МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«Самарский национальный исследовательский университет имени академика С.П. Королева»

(Самарский университет)

Институт информатики и кибернетики Кафедра информационных систем и технологий

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

«АВТОМАТИЗИРОВАННАЯ ИНФОРМАЦИОННАЯ СИСТЕМА РАБОТЫ КЛИНИНГОВОЙ КОМПАНИИ»

по направлению подготовки 09.03.01 Информатика и вычислительная техника

(уровень бакалавриата) профиль «Информационные системы»

Обучающийся Е.А. Шкайдуров

(подпись, дата)

Руководитель ВКР,

к.т.н., доцент И.В. Лёзина

(подпись, дата)

Нормоконтролер И.С Бельгер

(подпись, дата)

Самара 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«Самарский национальный исследовательский университет имени академика С.П. Королева»

(Самарский университет) Институт информатики и кибернетики

Кафедра информационных систем и технологий

УТВЕРЖДАЮ

Заведующий кафедрой ИСТ

С.А. Прохоров

« » 20 г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

обучающемуся Шкайдурову Егору Александровичу

группа № 6402-090301D Тема работы: Автоматизированная информационная система работы клининговой компании утверждена приказом по университету № 151-Т от « 15 » апреля 20 22 г. Исходные данные к работе:

* объект автоматизации: Самарская клининовая компания
* тип операционной системы: Windows 10;
* язык программирования: Java;
* среда разработки: IntelliJ IDEA 2023.1;
* система управления базами данных: PostgreSQL 16;
* технология доступа к данным: JPA;

− технология доступа к данным: ADO.NET;

* тип ЭВМ – IBM PC совместимый;
* объем внешней памяти – не менее 1 Гб, с учетом ВП для ОС;
* объем оперативной памяти – не менее 512 Мб, с учетом ОП для ОС;
* тип процессора – Intel Core i5-7200U с тактовой частотой 2,5 ГГц и выше;
* разрешение монитора – не менее 1024x768 точек.

Перечень вопросов, подлежащих разработке в работе:

* анализ предметной области;
* аналитический обзор существующих систем-аналогов;
* разработка логического проекта системы;
* разработка логической и физической моделей базы данных;
* разработка и отладка программного обеспечения;
* описание контрольного примера и результатов испытания.

Руководитель работы

доцент кафедры ИСТ И.В. Лёзина

(подпись)

« » 20 г.

Задание принял к исполнению Е.А. Шкайдуров

(подпись)

« » 20 г.

РЕФЕРАТ

Пояснительная записка 92 страницы, 66 рисунков, 20 таблиц, 14 источников, 3 приложения.

Графическая часть: 18 слайд Microsoft PowerPoint.

АВТОМАТИЗИРОВАННАЯ ИНФОРМАЦИОННАЯ СИСТЕМА, РАБОТА КЛИНИНГОВОЙ КОМПАНИИ, БАЗА ДАННЫХ, ВЕДЕНИЕ ДАННЫХ, МАНИПУЛИРОВАНИЕ ДАННЫМИ, ОТЧЁТЫ

В выпускной квалификационной работе спроектирована и разработана автоматизированная информационная система работы клининговой компании.

Цель работы – автоматизация процесса заказа клининга и облегчение ведения бизнеса.

В процессе работы был проведен анализ предметной области и доступных систем-аналогов, разработан проект системы по методологии UML, также разработаны логическая и физическая модели базы данные в среде Draw.io. Программа реализована в среде разработки IntelliJ IDEA Ultimate 2024.1 на языке программирования Java под управлением операционной системы Windows 10. База данных реализована под управлением СУБД PostgreSQL 16. Для доступа к базе данных использована технология JPA.

ВВЕДЕНИЕ

На сегодняшний день возможность делегирования рутинных обязанностей является неотъемлемой частью повседневной жизни людей, которые ценят своё время. К счастью, в настоящее время создано большое количество сервисов, позволяющих упросить нашу жизнь. Клининг является одним из таких упрощений. Клининг – это комплекс мер, обеспечивающих чистоту офисов, торговых комплексов, отелей, коттеджей, квартир и т.д. Использование автоматизированной системы позволяет ускорить процесс оформления заказов и подойти к этому оформлению более гибко.

Благодаря автоматизации можно достигнуть большего упрощения во взаимодействии между клиентами и персоналом, а также обеспечить получение обратной связи по каждому заказу. Элементы автоматизации подразумевают оформление заказа, отслеживание статуса заказа и систему отзывов.

В настоящее время многие рутинные обязанности уже делегированы или находятся на пути к этому. Например, доставка продуктов или доставка рациона питания. Таким образом можно сэкономить много времени для клиентов.

Цель данной работы – разработка автоматизированной информационной системы работы клининговой компании.

1. Постановка задачи и обоснование метода решения задачи
   1. Описание предметной области и постановка задачи

Предметная область – это часть реального мира, которая подлежит изучению с целью автоматизации организации управления. Любая предметная область может быть разбита на фрагменты. Каждый фрагмент оперирует со своими объектами и с множеством пользователей, которые имеют свои взгляды на предметную область, поэтому выявление предметной области и ее анализ является неотъемлемой частью разработки любой информационной системы [1].

Клининг – это комплекс действий, необходимый для обеспечения чистоты в промышленных, коммерческих и бытовых помещениях. Помимо технического обслуживания и аспектов гигиены, она также включает в себя контроль безопасности и порядка, а также улучшения с эстетической точки зрения [2].

* + 1. Типы уборки

По критерию проводимых мероприятий уборки делится на сухую и влажную. Из названия уже понятно, что во втором случае она проводится с использованием воды.

* + 1. Виды клининга

По частоте проведения уборка делится на четыре вида:

- ежедневная;

- еженедельная;

- генеральная;

- послестроительная.

Каждый из перечисленных видов клининга применяется в зависимости от типа помещения, частоты проведения, тщательности обработки объекта и, конечно, от требований к чистоте у жильцов.

1.1.3 Тип убираемого объекта

Представленная автоматизированная система предоставляет пользователям возможность выбирать тип объекта для уборки, включая участки, частные дома, квартиры или гаражи. Это обеспечивает гибкость и персонализированный подход к выполнению работы.

1.1.4 Процесс заказа клининга

Оформление заказа на уборку обычно состоит из следующих этапов:

- выбор вида клининга: пользователь выбирает, насколько тщательную уборку хотелось бы провести;

- выбор типа объекта: выбирается тип убираемого объекта для понимания клинером деталей работы;

- определение площади уборки: данный шаг уточняет объем уборки, чтобы клинеру было удобнее рассчитать количество моющих средств, время работы;

- определение удобной даты уборки: у пользователя есть возможность гибко управлять временем и вызвать уборку в любой удобный промежуток времени;

- выбор клинера: некоторым пользователям важно выбирать специалиста, с которым у них уже есть опыт сотрудничества или с которым они чувствуют наилучшее взаимопонимание. Этот функционал обеспечивает персонализированный подход к обслуживанию и удовлетворяет предпочтения клиентов, повышая качество обслуживания.

1.1.5 Процесс обратной связи

Обратная связь – в самом широком понимании ответная реакция человека или группы людей на получаемую информацию или совершаемое действие [3]. Обратная связь помогает решать ряд следующих задач:

- вдохновлять, мотивировать и развивать человека;

- устранять ошибки и избегать их в будущем;

- корректировать бизнес-процессы;

- искать новые идеи для развития.

В современных условиях зачастую удобно опираться на объективную оценку о какой-либо сущности. Например, при покупке телевизора можно обратиться к ранее оставленной обратной связи о качестве этого товара (в данном случае – отзывам), чтобы получить впечатления и объективные оценки людей. На основании отзывов можно делать решение о том, действительно необходим именно этот товар или эта услуга.

Для получения обратной связи проводят опросы и интервью, составляют анкеты и квизы, мотивируют написать отзыв – например, в обмен на скидку. Чтобы у пользователя не пропало желание дать обратную связь, нужны удобные каналы связи.

Возможность оперативно дать обратную связь бизнесу позволяет предупреждать конфликтные ситуации и появление негативных отзывов на внешних площадках [4].

В разрабатываемом приложении процесс обратной связи реализован через систему отзывов. Пользователь вправе оставить отзыв к каждому заказу, при этом не только в текстовом формате, но и в виде фотографий и оценки по пятибальной шкале.

1.2 Анализ существующих методов решения и обоснование выбора реализуемого метода

Существует два основных способа заказа клининга: по номеру телефона и через приложение. Каждый из способов имеет свои плюсы и минусы для каждого типа граждан. Рассмотрим каждый из описанных способов подробнее.

1.2.1 Заказ клининга по номеру телефона

Несмотря на век быстроразвивающихся технологий, из жизни людей не уходят более привычные способы взаимодействия с сервисами. Один из этих способов - коммуникация через мобильную связь.

Описываемый способ заказа клининга хоть и устарел, но имеет ряд преимуществ:

- скорость оформления заказа;

- простота использования;

- доступность всем возрастам;

- отсутствие необходимости в интернет подключении.

Из вышеперечисленных положительных аспектов данного метода можно сделать вывод, что он предпочтителен для более старшего поколения.

1.2.2 Заказ клининга через автоматизированную систему

Данный метод появился в результате прогрессивного развития ИТ-технологий и представляет собой отлаженную систему для взаимодействия клининговой компании и конечного клиента. Поскольку описываемый способ заказа клининга опирается на цифровую инфраструктуру, он имеет ряд достойных преимуществ:

- более гибкая и персонализированная настройка параметров заказа;

- возможность изучить все вариации уборки перед оформлением заказа;

- отсутствие необходимости в операторе, принимающем заказы;

- клиент может наглядно отслеживать заказ;

- предоставляется возможность выбора любого клинера, а также оставить ему отзыв после проделанной им работы.

Таким образом, подход с использованием автоматизированной информационной системы имеет подавляющие преимущества как для клиента, так и для бизнеса. Вдобавок к этому, информационную систему можно развивать и интегрировать с другими сервисами автоматизации, что позволит умножить преимущество этого метода

* 1. Обзор имеющихся систем-аналогов

1.3.1 ООО «Первая клининговая служба»

ООО «Первая клининговая служба» - это команда профессионалов с пятнадцатилетним опытом работы в сфере клининговых услуг. Компания занимается профессиональной уборкой квартир и частных домов, офисов и помещений разного назначения, чисткой мебели, ковров и мойкой окон в Москве и Московской области [5].

В качестве преимуществ данного сервиса хочется отметить чрезвычайно прозрачное ценообразование, а также подробный перечень услуг и обязанностей для каждого вида уборки. На рисунке 1 представлен раздел ценообразования данной программы. В совокупности с демократичными ценами пользователь остается крайне заинтересован к заказу клининга именно в этом приложении.

Однако, как у любой другой системы, ООО «Первая клининговая служба» не обошлась и без них. Первоочередный нюанс заключается в неудачном расположении разделов приложения. Начиная просматривать один раздел, случайно можешь перейти на другой, совсем не интересующий. Вдобавок к этому не совсем приемлемо выбраны цветовые решения и спроектирован дизайн приложения, что в совокупности с предыдущим недостатком создает неинтуитивную и непонятную навигацию по сайту.



Рисунок 1 – Ценообразование ООО «Первая клининговая компания»

1.3.2 Qlean

Qlean – это сервис, предоставляющий книнговые услуги по Москве и Московской области. Компания предлагает опытных клинеров, гарантию качества и надёжность уборки, а также честную цену и скидки для постоянных клиентов. Qlean имеет веб-сайт, через который будет удобно заказать клининг, работая за компьютером, и мобильное приложение, через которое можно не только заказывать клининг, но и узнавать об акциях и специальных предложениях. Интерфейс системы продемонстрирован на рисунке 2.

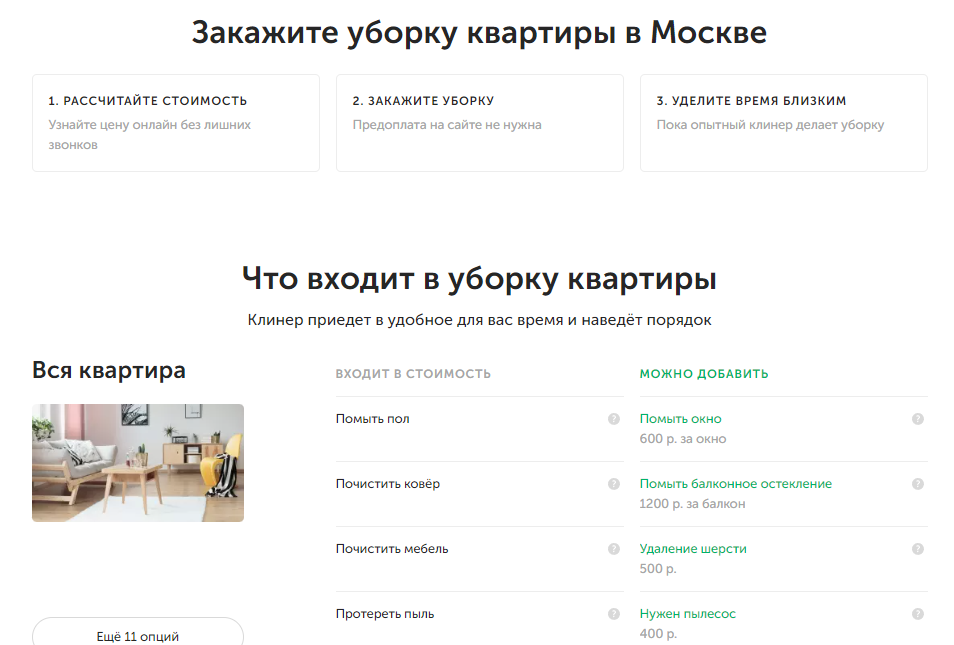


Рисунок 2 – Интерфейс системы Qlean

Из явных преимуществ можно выделить приятный интерфейс одностраничного сайта. Всю необходимую для заказа информацию можно получить порционно, пролистывая сайт. Вдобавок к этому подобраны не броские цвета, что создает приятное ощущение при взаимодействии с интерфейсом системы. Немаловажное преимущество – понятное ценообразование: некоторые услуги уже включены в стоимость уборки, но также существует ряд услуг за дополнительную плату. Рисунок 3 прекрасно описывает ценообразование в данной системе.

Говоря о минусах, можно подметить, что в системе отсутствует возможность выбора определенного клинера. Это немаловажный фактор, который хорошо сказывается как на клиенте, так и на сотруднике. Также у системы отсутствует возможность заказать уборку через телефонный звонок. Пусть эта возможность довольна редкая, однако некоторому слою населения удобно выстраивать коммуникацию по средствам телефонной связи.

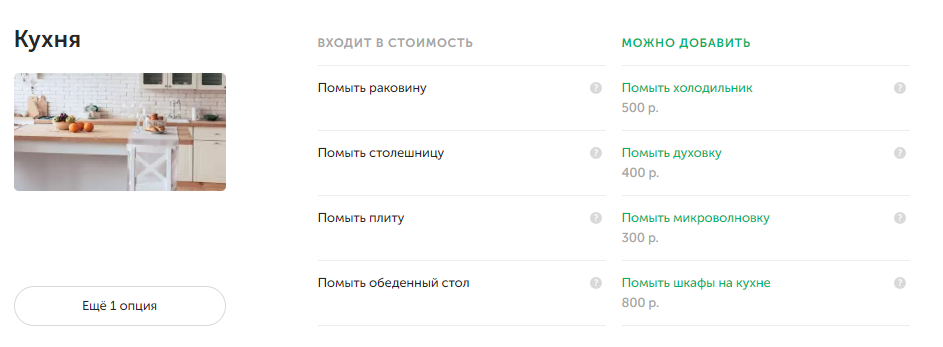


Рисунок 3 – Ценообразование в Qlean

В заключении хочется отметить, что данная система имеет больше преимуществ, чем недостатков, поэтому она может вправе считаться эталонной.

1.3.3 Клининговая компания «Avalon»

Avalon – это Самарская клининговая компания, которая предлагает уборку жилых и коммерческих помещений, химчистку матрасов, мебели и ковровых покрытий.

Сходу хочется отметить недостатки системы в виде перебивающегося интерфейса, неуместных анимаций и слабой адаптации веб-сайта под мобильные устройства. Функционал оформления заказа спрятан глубоко в логику системы, поэтому для оформления изначально требуется перейти по нескольким ссылкам, что может вполне отбить желание пользователей к заказу или требуется разглядеть неприметное меню в конце сайта. Крайне непонятное ценообразование, цены указаны только на основные виды работ. В форме для выбора степени загрязнения можно выбрать вариант «Срочно приехать», что является логической ошибкой. Рисунок 4 показывает это.



Рисунок 4 – Форма для выбора степени загрязнения

Из преимуществ можно отметить наличие номеров телефона для заказа клининга по сотовой связи, наличие физического офиса, куда можно обратиться в случае неопределенной ситуации. К преимуществам можно отнести довольно приемлемые цены, а также быстрый выезд клинера на объект: описана гарантия выезда на объект в течение двух часов.

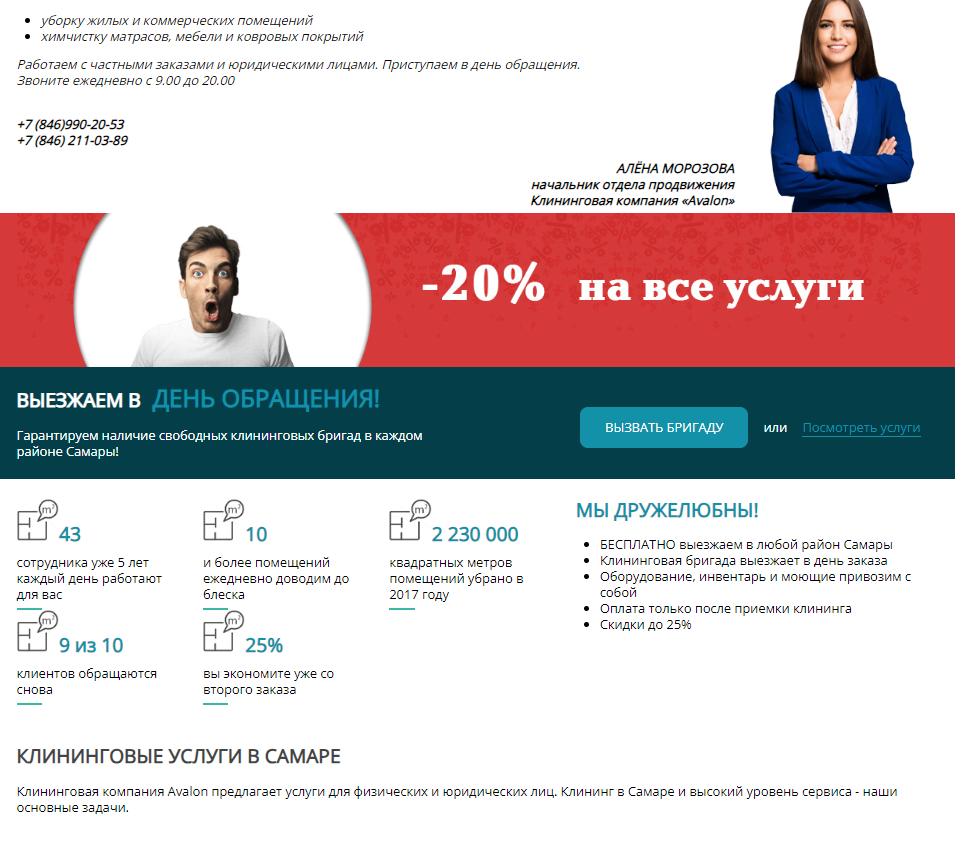


Рисунок 4 – Интерфейс «Avalon»

1. Проектирование и программная реализация метода решения задачи
   1. Обоснование и выбор комплекса программных средств, языка программирования, среды разработки, системы управления базой данных, операционной системы
      1. Выбор языка программирования

В настоящий момент существует большой выбор языков программирования. Они подразделяются на компилируемые и интерпретируемые, высокоуровневые и низкоуровневые, строго и слабо типизируемые. Каждый из вариантов содержит свои плюсы и минусы.

