ФЕДЕРАЛЬНОЕ АГЕНТСТВО МОРСКОГО И РЕЧНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Государственный университет морского и речного флота имени адмирала С.О. Макарова»
(ФГБОУ ВО «ГУМРФ имени адмирала С.О. Макарова»)
Воронежский филиал

Кафедра математики, информационных систем и технологий
Направление подготовки 09.03.02 Информационные системы и технологии
(код, наименование направления подготовки/специальности)
Форма обучения очная

«К ЗАЩИТЕ ДОПУЩЕН(А)»
Заведующий кафедрой

(подпись)
Черняева С. Н.
(ФИО)
21 июня 2024

Выпускная квалификационная работа

Обучающегося Кирмас Александра Дмитриевича
(фамилия, имя, отчество)
Вид работы выпускная квалификационная работа бакалавра
(выпускная квалификационная работа бакалавра, специалиста, магистра)

Пояснительная записка

Тема Разработка веб-ориентированной АИС учета оборудования транспортного предприятия (на примере ООО «Бизнес-Курьер»)

(полное название темы квалификационной работы, в соответствии с приказом об утверждении тематики ВКР)

Руководитель работы к.э.н., доцент каф. мисит, Скрипников О.А., 21.06.2024
(должность, подпись, фамилия, инициалы, дата)
Консультант
(при наличии) (должность, подпись, фамилия, инициалы, дата)
Консультант
(должность, подпись, фамилия, инициалы, дата)
Обучающийся Кирмас А.Д., 21.06.2024
(подпись, фамилия, инициалы, дата)

Воронеж
2024

ФЕДЕРАЛЬНОЕ АГЕНТСТВО МОРСКОГО И РЕЧНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Государственный университет морского и речного флота имени адмирала С.О. Макарова»
(ФГБОУ ВО «ГУМРФ имени адмирала С.О. Макарова»)
Воронежский филиал

Кафедра математики, информационных систем и технологий
Направление подготовки 09.03.02 Информационные системы и технологии
(код, наименование направления подготовки/специальности)
Форма обучения очная

УТВЕРЖДАЮ
Заведующий кафедрой
_____
(подпись)
Черняева С. Н.
(ФИО)
21 июня 2024

Задание
на выпускную квалификационную работу

Вид работы ВКР бакалавра
(ВКР бакалавра, ВКР специалиста, ВКР магистра)

Обучающемуся Кирмас Александру Дмитриевичу
(фамилия, имя, отчество)

Тема Разработка веб-ориентированной АИС учета оборудования транспортного предприятия (на примере ООО «Бизнес-Курьер»)

Утверждена приказом ректора университета от _____ 2024, № _____

Срок сдачи законченной работы _____ 2024

Исходные данные (или цель ВКР):

_____
_____
_____
_____

Перечень подлежащих исследованию, разработке, проектированию вопросов (краткое содержание ВКР):
(актуальность темы, цели и задачи ВКР; аналитический обзор литературных источников; постановка задачи исследования, разработки, проектирования; содержание процедуры исследования, разработки, проектирования; обсуждение результатов; дополнительные вопросы, подлежащие разработке; заключение – выводы по работе в целом, оценка степени решения поставленных задач, практические рекомендации; и др.)
– Введение. Актуальность выбранной темы, цель и задачи ВКР
(наименование вопроса, раздела и его краткое содержание)
– Исследовательский раздел.

(наименование вопроса, раздела и его краткое содержание)

– Проектный раздел.

(наименование вопроса, раздела и его краткое содержание)

– Заключение. Выводы по работе в целом. Оценка степени решения поставленных задач

(наименование вопроса, раздела и его краткое содержание)

Практические рекомендации

Перечень графического материала (или презентационного материала):

1. Титульный лист
2. Цель и задачи выпускной квалификационной работы
3. Анализ теоретических аспектов учета оборудования
4. Изучение основных инструментов и технологий для реализации учета оборудования
5. Разработка веб-ориентированной информационной системы учета оборудования транспортного предприятия ООО «Бизнес-Курьер»
6. Разработка веб-ориентированной информационной системы учета оборудования транспортного предприятия ООО «Бизнес-Курьер»
7. Разработка веб-ориентированной информационной системы учета оборудования транспортного предприятия ООО «Бизнес-Курьер»
8. Интерфейс пользователя
9. Интерфейс пользователя
10. Интерфейс пользователя
11. Интерфейс пользователя
12. Результаты выпускной квалификационной работы
13. Спасибо за внимание

Консультанты по разделам ВКР (при наличии):

1. _____

(наименование раздела, ученая степень, ученое звание и должность, ФИО консультанта)

2. _____

(наименование раздела, ученая степень, ученое звание и должность, ФИО консультанта)

3. _____

(наименование раздела, ученая степень, ученое звание и должность, ФИО консультанта)

Дата выдачи задания: _____ 2024

Задание согласовано и принято к исполнению: _____ 2024

Руководитель ВКР: к.э.н., доцент каф. мисит, Скрипников О.А. _____

(должность, ученая степень, ученое звание, ФИО) (подпись)

Обучающийся: Кирмас А.Д. _____

(учебная группа, ФИО) (подпись)

Содержание

Введение

В современном мире по причине очень большого объема данных, использование информационных систем становится необходимым почти в любой области. Инструменты, предоставляемые компьютерными технологиями, позволяют не только удобно хранить информацию, не затрагивая большое количество физических носителей, но и структурировать ее, и управлять ей. Также обеспечивают удобочитаемость таких данных.

Веб-ориентированные информационные системы являются отличным решением, благодаря эргономичному интерфейсу и общедоступностью в использовании. Правильная интеграция и оптимизация данных обеспечивают эффективное управление предприятием.

Проблема, исследуемая в выпускной квалификационной работе. Учет оборудования с применением веб-технологий. Анализ этого процесса, способы реализации, инструменты и технологии, используемые для разработки информационной системы.

Работа также включает в себя изучение подходящего шаблона проектирования сайта, определение преимуществ и выявление недостатков данного решения.

Разработка веб-ориентированной информационной системы позволит улучшить эффективность учета оборудования предприятия.

Объектом исследования является ООО «Бизнес-Курьер».

Предмет исследования выпускной квалификационной работы – учет оборудования транспортного предприятия.

Целью дипломной работы является разработка веб-ориентированной информационной системы учета оборудования транспортного предприятия.

Поставленная цель предполагает решение следующих задач выпускной квалификационной работы:

— Анализ теоретических аспектов учета оборудования;

— Изучение основных инструментов и технологий для реализации учета оборудования;

— Разработка веб-ориентированной информационной системы учета оборудования транспортного предприятия ООО «Бизнес-Курьер»;

Выпускная квалификационная работа состоит из четырех разделов:

Первый раздел выпускной квалификационной работы содержит в себе общую информацию о предприятии ООО «Бизнес-Курьер», анализ теоретических вопросов, связанных с учетом оборудования, выявление недостатков существующей информационной системы учета данных, а также проанализированы, существующие инструменты и технологии, предназначенные для разработки веб-ориентированных информационных систем.

Во втором разделе описана разработка веб-ориентированной информационной системы.

1 ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ

1.1 Общая информация о предприятии. Организационная структура ООО «Бизнес-Курьер» и ее характеристика

Компания ООО «Бизнес-Курьер» основана в 2005 году. «ООО «Бизнес-Курьер» является официальным представителем компании «Major Express» в Воронеже и специализируется на предоставлении услуг по экспресс-доставке корреспонденции и грузов по всей России и миру." Современные технологии, собственный автопарк и многолетний кропотливый труд позволили компании стать одним из лидеров на Воронежском рынке курьерских услуг.

Сеть Major Express осуществляет экспресс-доставку корреспонденции и грузов в более чем 7000 городов и населённых пунктов на всей территории РФ, 220 стран на всех континентах.

Качественная работа и стабильная ценовая политика позволяют завоевывать доверие крупнейших российских компаний из всех отраслей бизнеса и обеспечивать клиентам максимальный комфорт при работе с нами.

Общая информация о предприятии:

— Полное название фирмы: ООО «Бизнес-Курьер»;

— Адрес: 196641 г. Воронеж, ул. 45 Стрелковой Дивизии, д. 62А (остановка Детская больница №7);

— Контактные данные: тел. – +7(473)2023333, эл. почта – 397217@mail.ru;

— Дата создания: 28.02.2005.

Организационная структура управления предприятия представлена на рисунке 1.

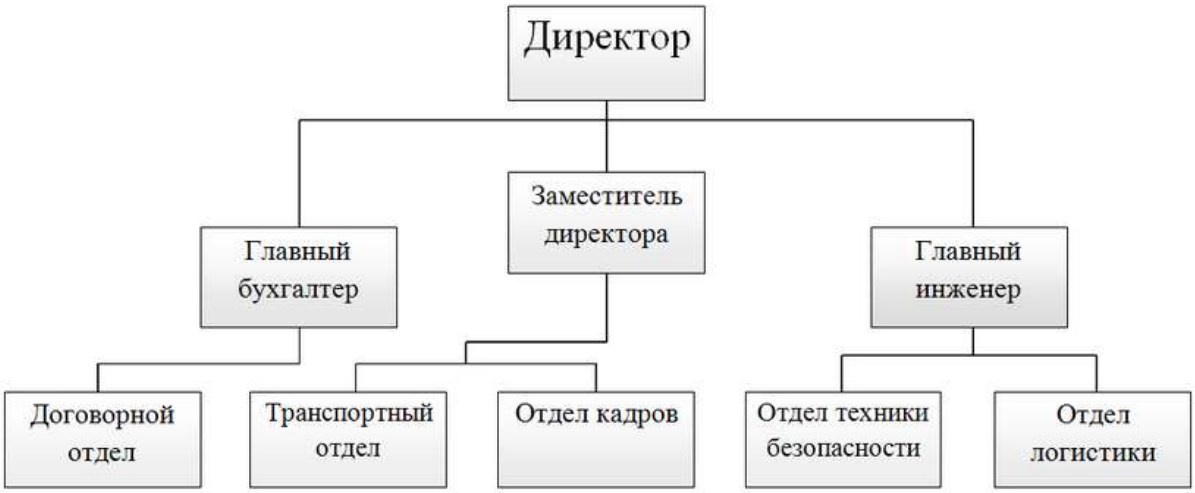

Рисунок 1 – Организационная структура предприятия

1.2 Элементы информационного и технического обеспечения, функционирующие на предприятии ООО «Бизнес-Курьер»

На момент прибытия в отделе находились инженер-программист и системный администратор.

Конфигурация персонального компьютера инженера-программиста в ООО «Бизнес-Курьер»:

— операционная система Windows 10 Pro;

— процессор Intel Core i3 9100K@ 3.6GHz;

— ОЗУ Apacer [DL.08G2K.KAM] 8 ГБ x 2;

— видеокарта интегрированная;

— монитор HP x24C.

