МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ»
Кафедра IT-инжиниринг

ДОПУЩЕН К ЗАЩИТЕ
Заведующий кафедрой
PhD, доцент
_____ Т.С. Картбаев
«____» _____ 2019 г.

ДИПЛОМНЫЙ ПРОЕКТ

На тему: Разработка CRM-системы для управления капиталом и ресурсами строительной компании ТОО Темиртас-1 на django(python)

Специальность: 5В070400 – «Вычислительная техника и программное обеспечение»

Выполнил: Овезов А.Б. Группа: ВТ-15-2
Научный руководитель: доц. Аманбаев А.А.

Консультанты:
по экономической части: к.э.н., профессор _____ Ж.Г. Аренбаева
«22» мая 2019 г.

по безопасности
жизнедеятельности: д.т.н., ст. преп. _____ Ш.Ш. Бекбасаров
«13» 05 2019 г.

по применению
вычислительной техники: ст. преп. _____ М.Н. Майкотов
«14» мая 2019 г.

Нормоконтролер: ассист. _____ А.А. Айтказина
«15» 05 2019 г.

Рецензент: асс. проф. каф _____ Н.К. Мукажанов
«____» _____ 2019 г.

Алматы 2019

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ»

Институт систем управления и информационных технологий

Кафедра IT-инжиниринг

Специальность 5В070400 – «Вычислительная техника и программное обеспечение»

ЗАДАНИЕ
на выполнение дипломного проекта

Студенту Овезов Адилбек Бекожанович

Тема проекта: Разработка CRM-системы для управления капиталом и ресурсами строительной компании ТОО Темиртас-1 на django(python)

Утверждена приказом по университету № 33 от «01» марта 2019 г.

Срок сдачи законченного проекта «24» мая 2019 г.

Исходные данные к проекту (требуемые параметры результатов исследования (проектирования) и исходные данные объекта): Руководство системы менеджмента качества на предприятии; международные стандарты ИСО-9001, данные преддипломной практики.

Перечень вопросов, подлежащих разработке в дипломном проекте, или краткое содержание дипломного проекта:

- аналитическая часть;
- проектная часть;
- экспериментальная часть;
- экономическая часть;
- безопасность жизнедеятельности;
- приложение А. Техническое задание;
- приложение Б. Листинг программы;
- приложение В. Акт внедрения.

Перечень графического материала (с точным указанием обязательных чертежей): представлены 15 таблиц, 47 иллюстрации.

Основная рекомендуемая литература:

1 Марк Лутц. Программирование на Python / Пер. с англ. — 4-е изд. — СПб.: Символ-Плюс, 2011. — Т. II. — ISBN 978-5-93286-211-7.Итан Браун. Веб-разработка с применением Node и Express. Полноценное использование стека — Санкт-Петербург: Питер, 2017. — 336 с.

2 Бизли, Дэвид М. Язык программирования Python. Справочник. — К.: ДиаСофт, 2000. — 336 с. — ISBN 966-7393-54-2, ISBN 0-7357-0901-7

3 У. Чан, П. Биссекс, Д. Форсье. Django. Разработка веб-приложений на Python = Python Web Development with Django / пер. с англ. А. Киселёв. — СПб.: Символ-Плюс, 2009. — 456 с. — (High Tech). — ISBN 978-5-93286-167-7

Раздел	Консультант	Сроки	Подпись
Экономическая часть	Аренбаева Ж.Г.	04.03.2019 – 22.05.2019	
Безопасность жизнедеятельности	Бекбасаров Ш.Ш.	04.02 - 13.05.19	
Программное обеспечение	Майкотов М.Н.	02.05. - 14.05	
Нормоконтролер	Айтказина А.А.	02.04.19 – 05.05.19	

ГРАФИК
подготовки дипломной проекта

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
Аналитическая часть	15.01.2019	Выполнил
Проектная часть	30.01.2019	Выполнил
Экспериментальная часть	15.05.2019	Выполнил

Дата выдачи задания «29» октября 2018 г.

Заведующий кафедрой _____ Т.С. Картбаев

Научный руководитель проекта _____ А.А. Аманбаев

Задание принял к
исполнению студент _____ А.Б. Овезов

Аңдатпа

Дипломдық жобаның тақырыбы ««Темиртас – 1» ЖШС құрылыс компаниясының ресурстары мен капиталын басқару үшін Django (python) фреймворкында CRM жүйесін құру».

Дипломдық жобаның мақсаты – орталықтандырылған база көмегімен «Теміртас - 1» ЖШС құрылыс компаниясының қызметкерлерінің жұмысын жеңілдету.

Алға қойған мақсатқа жету үшін келесідей технологиялар қолданылды:

Аннотация

Тема дипломного проекта: «Разработка CRM-системы для управления ресурсами и капиталом строительной компании ТОО «Темиртас-1» на Django(python)».

Цель дипломного проекта – упрощение работы сотрудников строительной компании ТОО «Темиртас – 1» с помощью централизованной базой.

Annotation

Theme of the graduation project: "Development of CRM-system for resource management and capital management of the construction company «Temirtas-1» LLP on Django(python)".

The purpose of the diploma project is to simplify the work of employees of the construction company Temirtas-1 LLP with the help of a centralized database.

Содержание

Введение

В последние несколько лет информационные технологии развиваются очень активно и получают все более широкое распространение и применение. Это связано с тем, что объем информации, которая обрабатывается и используется предприятием в процессе его функционирования, постоянно возрастает, а текущая информация обновляется. В наши дни трудно представить себе предприятие, которое обошлось бы без информационной системы, которая должна облегчить работу персонала организации.

В современном бизнесе необходимость автоматизация различных процессов стала уже привычным явлением. Уже становится сложно представить себе складской или бухгалтерский учет без применения специализированного программного обеспечения, торговые представители используют специальные приложения для оформления и отправки заказа в офис прямо с персонального компьютера.

Что происходит, если работа отдела продаж ведется без системы учета? Каждый менеджер по продажам работает так, как ему удобнее, ведет фиксацию звонков, других видов взаимодействия с клиентами по собственному усмотрению: кто-то – на бумаге, кто-то – в Excel таблицах, а кто-то вообще не считает нужным фиксировать процесс своей работы. Входящие звонки или заявки от заказчиков также не фиксируются, зачастую даже сложно понять, кто из менеджеров занимается входящей заявкой. В результате реальной учет ведется только на уровне оплаченных заказов и отгрузки товара. Кроме того, в случае увольнения или болезни сотрудника, все его неоконченные переговоры и необработанные контакты компания может потерять, что также крайне нежелательно для эффективной работы отдела продаж.

Выход из этой ситуации – автоматизация и стандартизация управления заявками, с помощью централизованной базой данных и программным обеспечением, т.е. внедрение CRM-системы. Это решение поможет:

– получить общую для компании стандартизированную базу клиентов;

– эффективно осуществлять контроль качества работы отдела продаж, а также следить за состоянием заявок в любой момент времени;

– получить статистику исходя от завершенных заявок и контролем склада, а также состоянием спецтехники;

– планировать повышение качества работы и разрабатывать стратегию развития бизнеса анализирую статистику.

1 Описание программного продукта

В настоящий момент конкуренция на всех уровнях рынка довольно высока. Чтобы компании выиграть конкурентную борьбу, она должна не только привлекать новых клиентов, но и удерживать уже существующих. Для удержания клиента необходимо учитывать его интересы и пожелания. Такой подход к ведению бизнеса называется «клиентоориентированным». Но при большой клиентской базе учет интересов каждого клиента является труднодостижмой целью.

Выходом из данной ситуации является внедрение в компанию CRM-систему, что в переводе на русский язык звучит как «управление взаимоотношениями с клиентами». В настоящий момент, сотрудники компании ведут записи о заявках клиентов вручную с помощью Excel. И, кажется, что даже без CRM-системы проконтролировать работу с покупателями проще простого. На самом деле, это не так. Очень быстро после внедрения автоматизированной системы выявляется огромное число недочетов, а качество работы отдела продаж вырастает в разы.

И так, CRM-система нужна для того, чтобы:

– не потерять потенциального клиента. В данной сфере в нашей стране конкуренция очень высокая. Компании прилагают значительные усилися для того, чтобы привлесь клиентов, чтобы на них обратили внимание. По сравнению с другими затратами на привлечение клиентов, выделяется значительный бюджет. И очень важно, чтобы все эти средства и усилия не пропали даром. Автоматизиваронные системы позволяют получить уверенность, что именно так и будет работать отдел. Каждая заявка будет зафиксирована для дальнейшей обработки;

– контроль работы сотрудников и стандартизация работы с клиентами. Без общей стандартизированной CRM-системы каждый сотрудник работает так, как он привык работать. Кто-то ведет учет в электронных таблицах, кто-то – в записной книжке или ежедневнике, кто-то не ведет учет вообще, ориентируясь лишь на собственную память. Контакты также происходят достаточно хаотично. CRM-система полностью решает эту проблему. Информация обо всех клиентах будет фиксироваться в базе, откуда ее можно в любой момент извлечь;

Программа будет использоваться сотрудниками компании, для оперативного доступа к информации о конкретном клиенте в процессе взаимодействия с ним в рамках обычных бизнес-процессов. Также фиксировать состояния оборудования и спецтехники. Потому что, они могут нуждаться в ремонте в результате продолжительной эксплуатации для погрузки стройматериалов и в других рабочик процессах.

1.1 Назначение программного продукта

Программный продукт предназначен для упрощения рабочего процесса путем контроля заявок и учетом склада с помощью централизованной базой данных. Первоначально, перед внедрением, сотрудники компании будут зарегистрированы с админ панели программы. После получения учетной записи, сотрудники компании должны авторизоваться в админ панели и заполнить персональную информацию. Программное обеспечение будет установлено на все рабочие персональные компьютеры. Таким образом, программное обеспечение, написанное для строительной фирмы ТОО «Темиртас – 1», упрощает следующие процессы:

– отслеживание и контроль заявок от клиентов, изменяя их статус по мере продвижения работы. Например, клиент заказал определенный материал. После внесения заявки, она моментально будет отображаться в списке всех заявок в программном обеспечении. Далее, сотрудник компании может открыть и получить детальную информацию о заявке. После обработки и доставки материала статус заявки меняется на «Завершена» и сохраняется в базе для дальнейшей статистики и аналитики;

– контроль и учет склада. В программе есть доступ к просмотру запаса товаров и стройматериалов на складе, также возможность сообщить администрации о том, что заканчивается определенный товар;

– отслеживание состояния спецтехники и оборудования. В программе также доступно отслежнивание спецтехники и оборудования для обработки и доставки товаров или стройматериалов;

– статистика. Все заявки, отклоненные или завершенные, надежно сохраняются в централизованной базе.

1.2 Выбор средств и технологий

В данном подразделе описывается стэк технологии, которые были использованы при разработке программного продукта.

1.2.1 Python 3.7

В основе проекта язык программирования Python, версию 3.7. Python – высокоуровненвый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Главные плюсы языка:

– качества программного обеспечения. Для многих, в том числе и для меня, основные преимущества – это удобочитаемый синтаксис. Не много языков могут похвастаться им. Программный код на Python читается легче, что значит, многократное его использование и обслуживание выполняется гораздо проще, чем использование программного кода на других языках сценариев. Python содержит самые современные механизмы многократного использования программного кода, каким является ООП;

– библиотеки поддержки. В составе Python поставляется большое число собранных и переносимых функциональных возможностей, известных как стандартная библиотека. Эта библиотека предоставляет Вам массу возможностей, востребованных в прикладных программах, начиная от поиска текста по шаблону и заканчивая сетевыми функциями. Python допускает расширение как за счёт ваших собственных библиотек, так и за счёт библиотек, созданных другими разработчиками;

– переносимость программ. Большая часть программ на языке Python выполняется без изменений на всех основных платформах. Перенос программного кода из Linux в Windows заключается в простом копировании файлов программ с одной машины на другую. Также Python предоставляет Вам массу возможностей по созданию переносимых графических интерфейсов;

– скорость разработки. По сравнению с компилирующим, или строго типизированными языками, такими как C, C++ или Java, Python во много раз повышает производительность труда разработчика. Объем программного кода на языке Python обычно составляет треть, или даже пятую часть эквивалентного программного кода на языке C++ или Java, что означает меньший объем ввода с клавиатуры, меньшее количество времени на откладку и меньший объем трудозатрат на сопровождение. Кроме того, программы на языке Python запускаются сразу же, минуя длительные этапы компиляции и связывания, необходимые в некоторых других языках программирования, что еще больше увеличивает производительность труда программиста.

1.2.2 Django 2.1 фреймворк

Django – свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC. Проект поддерживается организацией Django Software Foundation.

Сайт на Django строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Это одно из существенных архитектурных отличий этого фреймворка от некоторых других

(например, Ruby on Rails). Один из основных принципов фреймворка – DRY (англ. Don't repeat yourself).

Также, в отличие от других фреймворков, обработчики URL в Django конфигурируются явно при помощи регулярных выражений.

Для работы с базой данных Django использует собственный ORM, в котором модель данных описывается классами Python, и по ней генерируется схема базы данных.

Преимущества Django в качестве инструмента вэб-разработки:

– быстрота. Django был разработан, чтобы помочь разработчикам создать приложение настолько быстро, насколько это возможно. Это включает в себя формирование идеи, разработку и выпуск проекта, где Django экономит время и ресурсы на каждом из этих этапов. Таким образом, его можно назвать идеальным решением для разработчиков, для которых вопрос дедлайна стоит в приоритете;

– полная комплектация. Django работает с десятками дополнительных функций, которые заметно помогают с аутентификацией пользователя, картами сайта, администрированием содержимого, RSS и многим другим. Данные аспекты помогают осуществить каждый этап веб разработки;

– безопасность. Работая в Django, мы полчаем защиту от ошибок, связанных с безопасностью и ставящих под угрозу проект. Имеется в виду такие распространенные ошибки, как инъекции SQL, кросс-сайт подлоги, clickjacking и кросс-сайтовый скриптинг. Для эффективного использования логинов и паролей, система пользовательской аутентификации является ключом;

– масштабируемость. Фреймворк Django наилучшим образом подходит для работы с самыми высокими трафиками. Следовательно, логично, что великое множество загруженных сайтов используют Django для удовлетворения требований, связанных с трафиком.

1.2.3 Графический фреймворк PyQt5

PyQt5 – набор «привязок» графического фреймворка Qt для языка программирования Python, выполненный в виде расширения Python.

PyQt5 реализован как комплект Python-модулей. Он включает в себя около 620 классов и 6000 функций и методов включая:

– существующий набор виджетов графического интерфейса;

– ститли виджетов;

– доступ к базам данных с помощью SQL (ODBC, MySQL, PostgreSQL, Oracle);

– QScintilla, основанный на Scintilla виджет текстового редактора;

– поддержку интернационализации (i18n).

Это мульти-платформенный инструментарий, который запускается на большинстве операционных систем, среди которых Unix, Windows и MacOS. PyQt5 реализован под двумя лицензиями. Разработчики могут выбрать между GPL и коммерческой лицензией.

1.2.4 Django Rest Framework

Django Rest Framework (DRF) – это библиотека, которая работает со стандартными моделями Django для создания гибкого и мощного API для проекта. API DRF состоит из 3-х слоев: сериализатора, вида и маршрутизатора:

– сериализатор: преобразует информацию, хранящуюся в базе данных и определенную с помощью моделей Django, в формат, который легко и эффективно передается через API. Модели Django интуитивно представляют данные, хранящиеся в базе, но API должен передавать информацию в менее сложной структуре. Хотя данные будут представлены как экземпляры классов Model, их необходимо перевести в формат JSON для передачи через API;

– вид (ViewSet): определяет функции (чтение, создание, обновление, удаление), которые будут доступны через API. Сериализатор анализирует информацию в обоих направлениях (чтение и запись), тогда как ViewSet - это тот код, в котором определены доступные операции. Наиболее распространенным ViewSet является ModelViewSet;

– маршрутизатор: определяет URL-адреса, которые будут предоставлять доступ к каждому виду. Они предоставляют верхний уровень API. Чтобы избежать создания бесконечных URL-адресов вида: «списки», «детали» и «изменить», маршрутизаторы DRF объединяют все URL-адреса, необходимые для данного вида в одну строку для каждого ViewSet.

1.2.5 СУБД PostgreSQL

PostgreSQL является одной из наиболее популярных систем управления базами данных. Сам проект postgresql эволюционировал из другого проекта, который назывался Ingres. Формально развитие postgresql началось еще в 1986 году. Тогда он назывался POSTGRES. А в 1996 году проект был переименован в PostgreSQL, что отражало больший акцент на SQL. И собственно 8 июля 1996 года состоялся первый релиз продукта.

1.3 Постановка цели и задач

Целью данной дипломной работы заключается в разработке программного обеспечения с удобным графическим интерфейсом для облегчения рабочего

процесса путем контроля заявок и учетом склада с помощью централизованной базы данных.

Данное программное обеспечение позволяет очень эффективно фиксировать и отслеживать заявки от клиентов, и реактивно менять их статус, также вести учет запаса товаров и строиматериалов на складе.

Для достижения цели и разработки программного обеспечения была произведена декомпозиция, для разбиения всей работы на мелкие составляющие.

Основные задачи дипломного проекта:

– изучение серверных технологии, настройка сервера;

– определение технологического стека для разработки программного обеспечения;

– написание restful api серверной части;

– проектирование базы данных;

– настройка админ панели для внесении заявок и контроля товаров;

– разработка пользовательского графического интерфейса;

– интеграция пользовательского интерфейса с api серверной части;

– обоснование экономической целесообразности разрабатываемого продукта.

2 Проектирование программного продукта

В этой главе описывается пректирования базы данных, описание структуры программного обеспечения, также проектирования программного обеспечения.

2.1 Структура программного обеспечения

Программы – это упорядоченные последовательности команд. Конечная цель любой компьютерной программы – управление аппаратными средствами. Даже если на первый взгляд программа никак не взаимодействует с оборудованием, не требует никакого ввода данных с устройств ввода и не осуществляет вывод данных на устройства вывода, все равно ее работа основана на управлении аппаратными устройствами компьютера.

Программное и аппаратное обеспечение в компьютере работают в неразрывной связи и в непрерывном взаимодействии. Несмотря на то что эти две категории рассматриваются отдельно, нельзя забывать, что между ними существует диалектическая связь, и раздельное их рассмотрение является по меньшей мере условным.

Состав программного обеспечения (ПО) вычислительной системы называют программной конфигурацией. Между программами, как и между физическими узлами и блоками существует взаимосвязь – многие программы работают, опираясь на другие программы более низкого уровня, то есть, мы можем говорить о межпрограммном интерфейсе. Возможность существования такого интерфейса тоже основана на существовании технических условий и протоколов взаимодействия, а на практике он обеспечивается распределением программного обеспечения на несколько взаимодействующих между собой уровней.

Уровни программного обеспечения представляют собой пирамидальную конструкцию. Каждый следующий уровень опирается на программное обеспечение предшествующих уровней. Такое членение удобно для всех этапов работы с вычислительной системой, начиная с установки программ до практической эксплуатации и технического обслуживания. Каждый вышележащий уровень повышает функциональность всей системы. Так, например, вычислительная система с программным обеспечением базового уровня не способна выполнять большинство функций, но позволяет установить системное программное обеспечение.

Самый низкий уровень программного обеспечения представляет базовое программное обеспечение. Оно отвечает за взаимодействие с базовыми аппаратными средствами. Как правило, базовые программные средства непосредственно входят в состав базового оборудования и хранятся в специальных микросхемах, называемых постоянными запоминающими

устройствами (ПЗУ – Read Only Memory, ROM). Программы и данные записываются («прошиваются») в микросхемы ПЗУ на этапе производства и не могут быть изменены в процессе эксплуатации.