Мой выбор пал на компилируемый, высокоуровневый, строго типизированный язык программирования Java. Java – крайне обильно используемый язык программирования, который заточен под написание интернет-приложений. Однако ключевым преимуществом Java является то, что его код сначала транслируется в специальный байт-код, независимый от платформы. Затем байт-код выполняется виртуальной машиной JVM (Java Virual Machine). В этом плане Java отличается от стандартных интерпретируемых языков как PHP или Perl, код которых сразу же выполняется интерпретатором. В то же время Java не является и чисто компилируемым языком, как С или С++ [7].

Стоит отметить, что в моем случае использовался фреймворк Spring Boot, который упрощает написание приложений, избавляя программиста от написания однотипного кода. Spring Boot – фреймворк для создания приложений на языке Java. Он предоставляет возможность быстрого и удобного создания веб-приложений, используя уже готовый компоненты и шаблоны проектирования. Spring Boot включает все необходимые библиотеки, что позволяет разработчику сразу же начинать работу над проектом, не тратя время на конфигурацию приложения. Он также позволяет создавать микросервисы и интегрировать различные технологии, такие как базы данных, технологии обработки сообщений, без большого количества дополнительного кода [8].

* + 1. Выбор программных средств
       1. Средства для реализации клиентской части

Для реализации клиентской части системы выбор был сделан в сторону фреймворка Angular.

Angular представляет собой фреймворк от компании Google для разработки клиентских приложений. Прежде всего он нацелен на разработку SPA-решений (Single Page Application), то есть одностраничных приложений. Одной из ключевых особенностей Angular – использование в качестве языка программирования TypeScript. Однако стоит отметить, что TypeScript носит рекомендательный характер, поскольку Angular-приложение может использовать также Dart или JavaScript.

Angular обладает следующими преимуществами:

- модульность и компонентный подход: Angular-приложение разбивается на отдельные компоненты, которые подлежат многократному использованию. Такой подход значительно упрощает разработку и особенно поддержку;

- поддержка языка программирования TypeScript. Данный язык является строго типизированным, эта особенность значительно облегчает процесс разработки и повышает качество кода;

- поддержка пакетного менеджера npm: NPM – это менеджер пакетов, с помощью которого можно легко внедрять зависимости (библиотеки) в проект.

* + - 1. Средства для отладки серверной части

Перед использованием серверной части в совокупности с клиентской, хорошим тоном является проверка работоспособности всех аспектов разработанного API, инструмент Postman является основным для этой задачи.

Postman – это мощный и интуитивно понятный инструмент, предназначенный специально для тестирования и разработки API [9].

Postman позволяет отправлять HTTP-запросы, выбирая любой тип запроса, тело, заголовки, и просматривать тело и статус ответа. Также это средство позволяет открывать WebSocket-соединения, отправлять и получать сообщения в ws-канал.

* + 1. Выбор среды разработки

В настоящее время большую популярность набирают среды разработки от компании JetBrains, поскольку компания разрабатывает ряд высококачественных продуктов, каждый из которых отлично подходит под определенные языки программирования. В нашем случае для языка программирования Java описанной выше компанией разработана среда разработки IntelliJ IDEA. InelliJ IDEA – это интегрированная среда разработки (IDE) для языков программирования Java, Kotlin, Groovy и других языков. Она имеет множество функций и инструментов, которые облегчают процесс разработки.

Основные функции IntelliJ IDEA:

* автодополнение кода: среда разработки автоматически предлагает подсказки и завершение кода, что повышает производительность и уменьшает шанс ошибиться;
* интеграция с системами контроля версий: среда разработки поддерживает интеграцию с Git: позволяет делать операции commit, push, pull. Также есть возможность сделать patch-файл;
* сохраняет историю изменения кода: в описываемой среде разработки присутствует функционал, который фиксирует измененные файлы при каждом запуске программы. Можно вернуться к каждой зафиксированной точке;
* поддержка сопутствующих программных средств: IDEA предлагает решения для более удобной работы с такими сервисами, как: Maven, Spring, Hibernate, PostgreSQL и других.

На данный момент крайне популярен редактор кода Visual Studio Code (VS Code). Это средство крайне легковесное, поскольку изначально поддерживает лишь несколько языков и служит редактором кода. Однако VS Code имеет раздел расширений, в котором можно установить абсолютно любой плагин для работы с любым языком программирования. VS Code подходит для более примитивных языков, таких как JavaScript, Python, HTML, CSS.

Eclipse – свободная интегрированная среда разработки модульных кроссплатформенных приложений. На данный момент система поддерживается Eclipse Foundation. Eclipse написана на Java, потому является платформо-независимым продуктом [10]. Однако описываемая среда разработки имеет ряд критических недостатков:

- использует много системных ресурсов и процессов;

- использует много памяти;

- имеет медленный запуск;

- функция отладки не идет в ногу со временем по сравнению с другими IDE;

- неполная интеграция с Maven и Gradle.

Из всех вышеперечисленных вариантов наиболее прогрессивной и подходящей средой разработки является IntellJ IDEA. Вдобавок к этому она идеально подходит именно для Java-проектов из-за своих особенностей.

* + 1. Выбор системы управления базой данных

В данной автоматизированной информационной системе была выбрана СУБД PostgreSQL.

СУБД (система управления базами данных) – это комплекс программно-языковых средств, позволяющих создать базы данных и управлять данными. СУБД обеспечивает доступ к данным, позволяет выполнять запросы и обеспечивает целостность и безопасность данных. Она также может обеспечивать масштабируемость и производительность базы данных. СУБД является неотъемлемой частью большинства приложений, которые требуют хранения и управления данными [11].

PostgreSQL – это открытая система управления базами данных, которая обладает высокой степенью надежности и устойчивости к сбоям благодаря использованию транзакций и механизмов восстановления данных. Она обладает высоким уровнем безопасности благодаря использованию механизмов аутентификации, авторизации и шифрования данных. PostgreSQL поддерживает множество стандартов SQL, что делает её совместимой с другими СУБД. Кроме того, PostgreSQL позволяет добавлять новые функции и возможности с помощью расширений, что делает её еще более гибкой и мощной [12].

* + 1. Выбор операционной системы

Поскольку язык программирования, среда разработки и СУБД были выбраны кроссплатформенные, нет рамок в выборе операционной системы и рациональнее использовать ту, которая привычнее.

Исходя из этого, для разрабатываемой информационной системы была выбрана операционная система Windows 10. Windows 10 – это операционная система для персональных компьютеров и рабочих станций, разработанная корпорацией Microsoft в рамках семейства Windows NT.

* 1. Описание информационно-логической модели системы и логической модели базы данных

Проектирование – один из важнейших шагов при разработке сложных систем, который крайне часто игнорируется начинающими разработчиками. Зачастую неопытные программисты пытаются удержать всё у себя в голове. Как результат, с таким подходом не будет ни четкого плана действий, ни удобного варианта, в который можно было бы вносить правки без последствий.

Обычно при проектировании система изображается графически, поскольку человеку легче разобраться в таком представлении. Вследствие этого вместо громоздких текстов строятся различные диаграммы для описания систем [13].

* + 1. Диаграмма вариантов использования

Диаграмма вариантов использования (use-case diagram) – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей [13]. Разработку сложной автоматизированной системы принято начинать именно с проектирования диаграммы вариантов использования. Каждый блок из этой диаграммы определяет набор действий, которые могут или должны выполняться проектируемой системой при взаимодействии с соответствующим актором.

На рисунке 5 представлена диаграмма вариантов использования для проектируемой автоматизированной информационной системы.

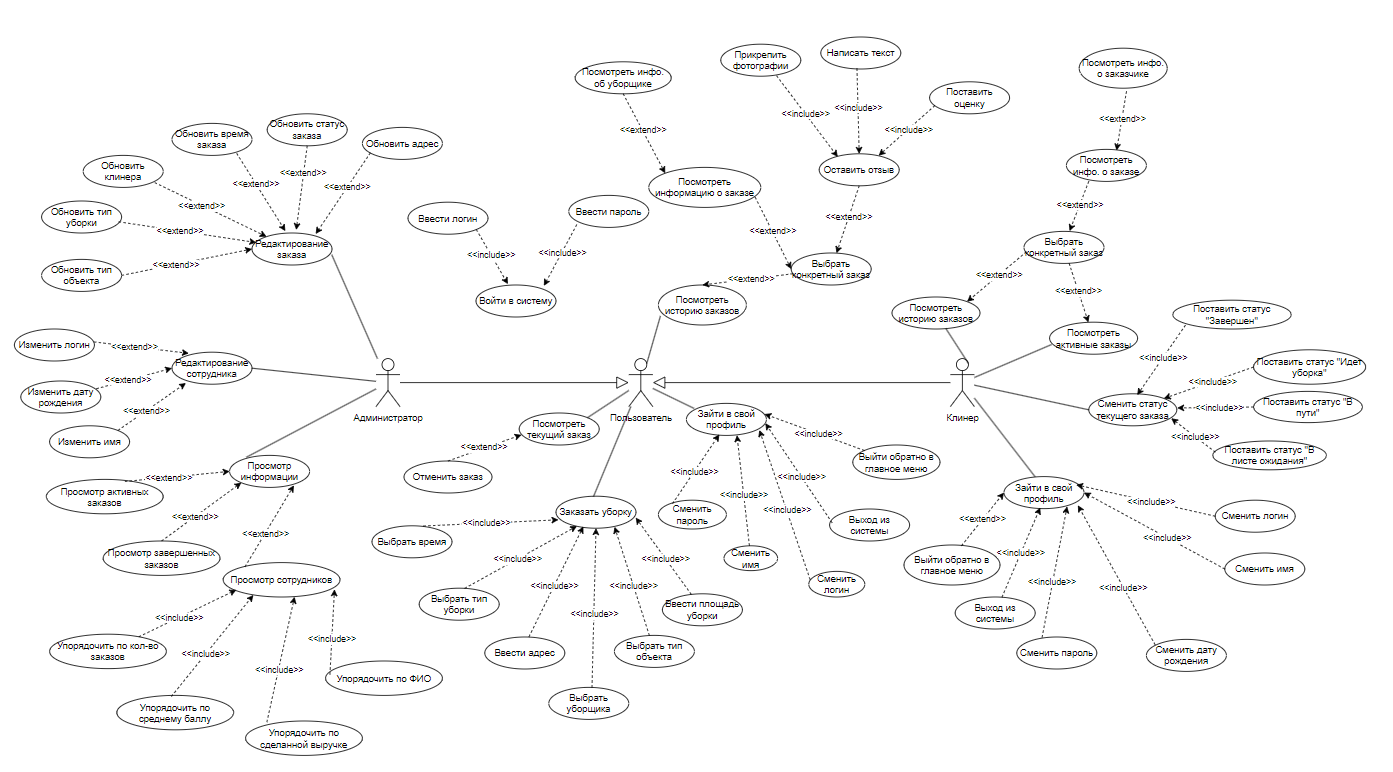


Рисунок 5 – Диаграмма вариантов использования

* + 1. Сценарии использования

Сценарии использования, варианты использования или прецеденты – спецификация последовательностей действий в UML, которые может осуществлять система, подсистема или класс, взаимодействуя с внешними действующими лицами (англ. Actors) [14].

Рассмотрим сценарии использования для нескольких вариантов использования.

Вариант использования: Войти в систему.

Краткое описание: Предоставляет возможность авторизоваться в системе.

Предусловие: Пользователь открыл нужный веб-сайт, на экране форма авторизации.

Актант: Клинер

Основной поток событий:

1. Клинер вводит логин и пароль.
2. Актант нажимает левой кнопкой мыши на кнопку «Войти».

А1: Система выводит сообщение о неправильном формате введённых данных.

А2: Система выводит сообщение о неправильном логине или пароле.

Альтернативы:

А1: Система выводит сообщение об отсутствии введённого логина.

1. Система закрывает форму авторизации.

Постусловие: При успешной авторизации в системе на экране появляется панель управления для сотрудника.

Вариант использования: Оформить заказ на клининг.

Краткое описание: Предоставляет клиенту возможность оформить заказ на уборку помещения.

Предусловие: Клиент открыт нужный веб-сайт, авторизация пройдена от роли клиента и открыта панель управления для пользователей.

Актант: Пользователь.

Основной поток событий:

1. Пользователь нажимает на поле с выбором типа объекта.
2. Система реагирует на нажатие открытием выпадающего списка.

А1: Пользователь нажал на область за пределами списка.

А2: Пользователь выбрал наиболее подходящий вариант.

1. Система закрыла выпадающий список выбора типа объекта.
2. Пользователь нажал на поле для ввода площади уборки.
3. Пользователь вводит площадь уборки.

А3: Система успешно проверила значение площади.

1. Пользователь выбирает дату и время.
2. Пользователь выбирает тип уборки из выпадающего списка

А4: Пользователь нажал на область за пределами списка.

А5: Пользователь выбрал подходящий вариант.

1. Пользователь ввёл адрес уборки.

А6: Система успешно проверила адрес по базе районов и улиц.

1. Пользователь выбирает клинера из выпадающего списка.

А7: Пользователь нажал на область за пределами списка.

А8: Пользователь выбрал подходящий вариант.

1. Пользователь нажал кнопку «Оформить заказ».

А9: Система проверила заказ и занесла в БД.

Альтернативы:

А1: Пользователь нажатием перешёл в другое окно.

А1.1: Система отобразила соответствующее окно.

А6: Система не нашла совпадений в базе районов или улиц.

А6.1: Система выдала сообщение об ошибке.

А9: Система отклонила заказ в силу неправильно введённых данных.

Постусловие: При успешном оформлении заказа в системе пользователь увидит его в списке активных заказов, сам заказ создастся и занесётся в базу данных. Выбранный клинер увидит добавленный заказ.

* + 1. Диаграмма классов системы

Диаграмма классов описывает типы объектов системы и различного рода статические отношения, которые существуют между ними. На диаграммах классов отображаются свойства классов, операции классов и ограничения, которые накладываются на связи между объектами [14].

На рисунке 6 представлена диаграмма классов-сущностей системы.

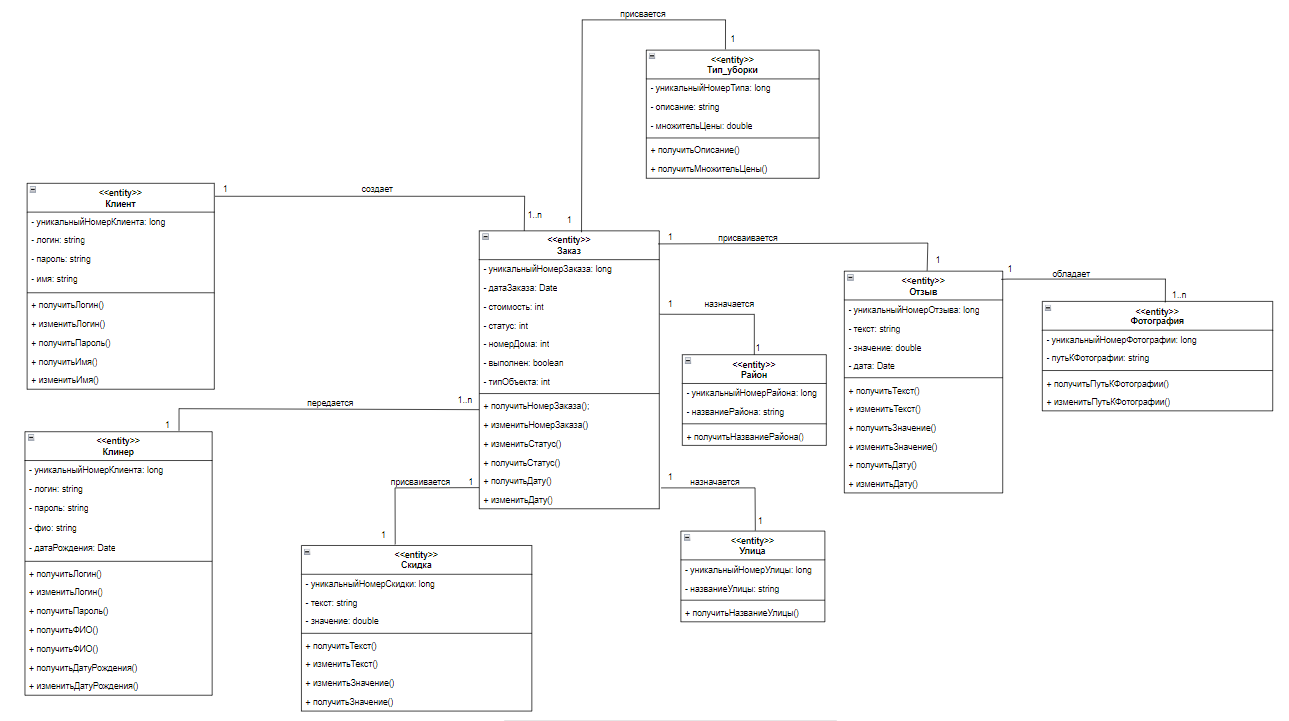


Рисунок 6 – Диаграмма классов-сущностей системы

Класс «Order» отвечает за взаимодействие с таблицей «Order» из БД.

Класс «Customer» отвечает за взаимодействие с таблицей «Customer» из БД.

Класс «Cleaner» отвечает за взаимодействие с таблицей «Cleaner» из БД.

Класс «Sale» отвечает за взаимодействие с таблицей «Sale» из БД.

Класс «Street» отвечает за взаимодействие с таблицей «Street» из БД.

Класс «Area» отвечает за взаимодействие с таблицей «Area» из БД.

Класс «Review» отвечает за взаимодействие с таблицей «Review» из БД.

Класс «Photo» отвечает за взаимодействие с таблицей «Photo» из БД.

Класс «CleanType» отвечает за взаимодействие с таблицей «CleanType» из БД.

На рисунке 7 показана диаграмма классов системы, реализующие всю логику приложения.