Программное обеспечение, установленное на персональном компьютере инженера-программиста в ООО «Бизнес-Курьер»:

— VS code;

— браузер Mozilla Firefox;

— Microsoft_Office 2016;

— Adobe Photoshop;

— AIDA64.

Конфигурация персонального компьютера системного администратора в ООО «Бизнес-Курьер»:

— операционная система Debial Linux;

— процессор Intel Core i5 12400K@ 3.6GHz;

— ОЗУ Apacer [DL.08G2K.KAM] 8 ГБ x 2;

— видеокарта интегрированная;

— монитор HP x24C.

Программное обеспечение, установленное на персональном компьютере системного администратора в ООО «Бизнес-Курьер»:

— VS code;

— браузер Mozilla Firefox;

— LibreOffice 2016;

— Netcat.

Целью инженера-программиста была поддержка клиентской части веб-ориентированной информационной системы.

Целью системного администратора была поддержка серверной части веб-ориентированной информационной системы.

Контекстная диаграмма IDEF0 ООО «Бизнес-Курьер» перевозки грузов изображена на рисунке 2.

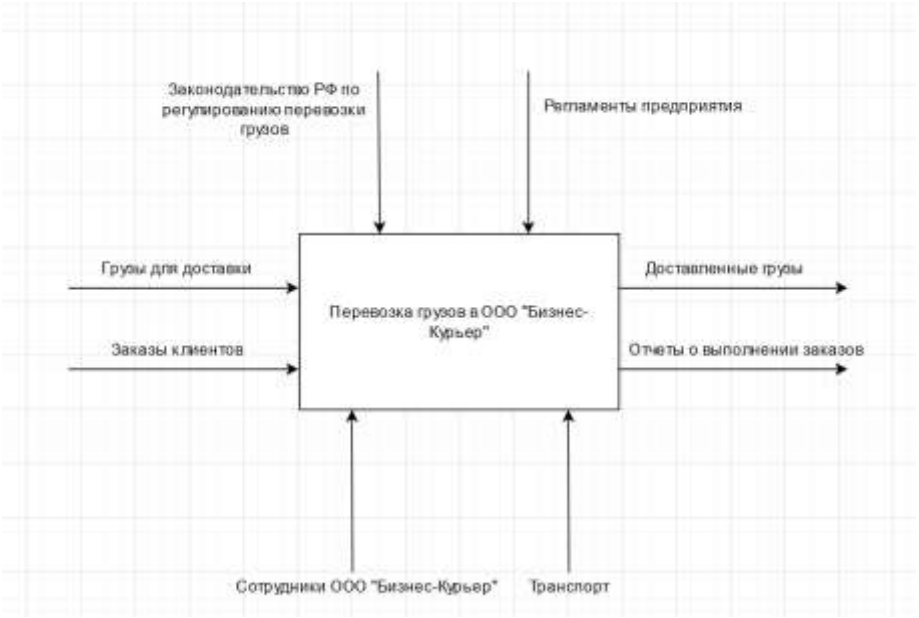

Рисунок 2 – Контекстная диаграмма IDEF0

Входными данными здесь являются «Грузы для доставки» и «Заказы клиентов».

Выходные данные – «Доставленные грузы» и «Отчеты о выполнении заказов».

За механизмы перевозки грузов отвечают «Сотрудники ООО «Бизнес-Курьер»» и «Транспорт».

Регулирование перевозки грузов осуществляют «Законодательство РФ по регулированию перевозки грузов» и «Регламенты предприятия».

Декомпозиция контекстной диаграммы ООО «Бизнес-Курьер» перевозки грузов показана на рисунке 3.

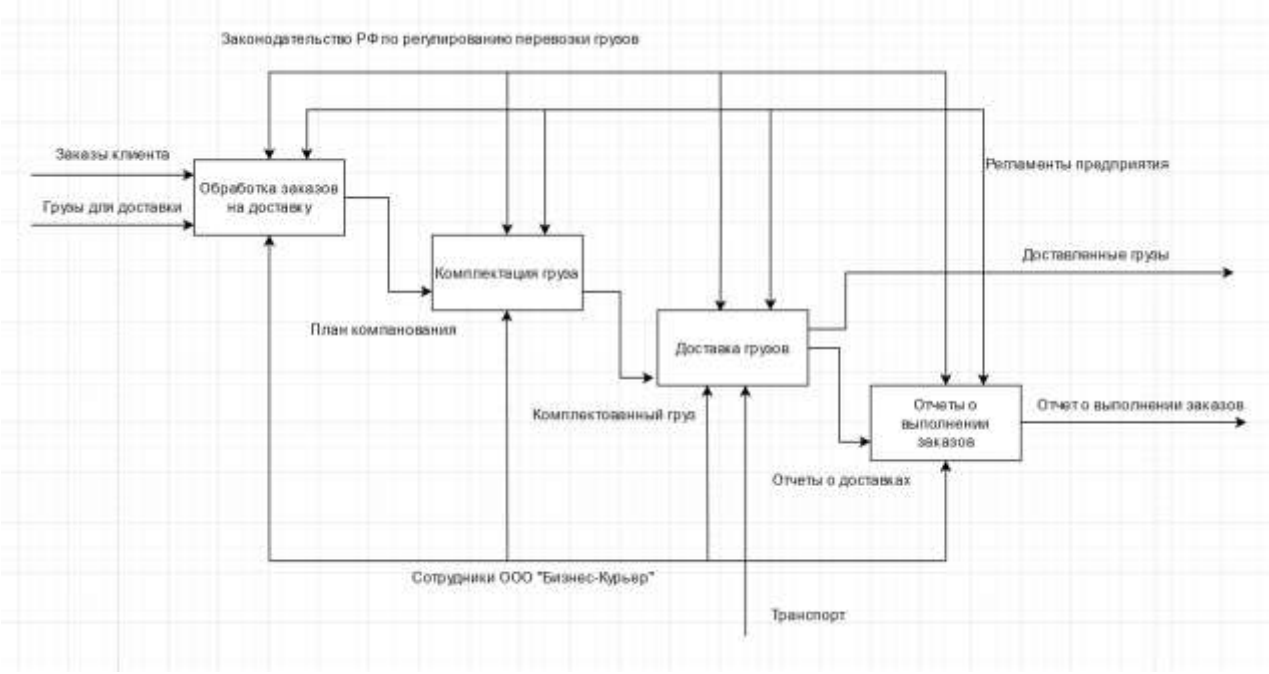

Рисунок 3 – Декомпозиция контекстной диаграммы IDEF0

1.3 Постановка задачи и формирование функциональных требований к проектируемой системе

Согласно индивидуальному заданию, требуется разработать веб-ориентированную информационную систему учета транспортного предприятия.

Веб-ориентированная информационная система предприятия была разработана с помощью системы управления контента (Content Management

System – CMS). Данная система достаточно длительное время уже не поддерживается разработчиками и устарела. На сегодняшний день она стала неактуальной для дальнейшей работы на мировом рынке. CMS – позволяет быстро разрабатывать сайты, но из-за того, что она была разработана достаточно давно она имеет множество отрицательных сторон, например, такие как невозможность соответствия всем требованиям для обеспечения безопасности данных и низкую производительность. На предприятии поддержка предыдущей информационной системы осуществлялась двумя сотрудниками.

За счет внедрения новой информационной системы были повышены производительность, безопасность и масштабируемость информационной системы. Немаловажным плюсом служило и то что предприятие перестало зависеть от конкретных сотрудников и появилась возможность автоматизировать процесс.

В проектируемой веб-ориентированной информационной системе в зависимости от уровня пользователя будут доступны различные возможности.

Первый уровень доступа – гости, имеют минимальный набор функциональных возможностей. Основными возможностями информационной системы, доступными обычному пользователю будут:

— авторизация или регистрация;

— формирование, изменение и удаление только своих заказов.

Диаграмма возможностей доступных обычному пользователю изображена на рисунке 4.

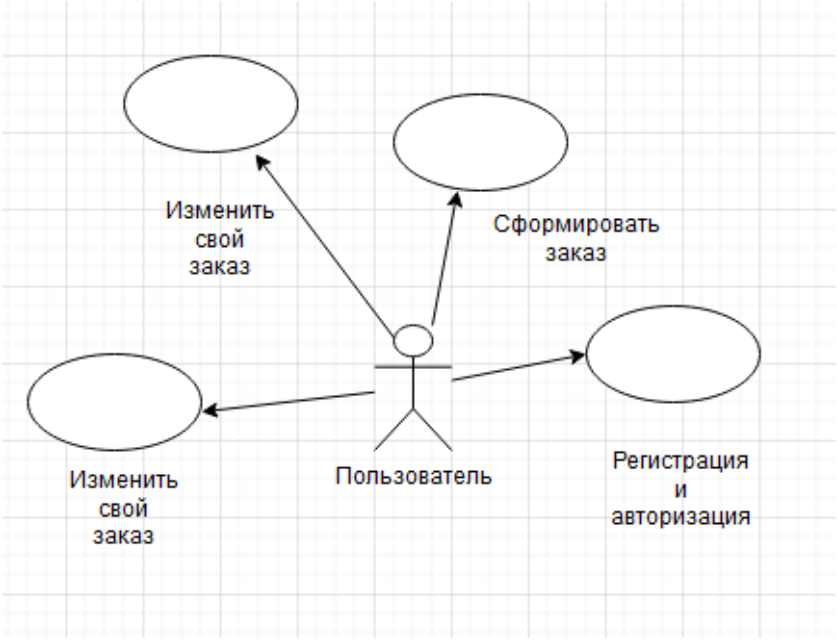

Рисунок 4 – Диаграмма возможностей доступных обычному пользователю

Второй уровень доступа – администратор, помимо доступа к функциям обычного пользователя, имеет следующие возможности:

— изменение и удаление заказов без ограничений;

— просмотр списков: грузов, отправителей и получателей.

Также в информационной системе с веб-интерфейсом будут следующие разделы:

1. Раздел: «Курьеры».

Данный раздел содержит информацию о курьерах, их имя, фамилия, отчество, номер телефона, email, марку и модель транспорта курьера, номер и регион транспорта и его цвет.

2. Разделы: «Отправители» и «Получатели».

Данные разделы содержат информацию об отправителях и получателях, а именно: имя, фамилия, отчество, номер телефона, email, организация, ИНН, а также индекс и адрес отправителя.

3. Раздел: «Грузы».

Данный раздел содержит информацию о доставляемых грузах, а именно: содержимое, вес, габариты, количество пакетов.

4. Раздел: «Заказы».

Данный раздел содержит информацию о всех предыдущих разделах кроме раздела «Транспорт», а также имеет несколько уникальных полей: тип заказа, дата исполнения, кто оплатит (отправитель или получатель), форма оплаты, комментарий, нужна ли доверенность на забор груза и стоимость страховки.