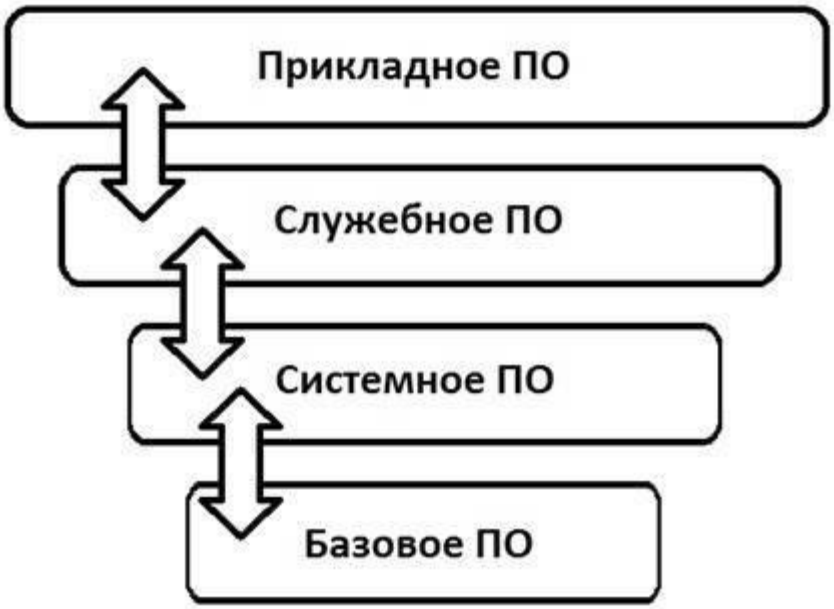

Рисунок 2.1.1 – Структура программного обеспечения

В тех случаях, когда изменение базовых программных средств во время эксплуатации является технически целесообразным, вместо микросхем ПЗУ применяют перепрограммируемые постоянные запоминающие устройства (ППЗУ – Erasable and Programmable Read Only Memory, EPROM). В этом случае изменение содержания ПЗУ можно выполнять как непосредственно в составе вычислительной системы (такая технология называется флэш-технологией), так и вне ее, на специальных устройствах, называемых программаторами.

Системный уровень – переходный. Программы, работающие на этом уровне, обеспечивают взаимодействие прочих программ компьютерной системы с программами базового уровня и непосредственно с аппаратным обеспечением, то есть выполняют «посреднические» функции.

2.2 Проектирование программной части

Проектирование программной части – это один из главных моментов в разработке программного обеспечения, т.к. именно, в этой части определяется вся логика, на которой будет работать программное обеспечение.

На рисунке 2.2.1 представлена блок-схема работы программного обеспечения.

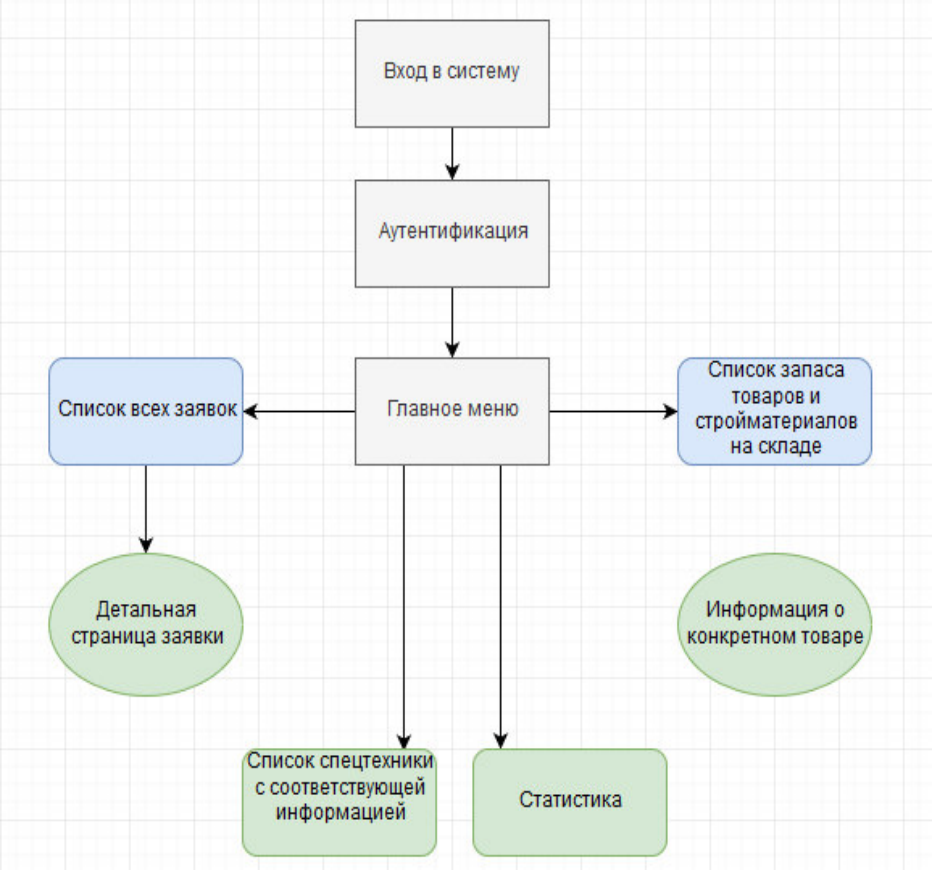

Рисунок 2.2.1 – Блок-схема, которая описывает логику работы программного обечпечения

2.3 Проектирование базы данных

Проектирование баз данных – процесс создания схемы базы данных и определения необходимых ограничений целостности.

Создание информационной системы – это сложный процесс, в котором принимает участие коллектив разработчиков, разбивается на стадии проектирования, программной реализации и эксплуатации.

В процессе создания информационной системы подготавливаются рабочие документы, служащие основой для всех разработчиков и пользователей системы.

Проектирование базы данных заключается в многоступенчатом описании будущей БД с различной степенью детализации и формализации, в ходе которого производится уточнение и оптимизация ее структуры.

Проектирование включает описание предметной области и задач информационной системы, далее идет к логическому описанию данных и затем – к физической модели БД. Различают три этапа детализации описания объектов БД

и их взаимосвязей по трем основным уровням моделирования системы – концептуальному, логическому и физическому.

На концептуальном уровне проектирования производится смысловое (семантическое) описание информационного содержания предметной области, определяются границы предметной области, производится абстрагирование от несущественных для данной информационной системы деталей. В результате определяются моделируемые объекты, их свойства и связи. Выполняется структуризация знаний о предметной области, стандартизируется терминология. Затем строится концептуальная модель, описываемая на естественном языке. Для описания свойств и связей объектов применяют различные диаграммы. Такая модель создаётся без ориентации на какую-либо конкретную СУБД и модель данных. Термины «семантическая модель», «концептуальная модель» и «инфологическая модель» являются синонимами. Кроме того, в этом контексте равноправно могут использоваться слова «модель базы данных» и «модель предметной области» (например, «концептуальная модель базы данных» и «концептуальная модель предметной области»), поскольку такая модель является как образом реальности, так и образом проектируемой базы данных для этой реальности.

Конкретный вид и содержание концептуальной модели базы данных определяется выбранным для этого формальным аппаратом. Обычно используются графические нотации, подобные ER-диаграммам.

Чаще всего концептуальная модель базы данных включает в себя:

– описание информационных объектов или понятий предметной области и связей между ними;

– описание ограничений целостности, то есть требований к допустимым значениям данных и к связям между ними.

На следующем шаге принимается решение о том, в какой СУБД будет реализована БД. Определяющими параметрами являются вид программного продукта и категория пользователей (профессиональные программисты или конечные пользователи, или и то, и другое). Другими показателями, влияющими на выбор СУБД, являются: удобство и простота использования; качество средств разработки, защиты и контроля БД; уровень коммуникационных средств (применение в сетях); фирма-разработчик; стоимость. Каждая конкретная СУБД работает с определенной моделью данных.

Логическое (даталогическое) проектирование – создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных. Для реляционной модели данных даталогическая модель – набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи. На логическом уровне производится отображение данных концептуальной модели в логическую модель,

поддерживаемую выбранной СУБД. Здесь объектом работы выступают сами данные, их структура и правила построения. Логическая модель не зависит от конкретной СУБД – построенная на основе таблиц логическая модель может быть реализована на любой СУБД реляционного типа.

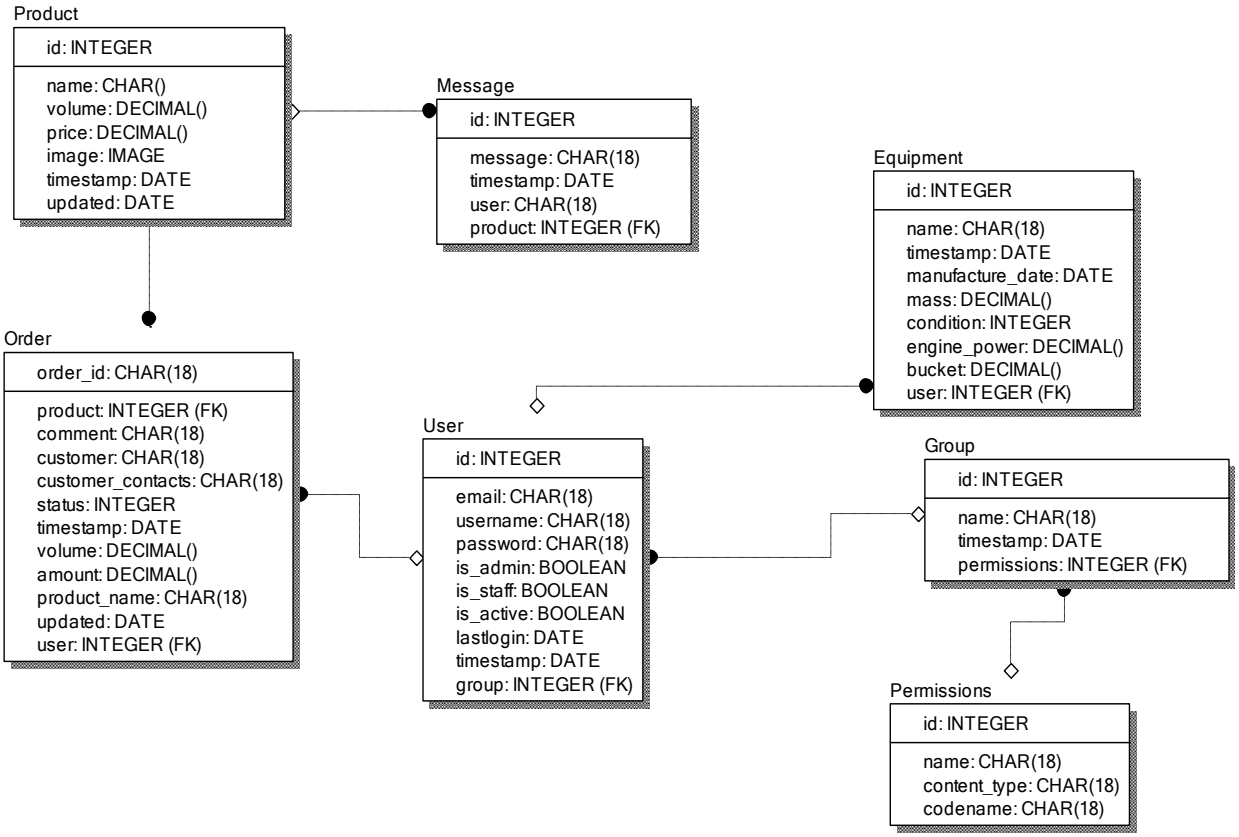

Рисунок 2.3.1 – Логическая модель БД

На физическом уровне производится выбор рациональной структуры хранения данных и методов доступа к ним, решаются вопросы эффективного выполнения запросов к БД, строятся дополнительные структуры, например, индексы. В физической модели содержится информация обо всех объектах БД (таблицах, индексах, процедурах и др.) и используемых типах данных. Физическая модель зависит от конкретной СУБД. Одной и той же логической модели может соответствовать несколько разных физических моделей. Физическое проектирование является начальным этапом реализации БД. Преобразование концептуальной модели в логическую модель, как правило, осуществляется по формальным правилам. Этот этап может быть в значительной степени автоматизирован.

Первая таблица базы данных имеет название «Product»(товары, стройматериалы). Таблица хранит в себе информацию, о доступной на складе товаров и стройматериалов (см рисунок 2.3.2).

```
--
-- Create model Product
--
CREATE TABLE "api_product" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "na
me" varchar(255) NULL, "volume" integer NOT NULL, "price" integer NOT NULL, "ima
ge" varchar(100) NULL);
--
```

Рисунок 2.3.2 – Создание таблицы Product

В миграции Django framework данная таблица выглядит следующим образом (см. рисунок 2.3.3).

```
migrations.CreateModel(
    name='Product',
    fields=[
        ('id', models.AutoField(auto_created=True, primary_key=True,
            serialize=False, verbose_name='ID')),
        ('name', models.CharField(max_length=255, null=True)),
        ('volume', models.IntegerField(default=0)),
        ('price', models.IntegerField(default=0)),
        ('image', models.ImageField(blank=True, null=True,
            upload_to='product/')),
    ],
),
```

Рисунок 2.3.3 – Таблица Product в миграции Django

Вторая таблица «Equipment» – содержит информацию о спецтехнике для погрузки и доставки стройматериалов. Хранит данные об объеме двигателя, подъемной площади, а также состояние транспорта и т.д. (см. рисунок 2.3.4).

```
--
-- Create model Equipment
--
CREATE TABLE "api_equipment" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "
name" varchar(255) NULL, "m_date" datetime NOT NULL, "date" datetime NOT NULL, "
condition" integer NOT NULL);
--
```

Рисунок 2.3.4 – Создание таблицы Equipment

Структура таблицы в миграциях Django framework (см. рисунок 2.3.5).

```
migrations.CreateModel(
    name='Equipment',
    fields=[
        ('id', models.AutoField(auto_created=True, primary_key=True,
            serialize=False, verbose_name='ID')),
        ('name', models.CharField(max_length=255, null=True)),
        ('m_date', models.DateTimeField()),
        ('date', models.DateTimeField(auto_now_add=True)),
        ('condition', models.IntegerField(default=2)),
        ('timestamp', models.DatatimeField(auto_now_add=True)),
        ('updated', models.DatatimeField(auto_now=True)),
    ],
),
```

Рисунок 2.3.5 – Таблица Equipment в миграции Django

Следующая таблица называется «Order» (заявка). Таблица хранит всю информацию о заявках, контактные данные клиента, которые оставил заявку, также данные о заказанном товаре или стройматериале. Количество, и общая сумма всего заказа (см. рисунок 2.3.6).

```
-- Create model Order
--
CREATE TABLE "api_order" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "comm
ent" varchar(255) NULL, "service" varchar(255) NULL, "amount" integer NOT NULL,
"customer" varchar(255) NULL, "customer_contacts" varchar(255) NULL, "status" in
teger NOT NULL, "date" datetime NOT NULL);
--
```

Рисунок 2.3.6 – Создание модели Order

Та же модель «Order» в миграции Django framework (см. рисунок 2.3.7)

```
migrations.CreateModel(
    name='Order',
    fields=[
        ('id', models.AutoField(auto_created=True,
            primary_key=True, serialize=False, verbose_name='ID')),
        ('comment', models.CharField(max_length=255, null=True)),
        ('service', models.CharField(max_length=255, null=True)),
        ('amount', models.IntegerField(default=0)),
        ('customer', models.CharField(max_length=255, null=True)),
        ('customer_contacts', models.CharField(max_length=255,
            null=True)),
        ('status', models.IntegerField(
            choices=[
                (1, 'В очереди'),
                (2, 'В процессе'),
                (3, 'Завершена')],
                default=1)),
        ('date', models.DateTimeField(auto_now_add=True)),
    ],
),
```

Рисунок 2.3.7 – Модель Order в миграции Django

3 Разработка программного продукта

Клиент-серверное приложение представляет собой приложение, которое имеет две части. В моем случае клиентская часть разделяется на две состовляющие:

– админ панель. Взаимодействует через браузер;

– графический интерфейс для сотрудников компании. Десктопная программа с удобным графическим интерфейсом.

Клиент, это та часть приложения, которая отображается пользователю, выполняется в веб-браузере и взаимодействует визуально с пользователем. На этой стороне работают такие языки разметки, стилей и программирования как HTML, CSS и JavaScript.

Серверная часть приложения не имеет собственного визуального представления и взаимодействует с пользователем через веб-браузер и десктопное программное обеспечение. Название этой части вытекает из того, что все действия выполняются на сервере – специальном компьютере, который может быть расположен как за тысячи километров от браузера, так и в непосредственной близости, вплоть до одной машины. На сервере обычно располагается база данных и оперируют такие языки как Java, PHP, C# и т. д. Данное приложение разрабатывается на языке программирования Java.

3.1 Создание Django проекта для серверной части.

В начале разработки программного продукта необходимо создать проект, который будет включать в себя все необходимые средства для разработки данного проекта. Для начала создаем директорию для расположения нашего проекта. Первым делом создаем виртуальную среду для того, чтобы обеспечить независимую среду для пакетов и библиотек python. В корне своем, главная задача виртуальной среды Python – создание изолированной среды для проектов. При разработке Python-приложений или использовании решений на Python, созданных другими разработчиками, может возникнуть ряд проблем, связанных с использованием библиотек различных версий. Рассмотрим их более подробно.

– различные приложения могут использовать одну и ту же библиотеку, но при этом требуемые версии могут отличаться;

– может возникнуть необходимость в том, чтобы запретить вносить изменения в приложение на уровне библиотек, т.е. вы установили приложение и хотите, чтобы оно работало независимо от того обновляются у вас библиотеки или нет. Как вы понимаете, если оно будет использовать библиотеки из

глобального хранилища (/usr/lib/pythonXX/site-packages), то, со временем, могут возникнуть проблемы;

– у вас просто может не быть доступа к каталогу /usr/lib/pythonXX/site-packages.

Основное отличие venv в том, что он встроен в интерпретатор и может отрабатывать ещё до загрузки системных модулей. Для этого, при определении базовой директории с библиотеками, используется примерно такой алгоритм:

– в директории с интерпретатором или уровнем выше ищется файл с именем pyvenv.cfg;

– если файл найден, в нём ищется ключ home, значение которого и будет базовой директорией;

– в базовой директории идёт поиск системной библиотеки (по спец. маркеру os.py).

Для решения данных вопросов используется подход, основанный на построении виртуальных окружений – своего рода песочниц, в рамках которых запускается приложение со своими библиотеками, обновление и изменение которых не затронет другие приложение, использующие те же библиотеки.

ПО позволяющее создавать виртуальное окружение в Python:

– virtualenv. Это, наверное, одни из самых популярных инструментов, позволяющих создавать виртуальные окружения. Он прост в установке и использовании. В общем, этот инструмент нужно обязательно освоить, как минимум, потому что описание развертывания и использования многих систем, созданных с использованием Python, включает в себя процесс создания виртуального окружения с помощью virtualenv;

– pyenv. Инструмент для изоляции версий Python. Чаще всего применяется, когда на одной машине вам нужно иметь несколько версий интерпретатора для тестирования на них разрабатываемого вами ПО.

– Virtualenvwrapper – это обертка для virtualenv позволяющая хранить все изолированные окружения в одном месте, создавать их, копировать и удалять. Предоставляет удобный способ переключения между окружениями и возможность расширять функционал за счет plug-in’ов.

Создание виртуальной среды для изоляции библиотек (см. рисунок 3.1.1)

Рисунок 3.1.1 – Создание виртуальной среды

Виртуальная среда имеет следующую структуру (см. рисунок 3.1.2).

Здесь содержаться все необходимые библиотеки т.е., все стандартные библиотеки python. Говоря другими словами, у нас независимый с фиксированными версиями стандартных, так и сторонних библиотек или фреймворков.

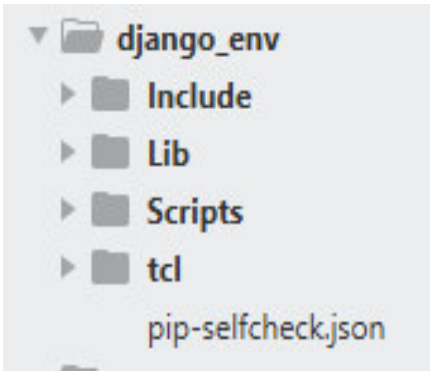

Рисунок 3.1.2 – Структура изолированной среды разработки

Далее, для установки необходимых пакетов и библиотиек (Django, pyqt5, Django-rest-framework и т.д.) необходимо запустить нашу изолированную среду (см.. рисунок 3.1.3).