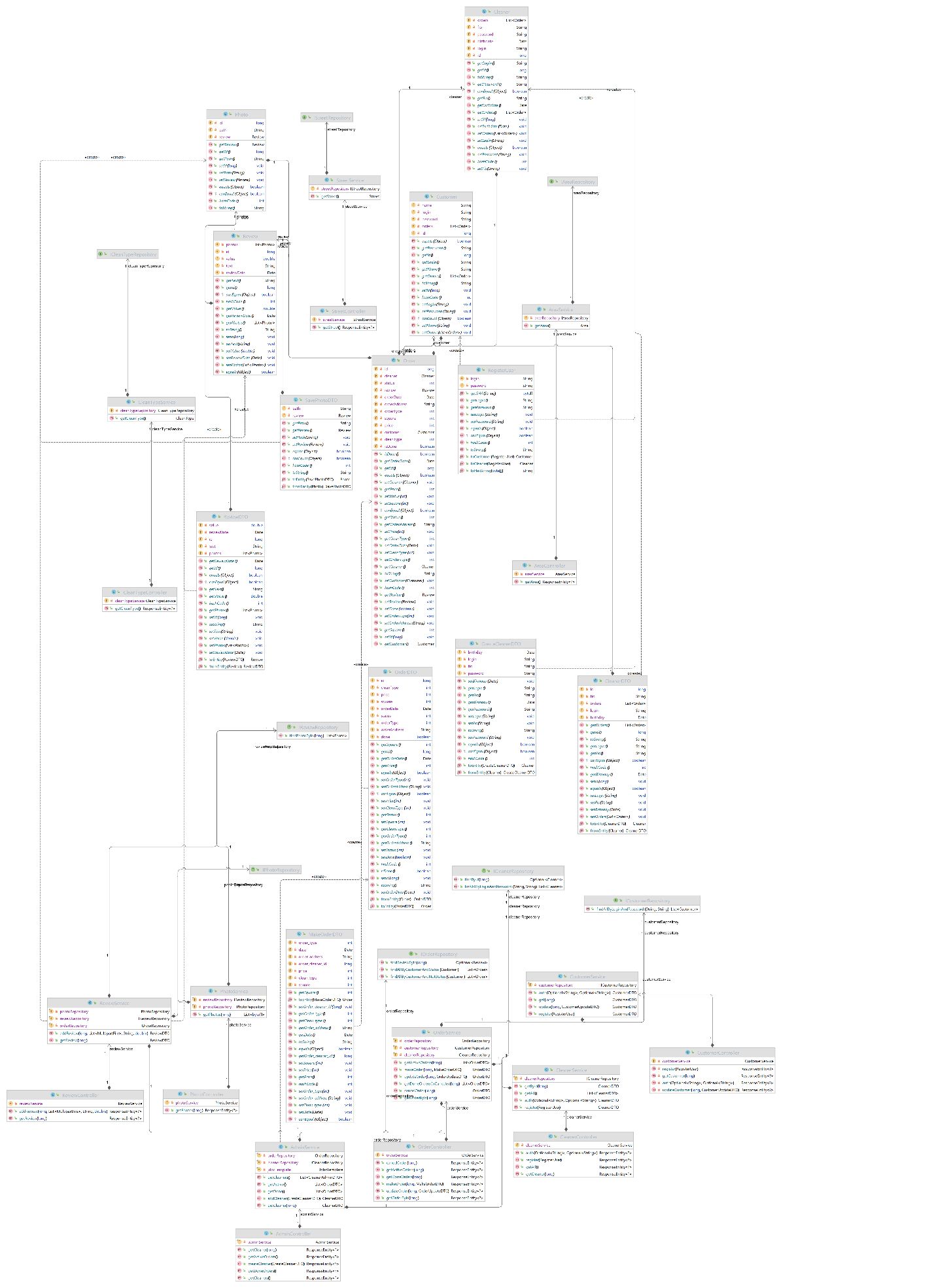


Рисунок 7 – Диаграмма классов системы

Классы «ICustomerRepository», «CustomerService» и «CustomerController» реализуют взаимодействие с сущностью «Клиент» в БД и бизнес-логику, которая затрагивает эту сущность.

Классы «ICleanerRepository», «CleanerService» и «CleanerController» реализуют взаимодействие с сущностью «Клинер» в БД и бизнес-логику, которая затрагивает эту сущность.

Классы «IOrderRepository», «OrderRepository» и «OrderController» реализуют взаимодействие с сущностью «Заказ» в БД и бизнес-логику, которая затрагивает эту сущность.

Классы «IPhotoRepository», «PhotoService» и «PhotoController» реализуют взаимодействие с сущностью «Фотография» в БД и бизнес-логику, которая затрагивает эту сущность.

Классы «ISaleRepository», «SaleService» и «SaleController» реализуют взаимодействие с сущностью «Скидка» в БД и бизнес-логику, которая затрагивает эту сущность.

Классы «ICleanTypeRepository», «CleanTypeService» и «CleanTypeController» реализуют взаимодействие с сущностью «Тип уборки» в БД и бизнес-логику, которая затрагивает эту сущность.

Классы «IAreaRepository», «AreaService» и «AreaController» реализуют взаимодействие с сущностью «Район» в БД и бизнес-логику, которая затрагивает эту сущность.

Классы «IStreetRepository», «StreetService» и «StreetController» реализуют взаимодействие с сущностью «Район» в БД и бизнес-логику, которая затрагивает эту сущность.

Классы «IAdminRepository» «AdminService» и «AdminController» содержат дополнительные методы для работы со всеми сущностями. Данная логика позволяет писать универсальные запросы к базе данных, не ломая архитектуру приложения.

Классы «MakeOrderDTO», «OrderDTO», «CleanerDTO», «ReviewDTO», «SavePhotoDTO», «RegisterUserDTO» представляют собой POJO-сущности, которые сервер отправляет клиенту в виде ответа в случае необходимости.

POJO-объект – это простой Java-объект, не унаследованный от какого-то специфического объекта и не реализующий никаких служебных интерфейсов помимо тех, которые нужны для бизнес-модели [15].

* + 1. Диаграмма состояний

Диаграмма состояний отображает поведение объектов одного класса в динамике, связь состояний объектов с событиями и определяет:

- какие типичные состояние проходит объект;

- какие события ведут к изменению состояния объекта;

- какие действия объект выполняет, когда он получает сообщение об изменении состояния;

- как объекты создаются и уничтожаются (входные и выходные точки диаграммы).

Данная диаграмма полезна при моделировании жизненного цикла объекта. От остальных диаграмм описываемая диаграмма отличается тем, что описывает процесс изменения состояний только одного экземпляра определенного класса – одного объекта, причем объекта реактивного, то есть объекта, поведение которого характеризуется его реакцией на внешние события [16].

Ниже, на рисунке 8, представлена диаграмма состояний для проектируемой системы.

* + 1. Диаграмма деятельности

Диаграмма деятельности – это один из пяти видов диаграмм, применяемых в UML для моделирования динамических аспектов систем. Диаграмма деятельности представляет собой блок-схему, которая показывает, как поток управления переходит от одной деятельности к другой. В отличие от традиционной блок-схемы диаграмма деятельности показывает параллелизм так же хорошо, как и ветвление потока управления.

Моделирование динамических аспектов систем при помощи диаграмм деятельности большей частью подразумевает моделирование последовательных шагов вычислительного процесса [17].

На рисунке 9 представлена диаграмма деятельности для разрабатываемой системы.

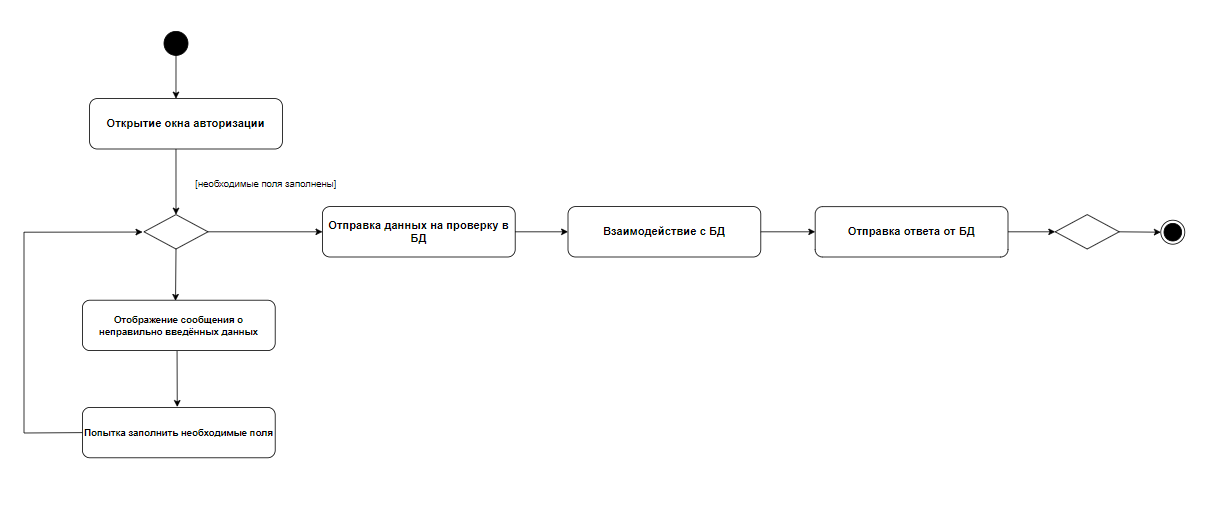


Рисунок 8 – Диаграмма состояний для варианта использования «Авторизация пользователя»

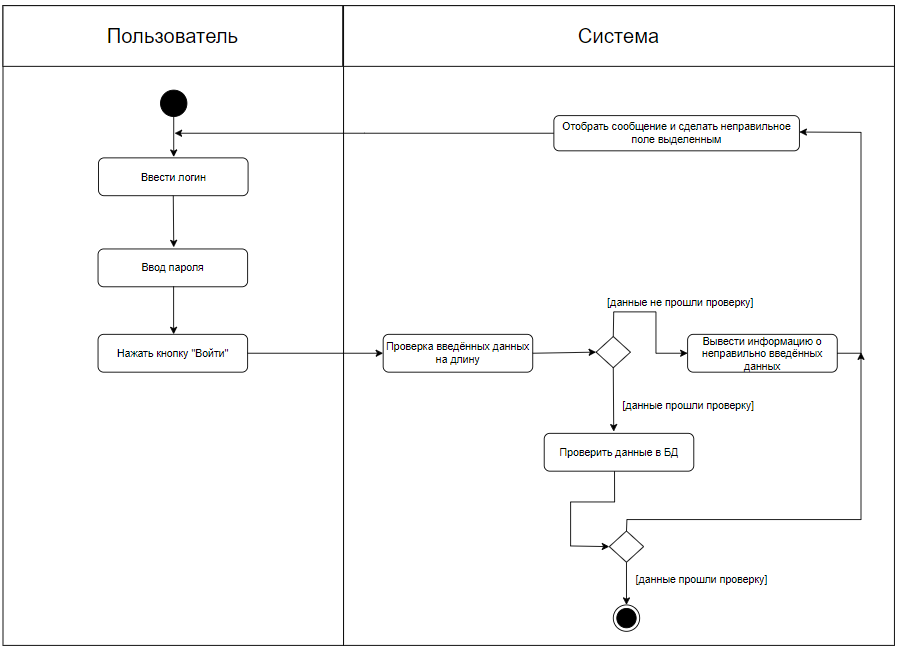


Рисунок 9 – Диаграмма деятельности для варианта использования «Авторизация пользователя»

* + 1. Диаграмма последовательности

Диаграммы последовательностей, используемые разработчиками, моделируют взаимодействия между объектами в едином сценарии использования. Они иллюстрируют, как различные части системы взаимодействуют друг с другом для выполнения функций, а также порядок, в котором происходит взаимодействие при выполнении конкретного случая использования [18].

Основными элементами диаграммы последовательности являются обозначения объектов, вертикальные «линии жизни», отображающие течение времени, прямоугольники, отображающие деятельность объекта или исполнение им определенной функции, и стрелки, показывающие обмен сигналами или сообщениями между объектами [19].

На рисунке 10 изображена диаграмма последовательности для разрабатываемой системы.

* + 1. Диаграмма кооперации

Диаграмма кооперации предназначена для спецификации структурных аспектов взаимодействия. Главная особенность – графически представить, как последовательность взаимодействия, так и все структурные отношения между объектами, участвующими в этом взаимодействии.

На диаграмме кооперации изображаются только отношения между объектами. Последовательность взаимодействия и параллельных потоков определяется с помощью порядковых номеров.

С помощью диаграммы кооперации описывают совокупность объектов, взаимодействующих между собой для выполнения определенной задачи [20].

На рисунке 11 представлена диаграмма кооперация для варианта использования «Авторизация пользователя» разрабатываемой системы.

* + 1. Диаграмма компонентов

Диаграмма компонентов описывает особенности физического представления системы. Определяет архитектуру разрабатываемой системы. Устанавливает зависимости между программными компонентами: исходных, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу [21].

На рисунке 12 представлена диаграмма компонентов для разрабатываемой системы.

* + 1. Диаграмма развёртывания

Диаграмма развёртывания предназначена для представления общей конфигурации или топологии распределенной программной системы и содержит изображение размещения различных артефактов по отдельным узлам системы.

Данная диаграмма предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения.

На рисунке 13 представлена диаграмма развёртывания для разрабатываемой информационной системы.

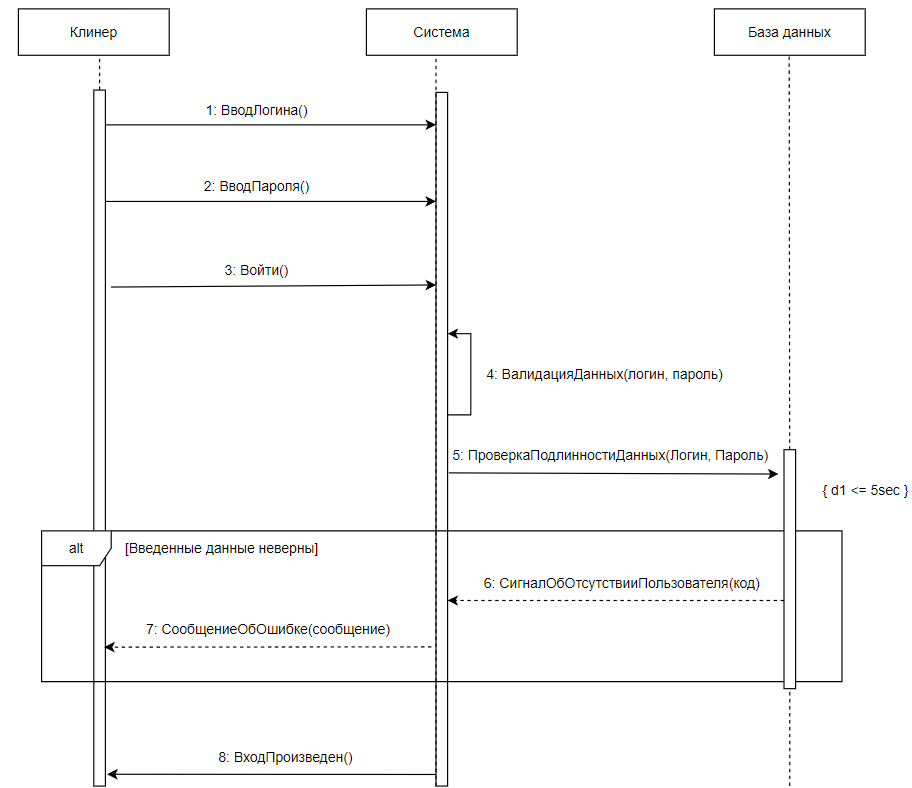


Рисунок 10 – Диаграмма последовательности для варианта использования «Авторизация пользователя»

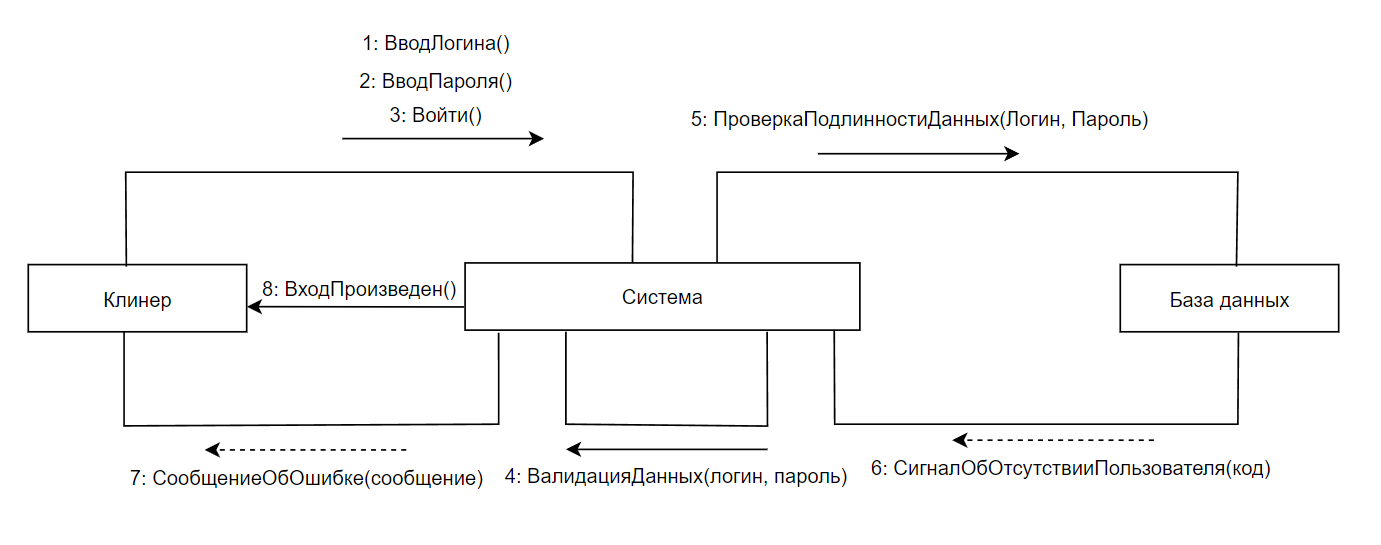


Рисунок 11 – Диаграмма кооперации для варианта использования «Авторизация пользователя»

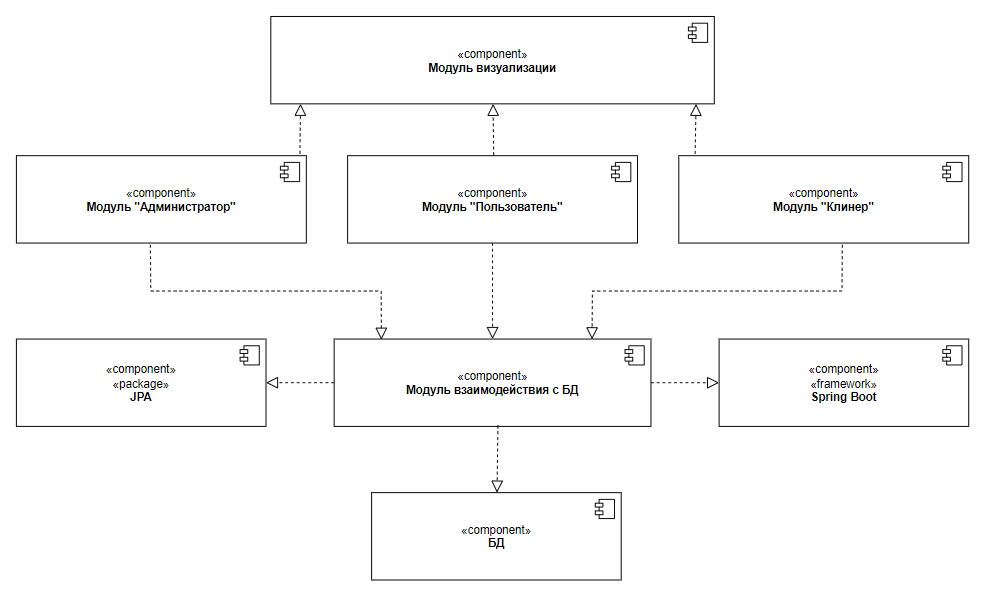


Рисунок 12 – Диаграмма компонентов для разрабатываемой системы

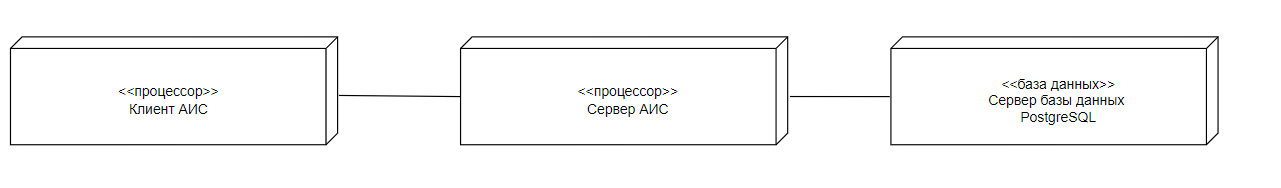


Рисунок 13 – Диаграмма развёртывания для разрабатываемой системы

* 1. Логическая модели баз данных

2.3.1 Описание логической модели баз данных

Логическая модель – это расширение концептуальной модели данных. Она включает в себя все сущности, атрибуты, ключи и взаимосвязи, которые представляют бизнес-информацию и определяют бизнес-правила [22].