Диаграмма вариантов использования веб-ориентированной информационной системы для обычного пользователя и администратора предоставлена на рисунке 5.

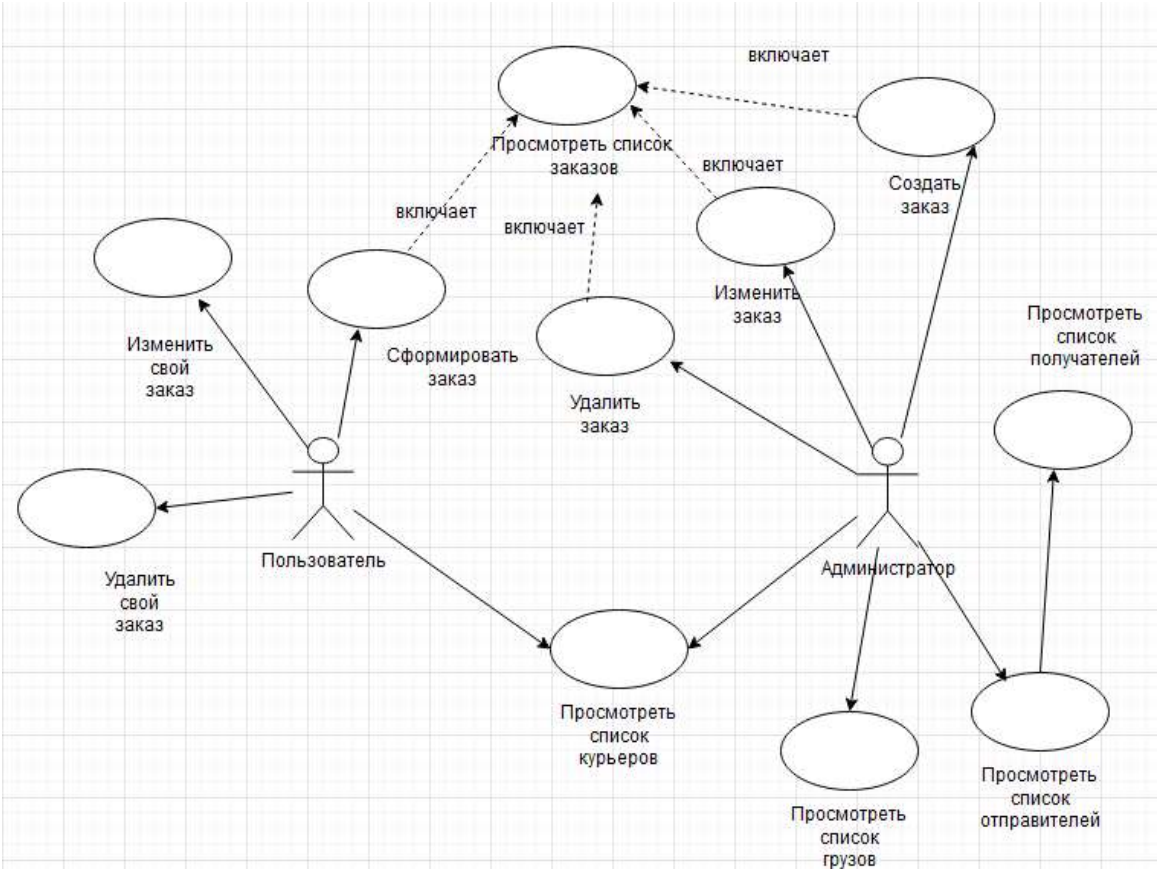

Рисунок 5 – Диаграмма вариантов использования системы

1.4 Выбор языков программирования и фреймворков для разработки серверной и клиентской части приложения

Компьютеры – это мощные машины, которые помогают человеку в самых разных сферах деятельности. Делающими их пригодными для использования являются языки программирования. Каждый язык будь то низкоуровневые или высокоуровневые языки, имеют свои сильные и слабые

стороны. Выбор языка программирования, который будет использоваться, при разработке веб-ориентированной информационной системы, зависит от типа программы и типа компьютера, на котором она будет работать. Наиболее подходящий язык программирования для информационной системы выбирается в соответствии со следующими критериями выбора языка:

— масштабируемость;

— параллелизм;

— надежность;

— безопасность;

— производительность;

— гибкость;

— переносимость;

— высокая целостность;

— простота использования.

Для анализа языков программирования, были изучены наиболее популярные среди разработчиков интернет ресурсы StackOverflow и GitHub.

В результате рассмотрения технологий – JavaScript, Java-applets и платформы Adobe Flash, выяснилось, что применимым можно считать только JavaScript, из-за невероятного удобства выполнения его кода в браузере.

Это объясняется тем, что JavaScript используется во многих областях (клиент браузер, серверная часть, мобильные платформы), а также в настольных приложениях, в то время как Java-апплеты и Adobe Flash технологии, требуют создание дополнительных компонентов в системе пользователя. Для разных операционных систем и браузеров требуются разные процессы установки и настройки.

Следует также отметить, что Adobe Flash и Java-апплеты не поддерживаются мобильными браузерами. А Adobe Flash еще и не имеет поддержки HTML5. JavaScript поддерживается всеми распространенными браузерами и включен в них по умолчанию. Поэтому, JavaScript – это лучший выбор для разработки клиентской части.

Поскольку разработка большого одностраничного приложения на чистом JavaScript – сложный и неэффективный процесс, поэтому необходимо использовать фреймворки, определяющие структуру приложения и имеющие базовый набор компонентов. Фреймворки оценивались по количеству применения их в проектах, размеру сообщества разработчиков, актуальности задачи и времени нахождения на рынке. Однако прямое сравнение фреймворков для разработки, не даст результатов, так как каждый из них позволит достичь конечного результата. Тем не менее некоторые фреймворки считаются более подходящими из-за лучшей масштабируемости, меньших затрат на обучение и большего количества готовых модулей.

Фреймворк – это структура, предоставляющая набор инструментов, библиотек и правил для разработки приложений. Использование готовых модулей и шаблонов ускоряет процесс разработки, так как программисту не нужно писать код с нуля, уменьшая время, затраченное на создание программы. Вместо того, чтобы переписывать одну и ту же функциональность снова и снова, можно просто использовать существующие модули, экономя время и силы программистов. Это также помогает улучшить качество кода, так как уже используются проверенные и отлаженные модули.

Наконец, использование фреймворков улучшает масштабируемость программы. Готовая архитектура позволяет легко добавлять новую функциональность, без необходимости полностью переделывать существующий код. Это позволяет создавать более гибкие и адаптивные информационные системы, которые могут легко расширяться вместе с потребностями предприятия. В целом, фреймворк – эффективный инструмент для создания качественного и масштабируемого программного обеспечения.

Все фрейморки обладают практически схожим функционалом и способны решить поставленную задачу. Поэтому, выбор основывался не на функционале фреймворка, а на требованиях и целях конкретного проекта.

Для анализа были выбраны наиболее известные фреймворки:

— Backbone.js;

— AngularJS;

— React;

— Ember.js;

— Vue.js;

— Polymer.

Backbone.js плохо подходит для разработки масштабных проектов, поскольку в нем отсутствуют компоненты, необходимые для создания сложной функциональности. Как следствие этот язык является непрактичным из за недостатка мощности.

Polymer – это библиотека, основанная на относительно новой технологии Web Components. Спецификация W3C для этой технологии еще не завершена. Могут возникнуть проблемы с поддержкой браузеров, проблемы со стабильностью работы, а также Polymer имеет большой порог входа для разработчиков. В связи с этим от использования данного фреймворка, было решено отказаться, из-за возможных рисков.

AngularJS, Ember.js и Vue.js имеют двунаправленную привязку данных, возможность создавать большие системы, хорошую документацию и сообщество.

Ember.js имеет сложную структуру проекта, что затрудняет работу начинающих программистов с ним, а в случае выхода за рамки стандартного использования является громоздким и не гибким. Также, фреймворк менее популярен, чем AngularJS и Vue.js.

Vue.js в настоящее время является самым быстрорастущим популярным фреймворком. Но Vue.js не поддерживает стиль REST и для этого требуется дополнительная библиотека Axios. Кроме того, фреймворк создан недавно и разрабатывается в основном одним человеком, а значит, существует множество рисков, связанных с его использованием.

В качестве фронтенд-фреймворка для разработки клиентской части веб-приложения был выбран React, написанный на JavaScript, и имеющим

хорошую поддержку со стороны разработчика. React использует виртуальную объектную модель документа (DOM).

Объектная модель документа (DOM) – это программный интерфейс для веб-документов, позволяющий получить доступ к HTML, XHTML и XML документам, а также изменять их содержимое, структуру и оформление.

Также React использует синтаксическое расширение JavaScript – JSX, которое позволяет использовать для описания структуры интерфейса использовать подобный синтаксису HTML документ, что сильно упрощает процесс разработки.

В результате фреймворк React лучше всех подходит для разработки проектируемой информационной системы. Также при создании веб-приложения были использованы HTML и CSS – очень популярные и самые востребованные инструменты на данный момент, которые используются для разработки почти всех сайтов.

HTML – это стандартизированный язык гипертекстовой разметки документов, используемый браузером для загрузки веб-страниц. При входе на сайт, браузер подгружает HTML-файл с информацией структуры элементов, их типов и вложенном контенте. HTML не загружает сайт, он только указывает положение, базовый дизайн и пути к импортированным элементам.

Код HTML состоит из тегов, определяющих вид элемента, некоторыми из них являются:

— гиперссылки;

— таблицы;

— блоки;

— формы;

— заголовки;

— изображения.

Имея в своем наборе только теги, помогающие браузеру правильно отображать содержимое, HTML немногим отличается от текстового файла и не является как таковым языком программирования.

CSS – это каскадные таблицы стилей, содержащие набор параметров форматирования, который применяется к компонентам документа для стилизации веб-страницы. Иерархическая структура CSS существенно упрощает верстку сайтов, для изменения стиля нескольких элементов достаточно внести правки в одно свойство. Также за счет хорошей поддержки со стороны разработчика, в CSS появляются новые возможности, упрощающие старую или добавляя новую функциональность. Немаловажным плюсом CSS, является кэширование файла, при первом заходе на страницу, что намного сильно повышает скорость загрузки сайта при последующих посещениях.

CSS состоит из селекторов, указывающих на элементы, к которым будут применяться созданные стили. Под каждым из селекторов находится блок объявлений, содержащий набор параметров и значений, заключенных в фигурные скобки.

Разработка серверной части платформы позволяет выбирать из достаточно широкого спектра технологий, по сравнению с клиентской частью. Это связано, прежде всего, с тем, что серверные технологии зависят от предпочтений разработчиков, оборудования и требований к проекту, тогда как клиентские технологии сильно ограничены. Выбор технологического решения для разработки серверных компонентов лучше начинать не с программирование языков, но с учетом фреймворков, поскольку они задают базовую структуру разработки приложения.