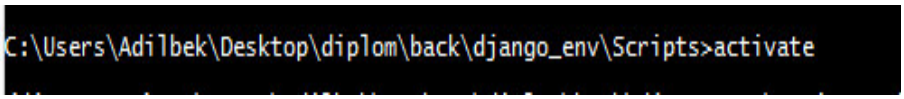

Рисунок 3.1.3 – Запуск виртуальной среды

После запуска мы модем заметить в консоли название нашей виртуальной среды (см. рисунок 3.1.4).

Рисунок 3.1.4 – Результат запуска виртуальной среды

Теперь, когда мы запустили нашу изолированную среду, необходимо создать Django проект (см. рисунок 3.1.5).

Рисунок 3.1.5 – Инициализация Django проекта

Структура нашего проекта после инициализации Django проекта (см. рисунок 3.1.6).

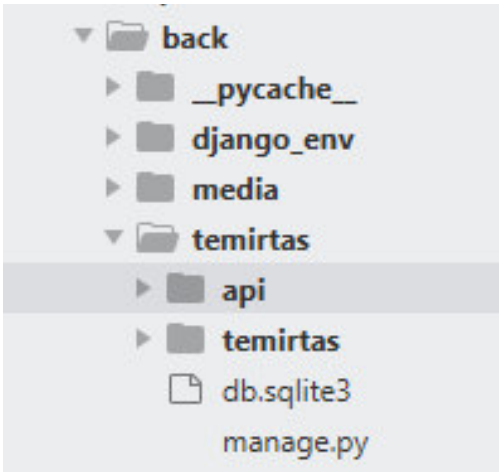

Рисунок 3.1.6 – Структура Django проекта

Рассмотрим эти файлы:

– внешний каталог «back» – это просто контейнер для нашего проекта. Его название никак не используется Django, и мы можем переименовать его во что угодно;

– manage.py: Скрипт, который позволяет вам взаимодействовать с проектом Django. Подробности о manage.py читайте в разделе django-admin.py и manage.py;

– внутренний каталог temirtas/ - это пакет Python вашего проекта. Его название – это название пакета Python, которое вы будете использовать для импорта чего-либо из проекта (например, mysite.urls);

– temirtas/__init__.py: Пустой файл, который указывает Python, что текущий каталог является пакетом Python;

– temirtas /settings.py: Настройки/конфигурация проекта. Раздел Django settings расскажет вам все о настройках проекта.

В файле settings.py содержится очень много информации о нашем проекте. Секретный ключь проекта – который используется для идентификации проекта. Статус Debug помогает при отладке программы. Если в значении debug стоит «true», то программа в случае возникновения ошибки выдает полную информацию об ошибке (тип ошибки, машина на котором возникла ошибка и т.д.). Также значение для переменной типа string SECRET_KEY значение генерируется автоматически при создании проекта. Ни в коем случае нельзя развертывать проект на сервере с положительным значением debug (см. рисунок 3.1.7).

```
SECRET_KEY = '(yipyj=_kqpo#92wd3b&spb9lx$y3c$bx1(9wc*&a+t$jc8%@p'
DEBUG = True
```

Рисунок 3.1.7 – Секретный ключь и debug для отладки внутри файла settings.py

Installed_ Apps – это список установленных приложении в нашем проекте (см. рисунок 3.1.8). Как мы можем видеть, проект изначально содержить некоторые стандартные приложения для админ панели, также для авторизации пользователей, контроля сессии и хранение статичных файлов.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'api',
    'rest_framework'
]
```

Рисунок 3.1.8 – Стандартные установленные приложения

Далее, настраиваем admin.py для коректного отображения наших данных. Для того чтобы войти сначала создаем пользователя (см. рисунок 3.1.9).

```
(django_env) C:\Users\Adilbek\Desktop\diplom\back\temirtas>py manage.py createsu
peruser

You have 1 unapplied migration(s). Your project may not work properly until you
apply the migrations for app(s): api.
Run 'python manage.py migrate' to apply them.
Username: adil
```

Рисунок 3.1.9 –Создание админа

Теперь можем приступить к настройке админ панели. Подключаем наши модели, которые должны отображаться. Подключаем нужные библиотеки (см. рисунок 3.1.10).

```
from django.contrib import admin
from api.models import Order, Product
from django.utils.safestring import mark_safe
from django.utils.encoding import force_text
from django.urls import reverse
```

Рисунок 3.1.10 – Подключение библиотек и моделей в admin.py

Подключение модели Product (см. рисунок 3.1.11).

```
@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    list_display = (
        'name', 'volume', 'price'
    )
    fieldsets = (
        (None,
            {'fields': (
                'name', 'volume',
                'price', 'image'
            )
        }),
    )
```

Рисунок 3.1.11 – Подключение модели Product в admin.py

Подключение модели Order (см. рисунок 3.1.12). Кортеж «list_display» отвечает за те данные, которые отображаются при открытии списка всех заявок. Переменная обязательно должна быть кортежом, либо массивом. Название полей, включенные в переменную «list_filter» определяют список полей, по которым будет предоставлена фильтрация в списке всех заявок.

```python
@admin.register(Order)
class OrderAdmin(admin.ModelAdmin):
    list_display = (
        'order_id', 'customer',
        'customer_contacts',
        'comment', 'product',
        'volume', 'amount',
        'product_name')
    list_filter = ('product',)
    fieldsets = (
        ('Заявка',
            {'fields': (
                'customer',
                'customer_contacts', 'comment',
                'product', 'volume', 'status')
        }),
    )
```

Рисунок 3.1.12 –Подключение модели Order

После регистрации необходимых полей в админке и их настройки для удобного вывода данных открываем браузер и переходим по «localhost:8000/admin». Первым делом можно увидеть окно входа (см. рисунок 3.1.13). В окне входа надо ввести «username» и «password», то есть логин и пароль пользователя у которого есть статус администрации или статус сотрудника компании ТОО «Темиртас – 1». При созданий пользователя для логина и пароля есть ограничения. Например, пароль должен состоять из:

– как минимум из 8 символов;

– обязательно должна быть как минимум одна заглавная буква;

– обязательно должна быть как минимум одна цифра;

– пароль не должен совпадать с именем пользователя;

– пароль не должен содержать часто встречающие символы (например: qwerty, 123456 и т.д.).

Авторизуемся под пользователем, которого создали ранее с помощью команды «python manage.py createsuperuser» в консоли (см. рисунок 3.1.13).

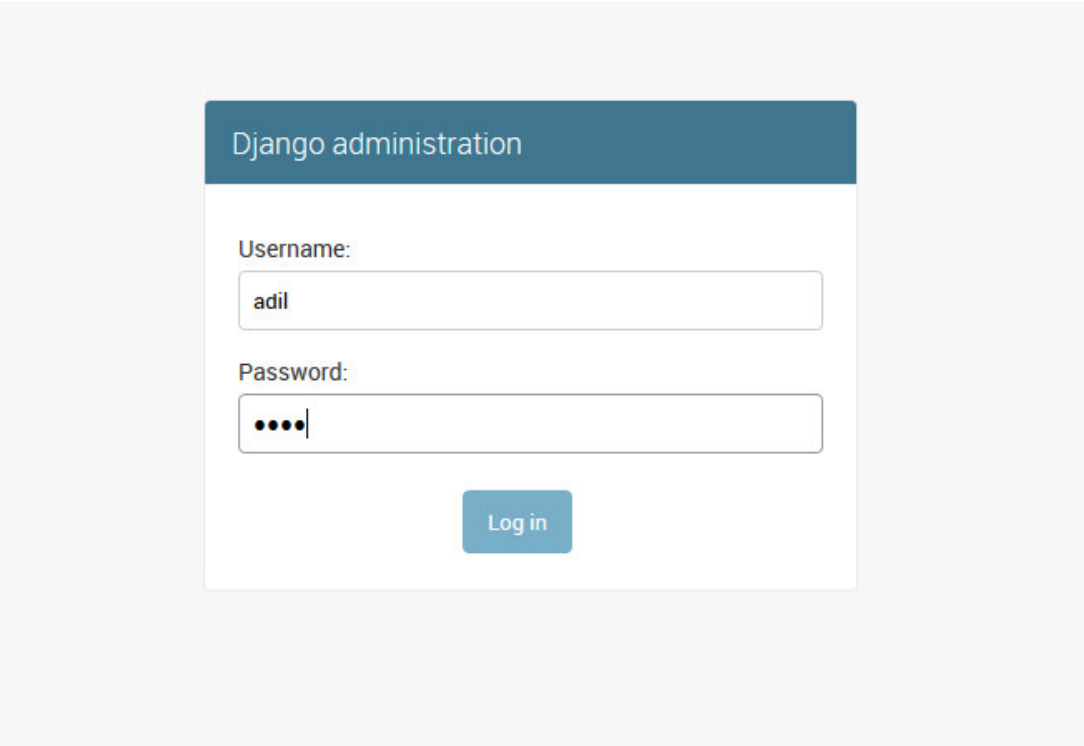

Рисунок 3.1.13 – вход в админ панель

Теперь после входа в админку мы можем видеть подключенные модели, а также стандартные модели пользователей и группы для распределении привелегии (см. рисунок 3.1.14).

После того как мы успешно авторизовались в администрацию приложения, также можно увидеть главную страницу администрации (см. рисунок 3.1.14). Первым делом можно заметить список моделей, зарегистрированных ранее. На данной странице доступны следующие действия:

– поле «Добавить продукты». Справа от названия поля есть кнопки для добавления или изменения продуктов. Также при переходе по полю откроется новая страница со всеми ранее добавленными продуктами;

– поле «Список заявок». Справа от названия также с помощью ссылок возможно добавить или изменить уже добавленную заявку. При переходе также откроется новая страница со списком ранее добавленных заявок;

– поле «Сообщения сотрудников». В десктопной программе для сотрудников есть возможность предупредить администрацию о малом запасе или о том, что товар или продукт полностью закончился на складе;

– поле «Спецтехника». При клике открывается страница со всей ранее добавленной спецтехникой. При добавлении указывается год, состояние, марка и технические характеристики техники.

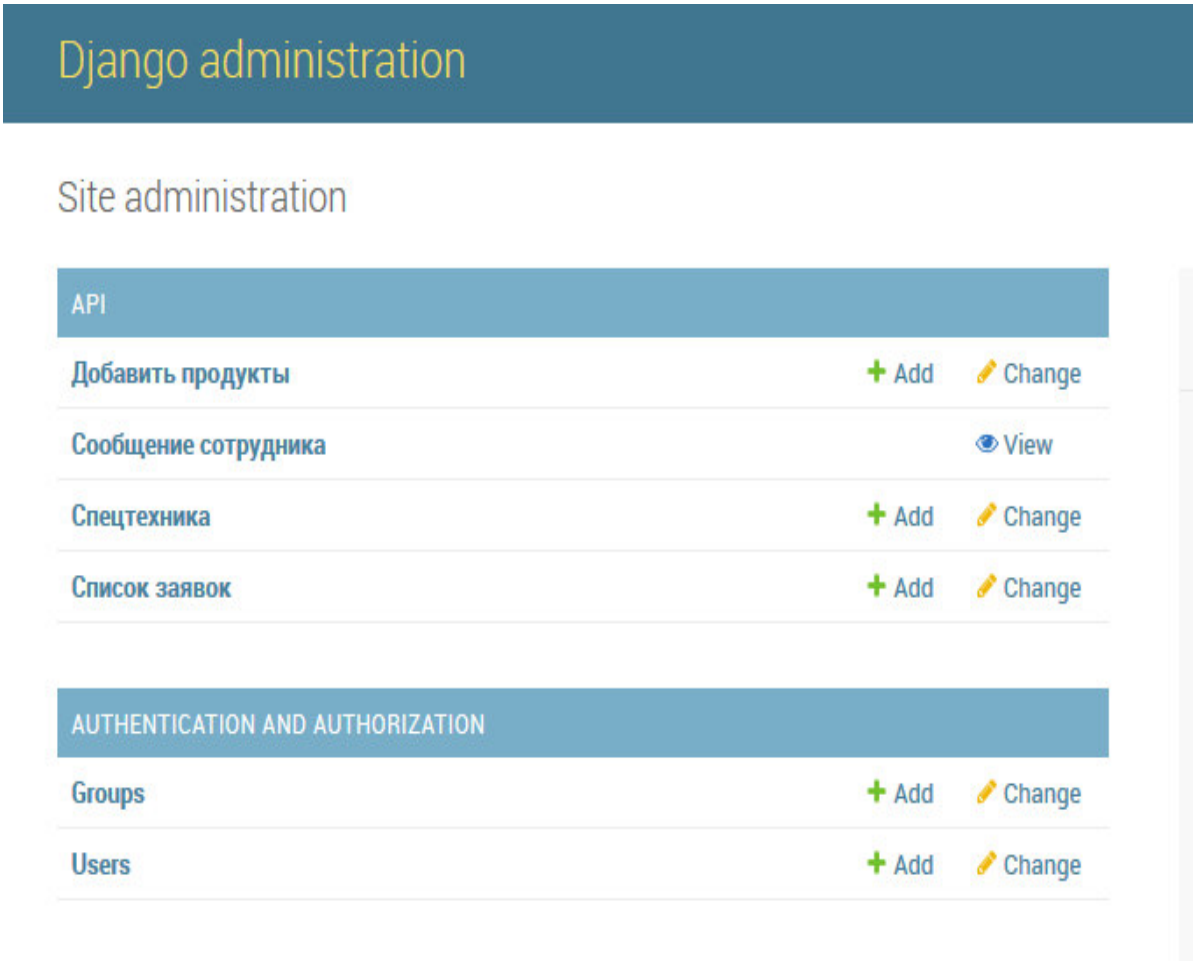

Рисунок 3.1.14 – Главная страница Django admin панели

После перехода по «Добавить продукты» мы получим список всех продуктов, которые есть на складе и добавлены в базу (см. рисунок 3.1.15). На данной странице возможны следующие действия:

– просмотреть детальную информацию о каждом товаре;

– добавить новые товары или стройматериалы в базу;

– удаление одного или нескольких товаров из базы;

Также при желании в файле «admin.py», где прописываются все настройки и регистрируется модели для отображения на главной странице администрации, можно изменить данные, которые будут отображаться в списке всех товаров, также указать поля по которым можно будет фильтровать список продуктов и указать список неизменяемых полей для данной модели. К примеру, время

добавления продукта берется текущее время с учетом часового пояса и сохраняется в базе.

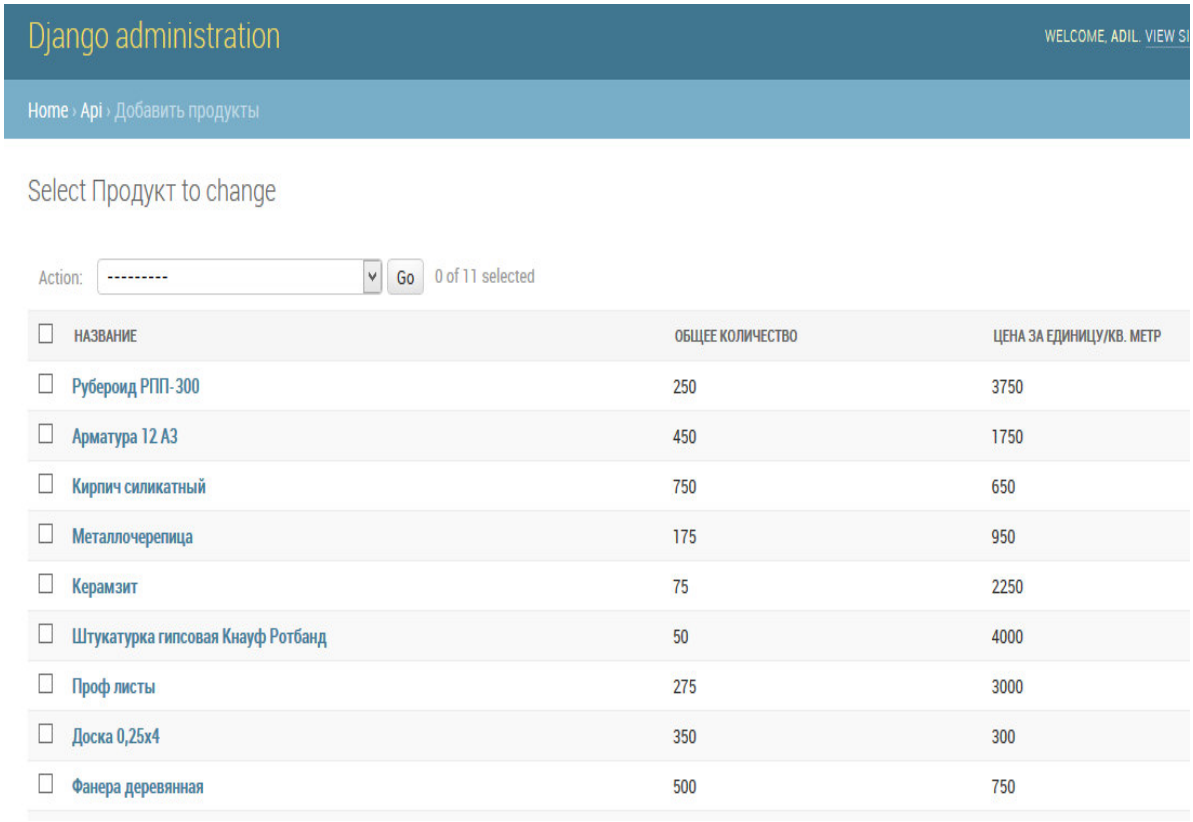

Рисунок 3.1.15 – Список товаров и стройматериалов

Кликнув на «добавить товар» мы можем добавить новый товар или стройматериал (см. рисунок 3.1.16).

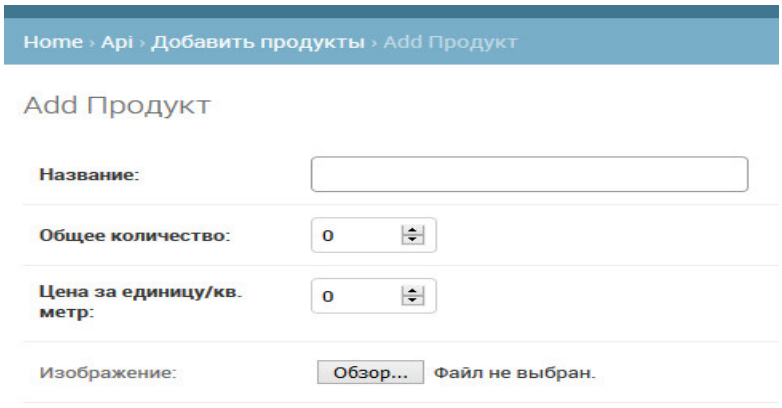

Рисунок 3.1.16 – Добавление нового товара

Еще одна модель которую мы добавили в панель админа – это заявки (Order) (см. рисунок 3.1.17)

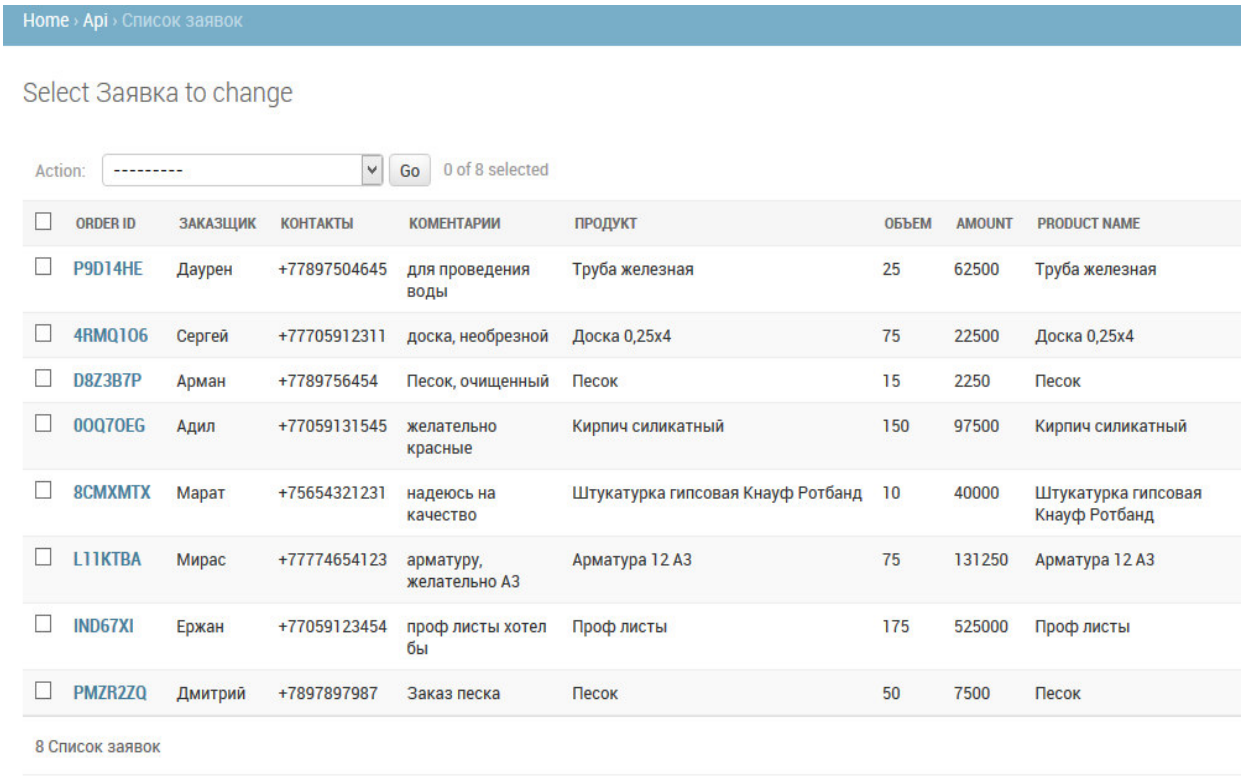

Рисунок 3.1.17 – Список заявок в панели админа

На данной странице также присутствуют фильтры, для сортировки и фильтрации товаров по нужным нам криетериям (см. рисунок 3.1.18).

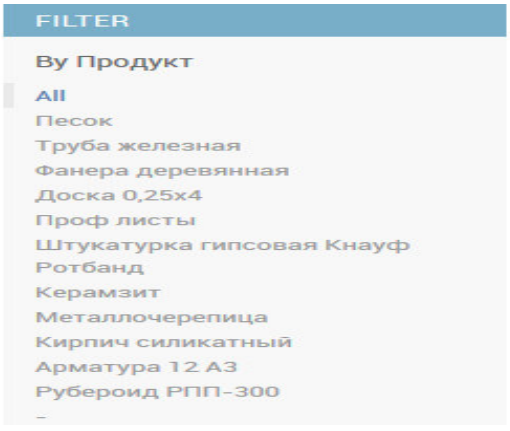

Рисунок 3.1.18 – Фильтрация заявок по товарам

Чтобы внести заявку клиента, нажимаем на кнопку «Добавить заявку». После добавления заявка моментально будет отображаться в списке всех заявок нашей десктопной программы (см. рисунок 3.1.19).

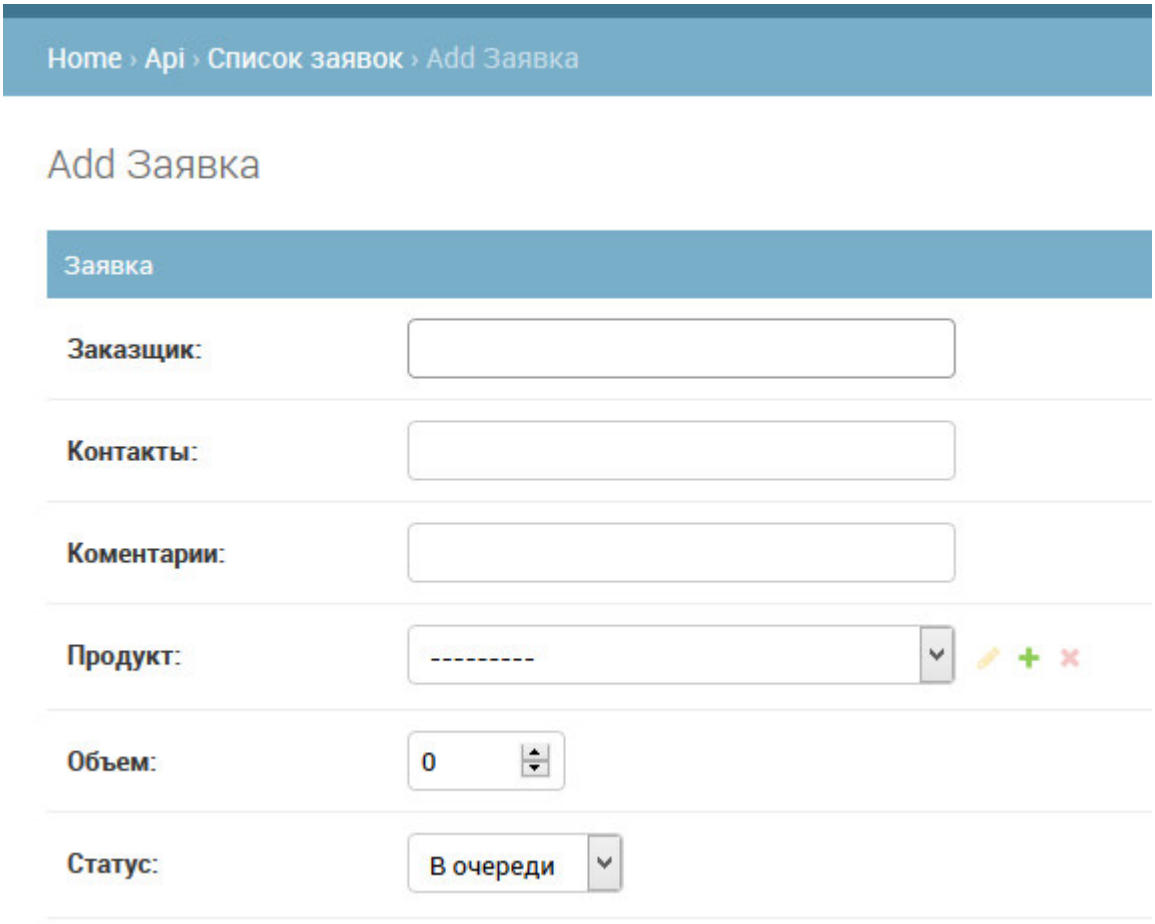

Рисунок 3.1.19 – Внесение клиентской заявки

3.2 Настройка API

REST (сокращение от англ. Representational State Transfer – «передача состояния представления») – архитектурный стиль взаимодействия компонентов распределённого приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы. В определённых случаях (интернет-магазины, поисковые системы, прочие системы, основанные на данных) это приводит к повышению производительности и упрощению архитектуры. В широком смысле компоненты в REST взаимодействуют наподобие взаимодействия клиентов и серверов во Всемирной паутине. REST является альтернативой RPC.

В сети Интернет вызов удалённой процедуры может представлять собой обычный HTTP-запрос (обычно «GET» или «POST»; такой запрос называют «REST-запрос»), а необходимые данные передаются в качестве параметров запроса. Для веб-служб, построенных с учётом REST (то есть не нарушающих накладываемых им ограничений), применяют термин «RESTful».

В отличие от веб-сервисов (веб-служб) на основе SOAP, не существует «официального» стандарта для RESTful веб-API. Дело в том, что REST является архитектурным стилем, в то время как SOAP является протоколом. Несмотря на то, что REST не является стандартом сам по себе, большинство RESTful-реализаций используют стандарты, такие как HTTP, URL, JSON и XML.

Для начала, определяем пути (url-пути) для нашего api. Переходим в файл urls.py и пишем следующие роуты (см. рисунок 3.2.1).

