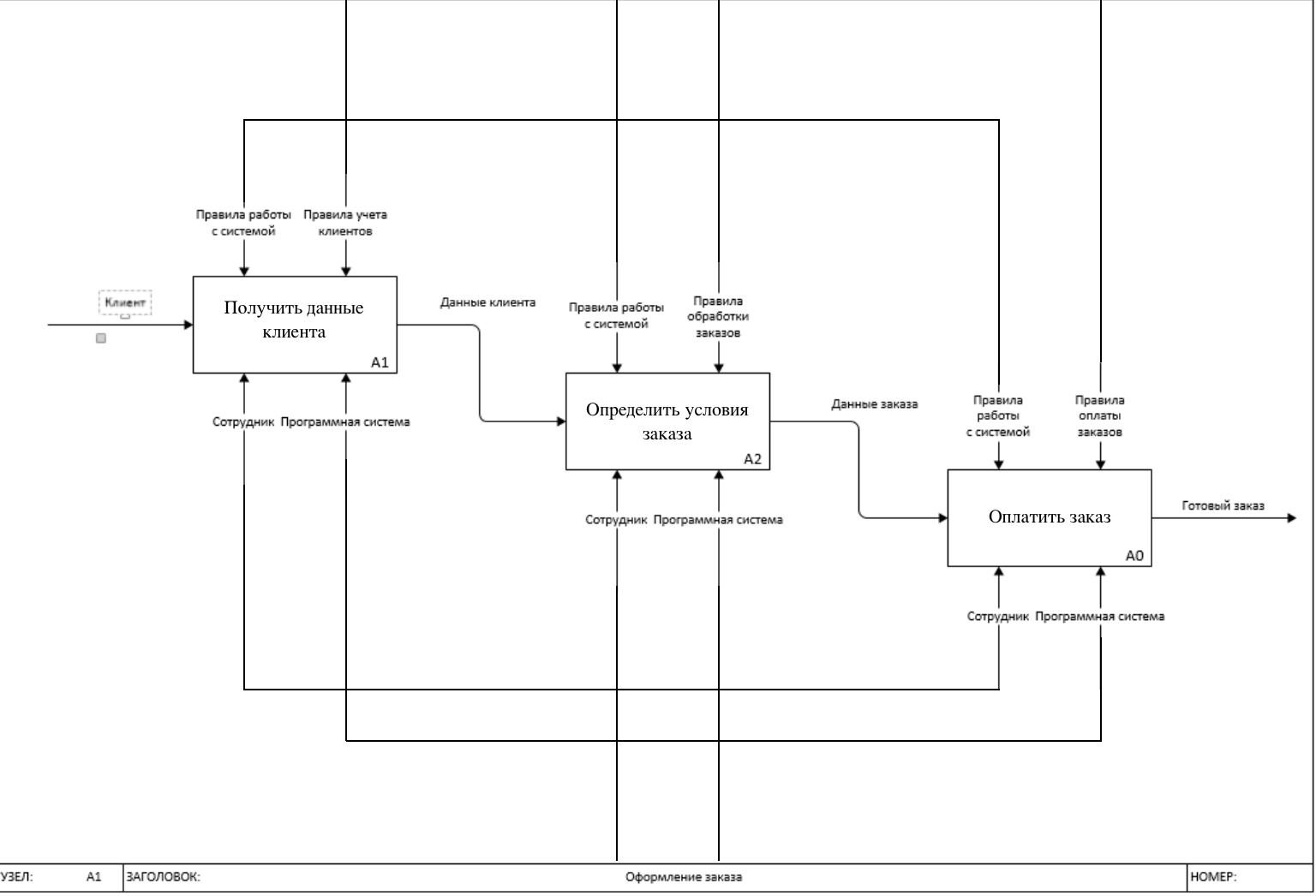
Продолжение приложения Б

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