Было установлено, что требуется высокая степень масштабируемости проекта, следует уделить внимание неблокирующим платформам ввода-вывода. В связи с этим необходимо исключить из рассмотрения Laravel, Symfony и Ruby on Rails. Кроме того, из-за сложности реализации неблокирующего ввода-вывода и пользовательских интерфейсов Vaadin Фреймворк не подходит для проекта.

ASP .NET MVC накладывает дополнительные ограничения на инфраструктуру при отсутствии существенных преимуществ, поэтому

фреймворк следует исключить из дальнейшего рассмотрения. Таким образом, основной выбор будет сделан между фреймворками Express.js, Loopback и Play.

Важным фактором является язык программирования на на котором написан фреймворк. Express.js и Loopback написаны на Node.js (JavaScript), а Play — на Java. Django написан на производительном и удобном в плане синтаксиса Python. Python является одними из самых популярных языков программирования в мире по статистике таких крупных сервисов, как GitHub и StackOverflow. В связи с этим, целесообразнее использовать фреймворки Django. Одним из основных принципов фреймворка является «не повторяйся» (DRY).

DRY (Don’t repeat yourself) – принцип разработки программного обеспечения, ориентированный на уменьшение количества повторяемой информации в коде. Согласно этому принципу код должен создаваться при помощи конвертации данных и генераторов кода, которые позволяют программисту не использовать операции копирайтинга.

Django использует концепцию программирования MVC (Model-View-Controller), которая разделяет данные и управляющую логику на три компонента:

— модель;

— представление;

— контроллер.

«Модель» представляет собой данные и изменяет свое состояние, реагируя на команды контроллера.

«Представление» реагирует на изменение состояния модели и визуализирует ее данные пользователю.

«Контроллер» взаимодействует с моделью, интерпретируя действия пользователя.

MVC не требует строгой реализации, поэтому нет общепринятого определения, где будет располагаться бизнес-логика. В разрабатываемой веб-

ориентированной информационной системе был выбран способ реализации бизнес-логики в модели по причине использования методологии программирования ООП.

При разработке с применением принципов ООП используется активная модель MVC, где «модель» отвечает не только за доступ к данным и СУБД, но и за, всю бизнес-логику продукта. Контроллер же отвечают за:

— прием запросов от пользователей;

— анализ запросов;

— определение следующего действия приложения, анализируя результаты анализа.

Визуальное представление концепции MVC представлена на рисунке 6.

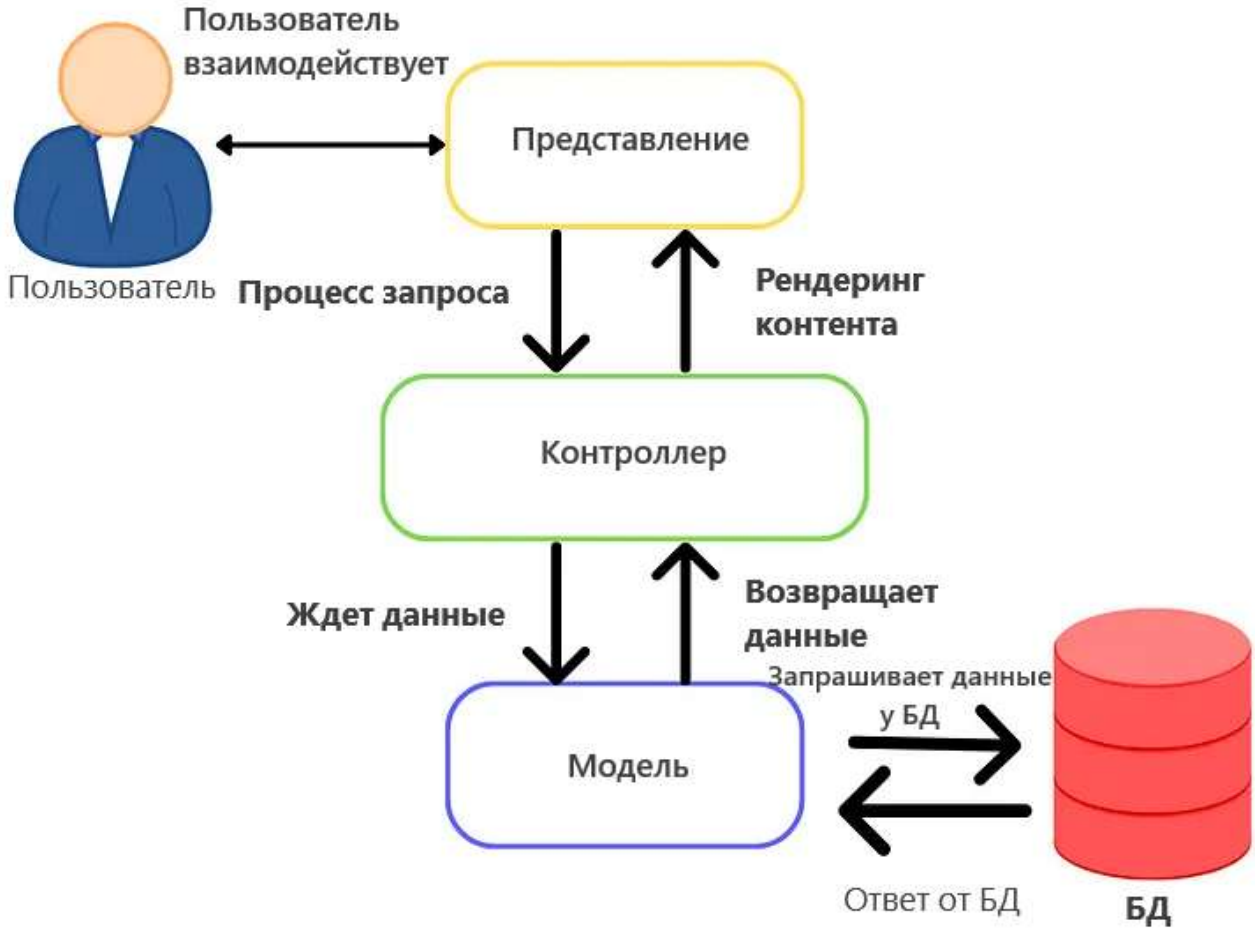

Рисунок 6 – Визуальное представление MVC

Объектно-ориентированное программирование (ООП) – парадигма разработки, состоящая из набора правил и критериев программирования,

основывающаяся на представления программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса. Классы в свою очередь могут наследоваться. Основополагающими принципами объектно-ориентированного программирования являются:

— инкапсуляция;

— полиморфизм;

— наследование.

Инкапсуляция отвечает за сокрытие, содержащихся внутренних данных отдельного компонента от других. Правильная инкапсуляция способствует переиспользованию компонентов, посредством API. Из-за слабой связанности компонентов, ускоряется разработка и тестирование информационной системы. Также улучшается читабельность кода.

Наследование позволяет создать иерархическую структуру объектов. Можно создать общий класс, который будет определять свойства и поведение связанных классов. Унаследованные классы в свою очередь могут добавить уникальные, свойственные только им характеристики, дополняя или изменяя поведение базового класса под себя.

Полиморфизм неразрывно связан с предыдущим принципом и заключается в способности объекта использовать методы производного класса, не существующего на момент создания базового.

1.5 Выбор среды разработки

Среда разработки (IDE) значительно облегчает разработку проектов. Среды поддерживают разные языки программирования и предоставляют разные возможности, но базовый функционал IDE обычно включает:

— компилятор – переводит написанный на языке программирования код в набор машинных кодов;

— интерпретатор – нужен для запуска скриптов, не нуждающихся в компиляции;

— встроенную отладку – выделяет ошибки;

— умное автодополнение кода – предлагает подсказки для разработки, исходя из контекста;

— наличие пользовательских расширений, обеспечивающие более комфортную разработку.

На сегодняшний день можно выделить три самые популярные среды разработки – это:

1. PyCharm – это IDE, произведенная компанией JetBrains, специализированной на создании профессиональных инструментов для разработки.

Pycharm был создан в большинстве своем для языка программирования Python. Помимо базовых функций, PyCharm предлагает умную навигацию, профилировщик Python, инструменты рефакторинга, а также имеет более продвинутую систему автодополнения кода. Также платная лицензионная версия данной среды разработки поддерживает HTML, JavaScript и CSS, являющимися одними из основополагающих инструментов для разработки современных веб-ориентированных приложений.

2. Intellij IDEA – платформа также разработанная компанией JetBrains. Данная среда разработки ориентирована на язык программирования Java. Большим преимуществом Intellij IDEA над PyCharm, является поддержка HTML, CSS, JavaScript в базовой версии продукта, а не в платной. Среди прочих возможностей среда разработки обладает интеграцией с инструментами сборки Apache Maven, Gradle, Webpack, а начиная с шестой версии продукта IDE предоставляет интегрированный инструментарий для разработки графического пользовательского интерфейса.

3. Visual Studio Code (VS code) – IDE разработанная одной из крупнейших в разработчиков в сфере проприетарного программного обеспечения Microsoft, поддерживающая все наиболее популярные языки программирования: JavaScript, HTML, CSS, Python, PHP и другие. Продукт предоставляет полный пакет возможностей стандартной среды разработки, в

том числе интеграцию с Git и Azure. Visual Studio Code – это полностью бесплатный редактор кода, поддерживающий 72 языка и имеющий огромную библиотеку расширений.

Для разработки веб-ориентированной информационной системы был выбран Visual Studio Code.

1.6 Выбор СУБД для разработки информационной системы

Выбор системы управления базой данных является немаловажной частью проектирования информационной системы. СУБД – это комплекс программ, позволяющих создать базу данных и манипулировать ими. Основными функциями СУБД являются:

— создание баз данных, изменение, удаление и объединения их по определенным признакам;

— структурированное хранение данных;

— обеспечение безопасности данных при помощи разделенного доступа;

— выгрузка и сортировка данных по заданным фильтрам при помощи запросов;

— обеспечение целостности данных, резервное копирование и восстановление после сбоев.

Чтобы управлять базой данных и находить нужную информацию, пишутся запросы на специальных языках. Самым популярным из них является язык структурированных запросов – SQL.

СУБД классифицируются по разным параметрам:

— хранение баз данных;

— хранение и обработка данных и запросов;

— структура и организация данных.

Классификация СУБД по хранению баз данных разделяется на:

— локальное, в которых все элементы системы и баз данных хранятся на одном сервере;

— распределенное, в которых элементы находятся на разных серверах, в том числе облачных.

Классификация СУБД по хранению баз данных разделяется на:

— клиент-серверное;

— файл-серверное.