Этап моделирование логической модели баз данных является главным элементом баз данных и представляется основным для физической реализации баз данных на устройстве хранения данных.

На рисунке 14 представлена логическая модель базы данных для разрабатываемой системы.

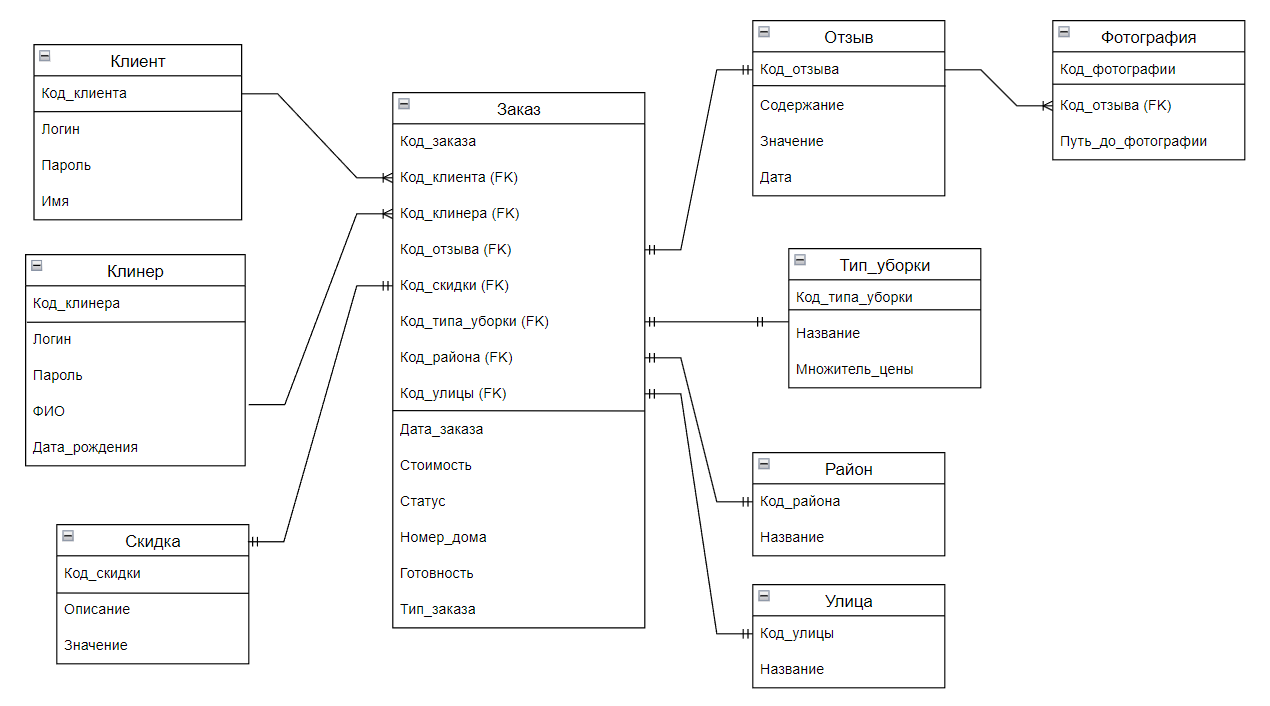


Рисунок 14 – Логическая модель базы данных

2.3.2 Нормализация отношений

Нормализация – это разбиения или декомпозиции исходного отношения на большее количество отношений для устранения нежелательных функциональных зависимостей, которые всегда приводят к возникновению избыточности хранения информации и аномалиям основных операций с данными.

Отношение находится в первой нормальной форме, если все его атрибуты являются простыми, все используемые домены содержат только скалярные значения. Не должно быть повторений строк в таблице.

Отношение находится во второй нормальной форме, если оно находится в первой нормальной форме и каждый не ключевой атрибут неприводимо зависит от первичного ключа.

Отношение находится в третьей нормальной форме, когда оно находится во второй нормальной форме и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа.

Согласно логической модели баз данных, приведённой выше, имеются отношения:

Клиент (Код\_клиента, Логин, Пароль, Имя);

Клинер (Код\_клинера, Логин, Пароль, ФИО, Дата\_рождения);

Скидка (Код\_скидки, Описание, Значение);

Улица (Код\_улицы, Название);

Район (Код\_района, Название);

Тип\_уборки (Код\_типа\_уборки, Название, Множитель\_цены);

Отзыв (Код\_отзыва, Содержание, Значение, Дата);

Фотография (Код\_фотографии, Код\_отзыва, Путь\_до\_фотографии);

Заказ (Код\_заказа, Код\_клинера, Код\_клиента, Код\_отзыва, Код\_скидки, Код\_типа\_уборки, Код\_района, Код\_улицы, Дата\_заказа, Стоимость, Статус, Номер\_дома, Готовность, Тип\_заказа).

Все отношения находятся в первой нормально форме, так как все значения атрибутов атомарные.

Все отношения находятся во второй нормально фотке, так как они находятся в первой нормальной форме и не имеют составных ключей.

Все отношения находятся в третьей нормальной форме, так как они находятся во второй нормальной форме и у них отсутствуют транзитивные зависимости атрибутов, не входящих в ключ от ключа.

* 1. Описание физической модели базы данных

Физическая модель данных включает в себя все необходимые таблицы, столбцы, связи и свойства базы данных для физической реализации баз данных. Производительность базы данных, стратегии индексации, физическое хранилище и денормализация – важные параметры физической модели.

На рисунке 15 представлена физическая модель базы данных разрабатываемой системы.

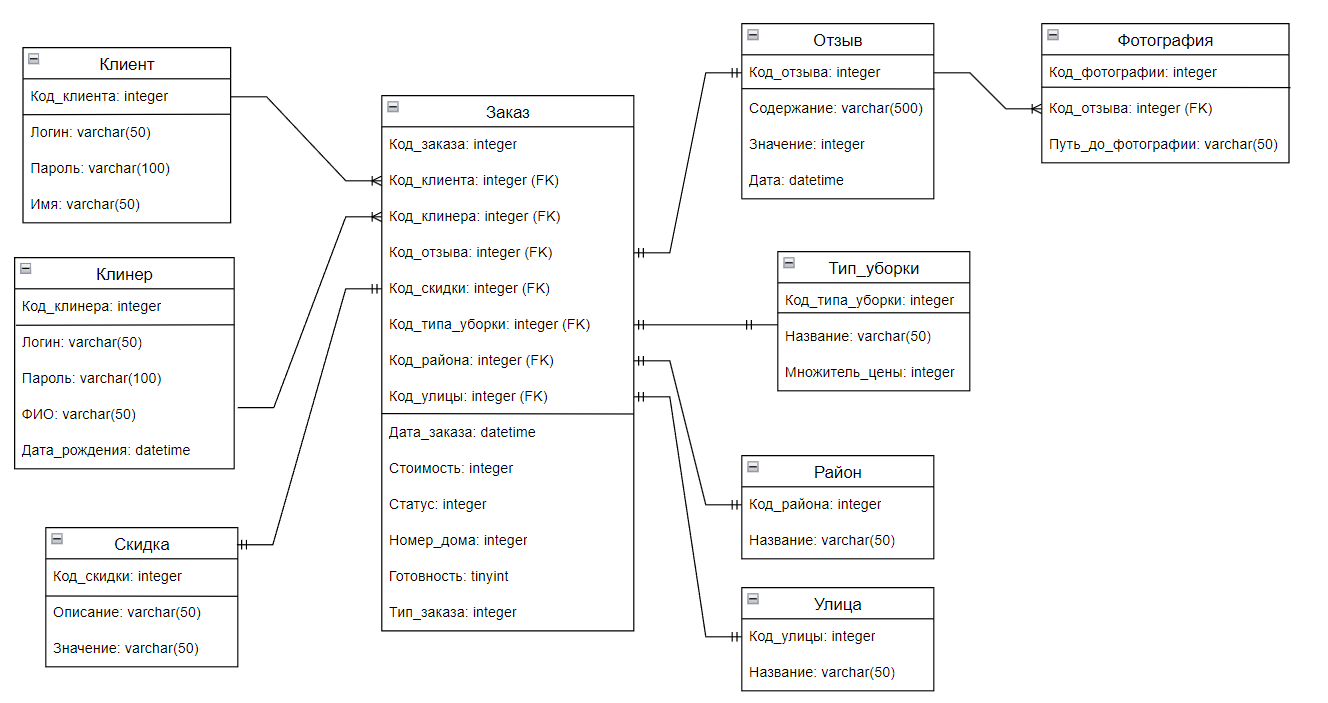


Рисунок 15 – Физическая модель базы данных

* 1. Разработка алгоритмов обработки данных

В ходе разработки автоматизированной информационной системы было составлено и реализовано некоторое количество алгоритмов для решения ряда задач по сохранению, передачи, извлечению или модифицированию данных. Каждый из алгоритмов на входе принимает массив данных, обрабатывает их, записывает необходимую информацию в базу данных и на выходе предоставляет другой массив данных.

Ключевым алгоритмом разрабатываемой системы является алгоритм оформления заказа, поскольку необходимо не только обработать ряд данных, но и отобразить пользователю большое количество изменений. Система должна:

- проверить данные;

- создать заказ;

- присвоить заказу назначенного клинера;

- присвоить заказу назначенного пользователя;

- отобразить новый заказ клинеру;

- отобразить новый заказ пользователю во вкладке «Список активных заказов»;

- очистить поле оформления заказа у пользователя.

На рисунке 16 приведена схема данного алгоритма.

Также немаловажен алгоритм завершения заказа, поскольку по окончанию жизненного цикла основной сущности – заказа необходимо закрыть ряд важных бизнес-процессов, таких как возможность оставить по завершенному заказу и отображение заказа как выполненного в интерфейсе клиента, а клинеру присвоить следующий по очереди заказ. Опишем основные шаги алгоритма:

- клинер завершает заказ;

- заказ отмечается как завершенный;

- помещаем текущий заказ в список выполненных в интерфейсе клинера;

- присваиваем клинеру новый заказ;

- помещаем текущий заказ в список выполненных в интерфейсе клиента;

- включаем клиенту функцию отправки отзыва по данному заказу;

1. Вычислительные эксперименты проверки эффективности программной реализации метода решения задачи
   1. Разработка контрольного примера обработки реальных данных или разработка плана исследования эффективности разработанной программной системы на модельных или реальных данных

Экспериментальные исследования будут заключаться в проверке работоспособности системы.

В качестве контрольного примера рассмотрим процесс оформления заказа, отслеживания его статуса в зависимости от этапа жизненного цикла

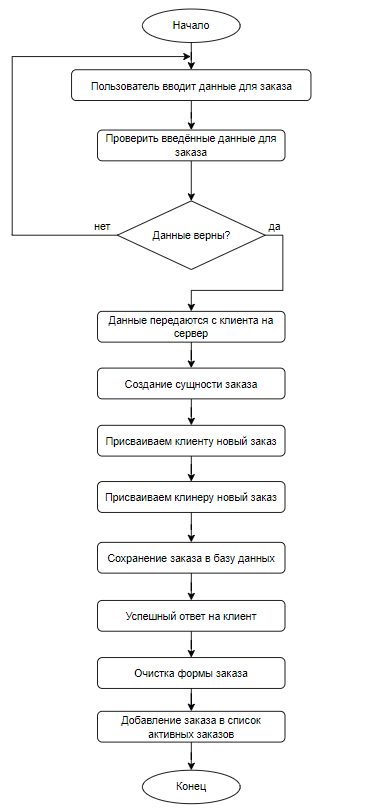


Рисунок 16 – схема алгоритма для оформления заказа

заказа и механизм обратной связи – отправку отзыва о выполненной работе.

3.1.1 Создание заказа

При создании заказа необходимо быть авторизированным в системе под ролью клиента. Затем необходимо в пользовательской панели управления

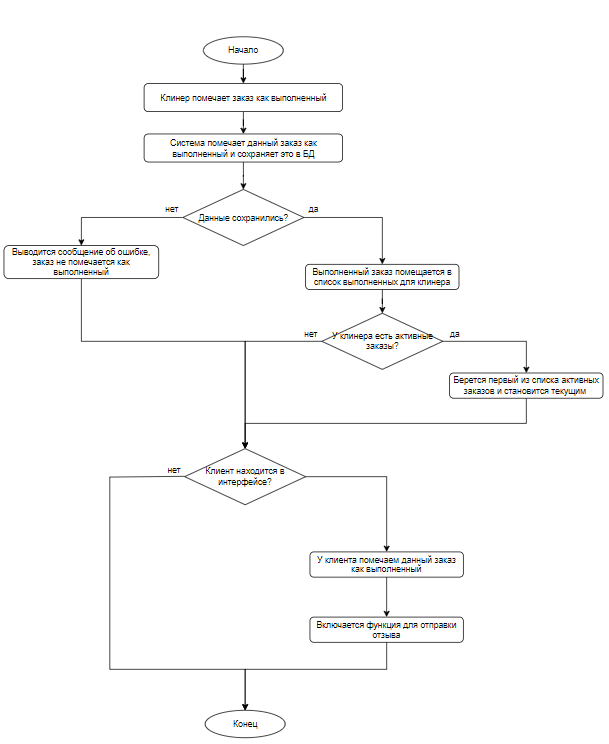


Рисунок 17 – Схема алгоритма завершения заказа

заполнить форму заказа, которая располагается внизу экрана.

Выбираем наиболее подходящий тип объекта из выпадающего меню. Это поле не влияет на стоимость заказа, оно служит для более детальной подготовки клинера к заказу. Если заказ оформлен на частный дом с участком, то клинеру необходимо вдобавок к обычным средствам взять с собой газонокосилку, грабли и другие садовые приборы. Затем система предлагает выбрать наиболее подходящий тип уборки – это свойство показывает, насколько детально и исключительно необходимо убрать помещение, от этого зависит стоимость заказа. За выбором типа уборки следует выбор площади уборки – этот параметр играет роль в финальной стоимости заказа. После необходимо ввести адрес уборки и выбрать клинера из доступных уборщиков. Затем станет доступна кнопка оформления заказа и пользователь сможет отправить заказ в систему.

На рисунке 18 представлена заполненная форма нового заказа.

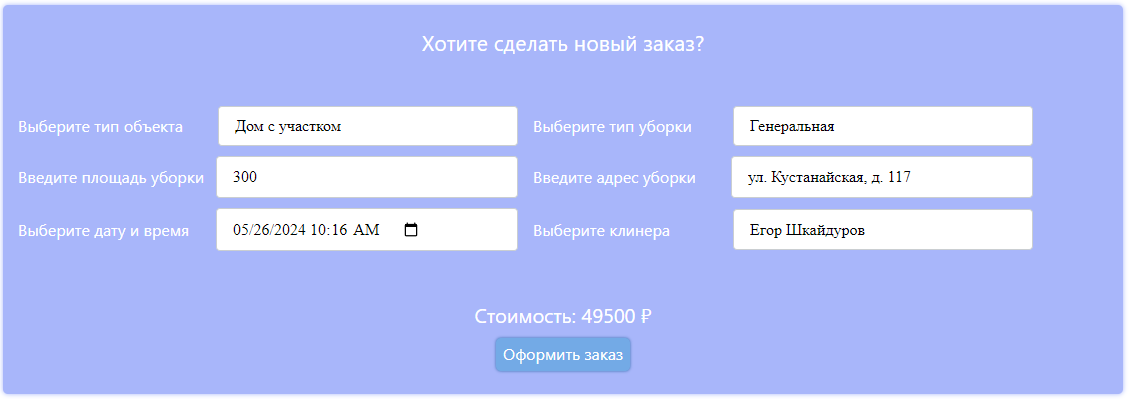


Рисунок 18 – Заполненная форма нового заказа

После нажатия на кнопку «Оформить заказ» он появляется в списке активных заказов у пользователя. В этом окне можно отслеживать текущий статус заказа. Этот этап продемонстрирован на рисунке 19.

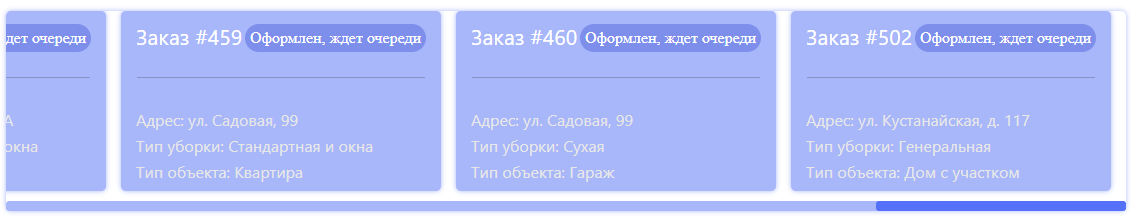


Рисунок 19 – Появление нового заказа в списке активных заказов

3.1.2 Выполнение заказа

Следующим этапом жизненного цикла заказа являются стадии его выполнения уборщиком. Для этого необходимо авторизироваться за клинера, будет в панели управления для сотрудников будет доступно окно текущего заказа, где описана полная информация о заказе, включая цену и текущий статус, а также кнопка для смены статуса заказа.

Кнопка смены статуса заказа называется по-разному, в зависимости от текущего статуса заказа. Название кнопки меняется по следующему алгоритму:

- если статус заказа «Оформлен, ждет очереди», то кнопка носит название «Я в пути!»;

- если статус заказа «Клинер в пути!», то кнопка носит название «Я на месте! Начинаю убираться»;

- если статус заказа «Происходит уборка», то кнопка носит название «Уборка завершена!».

Таким образом, кнопка служит для смены статуса заказа клинером. Изначально статус заказа «Оформлен, ждет очереди», поэтому кнопка называется «Я в пути!». Рисунок 20 демонстрирует окно текущего заказа. На рисунке 21 показан результат нажатия кнопки «Я в пути!» - смена статуса заказа.



Рисунок 20 – Изначальное окно текущего заказа



Рисунок 21 – Окно текущего заказа при смененном статусе

После того, как уборка будет завершена, клинер нажмет кнопку «Уборка завершена». Вследствие этого текущий заказ должен поместиться в раздел «История заказов», который доступен не только клинеру, но и клинту, а также сотруднику будет назначен следующий текущий заказ из списка активных заказов. Рисунок 22 показывает результат этого действия – взят новый заказ, а предыдущей добавлен в историю.



Рисунок 22 – Выдача нового заказа клинеру

* + 1. Отправка отзыва по выполненному заказу

Имея цель оставить отзыв по выполненному заказу, необходимо авторизоваться в роли клиента, который выполненный заказ оформлял и обратить внимание на историю заказов, поскольку в ней отображаются все выполненные или отменённые заказы.

На рисунке 23 представлена панель управления сотрудника с историей заказов, на которой отображён недавно выполненный заказ.