```
urlpatterns = [
    path('login/', Login.as_view()),
    path('orders/', OrderList.as_view()),
    path('order/<int:order_id>',
        OrderDetail.as_view()),
    path('order/<int:order_id>/',
        UpdateOrder.as_view()),
    path('products/', ProductList.as_view()),
    path('product/<int:product_id>',
        ProductDetail.as_view()),
    path('message/', MessageView.as_view()),
    path('equipment/', TexnikaList.as_view()),
    path('main/', Main.as_view())

]
```

Рисунок 3.2.1 – Роуты для api

Наш файл urls.py содержит следующие роуты:

– «login/» – для аутентификации пользователя и входа в систему. Вход в систему работает следующим образом: сначало проверяется логин на соответствие с логином из базы. Далее пароль хешируется и проверяется с хэшом, которая ранее, при регистрации сохранилась в базе;

– «orders/» – для получения списка всех заявок;

– «orders/int:order_id» – для получения детальной информации конкретной заявки по первичному ключу, точнее по «order_id»;

– «product/» – для получения списка всех товаров склада;

– «product/<int: product_id>» – для получения детельной информации определенного товара или стройматериала;

– «message/» – для уведомления администрации о том, что заканчаивается определенный товар;

– «equipment/» – для получения списка спецтехники и оборудования.

После того, как наши роуты готовы, пишем для них контроллеры. Аутентификация пользователя (см. рисунок 3.2.2).

```python
class Login(APIView):
    def post(self, request, *args, **kwargs):
        username = request.data.get('login', False)
        password = request.data.get('password', False)

        if not username or not password:
            return Response({
                "message": "Данные не подходят",
                "status": False
            })

        user = authenticate(request, username=username, password=password)
        if not user:
            return Response({
                "message": "Неправильный логин или пароль",
                "status": False
            })
        else:
            login(request, user)
            return Response({
                "message": f'Добро пожаловать, {user.first_name} {user.last_name}',
                "firstname": user.first_name,
                "lastname": user.last_name,
                "status": True
            })
```

Рисунок 3.2.2 – Аутентификация пользователя

Получение всех заявок, и детальная информация определенной заявки(см. рисунок 3.2.3).

```python
class OrderList(APIView):
    def get(self, request):
        orders = Order.objects.all()
        serializer = OrderSerializer(orders, many=True)
        return Response({
            "message": "",
            "status": True,
            "data": serializer.data
        })

class OrderDetail(APIView):
    def get(self, request, order_id):
        order = Order.objects.filter(id =order_id).first()
        data = {
            "order_id": order.order_id,
            "customer": order.customer,
            "customer_contacts": order.customer_contacts,
            "comment": order.comment,
            "status": order.status,
            "product": order.product.name,
            "volume": order.volume,
            "amount": order.amount,
            "date": order.date,
            "image": order.product.image.url
        }
        return Response({"message": "","status": True, "data": data})
```

Рисунок 3.2.3 – Получение списка всех заявок, и детальная информация определленой заявки

Также в программе сотрудник по мере продвижения работы может изменить статус заявки (см. рисунок 3.2.4). Новый статус приходит с клиентской части при изменении статуса и сохранении заявки. Доступны три статуса:

– в очереди. То есть, заявка внесена в базу, еще не обработана;

– в процессе. Заявка принята и ждет обработки и доставки;

– завершена. Заявка уже обработа и заказанные товары или стройматериалаы доставлены заказщикам.

```python
class UpdateOrder(APIView):
    def post(self, request, *args, **kwargs):
        order_id = request.data.get('order_id', False)
        status = request.data.get('status', False)

        if not order_id or not status:
            return Response({
                "message": "Вы не изменили статус",
                "status": False
            })

        order = Order.objects.filter(id=order_id).first()
        new_status = 0
        if status == "В очереди": new_status=1
        elif status == "В процессе": new_status=2
        elif status == "Завершена": new_status=3

        order.status=new_status
        order.save()
        return Response({
            "message": f'Статус заявки изменен на "{status}"',
            "status": True
        })
```

Рисунок 3.2.4 – Обновление статуса заявки

Напишем также контроллеры для получения списка всех товаров, и детальной информации определенного товара (см. рисунок 3.2.5). При запросе на адрес «products/», серверная часть принимает запрос и вызывает класс «ProductList», который берет из базы все добавленные продукты и изменяет их форму для того, чтобы можно было правильно отобразить данные, и отправить обратно в клиентскую часть. При запросе на «product/<int:product_id>», сервер принимает запрос и вызывает класс «ProductDetail». Данный класс при выполнении берет определенный товар или стройматериал по первичному ключу «product_id» и также сериализует обьект для правильного отображения и отправляет обратно в клиентскую часть, т.е. в нашу десктопную программу.

```python
class ProductList(APIView):
    def get(self, request):
        products = Product.objects.all()
        serializer = ProductSerializer(products, many=True)
        return Response({"data":serializer.data})

class ProductDetail(APIView):
    def get(self, request, product_id):
        product = Product.objects.filter(id=product_id).first()
        serializer = ProductSerializer(product)

        return Response({"data": serializer.data})
```

Рисунок 3.2.5 – Список всех товаров, детальная информация определенного товара

Функция для уведомления администрации о малом запасе определенного товара на складе (см. рисунок 3.2.6).

```python
class MessageView(APIView):
    def post(self, request, *args, **kwargs):
        message = request.data.get('message', False)
        product_id = request.data.get('product_id', False)

        if not message or not product_id:
            return Response({"message": "Не полная информация"})

        message = Message.objects.create(
            message=message,
            product_id=product_id,
        )
        message.save()
        return Response({"message": "Спасибо, мы проверим наличие продукта на складе"})
```

Рисунок 3.2.6 – Уведомление администрации о малом запасе определенного товара на складе

Абсолютно все вышенаписанные классы и их функции возвращают объект типа «Queryset». Для преобразования данных объектов в читамые строки, используется сериализация. В директории нашего проекта создаем и переходим в файл serializers.py. Далее, создаем наши serializer – классы для моделей (см. рисунок 3.2.7).

```python
from rest_framework import serializers
from api.models import Order, Product, Equipment


class OrderSerializer(serializers.ModelSerializer):
    class Meta:
        model = Order
        fields = '__all__'


class ProductSerializer(serializers.ModelSerializer):
    class Meta:
        model = Product
        fields = ('__all__')
        # fields = ('id', 'name', 'amount', 'price', 'image')
```

Рисунок 3.2.7 – Сериализация модели «Order» и «Product»

Теперь, когда наши пути запросов (url endpoints) готовы, протестируем их с помощью бесплатной программы Postman. Основное предназначение приложения – создание коллекций с запросами к вашему API. Любой разработчик или тестировщик, открыв коллекцию, сможет с лёгкостью разобраться в работе вашего сервиса. Ко всему прочему, Postman позволяет проектировать дизайн API и создавать на его основе Mock-сервер. Вашим разработчикам больше нет необходимости тратить время на создание "заглушек". Реализацию сервера и клиента можно запустить одновременно. Тестировщики могут писать тесты и производить автоматизированное тестирование прямо из Postman. А инструменты для автоматического документирования по описаниям из ваших коллекций сэкономят время на ещё одну "полезную фичу". Есть кое-что и для администраторов – авторы предусмотрели возможность создания коллекций для мониторинга сервисов.

Протестируем через Postman адрес для получения всех заявок (см. рисунок 3.2.8).

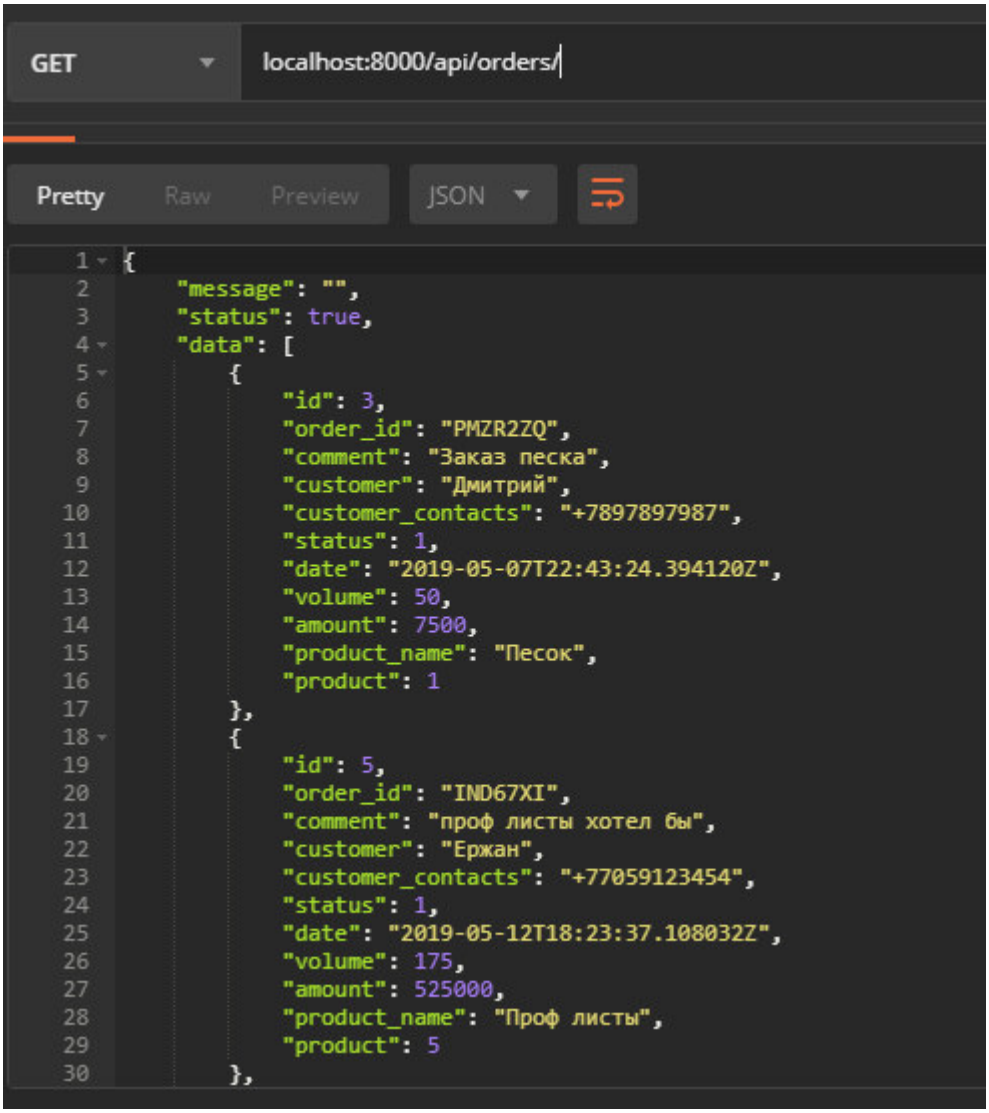

Рисунок 3.2.8 – Проверка адреса «/orders/»

3.3 Создание графического интерфейса с помощью PyQt5

Графический пользовательский интерфейс (GUI) – разновидность пользовательского интерфейса, в котором элементы интерфейса (меню, кнопки, значки, списки и т. п.), представленные пользователю на дисплее, исполнены в виде графических изображений. Также называется графической оболочкой управления. В отличие от интерфейса командной строки, в GUI пользователь имеет произвольный доступ (с помощью устройств ввода – клавиатуры, мыши, джойстика и т. п.) ко всем видимым экранным объектам (элементам интерфейса) и осуществляет непосредственное манипулирование ими. Чаще всего элементы интерфейса в GUI реализованы на основе метафор и отображают их назначение и

свойства, что облегчает понимание и освоение программ неподготовленными пользователями.

Графический интерфейс пользователя является частью пользовательского интерфейса и определяет взаимодействие с пользователем на уровне визуализированной информации.

Для начала установим библиотеку PyQt5 в нашу изолированную среду (см. рисунок 3.3.1).

```
(django_env) C:\Users\Adilbek\Desktop\diplom\back>pip install pyqt5
Requirement already satisfied: pyqt5 in c:\users\adilbek\desktop\diplom\back\dja
ngo_env\lib\site-packages (5.12.2)
```

Рисунок 3.3.1 – Установка библиотеки PyQt5

Первым делом напишем страницу входа в систему, которая делает запрос на адрес «login/» из нашей сервеной части (см. рисунок 3.3.2).