При клиент-серверном хранении, базы данных размещены на одном сервере с СУБД, к которому обращаются с запросами разные пользователи. Такими СУБД являются:

— Firebird;

— MS SQL Server;

— Oracle;

— PostgreSQL.

При файл-серверном хранении, базы данных хранятся на одном файл-сервере, а СУБД должно находиться на каждом устройстве, с которого отправляются запросы. К таким СУБД относятся:

— OpenEdge;

— SQLite;

— Microsoft SQL Server Compact.

Классификация СУБД по структуре и организации данных разделяется на:

— реляционные;

— ключ-значение;

— документные;

— графовые;

— колоночные.

При использовании реляционных СУБД данные представлены в виде таблиц, связанных между собой внешними ключами. Такими СУБД являются:

— MySQL;

— PostgreSQL.

В СУБД типа «ключ-значение» данные используют уникальный идентификатор, состоящий из двух частей – ключ и значение, который присваивается каждой единице данных. К таким СУБД относятся:

— Redis;

— Memcashed.

В документных СУБД в базах данных хранятся документы со структурированным текстом и особым синтаксисом. Это могут быть архивы, каталоги или журналы действий, логи для сайта. Примеры таких СУБД:

— Amazon DocumentDB;

— CouchDB;

— MongoDB.

Графовые СУБД применяют структуру графа для хранения данных и запросов. Такими СУБД выступают:

— Amazon Neptune;

— Neo4j;

— InfoGrid.

Колоночные СУБД похожи на реляционные, только здесь данные представлены в виде колонок, каждая из которых схожа с поведением таблиц. Примерами таких СУБД служат:

— SAP IQ;

— Google Bigtable;

— Vertica.

Для разрабатываемой веб-ориентированной системы была выбрана SQLite. SQLite – система управления базами данных, написанная на языке программирования C и предоставляющая возможность управления реляционными базами данных. В настоящее время SQLite, является одной из

самых популярных СУБД, ее используют большинство популярных браузеров – Mozilla Firefox, Google Chrome, Safari.

В отличии от множества СУБД, SQLite не требует сервер для базы данных, предоставляя интегрированный движок базы данных, который обращается напрямую к файлу базы данных на диске. Поэтому для работы СУБД не требует установки или конфигурации. Формат файла базы данных является кроссплатформенной, что позволяет создавать и управлять файлом базы данных на одном устройстве с одной операционной системой, а затем спокойно скопировать его на другое устройство с другой ОС. Также SQLite по умолчанию интегрирована в Django, что послужило большим плюсом для выбора именно этой СУБД.

Также была спроектирована база данных. Ее визуальное представление изображено на рисунке 7.

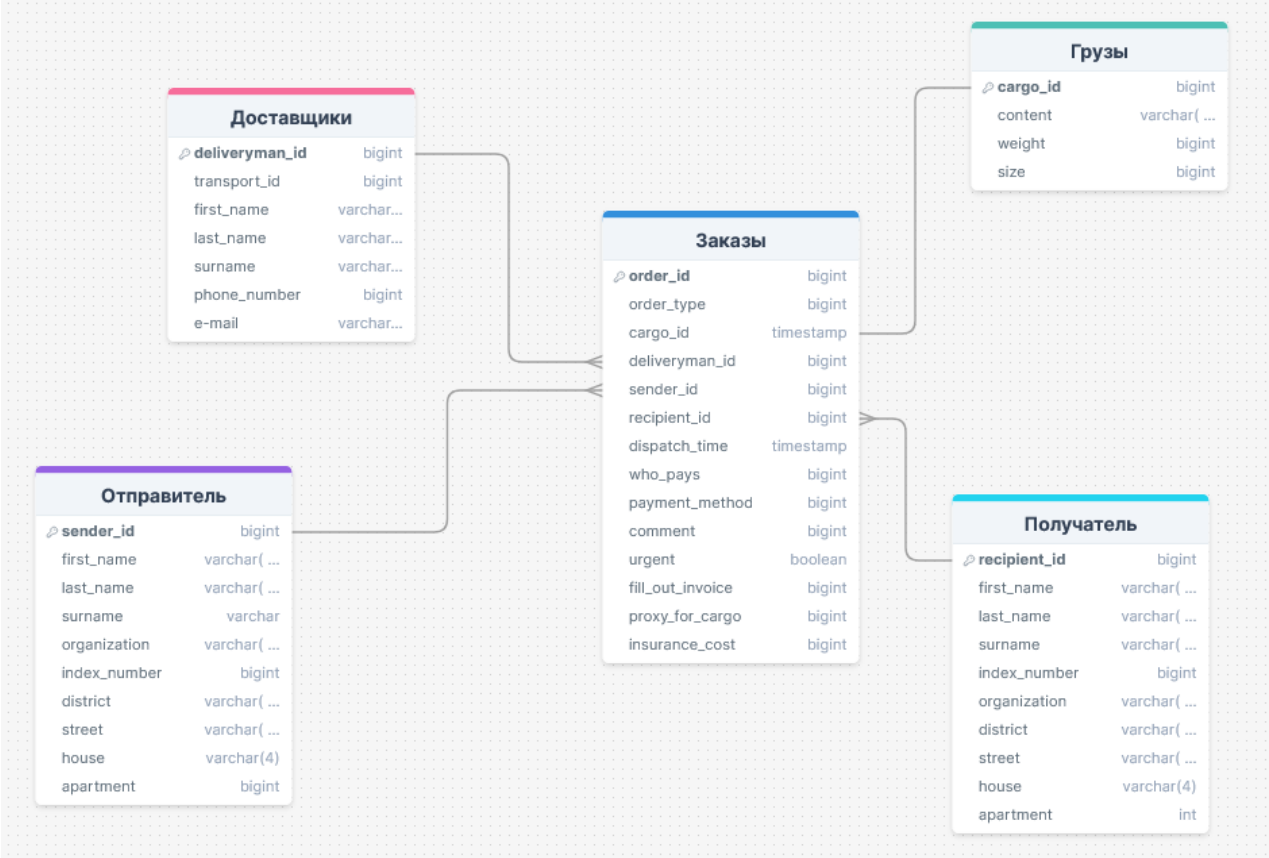

Рисунок 7 – Визуальное представление базы данных

2 ПРОЕКТНЫЙ РАЗДЕЛ

2.1 Создание файловой структуры и подключение фреймворков, билиотек для работы

Согласно схеме разделения данных MVC, файловая структура приложения должна быть разбита на разные компоненты, будь то каталоги, файлы конфигурации или исполняемые файлы. Информационная система состоит из двух каталогов, каждый из которых отвечает за одну из задач приложения:

1. Django – отвечает за серверную часть приложения. В каталоге было создано и активировано виртуальное окружение при помощи терминала. Создание и активация виртуального окружения представлены на рисунке 8.

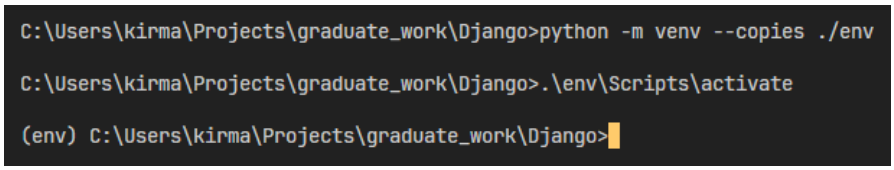

Рисунок 8 – Создание и активация виртуальной среды

Для инициализации фреймворка Django нужно создать проект, затем в каталоге проекта создать приложение. Инициализация Django, изображена на рисунке 9.

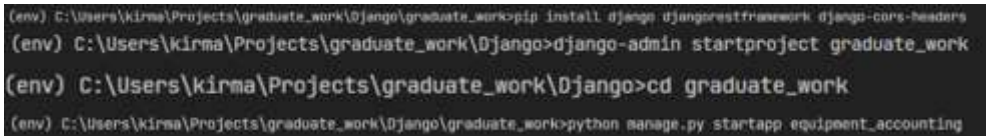

Рисунок 9 – Установка Django

Далее была установлена библиотека «psycorg2», предназначенная для взаимодействия с системой управления базами данных PostgreSQL. Установка библиотеки представлена на рисунке 10.

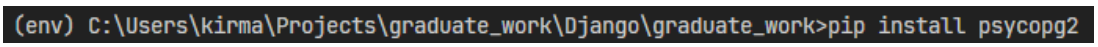

Рисунок 10 – Установка psycorg2

Модели в Django, обеспечивают доступ к хранилищу данных и делятся на несколько слоев. Для начала нужно определить модели информационной системы. В Python модели определяются как классы, которые будут инициализированы в базу данных. В классах создаются поля, влияющими на импорт данных в БД. Переменные в разрабатываемой информационной системе определяют такие свойства полей базы данных, как:

— максимальное количество символов;

— удобочитаемые имена полей и таблиц.

— поведение связанной таблицы при удалении данных из другой.

Код определения моделей изображен на рисунке 11.

```python
from django.db import models
from django.contrib.auth.models import User

class Contractor(models.Model):
    first_name = models.CharField(max_length=50, verbose_name='Имя')
    last_name = models.CharField(max_length=50, verbose_name='Фамилия')
    surname = models.CharField(max_length=50, verbose_name='Отчество')
    phone_number = models.CharField(max_length=11, verbose_name='Номер телефона', blank=True, null=True, default='')
    mail = models.EmailField(blank=True, null=True, default='', verbose_name='Почта')
    index_number = models.CharField(max_length=6, verbose_name='Индекс')
    district = models.CharField(max_length=50, verbose_name='Район')
    city = models.CharField(max_length=50, verbose_name='Город')
    street = models.CharField(max_length=50, verbose_name='Улица')
    house = models.CharField(max_length=4, verbose_name='Дом')
    apartment = models.CharField(max_length=4, verbose_name='Квартира', blank=True, null=True, default='')

    class Meta:
        abstract = True

class Deliveryman(models.Model):
    transport = models.ForeignKey("Transport", on_delete=models.CASCADE)
    first_name = models.CharField(max_length=50, verbose_name='Имя')
    last_name = models.CharField(max_length=50, verbose_name='Фамилия')
    surname = models.CharField(max_length=50, verbose_name='Отчество')
    phone_number = models.CharField(max_length=11, verbose_name='Номер телефона', blank=True, null=True, default='')
    mail = models.EmailField(verbose_name='Почта', blank=True, null=True, default = '')

    class Meta:
        verbose_name = 'Курьер'
        verbose_name_plural = 'Курьеры'

class Sender(Contractor):
    inn = models.PositiveIntegerField(verbose_name='ИНН', default = 000_000_000_000)
    class Meta:
        verbose_name = 'Отправитель'

class Recipient(Contractor):
    class Meta:
        verbose_name = 'Получатель'

class Cargo(models.Model):
    content = models.CharField(max_length=30, verbose_name='Содержимое')
    weight = models.IntegerField(verbose_name='Вес(кг)')
    size = models.CharField(max_length=12, verbose_name='Габариты(см)')

class Order(BaseDataModel):
    models.ForeignKey("Deliveryman", on_delete=models.PROTECT)
    sender = models.ForeignKey("Sender", on_delete=models.PROTECT)
    recipient = models.ForeignKey("Recipient", on_delete=models.PROTECT)
    cargo = models.OneToOneField("Cargo", on_delete=models.CASCADE)
    order_type = models.CharField(max_length=40, choices=ORDER_TYPES,  verbose_name='Тип')
    dispatch_time = models.DateField(verbose_name='Дата исполнения')
    comment = models.CharField(max_length=200, blank=True, null=True, default='', verbose_name='Комментарий')

    class Meta:
        verbose_name = 'Заказ'
        verbose_name_plural = 'Заказы'
        ordering = ['dispatch_time']
```