Стоит отметить, что каждая запись из истории заказа активна. По нажатию на конкретный заказ из истории заказа, открывается окно с деталями заказа, а также с возможностью оставить отзыв – этот функционал мы и будем проверять.

При отправке отзыва можно указать текстовое сообщение, оценку работы, а также фотографии, которые будут подтверждать факт плохой или хорошей уборки.

На рисунке 24 показано, как будет выглядеть отзыв, оставленный клиентом.

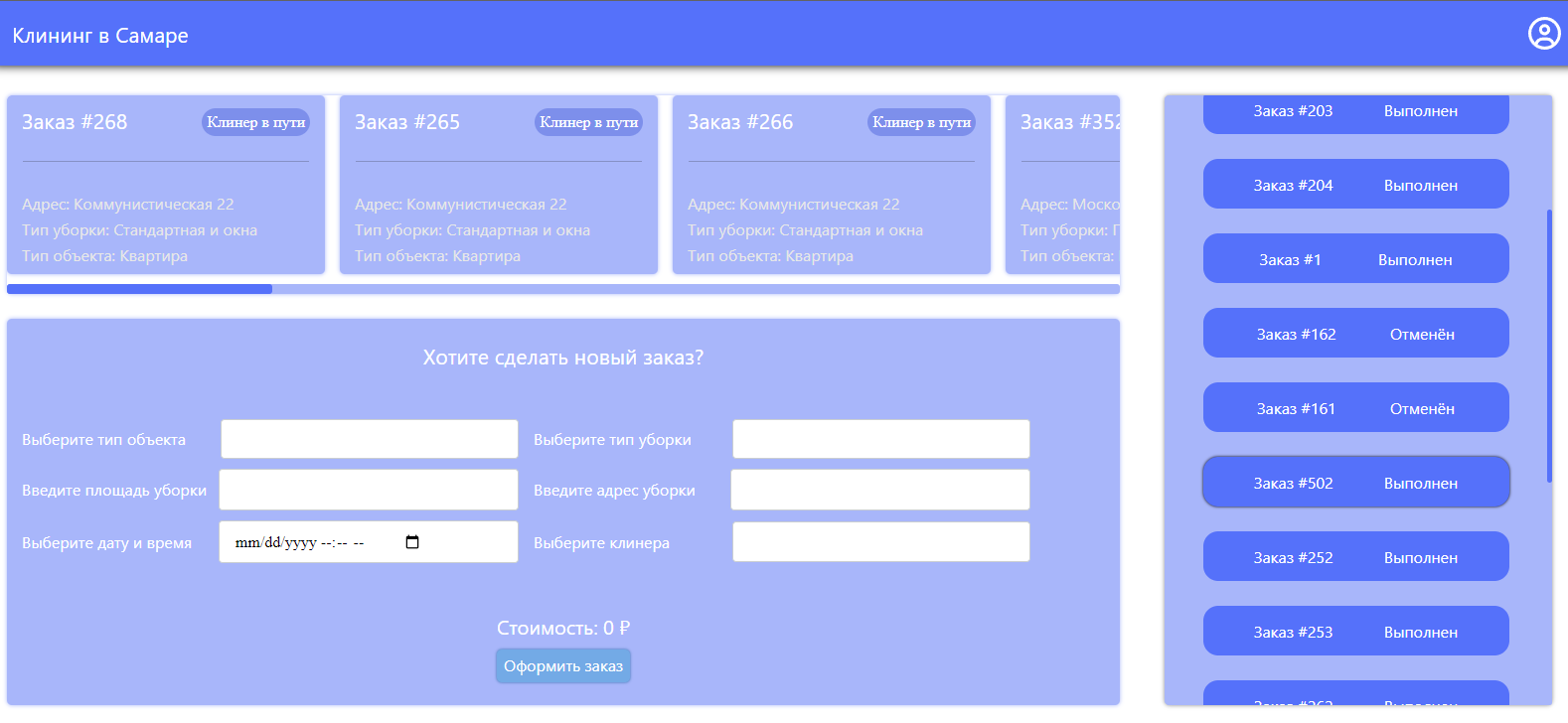


Рисунок 23 – История заказов сотрудника

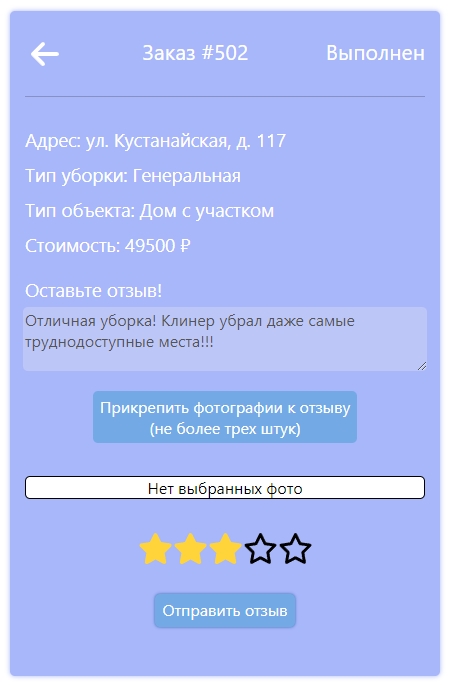


Рисунок 24 – Оставленный отзыв

Также стоит отметить, что отзыв доступен клинеру. Каждый сотрудник может просматривать отзывы в своей истории заказов, чтобы анализировать плюсы или минусы. Оценка, поставленная за отзыв, влияет на репутацию сотрудника, поскольку в системе также предусмотрена роль администратора с возможностью тщательного анализа работы каждого сотрудника.

* + 1. Анализ работы администратором

По окончанию жизненного цикла заказа, он не перестает учитываться в системе. Администратору предоставляется возможность просмотра всех сотрудников с основной выдержкой. Оставленный отзыв влияет на значение среднего рейтинга.

На рисунке 25 представлена таблица, доступная из панели администратора, в которой автоматически выносятся основные данные по всем сотрудникам.



Рисунок 25 – Аналитика по сотрудникам

* 1. Проверка эффективности разработанной программной системы для решения поставленной задачи

Завершающим этапом разработки программной системы является проверка эффективности разработанного приложения. Для этих целей обычно проводится ряд функциональных тестов и тестов на производительность. В силу того, что функциональность была проверена в предыдущей главе, настало время уделить внимание производительности и надёжности.

Для описанной цели будет использоваться программное средство Postman, которое позволяет отправлять HTTP-запросы, отслеживая статус ответа, время ответа и тело ответа.

3.2.1 Проверка одиночного запроса

Первым этапом проверки послужит обычный GET HTTP-запрос на сервер. Для примера, отправим запрос для получения администратором всех выполненных заказов. После выполнения запроса становятся доступны результаты, в данном случае пришел успешный ответ за 29 миллисекунд, заказы были получены.

Рисунок 26 демонстрирует результат выполнения запроса.

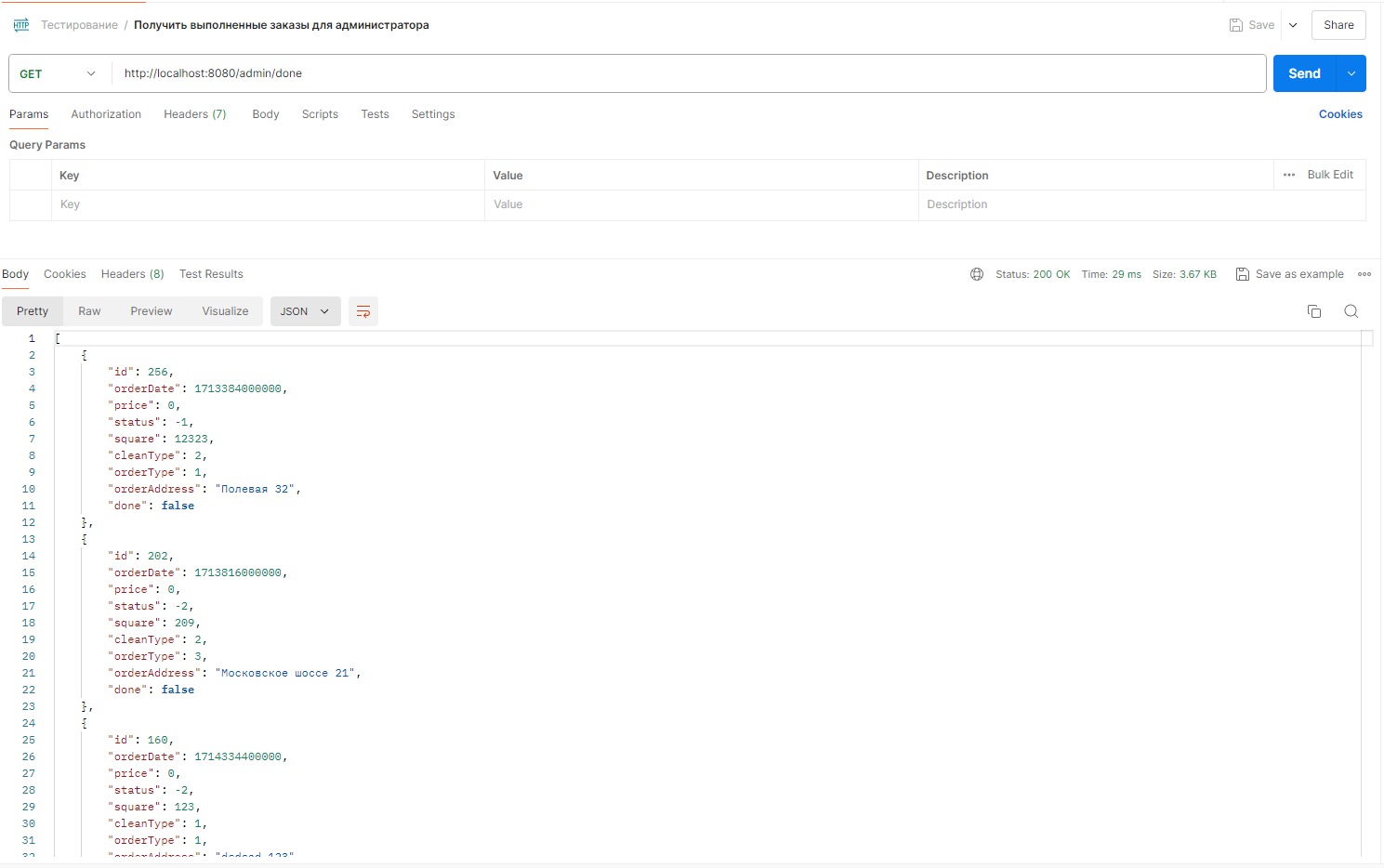


Рисунок 26 – Одиночный запрос к серверу

3.2.2 Нагрузочное тестирование

В описанное ранее программе – Postman – есть возможность запускать неограниченное количество запросов к одному серверу одновременно. Для этого необходимо поместить нужный запрос в коллекцию, а затем настроив количество итераций отправки запросов и задержку между ними, запустить тестирование.

Данный тест приближает систему к условиям реального пользования, поскольку реальных пользователей системы будет больше, чем один и выполнять они будут большое количество запросов.

В качестве первого тестирования запустим 50 итераций без задержки – это будет означать 50 непрерывных обращений к серверу. Результат выполнения отличный: каждый запрос выполнялся за 5 миллисекунд, а суммарно 50 запросов было выполнено за 4 секунды и 325 миллисекунд.

Рисунок 27 демонстрирует результаты такого тестирования.

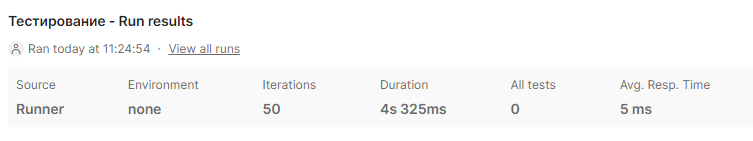


Рисунок 27 – нагрузочное тестирование из пятидесяти итераций

Теперь для большего приближения к реальным условиям будем проводить тестирование на разных запросах. Для этого создадим четыре GET-запроса и запустим тестирование на 10 итераций без задержек.

Результат продемонстрирован на рисунке 28. Здесь ситуация несколько отличается от предыдущей, поскольку время ответа, в среднем, составило 18 миллисекунд. Однако это все равно крайне хороший показатель времени.

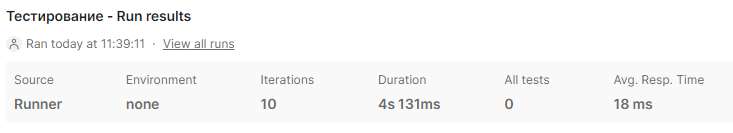


Рисунок 28 – нагрузочное тестирование из 10 итераций по 5 запросов

Представленные результаты демонстрируют крайне стабильную и эффективную работу системы при реальных нагрузках.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была разработана автоматизированная информационная система заказа клининга.

Система отвечает всем требованиям задания и реализует следующие функции:

- хранение данных о клиентах, сотрудниках, заказах, отзывах, фотографиях, улицах, районах;

- добавление, изменение и удаление сотрудников, заказов, а также изменение статуса заказов и информации о сотрудниках;

- просмотр информации о заказах, отзывах, сотрудниках, пользователях;

- хранение необходимых системе данных при помощи СУБД PostgreSQL.

Система разработана на языке программирования Java в среде разработки IntelliJ IDEA под управлением Windows 10. Вся работа с системой производится через Web-интерфейс, реализованный на фреймворке Angular. Хранение данных осуществляется при помощи СУБД PostgreSQL.

Разработан логический проект системы по методологии UML.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Вики. Предметная область (Object domain) [Электронный ресурс]. URL – <https://wiki.loginom.ru/articles/object-domain.html>
2. Википедия. Уборка [Электронный ресурс]. URL - <https://ru.wikipedia.org/wiki/%D0%A3%D0%B1%D0%BE%D1%80%D0%BA%D0%B0>
3. Википедия. Обратная связь (коммуникации) [Электронный ресурс]. URL - <https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D1%81%D0%B2%D1%8F%D0%B7%D1%8C_(%D0%BA%D0%BE%D0%BC%D0%BC%D1%83%D0%BD%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B8)>
4. Обратная связь [Электронный ресурс]. URL - <https://roistat.com/rublog/obratnaja-svjaz/>
5. ООО «Первая клининговая компания» [Электронный ресурс]. URL - <https://klining-msk.ru/>
6. Что такое Java? [Электронный ресурс]. URL - <https://aws.amazon.com/ru/what-is/java/>
7. Введение в Java [Электронный ресурс]. URL - <https://metanit.com/java/tutorial/1.1.php>
8. Введение в Spring Boot: создание простого REST API на Java [Электронный ресурс]. URL - <https://habr.com/ru/articles/435144>
9. Postman: Основы тестирования API и первые шаги с инструментом [Электронный ресурс]. URL - <https://habr.com/ru/companies/vk/articles/750096/>
10. Eclipse (среда разработки) [Электронный ресурс]. URL - <https://ru.wikipedia.org/wiki/Eclipse_(%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8)>
11. ItGlobal. СУБД (Система управления базами данных) [Электронный ресурс]. URL – <https://itglobal.com/ru-ru/company/glossary/subd-sistema-upravleniyabazami-dannyh/>
12. PostgreSQL — объектно-реляционная система управления базами данных [Электронный ресурс]. URL - <https://web-creator.ru/articles/postgresql>
13. Использование диаграммы вариантов использования UML при проектировании программного обеспечения [Электронный ресурс]. URL - <https://habr.com/ru/articles/566218/>
14. Systems Engineering Thinking Wiki. Сценарий использования [Электронный ресурс]. – <http://sewiki.ru/Сценарий_использования>
15. Создание POJO объекта [Электронный ресурс]. URL - <https://devcolibri.com/unit/%D1%83%D1%80%D0%BE%D0%BA-8-%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-pojo-%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%B0-user-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0-%D1%81-view-%D0%B8%D0%B7-java-2/>
16. Flex berry. Диаграмма состояний (Statechart diagram) [Электронный ресурс]. – <https://flexberry.github.io/ru/fd_statechart-diagram.html>
17. Унифицированный язык моделирования UML. Понятие диаграммы деятельности [Электронный ресурс]. URL - <https://it.kgsu.ru/UML/uml_0156.html>
18. Учебное пособие по диаграммам последовательности: полное руководство с примерами [Электронный ресурс]. URL - <https://creately.com/blog/ru/%D0%B4%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0/%D1%83%D1%87%D0%B5%D0%B1%D0%BD%D0%BE%D0%B5-%D0%BF%D0%BE%D1%81%D0%BE%D0%B1%D0%B8%D0%B5-%D0%BF%D0%BE-%D0%BF%D0%BE%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B9/>
19. Диаграмма последовательности [Электронный ресурс]. URL - <https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_%D0%BF%D0%BE%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D0%B8>
20. Диаграмма кооперации [Электронный ресурс]. URL - <https://portal.tpu.ru/SHARED/r/RYBALKA/academic/mct/TabMCTLect/MCTLect4.pdf>
21. Диаграмма компонентов [Электронный ресурс]. URL - <https://portal.tpu.ru/SHARED/r/RYBALKA/academic/mct/TabMCTLect/MCTLect6.pdf>
22. Моделирование данных: обзор [Электронный ресурс]. URL - <https://habr.com/ru/articles/556790/>

ПРИЛОЖЕНИЕ А

Руководство пользователя

А.1 Назначение системы

Автоматизированная информационная система работы клининговой компании – программное обеспечение для автоматизации процессов заказа клининга. Система обеспечивает работу с заказами, с клинерами, с заказчиками и предоставляет интерфейс управления для администраторов с функцией просмотра текущего бизнес-состояния: активные и выполненные заказы, сотрудники.

А.2 Условия работы системы

Для корректной работы системы необходимо наличие соответствующих программных и аппаратных средств.

1. Требования к обеспечению клиентской части:

− тип ЭВМ: x86-64 совместимый;

− объем ОЗУ – не менее 2 Гб;

− объем свободного дискового пространства – не менее 50 Гб;

− клавиатура или иное устройство ввода;

− мышь или иное манипулирующее устройство.

− процессор Intel Pentium не менее 1,5 ГГц (от 800 ГГц);

− операционная система Windows 7 и выше;

− СУБД – PostgreSQL 16.

1. Требования к обеспечению клиентской части:

− тип ЭВМ: x86-64 совместимый;

− объем ОЗУ – не менее 2 Гб;

− объем свободного дискового пространства – не менее 21 Гб;

− клавиатура или иное устройство ввода;

− мышь или иное манипулирующее устройство.

− процессор Intel Pentium не менее 1,5 ГГц (от 800 ГГц);

− операционная система Windows 7 и выше;

− интернет-браузер – Mozilla Firefox 55.0 и выше, Google Chrome 58 и выше, Яндекс Браузер 17.3 и выше.

А.3 Установка системы

Система поставляется в виде архива, который необходимо распаковать в любую директорию на дисковом пространстве. После чего необходимо:

− установить СУБД PostgreSQL 16 и пройти все необходимые этапы;

− создать базу данных «cleanApp»;

− открыть файл .env в распакованной папке server;

− вбить свои данные для подключения к базе данных;

− установить jdk 17 версии;

− находясь в папке server, необходимо открыть командную строку;

− запустить серверное приложение командой «java –jar cleanApp.jar»;

− далее необходимо установить node.js и npm с официального сайта;

− зайти в папку client, открыть там терминал;

− выполнить команду «npm install»;

− дождавшись завершения загрузки, выполнить команду «npm start».