```
class mywindow(QtWidgets.QMainWindow):
    def __init__(self):
        super(mywindow, self).__init__()
        self.setWindowIcon(QtGui.QIcon('icon.jpg'))
        self.ui = login.Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.pushButton.clicked.connect(self.signup)

    def signup(self):
        login = self.ui.lineEdit.text()
        password = self.ui.lineEdit_2.text()
        if not login or not password:
            QMessageBox.about(self, "Ошибка", "Введите логин и пароль")
        else:
            response = requests.post('http://127.0.0.1:8000/api/login/',
                data={"login": login, "password": password})
            if not response.status_code == 200:
                QMessageBox.about(self,
                    "Ошибка", "Технические неполадки на сервере")
            else:
                response = response.json()
                if not response.get("status"):
                    QMessageBox.about(self,
                        " ", response.get("message"))
                else:
                    QMessageBox.about(self,
                        "TOO Temirtas-1", response.get("message"))
                    fullname = response.get("lastname") + ' ' + response.get("firstname")
                    self.main_template(fullname)
```

Рисунок 3.3.2 – Страница входа в систему

При запуске нашей программы мы видим следующее окно (см. рисунок 3.3.4).

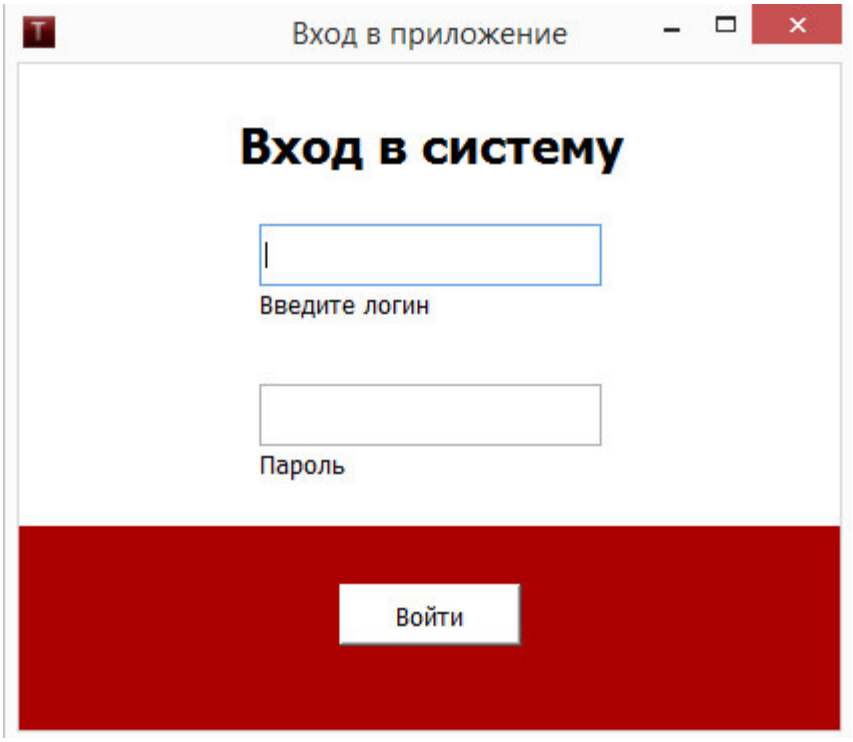

Рисунок 3.3.4 – Вход в систему

После того, как пользователь вводит свой логин и пароль данные отправляются на серверную часть для проверки валидности внесенных данных, и в результате успешной аутентификации, пропускает на следующую страницу, в противном случае, информирует о неправильности введенных данных (см. рисунок 3.3.5).

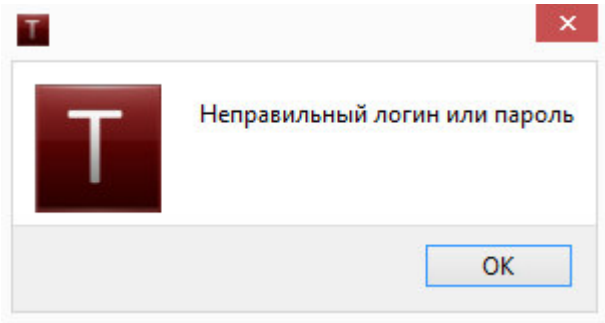

Рисунок 3.3.5 – Неправильные данные

Далее, пишем код для главной страницы, и подключаем роуты из нашего api для получения нужных данных (см. рисунок 3.3.6).

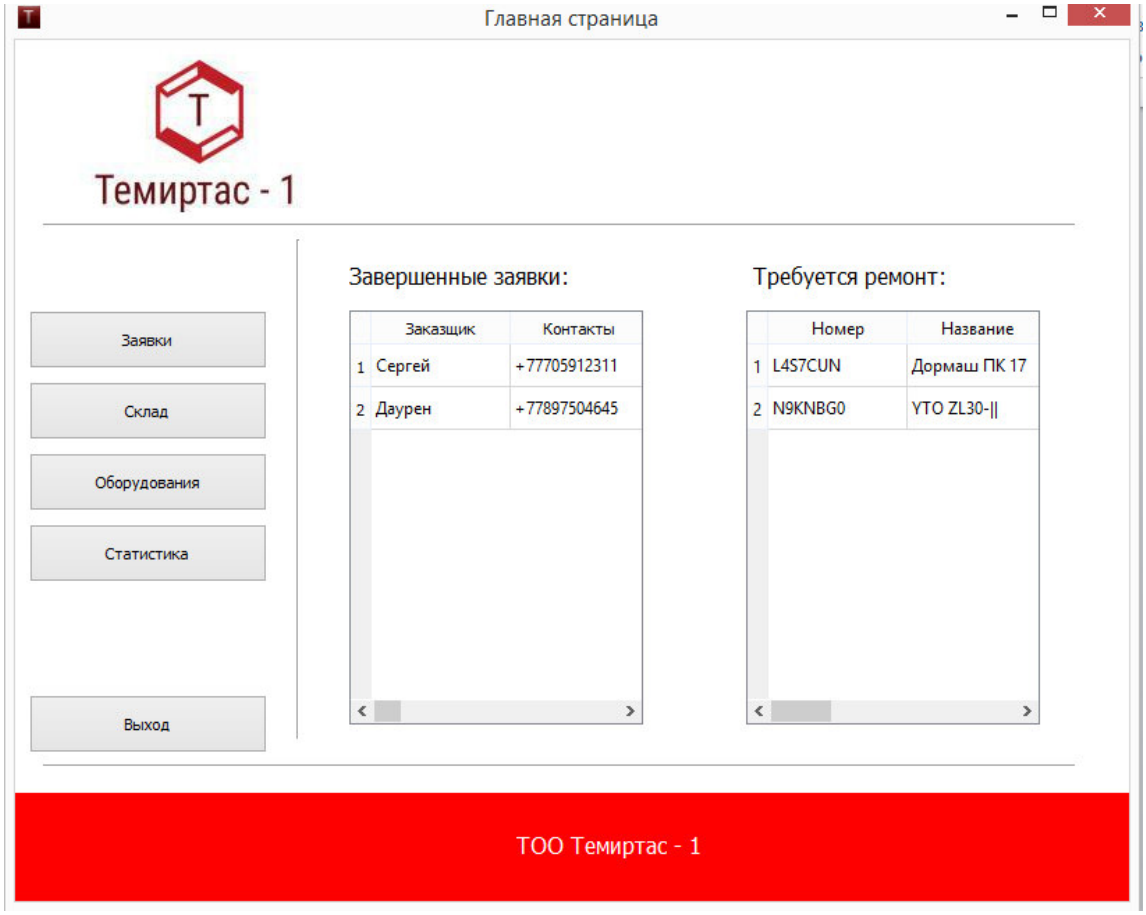

Рисунок 3.3.6 – Главная страница

В главной странице мы можем видеть список завершенных заявок и список спецтехники которые нуждаются в обслуживании. Также в левой части страницы мы можем перейти по следующим страницам:

– заявки – список всех заявок;

– склад – список доступных на складе товаров и стройматериалов;

– оборудования – список спецтехники и их характеристики;

– статистика – по завершенным заявкам.

При переходе в список заявок, получаем список всех заявок с первоначальной информацией (см. рисунок 3.3.7). Здесь можно видеть неполную информацию о заявках:

– заказщик;

– контакные данные заказщика;

– комментарий заказщика к заказу;

– статус заявки (в процессе, завершен);
– продукт, который был заказан;
– прочие характеристики и общая сумма.

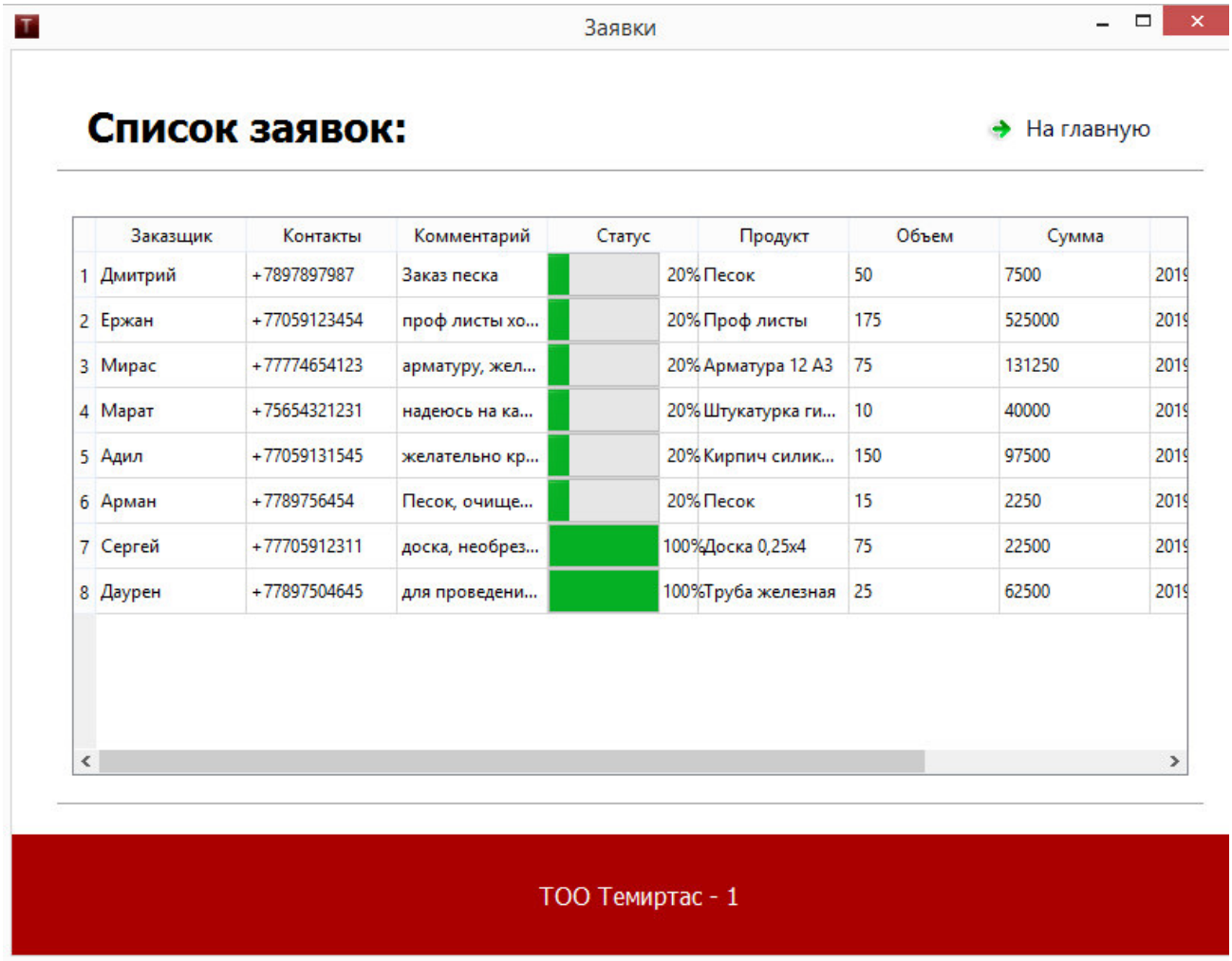

Рисунок 3.3.7 – Страница со всеми заявками

Статус отображается в виде элемента «StatusBar» чтобы интуитивно было понятно, в какой стадии находится заявка. При желании можно перейти к заявке и просмотреть детальную информацию, также реактивно изменить статус заявки (см. рисунок 3.3.8). В самом верху окны прогрммы можно увидеть уникальный идентификатор заявки. Данный идентификатор генерируется случайным образом, в программном коде, исключая возможности повторения одного значения уникального идентификатора для двух заявок. Ниже от уникального идентификатора показывается дата добавления заявки. Также, детальная информация в главной части окна программы.

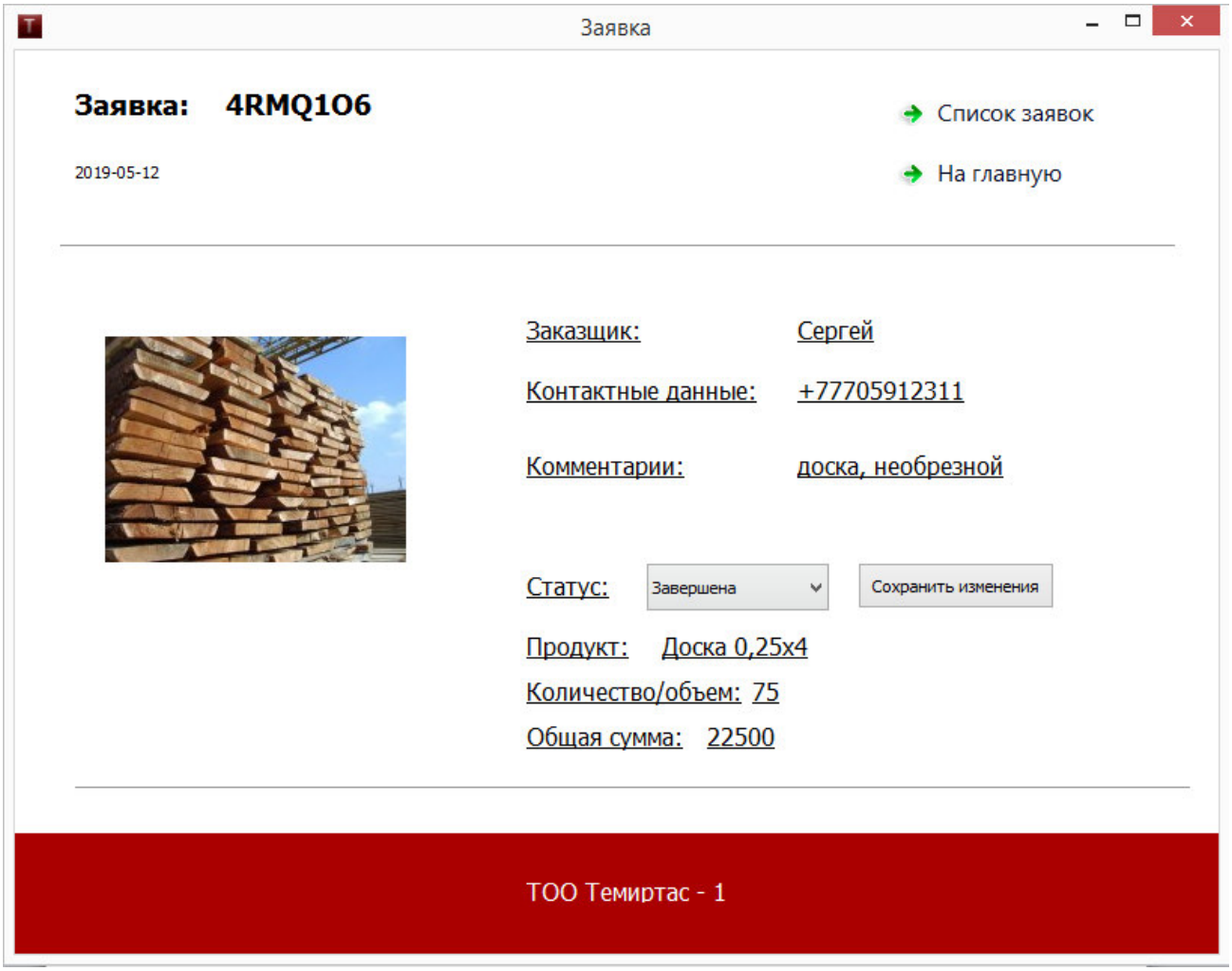

Рисунок 3.3.8 – Страница с полной информации о заявке

Переходим обратно в главное меню, нажимая на кнопку «На главную». Далее, переходим на страницу «Склад». Здесь получаем список всех товаров, которые имеются на складе (см. рисунок 3.3.9). Список товаров или стройматериалов, который отображаются в списке, добавляются через администрацию приложения. После добавления в базу, товар или стройматериал моментально отображается в списке товаров на складе.

Для получения информации о товара надо кликнуть по нему. После, в случае если запас товара на складе крайне мало, есть возможность сообщить об этом администрацию (см. рисунок 3.3.10). После отправки сообщения, в администрации приложения моментально появляется данное сообщение, и сотрудники, которые отвечают за товары, будут уведомлены о малом запасе определенного товара или о том, что определенный товар или стройматериал закончился на складе.

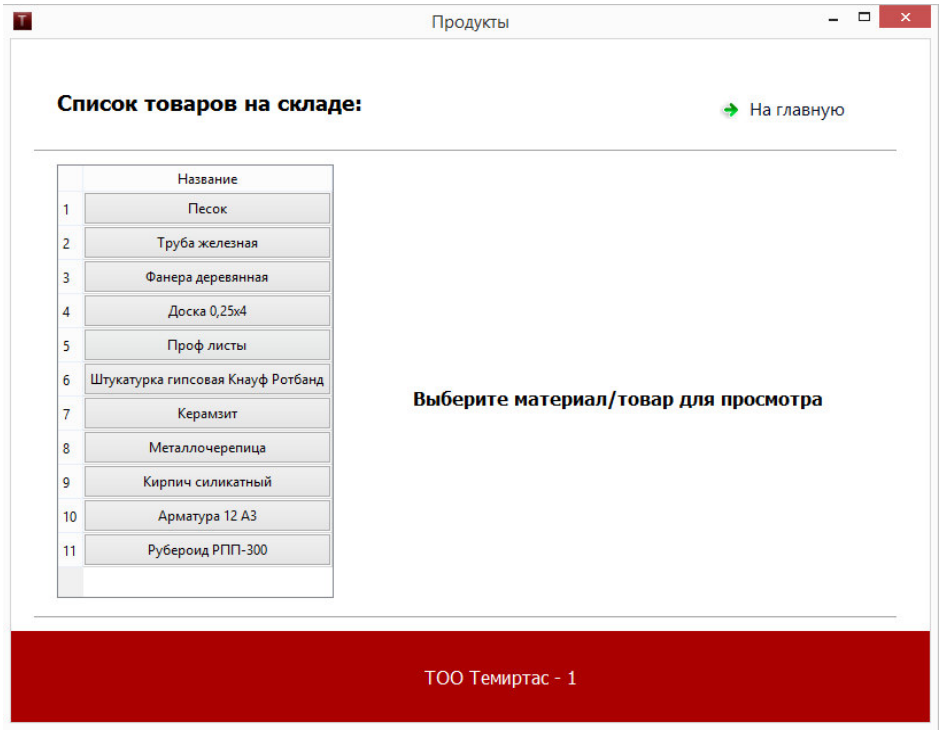

Рисунок 3.3.9 – Склад

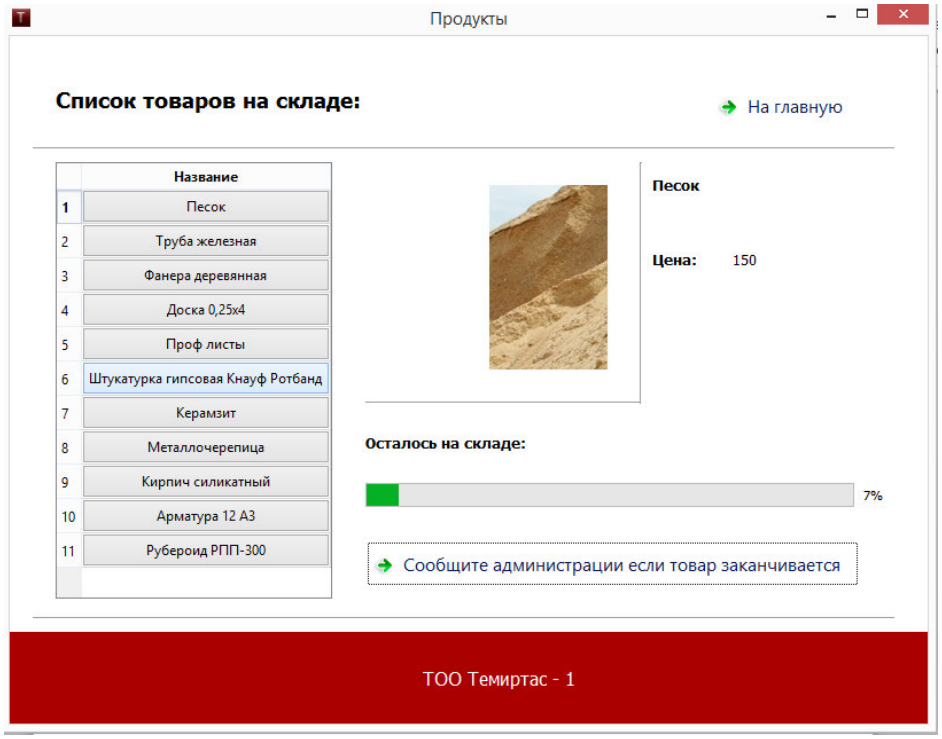

Рисунок 3.3.10 – Информация о товаре

После нажатии на «Сообщите администрации если товар заканчивается», в панели администрации (см. рисунок 3.3.11).

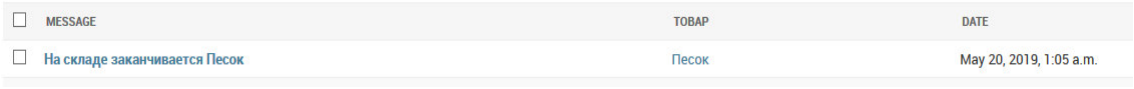

Рисунок 3.3.11 – Уведомление о малом запасе товара на складе

Также можно просмотреть список спецтехники, которая изменяется через панель админа (см. рисунок 3.3.12).

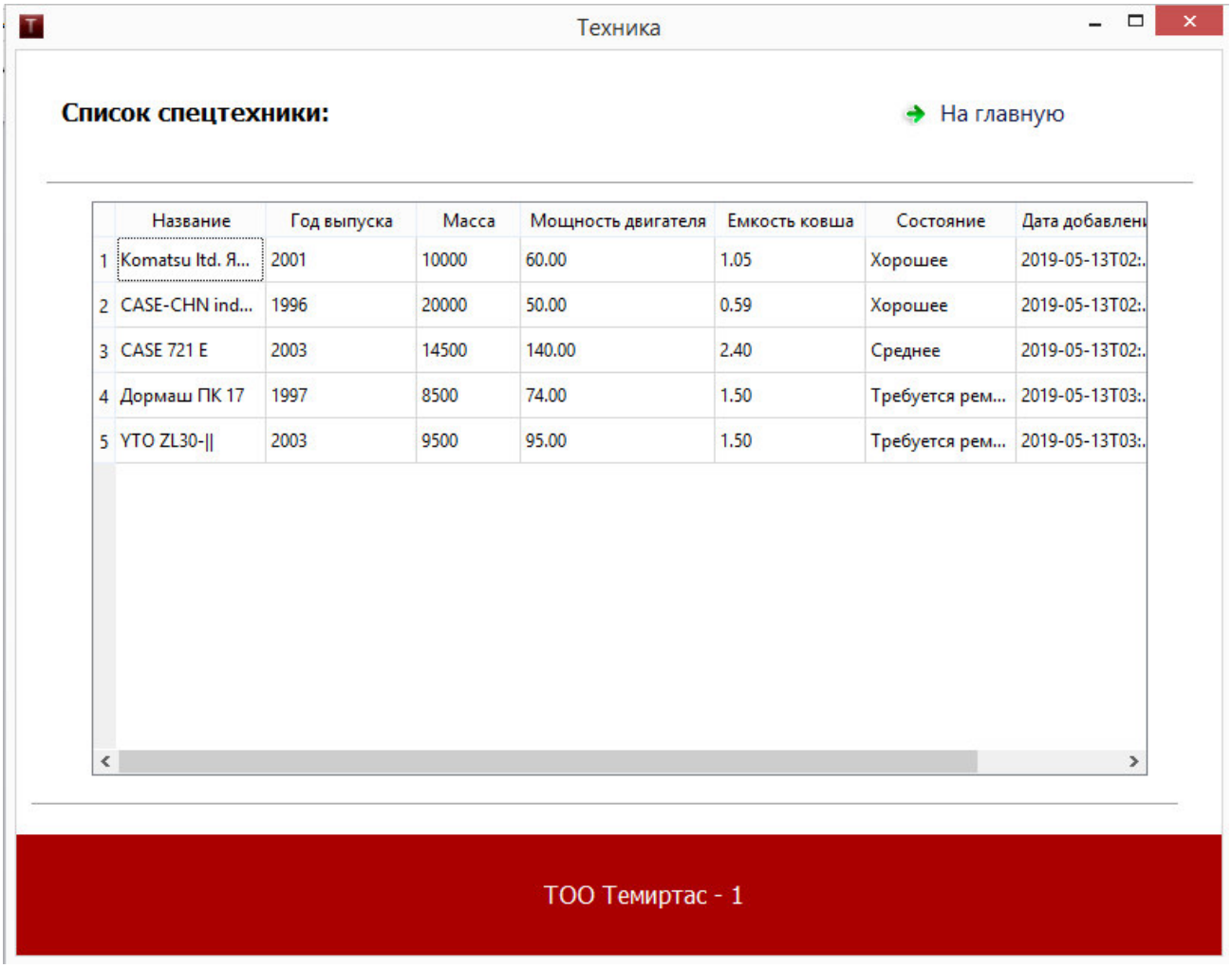

Рисунок 3.3.12 – Список спецтехники

4 Экономическая часть

Технико-экономическое обоснование содержит следующие пункты:
– определение сложности разработки программного обеспечения;
– расчет затрат на разработку ПО;
– определение ценности готового продукта;
– оценка результатов работы программного обеспечения.

4.1 Определение сложности разработки ПО

Чтобы определить трудоемкость разработки программного обеспечения, необходимо произвести разбиение всей задачи на более простые подзадачи, т.е. выполнить декомпозицию. С использованием методов декомпозиции определяется так называемая иерархическая структура работ (ИСР). Иерархическая структура разделяет проект на иерархически связанные, управляемые и контролируемые элементы (работы) и позволяет легче выстраивать систему управления по всем элементам проекта. ИСР является основой для составления в будущем сетевых графиков проекта, характеризующих сроки выполнения отдельных работ.

Модель распределения трудоемкости разработки ПО и стадии разработки представлены в таблице 4.1.

Таблица 4.1 – Этапы разработки ПО

Этапы разработки ПО	Вид работы	Трудоемкость, чел. час.
Этап 1	Декомпозиция проекта	15
Этап 2	Написание и утверждение ТЗ на разработку ПО	5
Этап 3	Поиск и изучение рынка CRM	20
Этап 4	Изучение документации технологического стека	30
Этап 5	Составление графиков ПО	20
Этап 6	Оформление теоретической части дипломной работы	10
Этап 7	Разработка практической части дипломного проекта	20
Этап 8	Реализация проекта	100
Этап 9	Тестирование и устранение багов	40
Этап 10	Организация отчета	5

Продолжение таблицы 4.1

Этапы разработки ПО	Вид работы	Трудоемкость, чел. час.
Этап 11	Тестирование ПО	5
Этап 12	Подведение итогов по разработанному ПО	5
Этап 13	Внедрение	30
Итого: трудоемкость выполнения дипломного проекта		305

Продолжительность рабочего дня равна 8 часам. В результате для реализации программного обеспечения необходимо 38 рабочих дней, т.е. трудоемкость выполнения дипломного проекта мы делим на продолжительность рабочего дня (305/8=38).

4.2 Расчет затрат на разработку ПО

Определение затрат необходимых для разработки программного обеспечения производится на основе имеющейся сметы, которая включает следующие элементы:

– материальные затраты;
– затраты на оплату труда;
– социальный налог;
– амортизация основных фондов;
– прочие затраты.

Материальные затраты делятся на основные и вспомогательные затраты на материалы, энергию и другие затраты необходимые для разработки ПО. Расчет материальных затрат происходит по форме, предоставленной в таблице 4.2.

Таблица 4.2 – Затраты на материальные ресурсы

Наименование материала	Марка	Ед. измерения	Количество	Цена за ед. в тенге	Сумма в тенге
Карандаш	Yamaha	Штук	2	90	180
Клавиатура	Toyota	Штук	1	3 000	3 000
Итого:		8 180			

В процессе разработки программного обеспечения будут использоваться три ноутбука Apple Mac, ASUS Alienware, Apple Mac Pro 2017 года. В каждом

ноутбуке установлены базовые операционные системы (Macentosh в ноутбуках от фирмы Apple и Windows 10 Pro в Asus Alienware) и нет необходимости дополнительных затрат на новую операционную систему.

Общую сумму, необходимую на материальные средства ($З_м$) можно рассчитать по следующей формуле (4.1):

$$З_м = \sum P_i * Ц_i, \quad (4.1)$$

где P_i - расход i-го вида материального ресурса, натуральные единицы;

$Ц_i$ - цена за единицу i-го вида материального ресурса, тг;

i - вид материального ресурса;

n - количество видов материальных ресурсов.

Расчет затрат на необходимое оборудование и программное обеспечение производится по форме, приведенной в таблице 4.3.

Таблица 4.3 – Расчет затрат на оборудование и ПО

Наименование материала	Марка	Ед. измерения	Количество	Цена за ед. в тенге	Сумма в тенге
Ноутбук	ASUS Alienware	Штук	1	350 000	350 000
Ноутбук	Apple Mac	Штук	1	450 000	450 000
Ноутбук	Apple Mac Pro 2017	Штук	1	600 000	600 000
Модем	Ericsson T073G	Штук	1	14 000	14 000
Домен	PS.kz	Штук	1	1 000	1 000
Итого:					1 415 000

Зм = 8180 + 1 415 000 = 1 423 180(тг)

Для реализации программного обеспечения необходимы материалы на сумму 1 423 180 тенге.

4.3 Расчет затрат на электроэнергию

В процессе разработки программного обеспечения электроэнергия играет ключевую роль, и без нее не обойтись.

Согласно таблице 4.1 для разработке программного обеспечения необходимо порядка 305 часов, теперь необходимо рассчитать стоимость электроэнергии, которая будет потрачена в течении 305 часов.

$$Э = З_{эл.эн.обор.} + З_{доп.нужды.}, \quad (4.2)$$

где $З_{эл.эн.обор.}$ – затраты на электроэнергию оборудования;

$З_{доп.нужды.}$ – затраты электроэнергии на дополнительные нужды.

Расчет электроэнергии, которая необходима для оборудования определяется по следующей формуле (4.3):

$$З_{эл.эн.обор.} = \sum W * К_{исц} * S * T, \quad (4.3)$$

где W – потребляемая мощность, Вт;

$К_{исц}$ – коэффициент использования ($К_{исц}$ = 0,7..0,9);

T – время работы;

S – тариф (1кВт/ч = 23,85 тг).

Итоги по расчетам стоимости затрачиваемой электроэнергии представлены в таблице 4.4.

Таблица 4.4 – Затраты на электроэнергию

Наименование приборов	Паспортная мощность, кВт	Коэффициент мощности	Время работы оборудования, ч	Цена ЭЭ тг/кВтч	Сумма, тг.
Ноутбук ASUS	0,7	0,7	305	23,85	3565
Модем	0,08	0,9	305	23,85	525
Apple Mac	0,6	0,9	305	23,85	3928
Apple Mac Pro	0.6	0.8	305	23.85	3500
Освещение	0,3	0,7	305	23,85	1530
Итого:					13 000

$$З_{эл.эн.обор.} = 13\,000 \text{ (тенге)}$$

На дополнительные потребности расходы подсчитываются на основе повышенного показателя в объеме 5% от расходов на электроэнергию(4.4):

$$З_{доп.нужды} = 5\% * З_{эл.эн.обор.}, \quad (4.4)$$

Определим затраты на дополнительные потребности согласно формуле (4.4):

$$З_{доп.нужды} = 0.05 * 13\,000 = 652(\text{тенге})$$

Исходя из всех расчетов, полные расходы на электроэнергию составляют:

$$Э = \mathbf{652 + 13000 = 13\,652}\ (\text{тенге})$$

4.4 Расчет затрат на оплату труда

Для реализации проекта, необходимо три работника:

- проект менеджер – декомпозиция, составление ТЗ по ней и контроль жизненного цикла программного обеспечения;
- разработчик серверной части – проектирование базы данных, настройка сервера, разработка серверной части программного обеспечения, тестирование и развертывание на сервере;
- разработчик клиентской части – создание графического интерфейса для конечных пользователей, интеграция с серверной частью по REST API

Сумму расходов на оплату труда можно рассчитать по следующей формуле:

$$З_{тр} = \sum ЧС_i * T_i \quad (4.5)$$

где $ЧС_i$ - часовая ставка i-го работника, тг;

T_i - трудоемкость разработки модели, чел.×ч; i - категория работника;

n - количество работников, занятых разработкой ПП.

Во время реализации проекта рабочее время участников не равномерно, поэтому имеет смысл установить часовую ставку каждого работника и общий объем заработной платы.

Часовую ставку сотрудника можно рассчитать по следующей формуле(4.6):

$$ЧС_i = \frac{ЗП_i}{ФРВ_i} \quad (4.6)$$

где $ЗП_i$ - месячная заработная плата i-го работника, тг;

$ФРВ_i$ - месячный фонд рабочего времени i-го работника, час.

Месячная заработная плата руководителя равняется 150 000 тенге, месячная заработная плата разработчика серверной части равняется 130 000 тенге, и

месячная заработная плата разработчика клиентской части составляет 115 000 тенге. Рассчитаем часовую ставку каждого работника согласно формуле (4.6):

$$ЧС_{менеджер} = \frac{198000}{22 * 8} = 1125 \text{ тг/ч}$$

$$ЧС_{разработчик(back-end)} = \frac{175000}{22 * 8} = 994 \text{тг/ч}$$

$$ЧС_{разработчик(front-end)} = \frac{155\,000}{22 * 8} = 881 \text{ тг/ч}$$

Часовая ставка проект-менеджера составляет 852,3 (тг/ч), трудоемкость разработки равняется 120 часам. Часовые ставки разработчиков серверной и клиентских частей составляют 739 (тг/ч) и 653,4 (тг/ч), соответственно. Трудоемкость разработки равняется 305 часам. Согласно формуле (4.5) можно рассчитать сумму расходов на заработную плату работников:

$$З_{тр} = 1125 * 120 + (994 + 881) * 305 = 135000 + 571875 = 706875$$

Расчеты затрат по оплате труда показаны в таблице (4.5).

Таблица 4.5. – Расчет заработной платы

Категория работника	Квалификация	Трудоемкость разработки ПП, час.	Часовая ставка, тг/ч	Сумма, тг.
Руководитель	Проектный руководитель	120	1125	135 000
Разработчик (back-end)	Back-end разработчик	305	994	303 170
Разработчик (front-end)	Web-разработчик	305	881	268 705
Итого:				706 875

4.5 Расчет затрат по социальному налогу

Согласно Налоговому кодексу Республики Казахстан социальный налог составляет 9,5% от фонда оплаты труда. Социальный налог можно рассчитать по следующей формуле(4.7):