Рисунок 11 – Модели

Затем создаются миграции моделей и инициализируются в базу данных. Миграция моделей представлена на рисунке 12.

```
(env) C:\Users\kirma\Projects\graduate_work\Django\graduate_work>python manage.py makemigrations
Migrations for 'equipment_accounting':
  equipment_accounting\migrations\0001_initial.py
    - Create model Cargo
    - Create model Deliveryman
    - Create model Recipient
    - Create model Sender
    - Create model Order
(env) C:\Users\kirma\Projects\graduate_work\Django\graduate_work>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, equipment_accounting, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying equipment_accounting.0001_initial... OK
  Applying sessions.0001_initial... OK
```

Рисунок 12 – Создание миграций

Для обмена данных с фронтендом данные должны быть преобразованы в файлы формата JSON. Компонент, выполняющий это называется сериалайзер – они прописываются для моделей и преобразуют их, перед передачей в другой компонент – «представление». Код сериалайзеров изображен на рисунке 13.

```python
from rest_framework import serializers
from .models import Transport, Deliveryman, Sender, Recipient, Cargo, Order

class DelivermanSerializer(serializers.ModelSerializer):
    class Meta:
        model = Deliveryman
        fields = "__all__"
class SenderSerializer(serializers.ModelSerializer):
    class Meta:
        model = Sender
        fields = "__all__"

class RecipientSerializer(serializers.ModelSerializer):
    class Meta:
        model = Recipient
        fields = "__all__"

class CargoSerializer(serializers.ModelSerializer):
    class Meta:
        model = Cargo
        fields = "__all__"

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

Рисунок 13 – Сериалайзеры

Взаимодействие пользовательского интерфейса с программно-аппаратной частью производиться посредством запросов типа GET, POST, PUT и DELETE, отвечающих за соответствующие элементы управления ресурсами. Реализация взаимодействия представлена на рисунке 14.

```python
from rest_framework import generics, status
from rest_framework.decorators import api_view
from rest_framework.response import Response
from .serializers import *



@api_view(['GET', 'POST'])
def deliverymans_list(request):
    if request.method == 'GET':
        data = Deliveryman.objects.all()
        serializer = DelivermanSerializer(data, context={'request': request}, many=True)
        return Response(serializer.data)
    elif request.method == 'POST':
        print('post')
        serializer = DeliverySerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Рисунок 14 – Алгоритм взаимодействия

Далее идет распределение элементов представления по обработке запросов каждого URL-адреса, реализованное при помощи метода re_path, предоставляющий возможность задавать URL-адреса регулярными выражениями. Код определения маршрутов изображен на рисунке 15.

```python
from django.contrib import admin
from django.urls import path, re_path
from api import views
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    re_path(r'^api/order$', views.orders_list),
    re_path(r'^api/create_order/(\d+)$', views.create_order),
]
```

Рисунок 15 – Определение URL-адресов

2. Reactapp – каталог, отвечающий за визуализацию пользовательского интерфейса, в нем были установлены React и дополнительные библиотеки. Установка изображена на рисунке 16.

```
C:\Users\kirma\Projects\graduate_work6\reactapp>npx create-react-app reactapp
C:\Users\kirma\Projects\graduate_work6\reactapp>npm i axios reactstrap bootstrap
```

Рисунок 16 – Установка библиотек для фронтенд части приложения

Затем производиться связывания React и Django. В файле, отвечающем за развертывание React-приложения производится рендеринг маршрутов, посредством метода Route, а также создается шаблон для последующего представления в виде документа HTML. Алгоритм рендеринга изображен на рисунке 17.

```jsx
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';
import OrdersList from './components/OrdersList';
import CreateOrder from './components/CreateOrder';

function App() {
  return (
    <Router>
      <div className="App">
        <nav>
          <ul>
            <li><Link to="/orders">Заказы</Link></li>
            <li><Link to="/create-order">Создать заказ</Link></li>
          </ul>
        </nav>
        <Routes>
          <Route path="/orders" element={<OrdersList />} />
          <Route path="/create-order" element={<CreateOrder />} />
        </Routes>
      </div>
    </Router>
  );
}

export default App;
```

Рисунок 17 – Алгоритм рендеринга маршрутов

В конечном итоге общая файловая структура проекта удовлетворяет всем требованиям разрабатываемой веб-ориентированной информационной системы. Файловая структура приложения показана на рисунке 18.

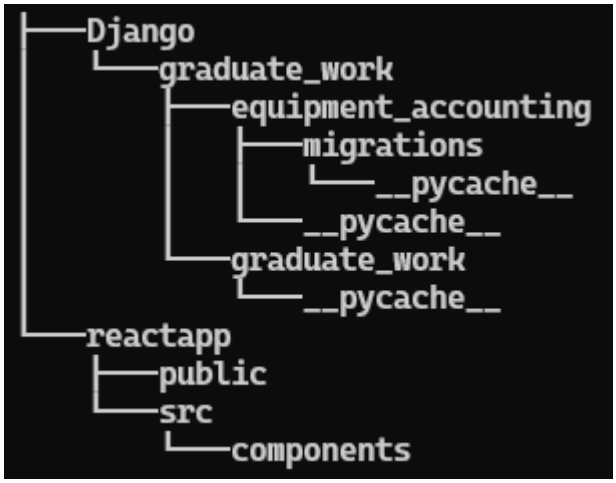

Рисунок 18 – Файловая структура приложения

2.2 Окна авторизации и регистрации

При разработке пользовательского интерфейса информационной системы, уделялось особое внимание удобству использования и интерактивности веб-приложения. Дизайн сайта выполнен в блочном стиле с использованием белого, черного, оранжевого цветов, а также был использован минималистичный стиль интерфейса, все это повысило удобство использования веб-приложения и позволило уменьшить вред для здоровья при длительном использовании, возникающего при использовании неприемлемых сочетаний цветов.

Заходя на сайт, пользователь переноситься в окно авторизации, содержащая следующие поля:

— «E-mail»;

— «Пароль».

А также следующие кнопки:

— «Войти»;

— «Зарегистрироваться».

Реализация окна авторизации изображена на рисунке 19.

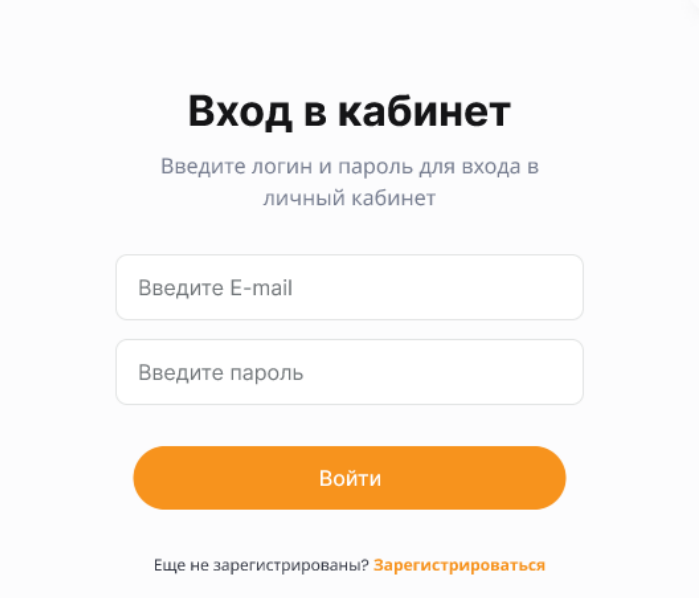

Рисунок 19 – Окно авторизации

Если у пользователя нет учетной записи, он может нажать на кнопку «Зарегистрироваться» и перенестись в окно регистрации, которое содержит следующие поля:

— «ФИО»;

— «E-mail»;

— «Номер телефона»;

— «Пароль»;

— «Подтверждение пароля».

Окно регистрации изображено на рисунке 20.

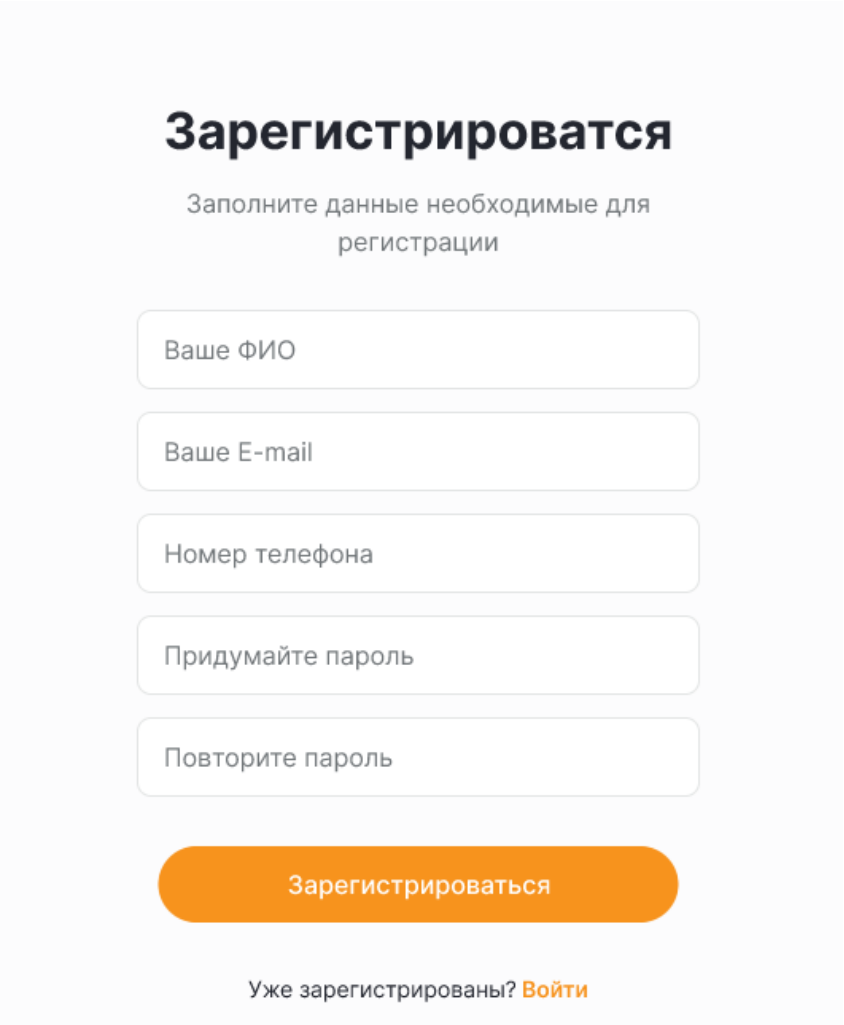

Рисунок 20 – Окно регистрации

Заполнив соответствующие поля окна регистрации и нажав кнопку «Зарегистрироваться», пользователь вноситься в базу данных как пользователь. Теперь пользователь может вернуться и авторизоваться, заполнив поля и нажав кнопу «Войти». Пример заполненных окон авторизации приведены на рисунках 21-22.