А.4 Описание операций

После запуска приложения откроется форма с начальным окном АИС, где будет доступна регистрация или авторизация (рисунок А.1)

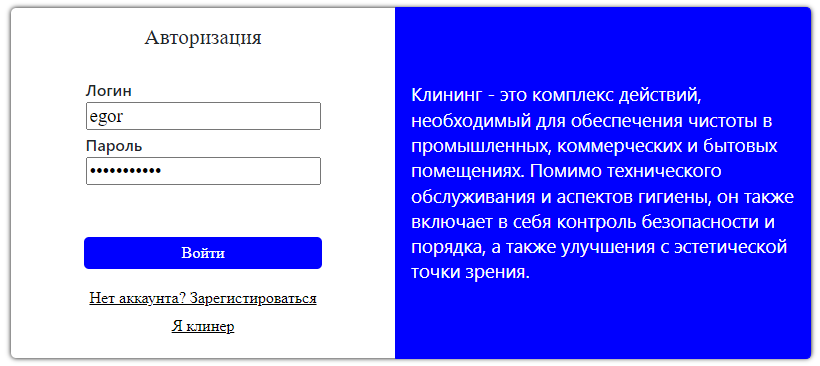


Рисунок А.1 – Начальное окно АИС

Пользователю предоставляется возможность выбрать роль из доступных (клинер и клиент). В случае с выбором клиентской роли система предоставит возможность авторизоваться или зарегистрироваться. Однако, при выборе роли клинера, будет доступна только функция авторизации. Поскольку в системе предусмотрена еще одна роль – администратор, функции которой включают добавление учётных записей для новых сотрудников. В случае необходимости авторизироваться под администраторской ролью, необходимо ввести логин и пароль, заранее предусмотренный разработчиком. При вводе этих данных, независимо от выбранной роли, пользователь попадет в панель администратора.

Авторизировавшись под клиентом, становится доступна панель управления для клиентов, в котором предоставляется функционал для отслеживания активных заказов, открытия истории выполненных или отмененных заказов, оформления нового заказа, а также вход в личный кабинет пользователя, где появляется возможность сменить логин, пароль, имя.

Рисунок А.2 демонстрирует главную панель клиента.

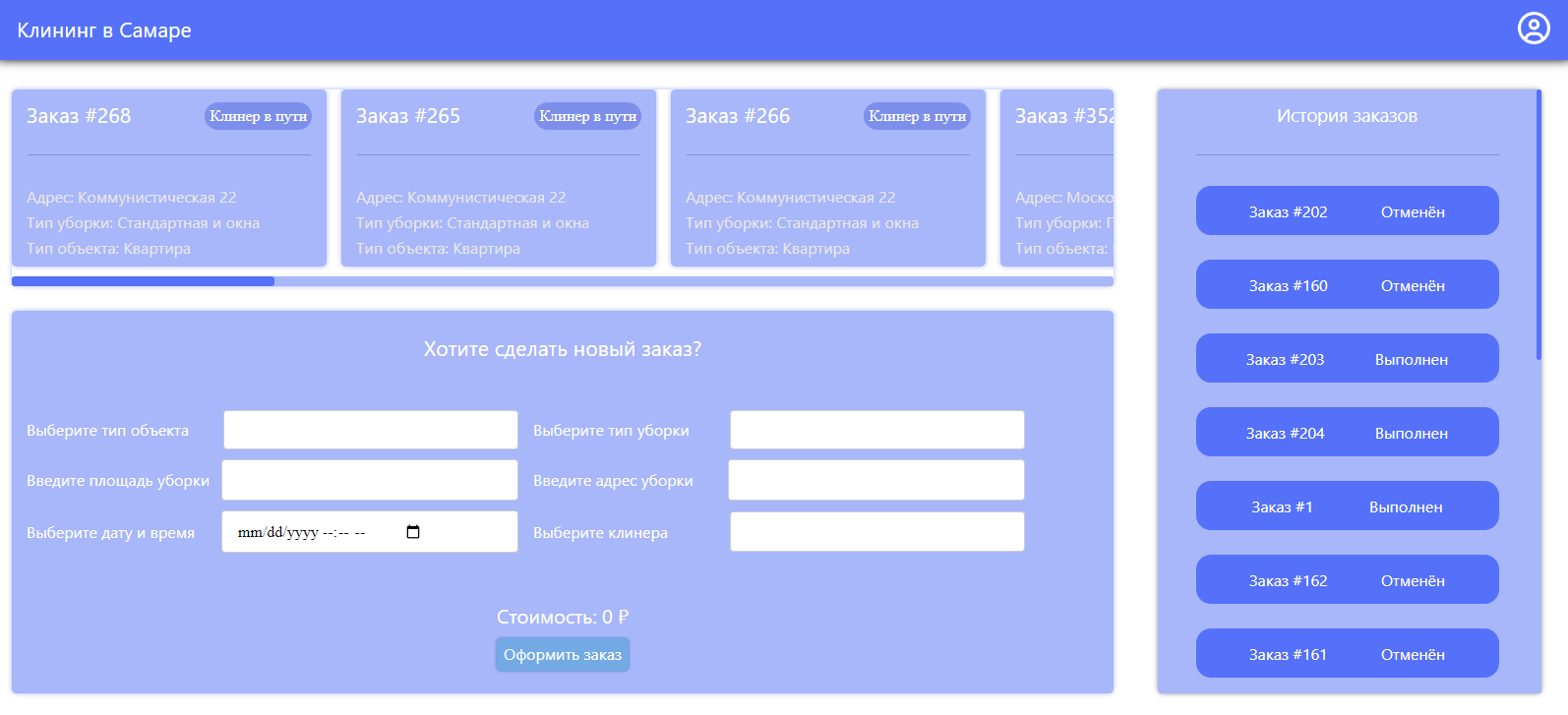


Рисунок А.2 – Главная панель клиента

Каждый заказ из списка активных заказов – это реактивное окошко, при нажатии на которое можно получить более подробную информацию о заказе, а также отменить заказ при необходимости. Рисунок А.3 это визуализирует.



Рисунок А.3 – Подробная информация о заказе

При отмене заказа он сразу же попадет в историю заказов. История заказов – это функционал, содержащий выполненные или отмененные заказы. Каждый элемент в этом меню также реактивный, поэтому запросто можно открыть любой интересующий заказ. Результатом этого действия станет переход на страницу выбранного заказа, в которой будет доступна основная информация по заказе, а также функционал для отправки отзыва по заказу. В случае, если отзыв уже отправлен – форма ввода отзыва будет заблокирована.

На рисунке А.4 представлен заказ, открытый из истории заказов, на который уже был оставлен отзыв.

Из интерфейса для клиента становится доступен самый главный функционал – оформление заказа. Для этого необходимо заполнить все требуемые поля в данной форме и нажать кнопку «Оформить заказ». В случае успешного оформления заказа в системе, пользователь сможет наблюдать новый заказ в списке активных заказов, соответственно сможет его и отменить. Также выбранному клинеру придет событие о добавлении нового заказа на его имя, поэтому список активных заказов в интерфейсе клинера также пополнится новым заказом.

На рисунке А.5 представлена заполненная форма для нового заказа.

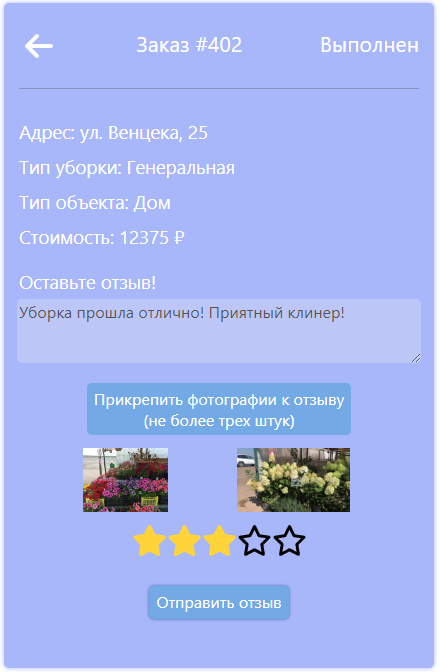


Рисунок А.4 – Выполненный заказ из истории заказов

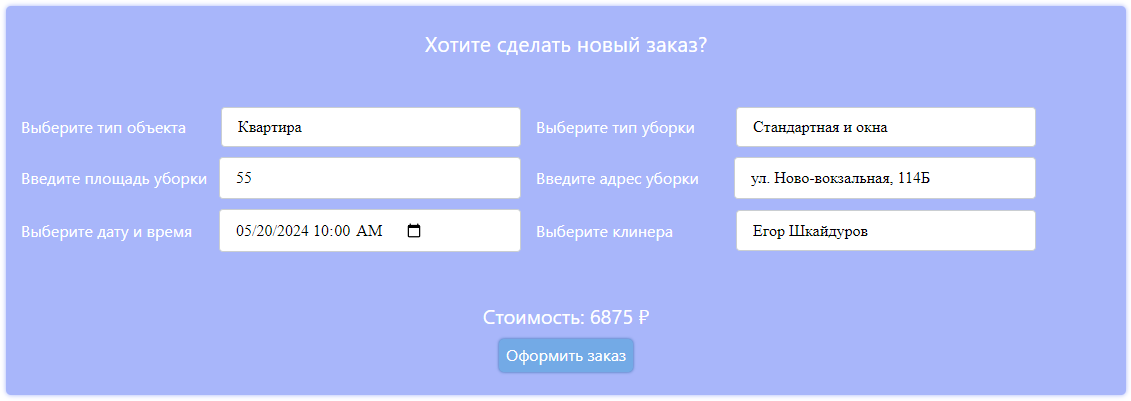


Рисунок А.5 – Оформление нового заказа

Последняя из доступных возможностей для пользователя остается в разделе «Профиль». Благодарю этому функционалу пользователь может менять свой логин, пароль и имя пользователя. Выход из системы также осуществляется через данное окно. Рисунок А.6 демонстрирует описанный функционал.

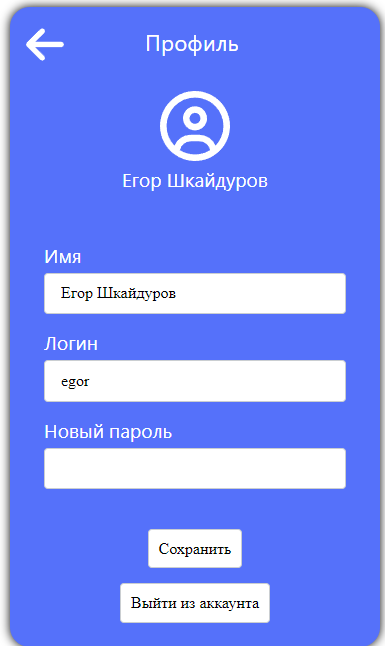


Рисунок А.6 – Профиль пользователя

Следующие функции системы доступны из другого интерфейса – интерфейса клинера. Для этого авторизовываемся в системе и попадаем в панель управления для сотрудников, которая показана на рисунке А.7.

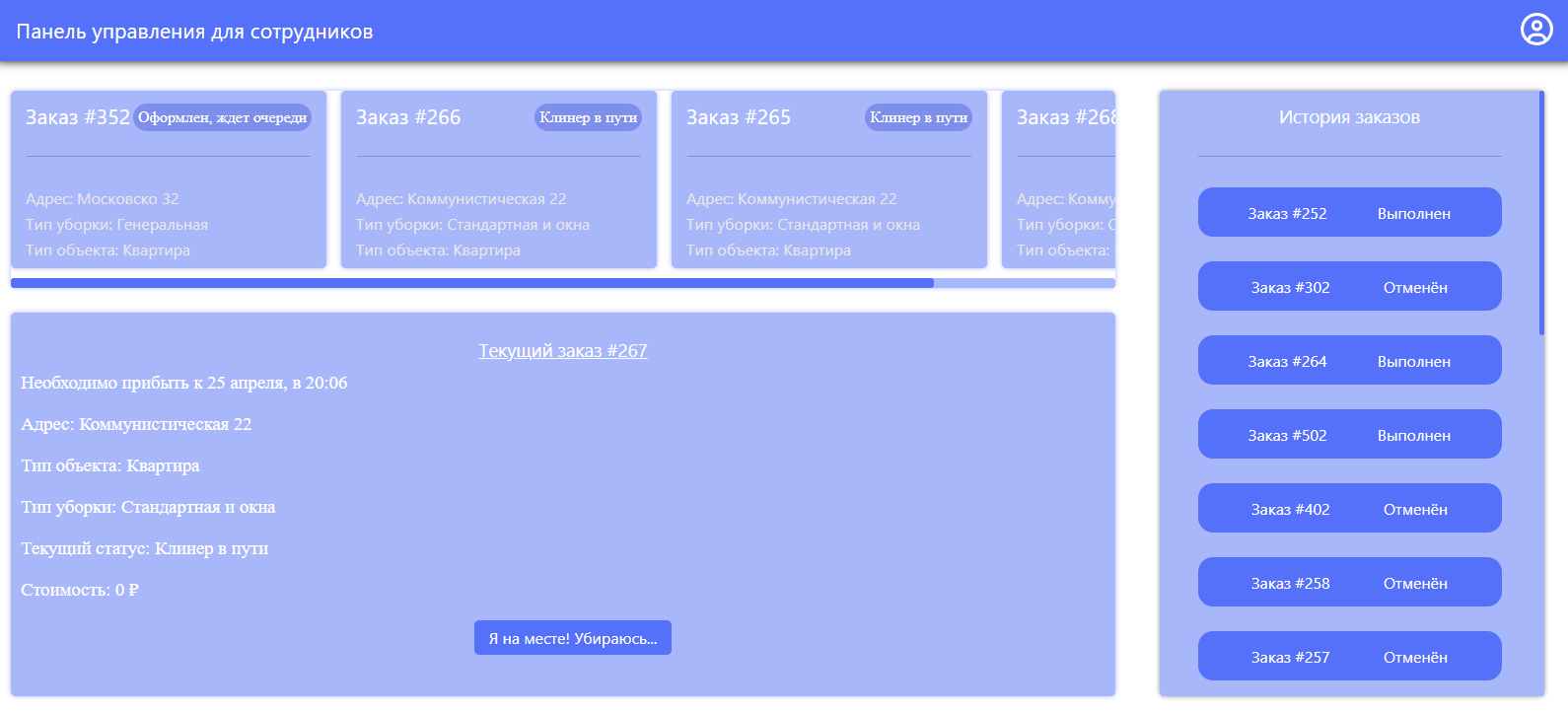


Рисунок А.7 – Панель управления для сотрудника

Данная панель имеет довольно похожий функционал с панелью управления для клиента: доступен список активных заказов и история выполненных заказов конкретного клинера. Отличительная черта роли клинера заключается в функции изменения статуса жизненного цикла заказа, то есть смены статуса заказа. Форма активного заказа, расположенная слева снизу, как раз отвечает за эту функцию.

Из этой формы доступно всего одно действие – нажатие кнопки смены статуса заказа. Стоит отметить, что название кнопки меняется в зависимости от статуса заказа, а именно по следующему алгоритму:

- если статус заказа «Оформлен, ждет очереди», то кнопка носит название «Я в пути!»;

- если статус заказа «Клинер в пути!», то кнопка носит название «Я на месте! Начинаю убираться»;

- если статус заказа «Происходит уборка», то кнопка носит название «Уборка завершена!».

Изначально статус заказа «Оформлен, ждет очереди», поэтому кнопка называется «Я в пути!». Рисунок А.8 демонстрирует окно текущего заказа. На рисунке А.9 показан результат нажатия кнопки «Я в пути!» - смена статуса заказа.



Рисунок А.8 – Изначальный статус заказа

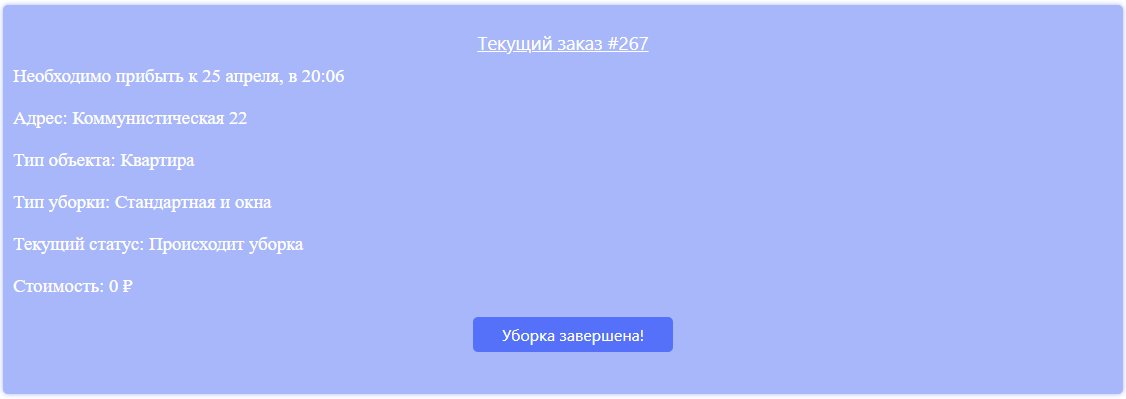


Рисунок А.9 – Предпоследний статус заказа

После нажатия на кнопку «Уборка завершена!» заказ исчезнет из этой формы и попадет в историю заказов, как для клинера, так и для клиента. На место текущего заказа встанет другой заказ из активных заказов.

Рисунок А.10 демонстрирует взятие нового текущего заказа клинером после завершения предыдущего.

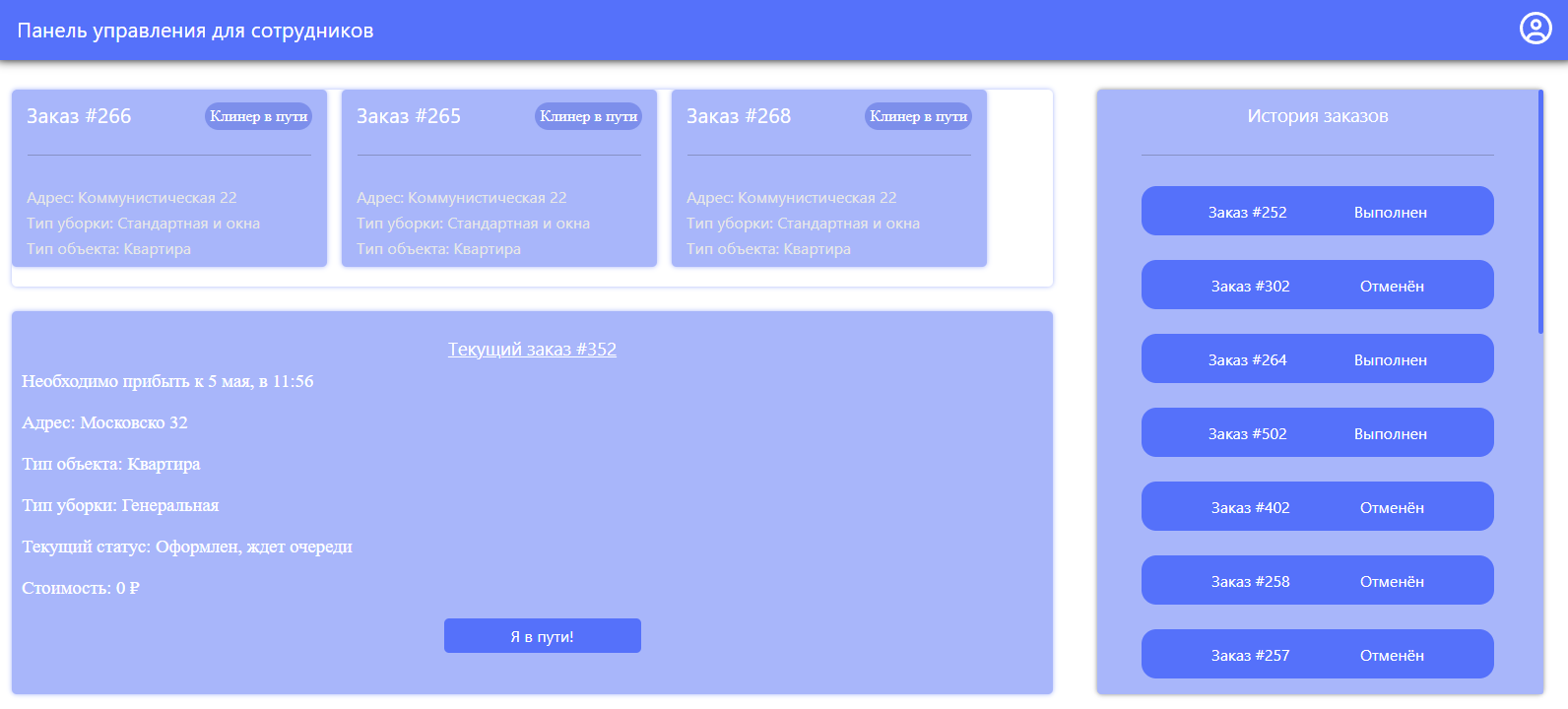


Рисунок А.10 – Новый текущий заказ

Стоит отметить, что история заказов клинера также реагирует на нажатия, поэтому есть возможность открыть любой заказ и посмотреть детали, а также наличие оставленного отзыва.