$$С_{н} = (ФОТ - ПО) * 0{,}095 \quad (4.7)$$

где ПО - отчисления в пенсионный фонд, они составляют 10% от ФОТ.

$$ПО = \ 706\,875 * 0{,}1 = 70\,687 \ \text{тенге}$$
$$С_н = (706\,875 - 70\,687) * 0{,}095 = 60\,437 \ \text{тенге}$$

Результаты расчетов представлены в таблице (4,6):

Таблица 4.6 – Начисление социального налога

Категория работника	Количество человек	Заработная плата, тг	Пенсионные отчисления, тг	Социальный налог, тг
Руководитель	1	135 000	13500	11 542
Разработчик (backend)	1	303 170	30 317	25 921
Разработчик (frontend)	1	268 705	26 870	22 974
Итого:				60 437

4.6 Амортизация основных фондов и прочие затраты

Нормы амортизации ОФ необходимо определить в соответствии с налоговым кодексом РК. Амортизацию ОФ можно определить по следующей формуле(4.8):

$$A_г = \frac{С_{об} * Н_а}{100} \quad (4.8)$$

где, $С_{об}$ – стоимость оборудования;

$Н_а$ – норма амортизации (норма амортизация = 25);

Формула (4.8) позволяет рассчитать нужную сумму для амортизационных отчислений за год для ноутбука:

$$A_г = \frac{600\,000 * 25}{100} = \ 150\,000 \ \text{тенге}$$

Теперь необходимо рассчитать норму амортизации за период разработки:

$$A_г = \frac{150\,000 * 38}{365} = 15\,616{,}44 \ \text{тенге}$$

Подобным образом необходимо рассчитать норму амортизации для всего оборудования. Результаты расчетов приведены в таблице (4.7).

Таблица 4.7 – Амортизация ОФ

Наименование оборудования и ПО	Стоимость оборудования и ПО, тг	Годовая норма амортизации, %	Сумма амортизации за год, тг	Сумма амортизации за время разработки, тг
ASUS	350 000	25	87 500	9 109
Apple Mac	450 000	25	112 500	42 750
Модем	14 000	20	2 800	1 064
Apple Mac Pro	600 000	25	150 000	57 000
Хостинг	4 700	20	940	357,2
Домен	3 338	15	500,7	190,266
Итого:			354240,7	110 470

На основе всех представленных расчетов необходимо оформить смету расходов на разработку ПО согласно форме, которая приведена в таблице (4.8). Также предусматриваем прочие расходы на оплату домена и интернета в сумме 27600.

Таблица 4.8 – Смета затрат на разработку ПО

Статьи затрат	Сумма, тг	Сумма в %
Затраты на оборудование	1 415 000	61%
Затраты на оплату труда	706 875	30%
Социальные налоги	60 437	3%
Затраты на электроэнергию	13 000	1%

Продолжение таблицы

Статьи затрат	Сумма, тг	Сумма в %
Амортизация основных фондов	110 470	4%
Прочие расходы	27 600	1%
Итого по смете:	2 333 382	100%

На диаграмме 5.2.1 представлена смета затрат на разработку.

Диаграмма 1 – Смета затрат на разработку проекта.

4.7 Определение возможной (договорной) цены ПО

Стоимость программного обеспечения определяется на основе качества разработанного продукта, сроков его разработки и производительности продукта. Стоимость $Ц_Д$ для программного обеспечения можно рассчитать по следующей формуле (4.9):

$$Ц_Д = З_{нир}\left(1 + \frac{P}{100}\right), \quad (4.9)$$

где $З_{нир}$ – затраты на разработку программного обеспечения, тг;

Р – средний уровень рентабельности ПО, (%). Данный параметр принят равным 25%.

$$Ц_Д = 2\,333\,382\left(1 + \frac{25}{100}\right) = 2\,333\,382 + (2\,333\,382 * 0{,}25) = 2\,916\,727 \text{ тенге}$$

Далее необходимо определить стоимость реализации с учетом НДС, ставка НДС устанавливается законодательством РК. На 2019 года ставка НДС составляет 12%. Стоимость реализации учитывая НДС можно рассчитать по следующей формуле:

$$Ц_р = Ц_д + Ц_д * НДС, \quad (4.10)$$

$$Ц_р = 2\,916\,727 + 2\,916\,727 * 0{,}12 = 2\,916\,727 + 350\,007 = 3\,266\,734 \text{ тенге}$$

Данную цену можно округлить до 3 267 000 тенге. Стоимость реализации с учетом НДС равна 2 916 727. Прибыль П равна $2\,333\,382 * 0{,}25 = 583\,345$

5 Охрана труда и безопасность жизнедеятельности

В данном разделе рассматриваются мероприятия по обеспечению безопасности жизнедеятельности и охране труда в рамках использования разрабатываемого программного продукта.

Разрабатываемое программное обеспечение предназначено для сотрудников компании ТОО «Темиртас-1» и нацелено на то, чтобы облегчить рабочий процесс путем автоматизации процесса контроля заявками.

Конечный программный продукт реализуется в виде десктопной программы, которая будет использоваться за персональным компьютером, Поэтому необходимо рассмотреть условий труда в офисе.

В помещениях отсутствует шум, который может потребовать дополнительной звукоизоляции и использования средств индивидуальной защиты от шума.

Микроклимат помещения также является комфортным для ежедневной работы, в том числе благодаря небольшой приточной системе и наличию кондиционеров, которые позволяют поддерживать температуру в помещении в пределах 23-25^{o}C в теплое время года.

Эргономика рабочего места (удобные рабочие кресла, просторные рабочие поверхности, аксессуары, обеспечивающие доступность рабочих материалов на расстоянии вытянутой руки) создает приятную рабочую обстановку для продуктивной работы.

Единственным фактором, который препятствует идеальным условиям труда является освещение помещений. Поэтому предстоит проверить степень освещенности каждого из помещений естественным и искусственным светом, и предложить мероприятия по улучшению условий труда.

В малом помещении есть два окна общей площадью 2,8 м^2, а в большом помещении три окна общей площадью 4,3 м^2. Они обеспечивают естественное освещение помещений. Искусственное освещение обеспечивается 6 лампами в малом помещении и 9 лампами в большом помещении. Используются лампы накаливания мощностью 40 Вт и световым потоком 420 Лм. Также важным фактором, оказывающим на коэффициент полезного действия программиста и усталости, становится корректное освещение. Усталость зрения зависит от следующих случаев:

– недостаточная освещенность помещения;

– избыточная освещенность;

– неправильное распределение света в помещении.

Недостаточная освещенность может быть причиной перебойного усилия и утраты концентрации, что соответственно вызывает чувство физической усталости и нагруженности.

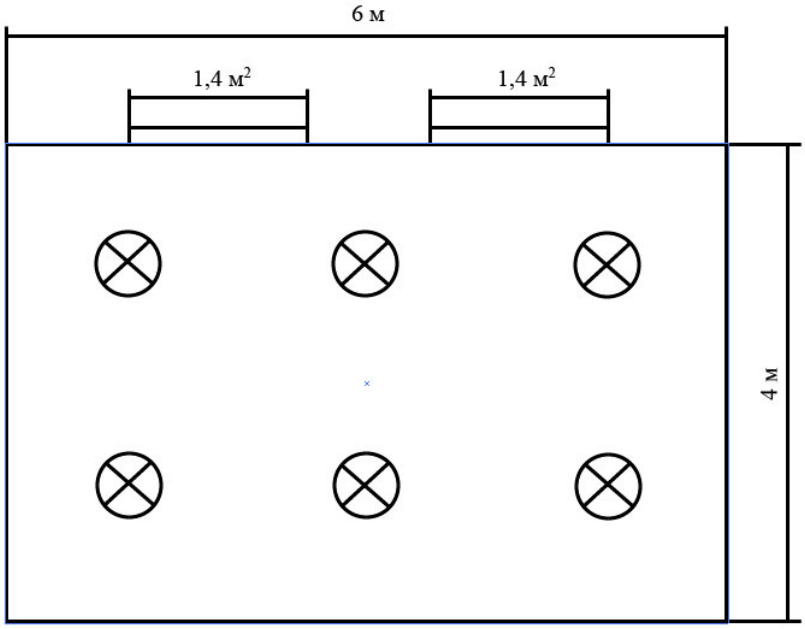

Рисунок 5.1 – Схема малого помещения

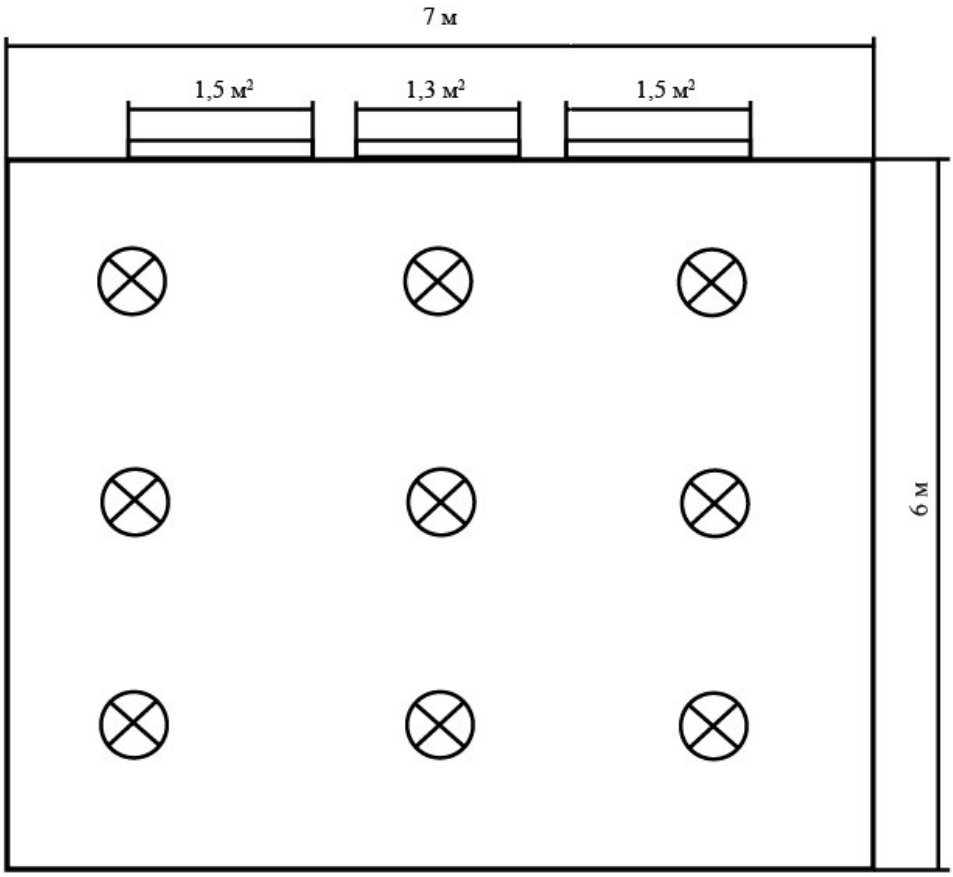

Рисунок 5.2 – Схема большого помещения

5.1 Расчет естественного освещения

Расчет естественного освещения включает в себя определение световых проемов в помещении. Световые проемы могут располагаться в стенах (боковое освещение) и в потолке (верхнее освещение). Так как предполагается офисная работа, и пользоваться разработанным программным продуктом будут в офисном помещении, то целесообразнее в качестве естественного освещения выбрать именно боковое освещение.

Необходимо рассчитать рекомендуемые размеры световых проемов для каждого помещения и сравним с фактическими размерами окон.

5.1.1 Расчет для малого помещения

Для расчета бокового освещения будет применена следующая формула(5.1):

$$100\frac{S_o}{S_n} = \frac{e_n \times K_з \times \eta_o}{\tau_o \times r_1} \times K_{зд} \qquad (5.1)$$

где: S_0 - суммарная площадь световых проемов при боковом освещении, м^2;

S_n - площадь помещения, м^2;

e_N – нормируемое значение КЕО;

$K_з$ – коэффициент запаса;

n_0 – световая характеристика окон;

t_0 - общий коэффициент светопропускания;

r_1 - коэффициент, учитывающий повышение КЕО при боковом освещении, благодаря свету, отраженному от поверхности помещения и подстилающего слоя, примыкающего к зданию;

$К_{зд}$ - коэффициент, учитывающий затемнение окон противостоящими зданиями.

Общий коэффициент светопропускания рассчитывается следующим образом:

$$t_0 = t_1 t_2 t_3 t_4 t_5,$$

где:

t_1 – коэффициент светопропускания;

t_2 – коэффициент потери света в переплетах проема;

t_3 – коэффициент потери света в несущих конструкциях, равен 1 при боковом освещении;

t_4 – коэффициент потери света в солнцезащитных устройствах;

t_5 – коэффициент потери света в защитной сетке, под фонарями, равен 0,9.

Так как световым проемом будет окно с тройным стеклом, то

$$t_1 = 0{,}75.$$

Стальные двойные и открывающиеся оконные переплеты, по таблице означают, что:

$$t_2 = 0{,}6.$$

Коэффициент потери света в регулируемых жалюзи равен:

$$t_4 = 1.$$

На основании собранных данных общий коэффициент светопропускания равен:

$$t_0 = 0{,}75 * 0{,}6 * 1 * 1 * 0{,}9 = 0{,}405.$$

Коэффициент запаса для офисов с вертикальным светопропускаемым материалом, равен:

$$K_з = 1{,}2.$$

Отношение длины помещения к глубине равно:

$$o_1 = \frac{6}{4} = 1{,}5.$$

Отношение глубины помещения, к разности высоты верхнего края окна и высоты рабочей поверхности:

$$o_2 = \frac{4}{2{,}5 - 1{,}2} = 3{,}1 \sim 3.$$

С помощью найденных значений находим значение световой характеристики окон при боковом освещении:

$$\eta_0 = 15.$$

Коэффициент r_1, с учетом определенных значений, равен:

$$r_1 = 4{,}3.$$

Ввиду отсутствия зданий напротив световых проемов, коэффициент затемнения близко стоящими зданиями принимается равным:

$$К_{зд} = 1.$$

Нормируемое значение КЕО рассчитывается по формуле:

$$e_N = e_н * m_N, \text{где}$$

N – номер группы обеспеченности естественным светом;
$e_н$ – значение КЕО;
m_N – коэффициент светового климата.

Алматы относится к четвертой группе обеспеченности естественным светом,

$$N = 4.$$

Световые проемы расположены на север, следовательно:

$$m_N = 0{,}75.$$

Значение КЕО, при высокой точности зрительной работы, которая соответствует работе разработчика:

$$e_н = 1{,}2.$$

Теперь выполним расчет нормируемого значения КЕО:

$$e_N = 1{,}2 * 0{,}75 = 0{,}9.$$

Подставив полученные значения в пропорцию, предназначенную для поиска суммарной площади световых проёмов, получим:

$$100\frac{S_0}{24} = \frac{0{,}9 * 1{,}2 * 15}{0{,}405 * 4{,}3} * 1 = 9{,}3.$$

Из полученной пропорции найдем суммарную площадь световых проемов:

$$S_0 = \frac{9{,}3 * 24}{100} = 2{,}2\ (\text{м}^2).$$

Таким образом, получается, что в малом помещении площадь окон превышает рекомендуемую суммарную площадь световых проемов ($2{,}8\text{м}^2 > 2{,}2\text{м}^2$) и обеспечивает достаточную освещенность в помещении.

5.1.2 Расчет для большого помещения

Для расчета бокового освещения будет применена следующая формула(5.2):

$$100\frac{S_o}{S_n} = \frac{e_n \times K_з \times \eta_o}{\tau_o \times r_1} \times K_{зд} \qquad (5.3)$$

где: S_0 - суммарная площадь световых проемов при боковом освещении, м^2;

S_n - площадь помещения, м^2;

e_N – нормируемое значение КЕО;

$K_з$ – коэффициент запаса;

n_0 – световая характеристика окон;

t_0 - общий коэффициент светопропускания;

r_1 - коэффициент, учитывающий повышение КЕО при боковом освещении, благодаря свету, отраженному от поверхности помещения и подстилающего слоя, примыкающего к зданию;

$К_{зд}$ - коэффициент, учитывающий затемнение окон противостоящими зданиями.

Общий коэффициент светопропускания рассчитывается следующим образом:

$$t_0 = t_1 t_2 t_3 t_4 t_5$$

где, t_1 – коэффициент светопропускания;

t_2 – коэффициент потери света в переплетах проема;

t_3 – коэффициент потери света в несущих конструкциях, равен 1 при боковом освещении;

t_4 – коэффициент потери света в солнцезащитных устройствах;

t_5 – коэффициент потери света в защитной сетке, под фонарями, равен 0,9.

Так как световым проемом будет окно с тройным стеклом, то

$$t_1 = 0{,}75.$$

Стальные двойные и открывающиеся оконные переплеты, по таблице означают, что:

$$t_2 = 0{,}6.$$

Коэффициент потери света в регулируемых жалюзи равен:

$$t_4 = 1.$$

На основании собранных данных общий коэффициент светопропускания равен:

$$t_0 = 0{,}75 * 0{,}6 * 1 * 1 * 0{,}9 = 0{,}405.$$

Коэффициент запаса для офисов с вертикальным светопропускаемым материалом, равен:

$$K_з = 1{,}2.$$

Отношение длины помещения к глубине равно:

$$o_1 = \frac{7}{6} = 1{,}2 \sim 1.$$

Отношение глубины помещения, к разности высоты верхнего края окна и высоты рабочей поверхности:

$$o_2 = \frac{6}{2{,}5 - 1{,}2} = 4{,}61 \sim 5.$$

С помощью найденных значений находим значение световой характеристики окон при боковом освещении:

$$\eta_0 = 23.$$

Коэффициент r_1, с учетом определенных значений, равен:

$$r_1 = 10.$$

Ввиду отсутствия зданий напротив световых проемов, коэффициент затемнения близко стоящими зданиями принимается равным:

$$К_{зд} = 1.$$

Нормируемое значение КЕО рассчитывается по формуле:

$$e_N = e_н * m_N$$

где, N – номер группы обеспеченности естественным светом;

$e_н$ – значение КЕО;

m_N – коэффициент светового климата.

Алматы относится к четвертой группе обеспеченности естественным светом,

$$N = 4.$$

Световые проемы расположены на север, следовательно:

$$m_N = 0{,}75.$$

Значение КЕО, при высокой точности зрительной работы, которая соответствует работе разработчика:

$$e_н = 1{,}2.$$

Теперь выполним расчет нормируемого значения КЕО:

$$e_N = 1{,}2 * 0{,}75 = 0{,}9.$$

Подставив полученные значения в пропорцию, предназначенную для поиска суммарной площади световых проёмов, получим:

$$100\frac{S_0}{42} = \frac{0{,}9 * 1{,}2 * 23}{0{,}405 * 10} * 1 = 6{,}13.$$

Из полученной пропорции найдем суммарную площадь световых проемов:

$$S_0 = \frac{6{,}13 * 42}{100} = 2{,}6\ (\text{м}^2).$$

Таким образом, получается, что в большом помещении площадь окон превышает рекомендуемую суммарную площадь световых проемов (4,3 м^2 > 2,6 м^2) и обеспечивает достаточную освещенность в помещении.

5.2 Расчет искусственного освещения

Расчет искусственного освещения включает в себя такие важные пункты, как подбор системы освещения, типа ламп, которые будут использоваться для помещения, их количество, а также месторасположение в помещении.

В большинстве случаев, для расчета освещения в помещениях используются лампы двух видов: люминесцентные лампы и лампы накаливания. В последние годы проектировщики освещенности все чаще и чаще делают выбор в пользу люминесцентных ламп, так как они имеют ряд ключевых преимуществ перед лампами накаливания:

– свет, излучаемый этими лампами, гораздо ближе по своему спектральному составу к обыкновенному дневному свету;

– КПД этих ламп в 1,5 – 2 раза превышает эту характеристику у ламп накаливания;

– при этом люминесцентные лампы имеют светоотдачу на порядок выше, чем лампы накаливания (в 3 – 4 раза);

– их срок службы более длительный, что позволяет предприятию экономить на замене ламп, которые вышли из строя, так как делать это приходиться реже.

Для расчета будет использоваться метод коэффициента использования светового потока. Он используется для помещений с общим равномерно распределенным по горизонтальной поверхности светом, что подходит для данного случая.

Необходимо рассчитать освещенность в помещении и сравнить ее с минимальной нормированной освещенностью. В том случае, если освещенность окажется недостаточной для полноценного освещения, то необходимо рассчитать план по замене ламп в помещении на другие, с более мощным световым потоком.

Нормированная минимальная освещенность для офисного помещения равна 300 Лк.

5.2.1 Расчет для малого помещения

Расчет производится для помещения площадью 24 м^2 (ширина А = 4 м и длина В = 6 м).

Освещенность рассчитывается по формуле (5.3):

$$E = \frac{F \times n}{K \times S \times Z} \quad (5.3)$$

где, E - рассчитываемая освещенность, Лк;

F - световой поток, обеспечиваемый нынешними лампами, Лм;

S - площадь освещаемого помещения (в нашем случае S = 24м^2);

Z - отношение средней освещенности к минимальной (обычно принимается равным 1,1…1,2, пусть Z = 1,1);

K - коэффициент запаса, учитывающий уменьшение светового потока лампы в результате загрязнения светильников в процессе эксплуатации (его значение зависит от типа помещения и характера проводимых в нем работ и в нашем случае K = 1,5);

n - коэффициент использования (выражается отношением светового потока, падающего на расчетную поверхность, к суммарному потоку всех ламп и исчисляется в долях единицы; зависит от характеристик светильника, размеров помещения, окраски стен и потолка, характеризуемых коэффициентами отражения от стен ($P_{стены}$), пола($P_{пол}$) и потолка ($P_{потолок}$)), значение коэффициентов: $P_{стены}$ = 50%, $P_{пол}$ = 30%, $P_{потолок}$ = 70%. Значение n определяется по таблице коэффициентов использования различных светильников. Для этого вычислим индекс помещения по формуле (5.4):

$$I = \frac{S}{h \times (A+B)} \quad (5.4)$$

где, S - площадь помещения, S = 24 м^2;

h - расчетная высота подвеса, h = 2.92 м;

A - ширина помещения, A = 4 м;

B - длина помещения, B = 6 м.

Подставив значения получим результат:

$$I = \frac{24}{2{,}92 \times (4+6)} = \frac{24}{29{,}2} = 0{,}82$$

Вычислив индекс помещения I = 0,82, определим по таблице на рисунке 5.3 значение коэффициента использования n = 0,46.

$\rho_п$, %	70	70	50	30	0
$\rho_с$, %	50	50	30	10	0
$\rho_р$, %	30	10	10	10	0
i					
0,5	24	22	20	17	16
0,6	34	32	26	23	21
0,7	42	39	34	30	29
0,8	46	44	38	34	33
0,9	49	47	41	37	36
1	51	49	43	39	37
1,1	53	50	45	41	39
1,25	56	52	47	43	41
1,5	60	55	50	46	44
1,75	63	58	53	48	46
2	66	60	55	51	49
2,25	68	62	57	53	51
2,5	70	64	59	55	53
3	73	66	62	58	56
3,5	76	68	64	61	59
4	78	70	66	62	60
5	81	73	69	64	62

Рисунок 5.3 – Коэффициент использования светового потока

Рассчитаем световой поток, обеспечиваемый нынешними лампами:
$F = F_Л \times N$, где:
$F_Л$ – световой поток одной лампы, Лм;
N – количество ламп в помещении.

$$F = 420 \times 6 = 2520 \text{ Лм}$$

Подставим все известные значения в формулу 1.1 и найдем величину освещенности:

$$E = \frac{2520 \times 0{,}46}{1{,}5 \times 24 \times 1{,}1} = 29{,}27 \text{ Лк}$$

Полученное значение освещенности приблизительно в 10 раз меньше, чем минимальное нормированное значение освещенности.

В качестве мероприятий по улучшению условий освещенности было выбрано изменить лампы таким образом, чтобы световой поток новых ламп был значительно выше, чем у предыдущих.

Для освещения выбираем компактную люминесцентную лампу фирмы Phillips типа TL-D 58W/33, световой поток которой F = 4600 Лк.

Рассчитаем световой поток, который будут обеспечивать новые лампы:

$$F = 4600 \times 6 = 27\,600 \text{ Лм}$$

Рассчитаем новую освещенность:

$$E = \frac{27\,600 \times 0{,}46}{1{,}5 \times 24 \times 1{,}1} = 320{,}6 \text{ Лк}$$

Значение освещенности с новыми лампами немногим превосходит значение минимальной нормированной освещенности.

Таким образом получается, что для увеличения освещенности до необходимой нормы, достаточно изменить тип ламп на более мощные.

5.2.2 Расчет для большого помещения

Расчет производится для помещения площадью 42 м^2 (ширина А = 6 м и длина В = 7 м).

Освещенность рассчитывается по формуле (5.5):

$$E = \frac{F \times n}{K \times S \times Z} \qquad (5.5)$$

где, Е - рассчитываемая освещенность, Лк.

F - световой поток, обеспечиваемый нынешними лампами, Лм;

S - площадь освещаемого помещения (в нашем случае S = 42м^2);

Z - отношение средней освещенности к минимальной (обычно принимается равным 1,1…1,2, пусть Z = 1,1);

К - коэффициент запаса, учитывающий уменьшение светового потока лампы в результате загрязнения светильников в процессе эксплуатации (его значение зависит от типа помещения и характера проводимых в нем работ и в нашем случае К = 1,5);

n - коэффициент использования (выражается отношением светового потока, падающего на расчетную поверхность, к суммарному потоку всех ламп и исчисляется в долях единицы; зависит от характеристик светильника, размеров помещения, окраски стен и потолка, характеризуемых коэффициентами отражения от стен ($Р_{стены}$), пола($Р_{пол}$) и потолка ($Р_{потолок}$)), значение коэффициентов: $Р_{стены}$ = 50%, $Р_{пол}$ = 30%, $Р_{потолок}$ = 70%. Значение n определяется по таблице коэффициентов использования различных светильников.

Для этого вычислим индекс помещения по формуле:

$$I = \frac{S}{h \times (A+B)} \qquad (1.2)$$

где, S - площадь помещения, S = 42 м^2;
h - расчетная высота подвеса, h = 2.92 м;
A - ширина помещения, A = 4 м;
B - длина помещения, B = 6 м.
Подставив значения получим результат:

$$I = \frac{42}{2{,}92 \times (6+7)} = \frac{42}{37{,}96} = 1{,}1$$

Вычислив индекс помещения I = 0,82, определим по таблице на рис.1 значение коэффициента использования n = 0,53.

Рассчитаем световой поток, обеспечиваемый нынешними лампами:

$$F = F_Л \times N,$$

где, $F_Л$ – световой поток одной лампы, Лм;
N – количество ламп в помещении.

$$F = 420 \times 9 = 3780 \text{ Лм}$$

Подставим все известные значения в формулу 1.1 и найдем величину освещенности:

$$E = \frac{3780 \times 0{,}53}{1{,}5 \times 42 \times 1{,}1} = 28{,}9 \text{ Лк}$$

Полученное значение освещенности приблизительно в 10 раз меньше, чем минимальное нормированное значение освещенности.

В качестве мероприятий по улучшению условий освещенности было выбрано изменить лампы таким образом, чтобы световой поток новых ламп был значительно выше, чем у предыдущих.

Для освещения выбираем компактную люминесцентную лампу фирмы Phillips типа TL-D 58W/33, световой поток которой F = 4600 Лк.

Рассчитаем световой поток, который будут обеспечивать новые лампы:

$$F = 4600 \times 9 = 41\,400 \text{ Лм}$$

Рассчитаем новую освещенность:

$$E = \frac{41\ 400 \times 0{,}53}{1{,}5 \times 42 \times 1{,}1} = 316{,}6\ \text{Лк}$$

Значение освещенности с новыми лампами немногим превосходит значение минимальной нормированной освещенности.

Таким образом получается, что для увеличения освещенности до необходимой нормы, достаточно изменить тип ламп на более мощные.

Проведя ряд расчетов для выяснения причин проблем с освещенностью в помещениях, были сделаны следующие выводы:

– в обоих помещениях соблюдаются нормы естественного освещения. Это означает, что в ясный день в этих помещениях не будет проблем с освещенностью;

– искусственное освещение в обоих помещениях слишком тусклое, и если его не изменить, то скоро это приведет к проблемам со здоровьем у служащих в помещениях.

Для устранения проблемы искусственного освещения в помещениях было принято следующее решение: изменить обычные лампы накаливания в обоих помещениях на новые, более мощные, дающие больший световой поток. Такое несложное мероприятие позволит улучшить условия работы в сумеречное время и пасмурные дни.

Заключение

В ходе выполнения дипломного проекта был разработан программный продукт, который облегчает процесс работы строительной фирмы ТОО «Темиртас – 1» путем фиксации и контроля клиентских заявок в централизованной базе.

В ходе выполнения дипломного проекта были выполнены следующие задачи:

- изучение предметной области;
- подбор подходящего технологического стека;
- изучение подхода REST API;
- настройка и запуск серверной части на Django framework;
- подключение Django-rest-framework для поднятия api;
- настройка админ панели, для корректного отображения данных, и возможности добавление товаров;
- создание графического интерфейса с помощью библиотеки PyQt5;
- интеграция серверной части с клиентской частью;
- произведено экономическое обоснование целесообразности разрабатываемого продукта, в ходе которого сделано следующее заключение: цена реализации окупает все затраты на разработку программного продукта;
- также были исследованы условия труда в предлагаемом помещении и предложены мероприятия по улучшению освещения в помещении в пасмурную погоду или темное время суток.

Таким образом, была проведена разработка приложения, которое полноценно выполняет все возложенные на него функции и предоставляет своим пользователям использование всего разработанного функционала.

Список литературы

1 Коэльё Л. П., Ричерт В. Построение систем машинного обучения на языке Python. – Перевод с английского. – М.: ДМК Пресс, 2015. – с. – ISBN 978-5-97060-330-7.

2 Марк Лутц. Программирование на Python / Пер. с англ. – 4-е изд. – СПб.: Символ-Плюс, 2011. – Т. II. – ISBN 978-5-93286-211-7.

3 Дэвид М. Бизли. Python. Подробный справочник, 4-е издание. – Перевод с английского. – СПб.: Символ-Плюс, 2010. – 864 с – ISBN 978-5-93286-157-8

4 Бизли, Дэвид М. Язык программирования Python. Справочник. – К.: ДиаСофт, 2000. – 336 с. – ISBN 966-7393-54-2, ISBN 0-7357-0901-7

5 Фёдоров Д. Ю. Основы программирования на примере языка Python / Учебное пособие. – СПб.: Юрайт, 2018. – 167 с. – ISBN 978-5-534-04479-9.

6 Сузи Р. А. Язык программирования Python: Учебное пособие. – М.: ИНТУИТ, БИНОМ. Лаборатория знаний, 2006. – 328 с. – ISBN 5-9556-0058-2, ISBN 5-94774-442-2

7 Доусон М. Программируем на Python. – СПб.: Питер, 2012. – 432 с. – ISBN 978-5-459-00314-7.

8 Марк Лутц. Программирование на Python / Пер. с англ. – 3-е изд. – СПб.: Символ-Плюс, 2011. – Т. I. – 992 с. – ISBN 978-5-93286-210-0.

9 Маккинли У. Python и анализ данных. – Перевод с английского. – М.: ДМК Пресс, 2015. – 482 с. – ISBN 978-5-97060-315-4.

10 Головатый А., Каплан-Мосс Дж. Django. Подробное руководство = Django. The definitive guide to / пер. с англ.. – СПб.: Символ-Плюс, 2010. – 560 с. – (High Tech). – ISBN 978-5-93286-187-5.

11 У. Чан, П. Биссекс, Д. Форсье. Django. Разработка веб-приложений на Python = Python Web Development with Django / пер. с англ. А. Киселёв. – СПб.: Символ-Плюс, 2009. – 456 с. – (High Tech). – ISBN 978-5-93286-167-7.

12 Макс Шлее. Qt 5.10 Профессиональное программирование на C++. – СПб.: БХВ-Петербург, 2018. – С. 1072. – ISBN 978-5-9775-3678-3.

13 Прохоренок Н. А. Python 3 и PyQt. Разработка приложений. – СПб.: БХВ-Петербург, 2012. – С. 704. – ISBN 978-5-9775-0797-4.

14 Mark Summerfield. Rapid GUI Programming with Python and Qt. – 1st. – Prentice Hall, 2008. – P. 648. – ISBN 978-0132354189.

15 Boudewijn Rempt. GUI Programming with Python: QT Edition. – OpenDocs, 2002. 9 апреля 2010 года

16 Ж. Бланшет, М. Саммерфилд. Qt 4: Программирование GUI на C++. 2-е дополненное издание. – М.: КУДИЦ-ПРЕСС, 2008. – С. 736. – ISBN 978-5-91136-059-7.

17 И. А. Хахаев. Практикум по алгоритмизации и программированию на Python. Учебник. – М.: Альт Линукс, 2010. – 126 с. – (Библиотека ALT Linux). – ISBN 978-5-905167-02-7.

18 Сузи Р. А. Язык программирования Python: Учебное пособие. – М.: ИНТУИТ, БИНОМ. Лаборатория знаний, 2006. – 328 с. – ISBN 5-9556-0058-2, ISBN 5-94774-442-2

19 Ноа Гифт, Джереми М. Джонс. Python в системном администрировании UNIX и Linux. – Перевод с английского. – СПб.: Символ-Плюс, 2009. – 512 с – ISBN 978-5-93286-149-3

20 Марк Саммерфилд. Программирование на Python 3. Подробное руководство. – Перевод с английского. – СПб.: Символ-Плюс, 2009. – 608 с – ISBN 978-5-93286-161-5

Приложение А
(обязательное)

Техническое задание

1 Настроить виртуальную среду, для того, чтобы обернуть серверную часть в независимую среду разработки, использую virtualenv(python 3.6).

2 Подключение серверной части для использывания Django-rest-framework.

3 Проектирования БД. Создание моделей и их миграции на стороне Django.

4 Написание url-адресов для запросов с клиентской части. Подготовить роутеры для всех возможных путей.

5 Подключение контроллеров для вызова из роутов.

6 Страница администрации. Добавление возможности CRUD (создание, считывание, обновление, удаление объектов) для товаров и заявок, также для спецхтеники. Реализовать обратную связь с клиенсткой программы с графическим интерфейсом.

7 Реализовать исполняемую программу с простым, удобным графическим интерфейсом и полным набором необходимого функционала.

8 Интеграция с серверной частью.

9 Обработка возможных ошибок.

Приложение Б
(обязательное)

Листинг программы