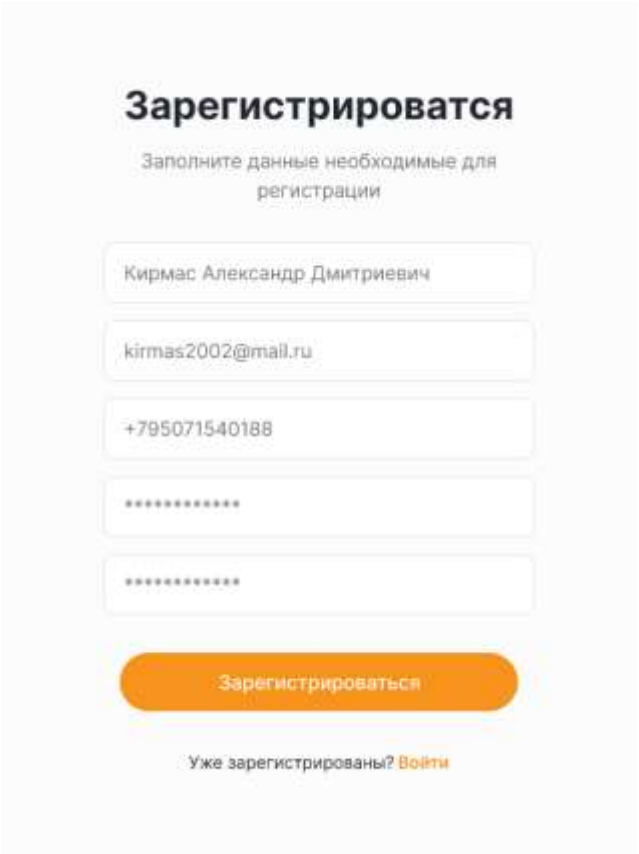

Рисунок 21 – Заполненное окно регистрации

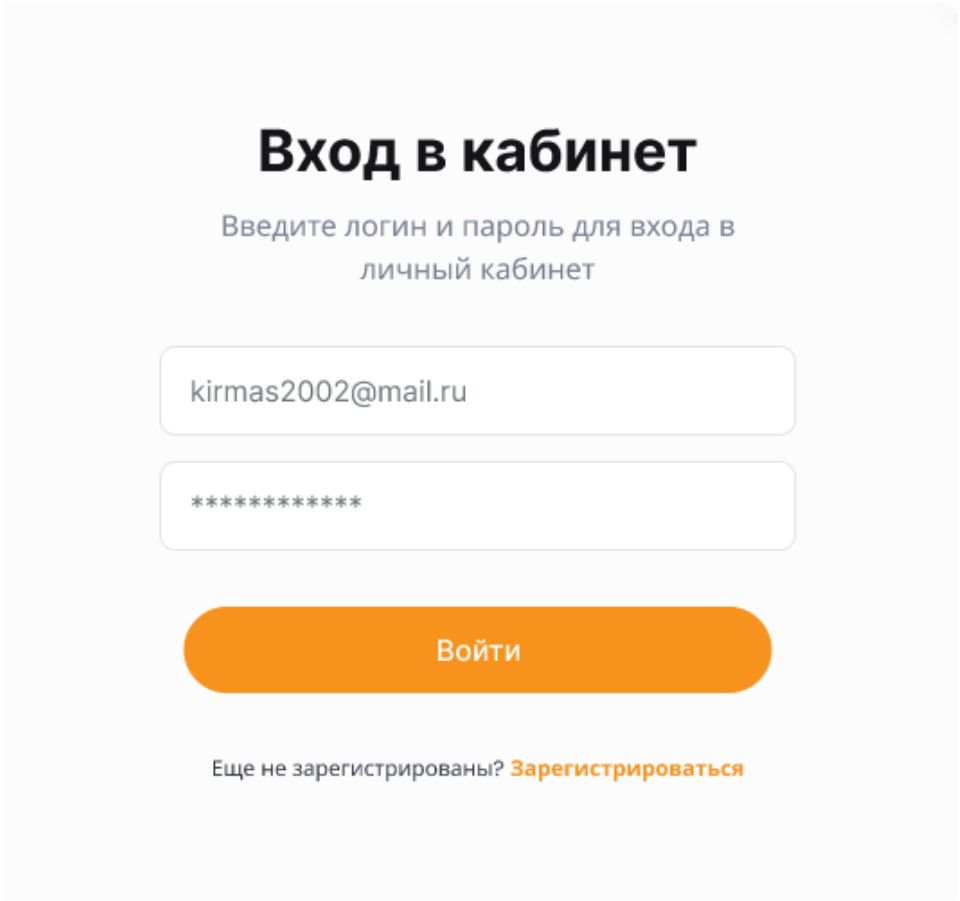

Рисунок 22 – Заполненное окно регистрации

2.3 Страница профиля

Далее пользователь переноситься на страницу профиля, со следующими полями:

— «ФИО»;

— «E-mail»;

— «Номер телефона»;

— «Пароль».

И следующими кнопками:

— «Оформить доставку»;

— «Профиль»;

— «История отправлений»;

— «Выйти»;

— «Посмотреть пароль»;

— «Обновить данные».

Страница профиля изображена на рисунке 23.

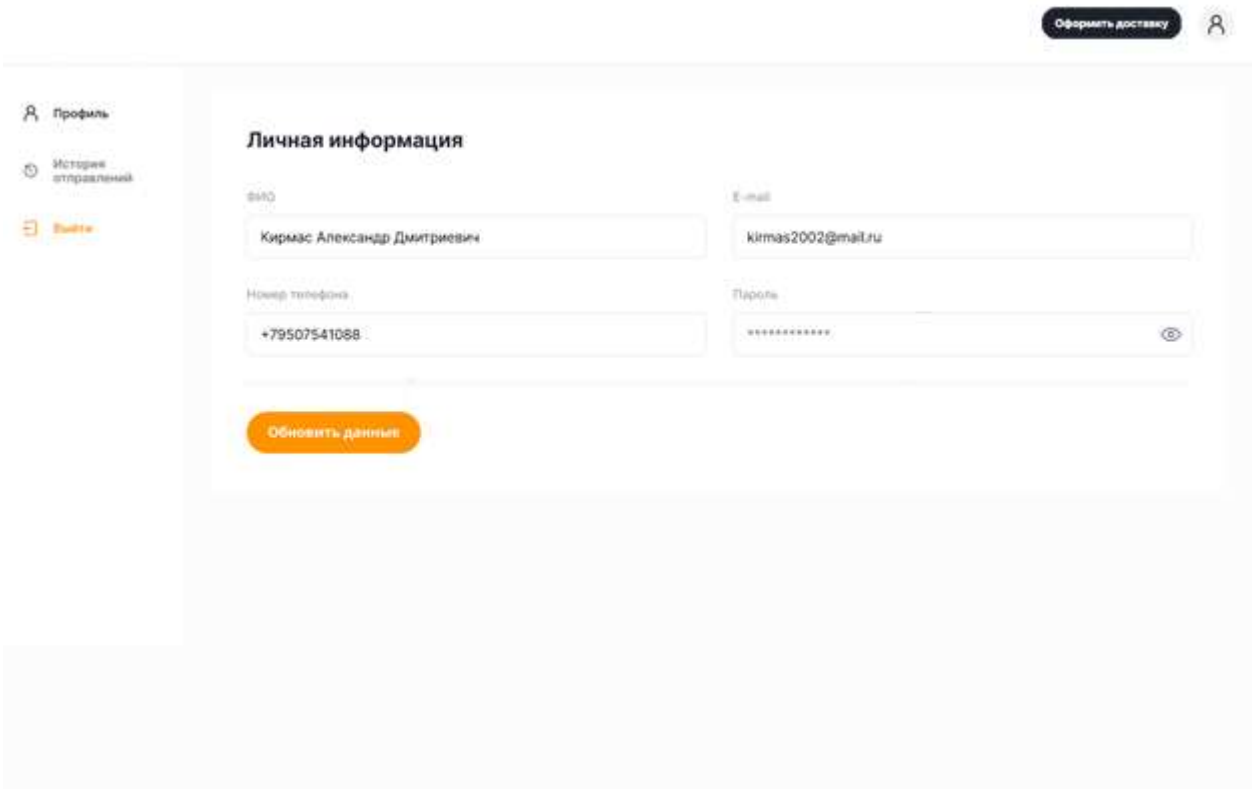

Рисунок 23 – Страница профиля

На этой странице, пользователь может нажать на соответствующие кнопки и перенестись на страницу, оформления доставки, истории отправлений, авторизации, выйдя при этом из учетной записи или обновить данные профиля. Также пользователь может нажать на кнопку находящуюся в поле «Пароль» и отобразить данные пароля в незашифрованном виде. Отображаемые данные при этом действии изображены на рисунках 24-25.