Следующая роль, присутствующая в системе, – роль администратора. После авторизации под этой ролью, становится доступна уже вполне знакомая панель управления, представленная на рисунке А.11.



Рисунок А.11 – Панель управления администратора

Администратору доступен список всех активных и выполненных или отмененных заказов, не зависимо от клинера или клиента.

Однако ключевым функционалом данной роли является просмотр таблицы сотрудников, которая содержит в себе всех клинеров системы, а также следующие параметры:

- номер сотрудника;

- ФИО;

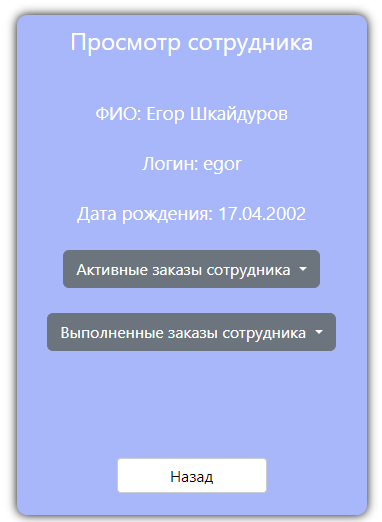
- средний рейтинг;

- количество выполненных заказов;

- выручка.

Таблица имеет возможность отсортировать сотрудников по каждому параметру по возрастанию или убыванию. Это крайне важная деталь для анализа работы сотрудников.

Каждая запись по сотруднику в таблице реагирует на нажатие левой кнопкой мыши. Выбрав конкретного сотрудника, откроется его карточка, где будет доступна информация о сотруднике, а также его выполненные заказы. Рисунок А.12 демонстрирует это функцию.



А.12 – Функционал просмотра сотрудника

Доступно к просмотру его активные заказы, по нажатию на соответствующую кнопку, будет открыто выпадающее меню, показанное на рисунке А.13.

По такому же принципу можно открыть выполненные заказы сотрудника, рисунок А.14 демонстрирует это.

Каждый заказ из обоих выпадающих список также реагирует на нажатие левой кнопкой мыши, при нажатии администратор перейдет в выбранный заказ. На рисунке А.15 показан результат этого действия.

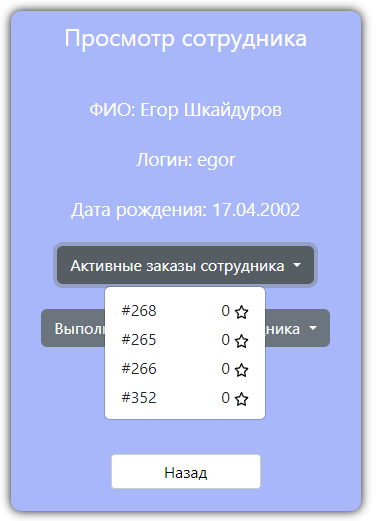
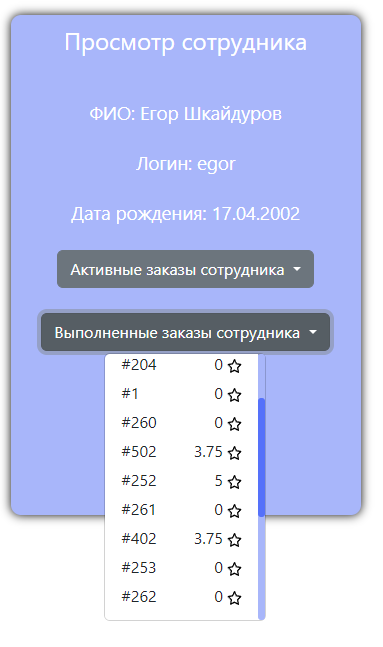


Рисунок А.13 – Активные заказы сотрудника



А.14 – Выполненные заказы сотрудника

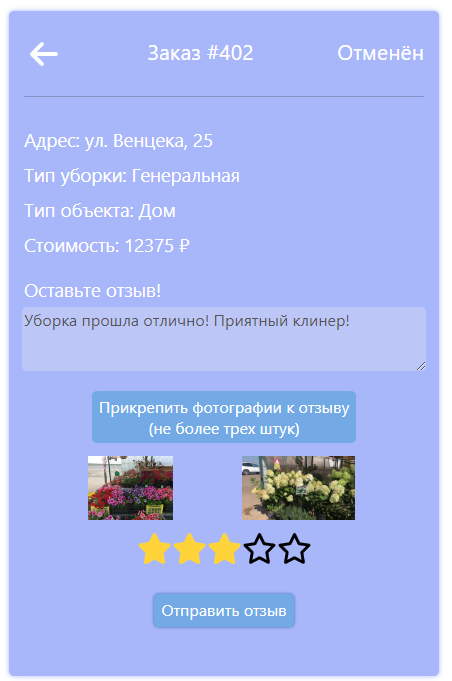


Рисунок А.15 – Просмотр выполненного заказа конкретного сотрудника администратором

Последнее действие, доступное администратору в таблице сотрудников, является добавление нового сотрудника. Для этого необходимо нажать «Добавить клинера» в таблице сотрудников. В открывшемся меню ввести всю необходимую информацию о новом клинере и нажать кнопку «Добавить».

Стоит отметить, что все поля являются обязательными, без этого клинер не будет создан, а также пароль для клинера создается единоразовый и не подлежит восстановлению.

После всех выполненных действий клинер будет добавлен в таблицу сотрудников.

На рисунке А.16 представлена форма добавления клинера.

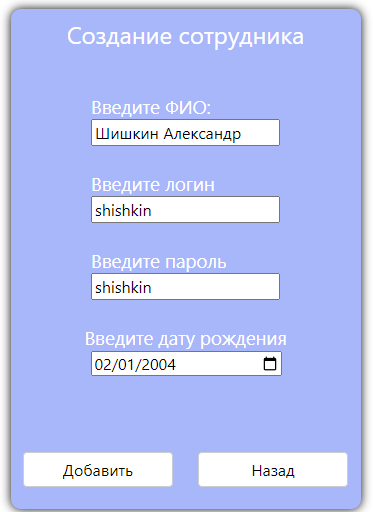


Рисунок А.16 – Добавление нового клинера

ПРИЛОЖЕНИЕ Б

Листинг кода программы

Классы сущностей

@Data

@Entity

@Table (name = "cleaners", schema = "public")

public class Cleaner {

@Id

@GeneratedValue(strategy = GenerationType.SEQUENCE)

private long id;

@Column

private String login;

@Column

private String password;

@Column

private String fio;

@Column

private Date birthdate;

@OneToMany(mappedBy = "cleaner",

fetch = FetchType.LAZY,

cascade = CascadeType.ALL)

@JsonBackReference

private List<Order> orders;

}

@Data

@Entity

@Table (name = "customers", schema = "public")

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.SEQUENCE)

private long id;

@Column

private String login;

@Column

private String password;

@Column

private String name;

@OneToMany(mappedBy = "customer",

fetch = FetchType.LAZY,

cascade = CascadeType.ALL)

@JsonBackReference

private List<Order> orders;

}

@Data

@Entity

@Table (name = "orders", schema = "public")

public class Order {

@Id

@GeneratedValue (strategy = GenerationType.SEQUENCE)

private long id;

private Date orderDate;

private int price;

private int status;

private String orderAddress;

private boolean isDone;

private int cleanType;

private int orderType;

private int square;

@ManyToOne(fetch = FetchType.EAGER,

cascade = CascadeType.MERGE)

@JoinColumn(name = "customer\_id", foreignKey = @ForeignKey(ConstraintMode.CONSTRAINT))

@JsonManagedReference

private Customer customer;

@ManyToOne(fetch = FetchType.EAGER,

cascade = CascadeType.MERGE)

@JoinColumn(name = "cleaner\_id", foreignKey = @ForeignKey(ConstraintMode.CONSTRAINT))

@JsonManagedReference

private Cleaner cleaner;

@OneToOne(cascade = CascadeType.ALL)

@JoinColumn(name = "fk\_review\_id")

private Review review;

}

@Data

public class Street {

private Long id;

private String name;

}

@Data

public class Area {

private Long id;

private String name;

}

@Data

public class CleanType {

private Long id;

private String text;

private double price;

}

@Data

@Entity

@Table (name = "reviews", schema = "public")

public class Review {

@Id

@GeneratedValue(strategy = GenerationType.SEQUENCE)

@Column(name = "id")

private long id;

private String text;

private double value;

private Date reviewDate;

@OneToMany(mappedBy = "review")

private List<Photo> photos;

}

@Entity

@Table (name = "photos", schema = "public")

@Data

public class Photo {

@Id

@GeneratedValue(strategy = GenerationType.SEQUENCE)

private long id;

private String path;

@ManyToOne(fetch = FetchType.EAGER, cascade = CascadeType.MERGE)

@JoinColumn(name = "review\_id", foreignKey = @ForeignKey(ConstraintMode.CONSTRAINT))

@JsonBackReference

private Review review;

}

Классы репозиториев

@Repository

public interface IAreaRepository {

}

public interface ICleanerRepository extends JpaRepository<Cleaner, Long> {

List<Cleaner> findAllByLoginAndPassword(String string, String hexString);

Optional<Cleaner> findById(long order\_cleaner\_id);

}

@Repository

public interface ICleanTypeRepository {

}

@Repository

public interface ICustomerRepository extends JpaRepository<Customer, Long> {

public List<Customer> findAllByLoginAndPassword(String login, String password);

}

@Repository

public interface IOrderRepository extends JpaRepository<Order, Long> {

@Query("SELECT o FROM Order o WHERE o.customer = :customer AND o.status != -1 AND o.status != -2")

List<Order> findAllByCustomerAndNotStatus(Customer customer);

@Query("SELECT o FROM Order o WHERE o.customer = :customer AND (o.status = -1 OR o.status = -2)")

List<Order> findAllByCustomerAndStatus(Customer customer);

Optional<Review> findReviewById(long orderId);

}

public interface IPhotoRepository extends JpaRepository<Photo, Long> {}

public interface IReviewRepository extends JpaRepository<Review, Long> {

List<Photo> findPhotoById(long reviewId);

}

@Repository

public interface IStreetRepository {

}

Классы сервисов

@Service

@RequiredArgsConstructor

public class AdminService {

private final ICleanerRepository cleanerRepository;

private final IOrderRepository orderRepository;

private final JdbcTemplate jdbcTemplate;

public List<OrderDTO> getActive () {

List<OrderDTO> list = jdbcTemplate.query(

"select \* from orders where status != -1 and status != -2",

new RowMapper<OrderDTO>() {

@Override

public OrderDTO mapRow(ResultSet rs, int rowNum) throws SQLException {

OrderDTO orderDTO = new OrderDTO();

orderDTO.setId(rs.getInt("id"));

orderDTO.setStatus(rs.getInt("status"));

orderDTO.setOrderAddress(rs.getString("order\_address"));

orderDTO.setOrderDate(rs.getDate("order\_date"));

orderDTO.setOrderType(rs.getInt("order\_type"));

orderDTO.setDone(rs.getBoolean("is\_done"));

orderDTO.setPrice(rs.getInt("price"));

orderDTO.setCleanType(rs.getInt("clean\_type"));

orderDTO.setSquare(rs.getInt("square"));

return orderDTO;

}

});

if (list.isEmpty())

return null;

return list;

}

public List<OrderDTO> getDone() {

List<OrderDTO> list = jdbcTemplate.query(

"select \* from orders where status = -1 or status = -2",

new RowMapper<OrderDTO>() {

@Override

public OrderDTO mapRow(ResultSet rs, int rowNum) throws SQLException {

OrderDTO orderDTO = new OrderDTO();

orderDTO.setId(rs.getInt("id"));

orderDTO.setStatus(rs.getInt("status"));

orderDTO.setOrderAddress(rs.getString("order\_address"));

orderDTO.setOrderDate(rs.getDate("order\_date"));

orderDTO.setOrderType(rs.getInt("order\_type"));

orderDTO.setDone(rs.getBoolean("is\_done"));

orderDTO.setPrice(rs.getInt("price"));

orderDTO.setCleanType(rs.getInt("clean\_type"));

orderDTO.setSquare(rs.getInt("square"));

return orderDTO;

}

});

if (list.isEmpty())

return null;

return list;

}

public List<CleanerAdminDTO> getCleaners () {

List<CleanerAdminDTO> listDTOs = jdbcTemplate.query(

"select " +

" cl.fio as fio," +

" cl.id as id," +

" sum(ord.price) as salary," +

" count(ord.id) as countOrders," +

" avg(rev.value) as avgReview" +

" from cleaners cl" +

" left join orders ord" +

" on ord.cleaner\_id = cl.id" +

" left join reviews rev" +

" on rev.id = ord.fk\_review\_id" +

" group by cl.id" +

" order by cl.id"

, CleanerAdminDTO.cleanerMapper);

return listDTOs;

}

public CleanerDTO getCleaner (long id) {

Optional<Cleaner> cleaner = cleanerRepository.findById(id);

return cleaner.map(CleanerDTO::fromEntity).orElse(null);

}

public CleanerDTO addCleaner (CreateCleanerDTO dto) throws NoSuchAlgorithmException {

Cleaner needCreate = CreateCleanerDTO.toEntity(dto);

Cleaner created = cleanerRepository.save(needCreate);

return CleanerDTO.fromEntity(created);

}

}

@Service

public class AreaService {

@Autowired

private IAreaRepository areaRepository;

public Area getArea() {

return areaRepository.findAll() ;

}

}

@Service

@RequiredArgsConstructor

public class CleanerService {

private final ICleanerRepository cleanerRepository;

public CleanerDTO auth (Optional<String> login, Optional<String> password) throws NoSuchAlgorithmException {

if (login.isEmpty() || password.isEmpty())

return null;

List<Cleaner> listCleaner = cleanerRepository.findAllByLoginAndPassword(login.get(), RegisterUser.toHexString(RegisterUser.getSHA(password.get())));

return listCleaner.size() == 0 ? null : CleanerDTO.fromEntity(listCleaner.get(0));

}

public CleanerDTO register (RegisterUser user) throws NoSuchAlgorithmException {

Cleaner cleaner = RegisterUser.toCleaner(user);

Cleaner saved = cleanerRepository.save(cleaner);

return saved == null ? null : CleanerDTO.fromEntity(saved);

}

public List<CleanerDTO> getAll () {

List<Cleaner> cleaners = cleanerRepository.findAll();

return cleaners.stream()

.map(CleanerDTO::fromEntity)

.toList();

}

public CleanerDTO getById (long id) {

Optional<Cleaner> cleaner = cleanerRepository.findById(id);

return cleaner.isEmpty() ? null : CleanerDTO.fromEntity(cleaner.get());

}

}

@Service

public class CleanTypeService {

@Autowired

private ICleanTypeRepository cleanTypeRepository;

public CleanType getCleanType() {

return cleanTypeRepository.findAll();

}

}

@Service

public class StreetService {

@Autowired

private IStreetRepository streetRepository;

public Street getStreet() {

return streetRepository.findAll();

}

}

@Service

@RequiredArgsConstructor

public class CustomerService {

private final ICustomerRepository customerRepository;

public CustomerDTO auth (Optional<String> login, Optional<String> pass) throws NoSuchAlgorithmException {

if (login.isEmpty() || pass.isEmpty())

return null;

List<Customer> listCustomer = customerRepository.findAllByLoginAndPassword(login.get(), RegisterUser.toHexString(RegisterUser.getSHA(pass.get())));

return listCustomer.size() == 0 ? null : CustomerDTO.fromEntity(listCustomer.get(0));

}

public CustomerDTO register (RegisterUser user) throws NoSuchAlgorithmException {

Customer customer = RegisterUser.toCustomer(user);

Customer registered = customerRepository.save(customer);

return registered == null ? null : CustomerDTO.fromEntity(registered);

}

public CustomerDTO get (long id) {

Optional<Customer> customer = customerRepository.findById(id);

if (customer.isEmpty())

return null;

return CustomerDTO.fromEntity(customer.get());

}

public CustomerDTO update (long id, CustomerUpdateDTO dto) throws NoSuchAlgorithmException {

Optional<Customer> customer = customerRepository.findById(id);

if (customer.isEmpty())

return null;

customer.get().setName(dto.getName());

customer.get().setLogin(dto.getLogin());

if (dto.getPassword() != null)

customer.get().setPassword(RegisterUser.toHexString(RegisterUser.getSHA(dto.getPassword())));

customerRepository.save(customer.get());

return CustomerDTO.fromEntity(customer.get());

}

}

@Service

@RequiredArgsConstructor

public class OrderService {

private final IOrderRepository orderRepository;

private final ICustomerRepository customerRepository;

private final ICleanerRepository cleanerRepository;

public List<OrderDTO> getActiveOrders(long userId) {

Optional<Customer> customer = customerRepository.findById(userId);

if (customer.isEmpty())

return null;

List<Order> orders = orderRepository.findAllByCustomerAndNotStatus(customer.get());

if (orders.isEmpty())

return null;

return orders.stream()

.map(OrderDTO::fromEntity)

.toList();

}

public List<OrderDTO> getDoneOrdersOrCanceled(long userId) {

Optional<Customer> customer = customerRepository.findById(userId);

if (customer.isEmpty())

return null;

List<Order> orders = orderRepository.findAllByCustomerAndStatus(customer.get());

if (orders.isEmpty())

return null;

return orders.stream()

.map(OrderDTO::fromEntity)

.toList();

}

public OrderDTO makeOrder(long userId, MakeOrderDTO dto) {

Optional<Customer> customer = customerRepository.findById(userId);

Optional<Cleaner> cleaner = cleanerRepository.findById(dto.getOrder\_cleaner\_id());

if (customer.isEmpty() || cleaner.isEmpty())

return null;

Order order = MakeOrderDTO.toEntity(dto);

order.setCleaner(cleaner.get());

order.setCustomer(customer.get());

Order savedOrder = orderRepository.save(order);

return savedOrder == null ? null : OrderDTO.fromEntity(savedOrder);