```
from django.db import models
from django.db.models.signals import pre_save
from django.contrib.auth.models import User

import string
import random
class StaffUser(User):
pass
class Product(models.Model):
name = models.CharField(max_length=255, null=True, verbose_name='Название')
volume = models.IntegerField(default=0, verbose_name = 'Общее количество')
price = models.IntegerField(default=0, verbose_name='Цена за единицу/кв. метр')
image = models.ImageField(upload_to='product/', null=True, blank=True, verbose_name='Изображение')

def __str__(self):
    return self.name
class Meta:
    verbose_name_plural= "Добавить продукты"
    verbose_name = 'Продукт'

class Order(models.Model):
order_id = models.CharField(max_length=255, null=True, blank=True, editable=False)
comment = models.CharField(max_length=255, null=True, verbose_name='Коментарии')
customer = models.CharField(max_length=255, null=True, verbose_name='Заказщик')
customer_contacts = models.CharField(max_length=255, null=True, verbose_name='Контакты')

STATUS_CHOICE = (
    (1, 'В очереди'),
    (2, 'В процессе'),
    (3, 'Завершена')
)
status = models.IntegerField(choices=STATUS_CHOICE, default=1, verbose_name='Статус')

date = models.DateTimeField(auto_now_add=True, verbose_name='Дата')
product = models.ForeignKey(Product, null=True, on_delete=models.SET_NULL, verbose_name='Продукт')
volume = models.IntegerField(default=0, verbose_name='Объем')
amount = models.IntegerField(default=0)
```

```
        product_name = models.CharField(max_length=255)

        class Meta:
                verbose_name_plural = 'Список заявок'
                verbose_name = 'Заявка'

        import string
        import random
        from django.db.models.signals import pre_save

        def id_generator(instance, sender, *args, **kwargs):
        instance.amount = instance.product.price * instance.volume
        instance.product_name = instance.product.name
        chars=string.ascii_uppercase + string.digits
        the_id =  "".join(random.choice(chars) for x in range(7))
        try:
                order = instance.__class__.objects.get(order_id = the_id)
                id_generator()
        except Order.DoesNotExist:
                instance.order_id = the_id

        pre_save.connect(id_generator, sender=Order)


        class Equipment(models.Model):
        eqp_id = models.CharField(max_length=20, default="ADDDDADD")
        name = models.CharField(max_length=255, null=True, verbose_name='Название')
        m_date = models.IntegerField(verbose_name='Год выпуска')
        date = models.DateTimeField(auto_now_add=True)
        massa = models.IntegerField(default=10000)
        CONDITION_CHOICE = (
                ('Требуется ремонт', 'Требуется ремонт'),
                ('Среднее', 'Среднее'),
                ('Хорошее', 'Хорошее')
        )
        condition = models.CharField(max_length=255, default='Среднее',
verbose_name='Состояние', choices=CONDITION_CHOICE)
        engine_power = models.DecimalField(max_digits=5, decimal_places=2,
verbose_name='Мощность двигателя', default=50.00)
        bucket = models.DecimalField(max_digits=10, decimal_places=2, verbose_name='Емкость
ковша', default=0.15)

        class Meta:
                verbose_name='Список спецтехники'
                verbose_name_plural = 'Спецтехника'
```

```
def __str__(self):
        return self.name

def id_generator(instance, sender, *args, **kwargs):
chars=string.ascii_uppercase + string.digits
the_id =  "".join(random.choice(chars) for x in range(7))
try:
        equipment = instance.__class__.objects.get(eqp_id = the_id)
        id_generator()
except Equipment.DoesNotExist:
        instance.eqp_id = the_id

pre_save.connect(id_generator, sender=Equipment)


class Message(models.Model):
message = models.CharField(max_length=255)
product = models.ForeignKey(Product, on_delete=models.CASCADE)
date = models.DateTimeField(auto_now_add=True)

def __str__(self):
        return f'self.message'
class Meta:
        verbose_name = 'Сообщения сотрудников'
        verbose_name_plural = 'Сообщение сотрудника'


from api.models import Product, Order, Equipment
from rest_framework.views import APIView
from rest_framework.response import Response
from django.contrib.auth.models import User
from django.contrib.auth import authenticate, login

from api.models import Order, Product, Message, Equipment

from api.serializers import OrderSerializer, ProductSerializer, EquipmentSerializer

class Login(APIView):
def post(self, request, *args, **kwargs):
        username = request.data.get('login', False)
        password = request.data.get('password', False)

        if not username or not password:
                return Response({
                        "message": "Данные не подходят",
```

```
        "status": False
                })

        user = authenticate(request, username=username, password=password)
        if not user:
                return Response({
                        "message": "Неправильный логин или пароль",
                        "status": False
                })
        else:
                login(request, user)
                return Response({
                        "message": f'Добро пожаловать, {user.first_name} {user.last_name}',
                        "firstname": user.first_name,
                        "lastname": user.last_name,
                        "status": True
                })

class OrderList(APIView):
def get(self, request):
        orders = Order.objects.all()
        serializer = OrderSerializer(orders, many=True)
        return Response({
                "message": "",
                "status": True,
                "data": serializer.data
        })

class OrderDetail(APIView):
def get(self, request, order_id):
        order = Order.objects.filter(id =order_id).first()
        data = {
                "order_id": order.order_id,
                "customer": order.customer,
                "customer_contacts": order.customer_contacts,
                "comment": order.comment,
                "status": order.status,
                "product": order.product.name,
                "volume": order.volume,
                "amount": order.amount,
                "date": order.date,
                "image": order.product.image.url
        }
        return Response({
                "message": "",
```

```
        "status": True,
                "data": data
        })

class UpdateOrder(APIView):
def post(self, request, *args, **kwargs):
        order_id = request.data.get('order_id', False)
        status = request.data.get('status', False)

        if not order_id or not status:
                return Response({
                        "message": "Вы не изменили статус",
                        "status": False
                })

        order = Order.objects.filter(id=order_id).first()
        new_status = 0
        if status == "В очереди": new_status=1
        elif status == "В процессе": new_status=2
        elif status == "Завершена": new_status=3

        order.status=new_status
        order.save()
        return Response({
                "message": f'Статус заявки изменен на "{status}"',
                "status": True
        })

class ProductList(APIView):
def get(self, request):
        products = Product.objects.all()
        serializer = ProductSerializer(products, many=True)
        return Response({"data":serializer.data})

class ProductDetail(APIView):
def get(self, request, product_id):
        product = Product.objects.filter(id=product_id).first()
        serializer = ProductSerializer(product)

        return Response({"data": serializer.data})


class MessageView(APIView):
def post(self, request, *args, **kwargs):
        message = request.data.get('message', False)
```

```
            product_id = request.data.get('product_id', False)

            if not message or not product_id:
                return Response({"message": "Не полная информация"})

            message = Message.objects.create(
                message=message,
                product_id=product_id,
            )
            message.save()
            return Response({"message": "Спасибо, мы проверим наличие продукта на складе"})

        class TexnikaList(APIView):
        def get(self, request):
            texnika = Equipment.objects.all()
            serializer = EquipmentSerializer(texnika, many=True)
            return Response({"data": serializer.data})

        class Main(APIView):
        def get(self, request):
            orders = Order.objects.filter(status=3)
            serializer = OrderSerializer(orders, many=True)

            texnika = Equipment.objects.filter(condition="Требуется ремонт")
            serializer2 = EquipmentSerializer(texnika, many=True)
            data = {
                "orders" : serializer.data,
                "texnika" : serializer2.data
            }
            return Response({"data": data})

        from rest_framework import serializers
        from api.models import Order, Product, Equipment


        class OrderSerializer(serializers.ModelSerializer):
        class Meta:
            model = Order
            fields = '__all__'


        class ProductSerializer(serializers.ModelSerializer):
        class Meta:
            model = Product
```

```
        fields = ('__all__')
                # fields = ('id', 'name', 'amount', 'price', 'image')

        class EquipmentSerializer(serializers.ModelSerializer):
        class Meta:
                model = Equipment
                fields = ('__all__')

        from django.urls import path, include
        from api.views import Login, OrderList, OrderDetail, UpdateOrder, ProductList, ProductDetail,
MessageView, TexnikaList, Main

        urlpatterns = [
        path('login/', Login.as_view()),
        path('orders/', OrderList.as_view()),
        path('order/<int:order_id>', OrderDetail.as_view()),
        path('order/<int:order_id>/', UpdateOrder.as_view()),
        path('products/', ProductList.as_view()),
        path('product/<int:product_id>', ProductDetail.as_view()),
        path('message/', MessageView.as_view()),
        path('equipment/', TexnikaList.as_view()),
        path('main/', Main.as_view())

        ]

        from django.contrib import admin

        from api.models import Order, Product
        from django.utils.safestring import mark_safe
        from django.utils.encoding import force_text
        from django.urls import reverse
        @admin.register(Product)
        class ProductAdmin(admin.ModelAdmin):
        list_display = ('name', 'volume', 'price')
        fieldsets = (
           (None,
                {'fields': ('name', 'volume', 'price', 'image')
              }),
           )



        @admin.register(Order)
        class OrderAdmin(admin.ModelAdmin):
```

```
        list_display = ('order_id', 'customer', 'customer_contacts', 'comment', 'product', 'volume',
'amount', 'product_name')
        list_filter = ('product',)
        fieldsets = (
           ('Заявка',
                {'fields': ('customer', 'customer_contacts', 'comment', 'product', 'volume', 'status')
            }),
           )

        from api.models import Message

        @admin.register(Message)
        class MessageAdmin(admin.ModelAdmin):
        list_display = ('message', 'product_edit_link', 'date')

        def product_edit_link(self, instance=None):
                if instance.product.name:
                        url = reverse('admin:%s_%s_change' % (instance.product._meta.app_label,
instance.product._meta.model_name), args = [force_text(instance.product.pk)])
                        return mark_safe(f'<a href="{url}">{instance.product.name}</a>')
        product_edit_link.short_description = 'Товар'

        def has_add_permission(self, request):
                return False
        def has_change_permission(self, request, instance=None):
                return False



        from api.models import Equipment

        @admin.register(Equipment)
        class EquipmentAdmin(admin.ModelAdmin):
        list_display = ('name', 'm_date', 'condition', 'engine_power', 'bucket', 'massa')

        import os
        BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


        SECRET_KEY = '(yipyj=_kqpo#92wd3b&spb9lx$y3c$bx1(9wc*&a+t$jc8%@p'
        DEBUG = True

        ALLOWED_HOSTS = []
```

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'api',
    'rest_framework'
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'temirtas.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'temirtas.wsgi.application'


# Database
# https://docs.djangoproject.com/en/2.1/ref/settings/#databases
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}


# Password validation
# https://docs.djangoproject.com/en/2.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/2.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.1/howto/static-files/

REPOSITORY_ROOT = os.path.dirname(BASE_DIR)
```

```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(REPOSITORY_ROOT, 'staticfiles/')

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(REPOSITORY_ROOT, 'media/')
```