}

public OrderDTO cancelOrder(long id) {

Optional<Order> order = orderRepository.findById(id);

if (order.isEmpty())

return null;

order.get().setStatus(-2);

Order saved = orderRepository.save(order.get());

return saved == null ? null : OrderDTO.fromEntity(saved);

}

public OrderDTO getOrderById(long id) {

Optional<Order> order = orderRepository.findById(id);

return order.isEmpty() ? null : OrderDTO.fromEntity(order.get());

}

public OrderDTO updateOrder(long id, OrderUpdateDTO dto) {

Optional<Order> order = orderRepository.findById(id);

if (order.isEmpty())

return null;

order.get().setStatus(dto.getStatus());

Order saved = orderRepository.save(order.get());

return saved == null ? null : OrderDTO.fromEntity(saved);

}

}

@Service

@RequiredArgsConstructor

public class PhotoService {

private final IPhotoRepository photoRepository;

private final IReviewRepository reviewRepository;

public List<byte[]> getPhotos (long reviewId) {

Optional<Review> review = reviewRepository.findById(reviewId);

if (review.isEmpty())

return null;

List<Photo> listPhoto = review.get().getPhotos();

List<byte[]> photoBytes = new ArrayList<>();

for (Photo photo : listPhoto) {

File file;

try {

file = ResourceUtils.getFile(photo.getPath());

Path path = file.toPath();

photoBytes.add(Files.readAllBytes(path));

} catch (IOException e) {

e.printStackTrace();

};

}

return photoBytes;

}

}

@Service

@RequiredArgsConstructor

public class ReviewService {

private final IReviewRepository reviewRepository;

private final IPhotoRepository photoRepository;

private final IOrderRepository orderRepository;

public ReviewDTO addReview(long orderId, List<MultipartFile> photos, String text, double value) {

Optional<Order> order = orderRepository.findById(orderId);

if (order.isEmpty())

return null;

Review review = new Review();

review.setText(text);

review.setValue(value);

Review savedReview = reviewRepository.save(review);

List<Photo> savedPhotos = new ArrayList<>();

for (MultipartFile photo : photos) {

if (photo.isEmpty())

continue;

try {

String uploadDir = "D:/university/cleanApp/server/photos";

File uploadPath = new File(uploadDir);

if (!uploadPath.exists()) {

uploadPath.mkdirs();

}

String filename = photo.getOriginalFilename();

String filePath = uploadDir + File.separator + filename;

File destFile = new File(filePath); // path to database

photo.transferTo(destFile);

// Create PHOTO entity

Photo photoEntity = new Photo();

photoEntity.setPath(filePath);

photoEntity.setReview(review);

Photo savedPhoto = photoRepository.save(photoEntity);

savedPhotos.add(savedPhoto);

List<Photo> photosReview = review.getPhotos();

if (photosReview == null) {

photosReview = new ArrayList<>();

}

photosReview.add(savedPhoto);

review.setPhotos(photosReview);

} catch (IOException e) {

continue;

}

}

savedReview.setPhotos(savedPhotos);

reviewRepository.save(review);

// Взять сущность Order и проставить ему Review

order.get().setReview(savedReview);

orderRepository.save(order.get());

return savedReview == null ? null : ReviewDTO.fromEntity(savedReview);

}

public ReviewDTO getReview(long orderId) {

Optional<Order> order = orderRepository.findById(orderId);

if (order.isEmpty())

return null;

Review review = order.get().getReview();

return review == null ? null : ReviewDTO.fromEntity(review);

}

}

Классы контроллеров

@RequiredArgsConstructor

@RestController

@RequestMapping("admin")

@CrossOrigin

public class AdminController {

private final AdminService adminService;

@GetMapping("/active")

public ResponseEntity<?> getActiveOrders () {

List<OrderDTO> dtoList = adminService.getActive();

return new ResponseEntity<>(dtoList, dtoList == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

@GetMapping("/done")

public ResponseEntity<?> getDoneOrders () {

List<OrderDTO> dtoList = adminService.getDone();

return new ResponseEntity<>(dtoList, dtoList == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

@GetMapping("/cleaners")

public ResponseEntity<?> getCleaners () {

List<CleanerAdminDTO> cleanerDTOs = adminService.getCleaners();

return new ResponseEntity<>(cleanerDTOs, cleanerDTOs == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

@GetMapping("/cleaner/{id}")

public ResponseEntity<?> getCleaner(@PathVariable long id) {

CleanerDTO cleaner = adminService.getCleaner(id);

return new ResponseEntity<>(cleaner, cleaner == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

@PostMapping("/cleaner")

public ResponseEntity<?> createCleaner(@RequestBody CreateCleanerDTO dto) throws NoSuchAlgorithmException {

CleanerDTO createdCleaner = adminService.addCleaner(dto);

return new ResponseEntity<>(createdCleaner, createdCleaner == null ? HttpStatus.CONFLICT : HttpStatus.CREATED);

}

}

@RestController

public class AreaController {

@Autowired

private AreaService areaService;

public ResponseEntity<?> getArea () {

return new ResponseEntity<>(areaService.getArea(), HttpStatus.OK);

}

}

public class StreetController {

@Autowired

private StreetService streetService;

public ResponseEntity<?> getStreet () {

return new ResponseEntity<>(streetService.getStreet(), HttpStatus.OK);

}

}

public class CleanTypeController {

@Autowired

private CleanTypeService cleanTypeService;

public ResponseEntity<?> getCleanType () {

return new ResponseEntity<>(cleanTypeService.getCleanType(), HttpStatus.OK);

}

}

@RestController

@RequestMapping("/cleaner")

@CrossOrigin

public class CleanerController {

@Autowired

private CleanerService cleanerService;

@GetMapping("/auth")

public ResponseEntity<?> auth (@RequestParam("login") Optional<String> login, @RequestParam("password") Optional<String> password) throws NoSuchAlgorithmException {

CleanerDTO cleaner = cleanerService.auth(login, password);

return new ResponseEntity<>(cleaner, cleaner == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

@PostMapping("/register")

public ResponseEntity<?> register (@RequestBody RegisterUser user) throws NoSuchAlgorithmException {

CleanerDTO cleaner = cleanerService.register(user);

return new ResponseEntity<>(cleaner, cleaner == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

@GetMapping("/all")

public ResponseEntity<?> getAll () {

List<CleanerDTO> cleaners = cleanerService.getAll();

return new ResponseEntity<>(cleaners, cleaners == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

@GetMapping("/{id}")

public ResponseEntity<?> getCleaner (@PathVariable("id") long id) {

CleanerDTO cleaner = cleanerService.getById(id);

return new ResponseEntity<>(cleaner, cleaner == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

}

@CrossOrigin

@RestController

@RequestMapping("/customer")

public class CustomerController {

@Autowired

private CustomerService customerService;

@GetMapping("/auth")

private ResponseEntity<?> auth (@RequestParam("login") Optional<String> login, @RequestParam("password") Optional<String> password) throws NoSuchAlgorithmException {

CustomerDTO customer = customerService.auth(login, password);

return new ResponseEntity<>(customer, customer == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

@PostMapping("/register")

private ResponseEntity<?> register (@RequestBody RegisterUser user) throws NoSuchAlgorithmException {

CustomerDTO customer = customerService.register(user);

return new ResponseEntity<>(customer, customer == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

@GetMapping("/{id}")

private ResponseEntity<?> getCustomer (@PathVariable long id) {

CustomerDTO customer = customerService.get(id);

return new ResponseEntity<>(customer, customer == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

@PutMapping("/{id}")

private ResponseEntity<?> updateCustomer (@PathVariable long id, @RequestBody CustomerUpdateDTO dto) throws NoSuchAlgorithmException {

CustomerDTO customer = customerService.update(id, dto);

return new ResponseEntity<>(customer, customer == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

}

@CrossOrigin

@RestController

@RequestMapping("/{userId}/order")

public class OrderController {

@Autowired

private OrderService orderService;

@GetMapping("/active")

public ResponseEntity<?> getActiveOrders (@PathVariable("userId") long userId) {

List<OrderDTO> orders = orderService.getActiveOrders(userId);

return new ResponseEntity<>(orders, orders == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

@GetMapping("/done")

public ResponseEntity<?> getDoneOrders (@PathVariable("userId") long userId) {

List<OrderDTO> orders = orderService.getDoneOrdersOrCanceled(userId);

return new ResponseEntity<>(orders, orders == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

@PostMapping

public ResponseEntity<?> makeOrder (@PathVariable ("userId") long userId, @RequestBody MakeOrderDTO dto) {

OrderDTO order = orderService.makeOrder(userId, dto);

return new ResponseEntity<>(order, order == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

@GetMapping("/{orderId}/cancel")

public ResponseEntity<?> cancelOrder (@PathVariable ("orderId") long orderId) {

OrderDTO order = orderService.cancelOrder(orderId);

return new ResponseEntity<>(order, order == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

@GetMapping("/{orderId}")

public ResponseEntity<?> getOrderById (@PathVariable("orderId") long id) {

OrderDTO dto = orderService.getOrderById(id);

return new ResponseEntity<>(dto, dto == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

@PutMapping("/{orderId}")

public ResponseEntity<?> updateOrder (@PathVariable("orderId") long id,

@RequestBody OrderUpdateDTO dto) {

OrderDTO order = orderService.updateOrder(id, dto);

return new ResponseEntity<>(order, order == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

}

@CrossOrigin

@RestController

@RequestMapping("/{reviewId}/photo")

public class PhotoController {

@Autowired

private PhotoService photoService;

@GetMapping()

public ResponseEntity<?> getPhotos (@PathVariable("reviewId") long reviewId) {

List<byte[]> dto = photoService.getPhotos(reviewId);

return new ResponseEntity<>(dto, dto == null || dto.size() == 0? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

}

@RestController

@RequestMapping("/{orderId}/review")

@CrossOrigin

public class ReviewController {

@Autowired

private ReviewService reviewService;

@PostMapping

public ResponseEntity<?> addReview (@PathVariable("orderId") long orderId,

@RequestParam("files") List<MultipartFile> photos,

@RequestParam("text") String text,

@RequestParam("value") double value) {

ReviewDTO dto = reviewService.addReview(orderId, photos, text, value);

return new ResponseEntity<>(dto, dto == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

@GetMapping

public ResponseEntity<?> getReview (@PathVariable("orderId") long orderId) {

ReviewDTO dto = reviewService.getReview(orderId);

return new ResponseEntity<>(dto, dto == null ? HttpStatus.NOT\_FOUND : HttpStatus.OK);

}

}

DTO-классы

@Data

public class CleanerAdminDTO {

private long id;

private String fio;

private long countOrders;

private double avgReview;

private long salary;

public static CleanerAdminDTO fromEntity (Cleaner cleaner) {

CleanerAdminDTO dto = new CleanerAdminDTO();

dto.setId(cleaner.getId());

dto.setFio(cleaner.getFio());

return dto;

}

public static Cleaner toEntity (CleanerAdminDTO dto) {

Cleaner cleaner = new Cleaner();

cleaner.setId(dto.getId());

cleaner.setFio(dto.getFio());

return cleaner;

}

public static RowMapper<CleanerAdminDTO>cleanerMapper = (rs, rowNum) -> {

CleanerAdminDTO dto = new CleanerAdminDTO();

dto.setFio(rs.getString("fio"));

dto.setCountOrders(rs.getLong("countOrders"));

dto.setSalary(rs.getLong("salary"));

dto.setAvgReview(roundTo(rs.getDouble("avgReview")));

dto.setId(rs.getLong("id"));

return dto;

};

private static double roundTo(double value) {

double scale = Math.pow(10, 2);

return Math.ceil(value \* scale) / scale;

}

}

@Data

public class CleanerDTO {

private long id;

private String login;

private String fio;

private Date birthday;

private List<Order> orders;

public static CleanerDTO fromEntity (Cleaner cleaner) {

CleanerDTO dto = new CleanerDTO();

dto.setId(cleaner.getId());

dto.setLogin(cleaner.getLogin());

dto.setFio(cleaner.getFio());

dto.setBirthday(cleaner.getBirthdate());

dto.setOrders(cleaner.getOrders());

return dto;

}

public static Cleaner toEntity (CleanerDTO dto) {

Cleaner cleaner = new Cleaner();

cleaner.setId(dto.getId());

cleaner.setLogin(dto.getLogin());

cleaner.setFio(dto.getFio());

cleaner.setBirthdate(dto.getBirthday());

cleaner.setOrders(dto.getOrders());

return cleaner;

}

}

@Data

public class CreateCleanerDTO {

private String login;

private String fio;

private Date birthday;

private String password;

public static CreateCleanerDTO fromEntity (Cleaner cleaner) {

CreateCleanerDTO dto = new CreateCleanerDTO();

dto.setLogin(cleaner.getLogin());

dto.setFio(cleaner.getFio());

dto.setBirthday(cleaner.getBirthdate());

dto.setPassword(cleaner.getPassword());

return dto;

}

public static Cleaner toEntity (CreateCleanerDTO dto) throws NoSuchAlgorithmException {

Cleaner cleaner = new Cleaner();

cleaner.setLogin(dto.getLogin());

cleaner.setFio(dto.getFio());

cleaner.setBirthdate(dto.getBirthday());

cleaner.setPassword(RegisterUser.toHexString(RegisterUser.getSHA(dto.getPassword())));

return cleaner;

}

}

@Data

public class CustomerDTO {

private long id;

private String name;

private String login;

private List<Order> orders;

public static Customer toEntity (CustomerDTO dto) {

Customer customer = new Customer();

customer.setId(dto.getId());

customer.setName(dto.getName());

customer.setLogin(dto.getLogin());

customer.setOrders(dto.getOrders());

return customer;

}

public static CustomerDTO fromEntity (Customer customer) {

CustomerDTO dto = new CustomerDTO();

dto.setId(customer.getId());

dto.setName(customer.getName());

dto.setLogin(customer.getLogin());

dto.setOrders(customer.getOrders());

return dto;

}

}

@Data

public class CustomerUpdateDTO {

private String name;

private String login;

private String password;

public static Customer toEntity (CustomerUpdateDTO dto) {

Customer customer = new Customer();

customer.setName(dto.getName());

customer.setLogin(dto.getLogin());

customer.setName(dto.getName());

return customer;

}

public static CustomerUpdateDTO fromEntity (Customer customer) {

CustomerUpdateDTO dto = new CustomerUpdateDTO();

dto.setName(customer.getName());

dto.setLogin(customer.getLogin());

dto.setName(customer.getName());

return dto;

}

}@Data

public class MakeOrderDTO {

private int order\_type;

private int clean\_type;

private int square;

private String order\_address;

private Date date;

private long order\_cleaner\_id;

private int price;

public static Order toEntity (MakeOrderDTO dto) {

Order order = new Order();

order.setCleanType(dto.getClean\_type());

order.setOrderType(dto.getOrder\_type());

order.setSquare(dto.getSquare());

order.setOrderAddress(dto.getOrder\_address());

order.setOrderDate(dto.getDate());

order.setPrice(dto.getPrice());

return order;

}

}

@Data

public class OrderDTO {

private long id;

private Date orderDate;

private int price;

private int status;

private int square;

private int cleanType;

private int orderType;

private String orderAddress;

private boolean done;

public static OrderDTO fromEntity (Order order) {

OrderDTO dto = new OrderDTO();

dto.setId(order.getId());

dto.setOrderDate(order.getOrderDate());

dto.setPrice(order.getPrice());

dto.setStatus(order.getStatus());

dto.setCleanType(order.getCleanType());

dto.setOrderType(order.getOrderType());

dto.setOrderAddress(order.getOrderAddress());

dto.setDone(order.isDone());

dto.setSquare(order.getSquare());

return dto;

}

public static Order toEntity (OrderDTO dto) {

Order order = new Order();

order.setId(dto.getId());

order.setOrderDate(dto.getOrderDate());

order.setPrice(dto.getPrice());

order.setStatus(dto.getStatus());

order.setCleanType(dto.getCleanType());

order.setOrderType(dto.getOrderType());

order.setOrderAddress(dto.getOrderAddress());

order.setDone(dto.isDone());

order.setSquare(dto.getSquare());

return order;

}

}

@Data

public class OrderUpdateDTO {

private Date orderDate;

private int price;

private int status;

private int square;

private int cleanType;

private int orderType;

private String orderAddress;

private boolean done;

public static OrderUpdateDTO fromEntity (Order order) {

OrderUpdateDTO dto = new OrderUpdateDTO();

dto.setOrderDate(order.getOrderDate());

dto.setPrice(order.getPrice());

dto.setStatus(order.getStatus());

dto.setCleanType(order.getCleanType());

dto.setOrderType(order.getOrderType());

dto.setOrderAddress(order.getOrderAddress());

dto.setDone(order.isDone());

dto.setSquare(order.getSquare());

return dto;

}

public static Order toEntity (OrderUpdateDTO dto) {

Order order = new Order();

order.setOrderDate(dto.getOrderDate());

order.setPrice(dto.getPrice());

order.setStatus(dto.getStatus());

order.setCleanType(dto.getCleanType());

order.setOrderType(dto.getOrderType());

order.setOrderAddress(dto.getOrderAddress());

order.setDone(dto.isDone());

order.setSquare(dto.getSquare());

return order;

}

}

@Data

public class RegisterUser {

private String login;

private String password;

public static Customer toCustomer (RegisterUser user) throws NoSuchAlgorithmException {

Customer customer = new Customer();

customer.setLogin(user.getLogin());

customer.setPassword(RegisterUser.toHexString(RegisterUser.getSHA(user.getPassword())));

return customer;

}

public static Cleaner toCleaner (RegisterUser user) throws NoSuchAlgorithmException {

Cleaner cleaner = new Cleaner();

cleaner.setLogin(user.getLogin());

cleaner.setPassword(RegisterUser.toHexString(RegisterUser.getSHA(user.getPassword())));

return cleaner;

}

public static byte[] getSHA(String input) throws NoSuchAlgorithmException {

MessageDigest md = MessageDigest.getInstance("SHA-256");

return md.digest(input.getBytes(StandardCharsets.UTF\_8));

}

public static String toHexString(byte[] hash) {

BigInteger number = new BigInteger(1, hash);

StringBuilder hexString = new StringBuilder(number.toString(16));

while (hexString.length() < 32) {

hexString.insert(0, '0');

}

return hexString.toString();

}

}

@Data

public class ReviewDTO {

private long id;

private String text;

private double value;

private List<Photo> photos;

private Date reviewDate;

public static Review toEntity (ReviewDTO dto) {

Review review = new Review();

review.setId(dto.getId());

review.setText(dto.getText());

review.setValue(dto.getValue());

review.setPhotos(dto.getPhotos());

review.setReviewDate(dto.getReviewDate());

return review;

}

public static ReviewDTO fromEntity (Review review) {

ReviewDTO dto = new ReviewDTO();

dto.setId(review.getId());

dto.setText(review.getText());

dto.setValue(review.getValue());

dto.setPhotos(review.getPhotos());

dto.setReviewDate(review.getReviewDate());

return dto;

}

}

@Data

public class SavePhotoDTO {

private String path;

private Review review;

public static Photo toEntity (SavePhotoDTO dto) {

Photo photo = new Photo();

photo.setPath(dto.getPath());

photo.setReview(dto.getReview());

return photo;

}

public static SavePhotoDTO fromEntity (Photo photo) {

SavePhotoDTO dto = new SavePhotoDTO();

dto.setPath(photo.getPath());

dto.setReview(photo.getReview());

return dto;

}

}