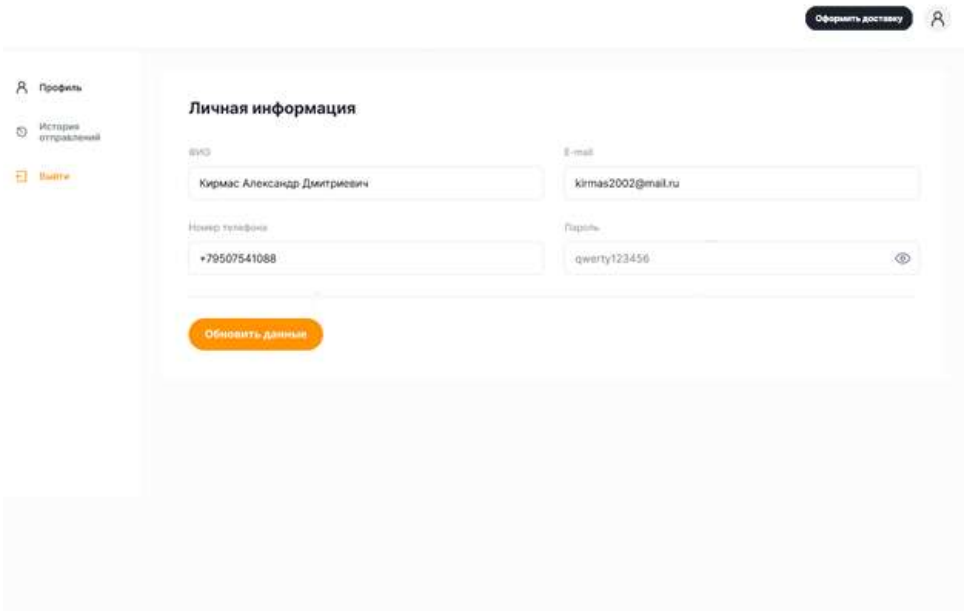

Рисунок 24 – Страница профиля с отображаемым паролем

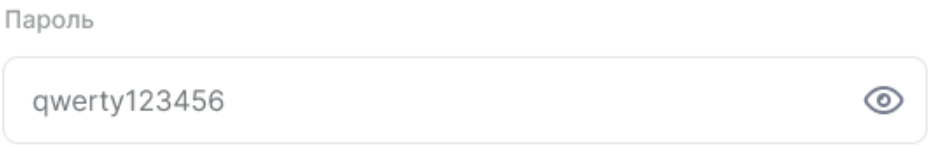

Рисунок 25 – Незашифрованный вид поля «Пароль»

Если пользователь хочет изменить данные профиля, то ему нужно ввести новые значения в поле или поля и нажать на кнопку «Обновить данные». Пример обновления данных изображен на рисунке 26.

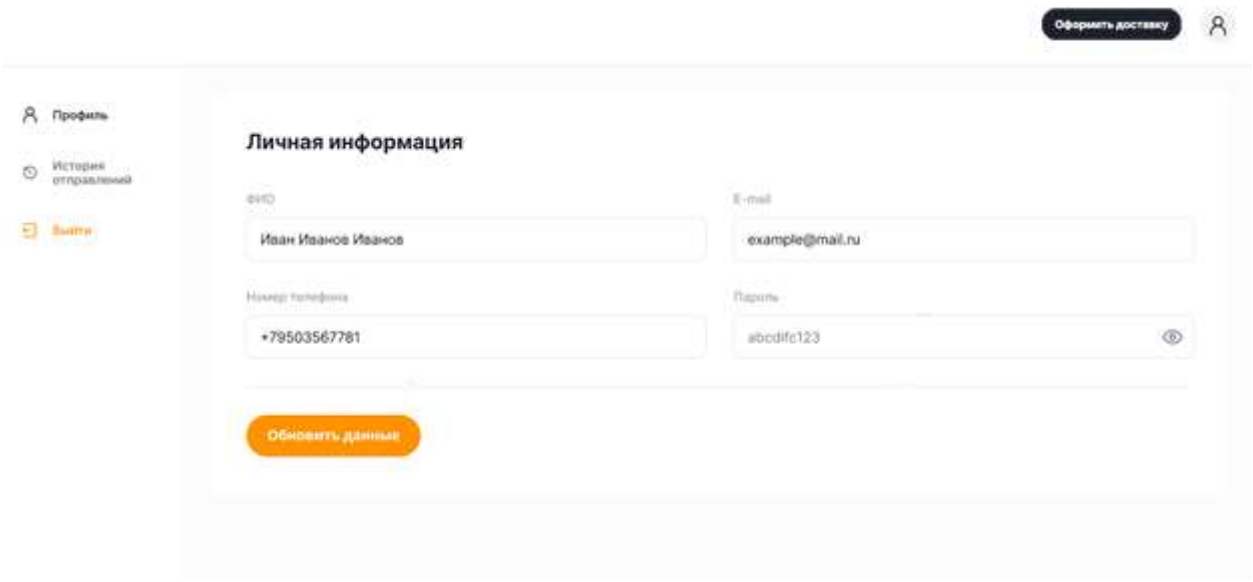

Рисунок 26 – Измененные данные профиля

2.4 Страница оформления доставки

Нажав на кнопку «Оформить доставку», пользователь переносится на страницу оформления доставки, содержащую следующие поля:

— «Тип доставки»;

— «Тип посылки»;

— «Вес»;

— «Длина»;

— «Ширина»;

— «Высота»;

— «Город» отправителя и получателя;

— «Адрес» отправителя и получателя;

— «Почтовый индекс» отравителя и получателя;

— «Дата отправки»;

— «ФИО» отправителя и получателя;

— «Телефон» отправителя и получателя;

— «Комментарий».

И такие кнопки как:

— «Вызов курьера в Воронеже»;

— «Забор груза из другого города»;

— «Доставка груза по Воронежу»;

— «Посылка»;

— «Документы»;

— «Отмена»;

— «Сохранить».

Страница оформления доставки изображена на рисунках 27-28.

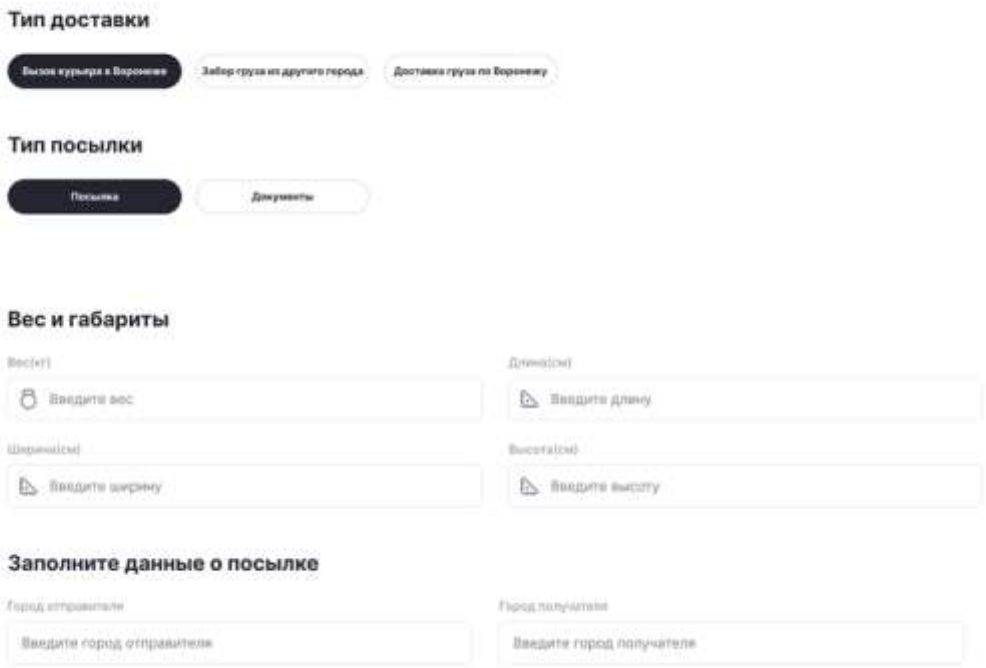

Рисунок 27 – Верхняя часть страницы оформления доставки

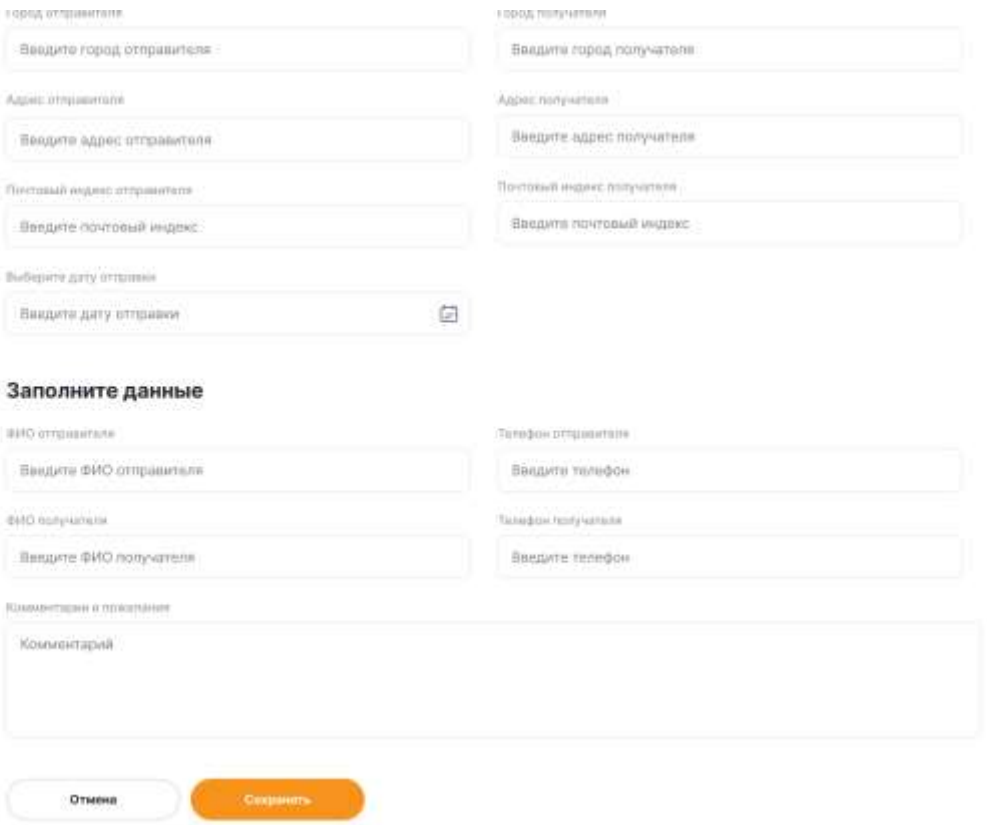

Рисунок 28 – Нижняя часть страницы оформления доставки

Пользователь может выбрать тип доставки и посылки, нажав на соответствующие кнопки, иначе они будут выбраны значением по умолчанию. Далее заполнив оставшиеся поля, пользователь может нажать на одну из кнопок, каждая из которых вернет пользователя на страницу профиля, но нажав на кнопку «Отмена» данные не сохраняться, а на кнопку «Сохранить» данные сохранятся в информационную систему.

2.5 Страница истории отправлений

Страница истории отправлений отображает заказы на доставку сохраненные в информационной системе в виде таблицы со следующими колонками и соответствующими данными под ними:

— «Номер заказа»;

— «Адрес отправления»;

— «Адрес доставки»;

— «Тип доставки»;

— «Дата отправки»;

— «Трек номер»;

— «ФИО доставщика».

Также у каждого элемента таблицы есть кнопка удаления. Страница истории отправлений с данными отображена на рисунке 29.

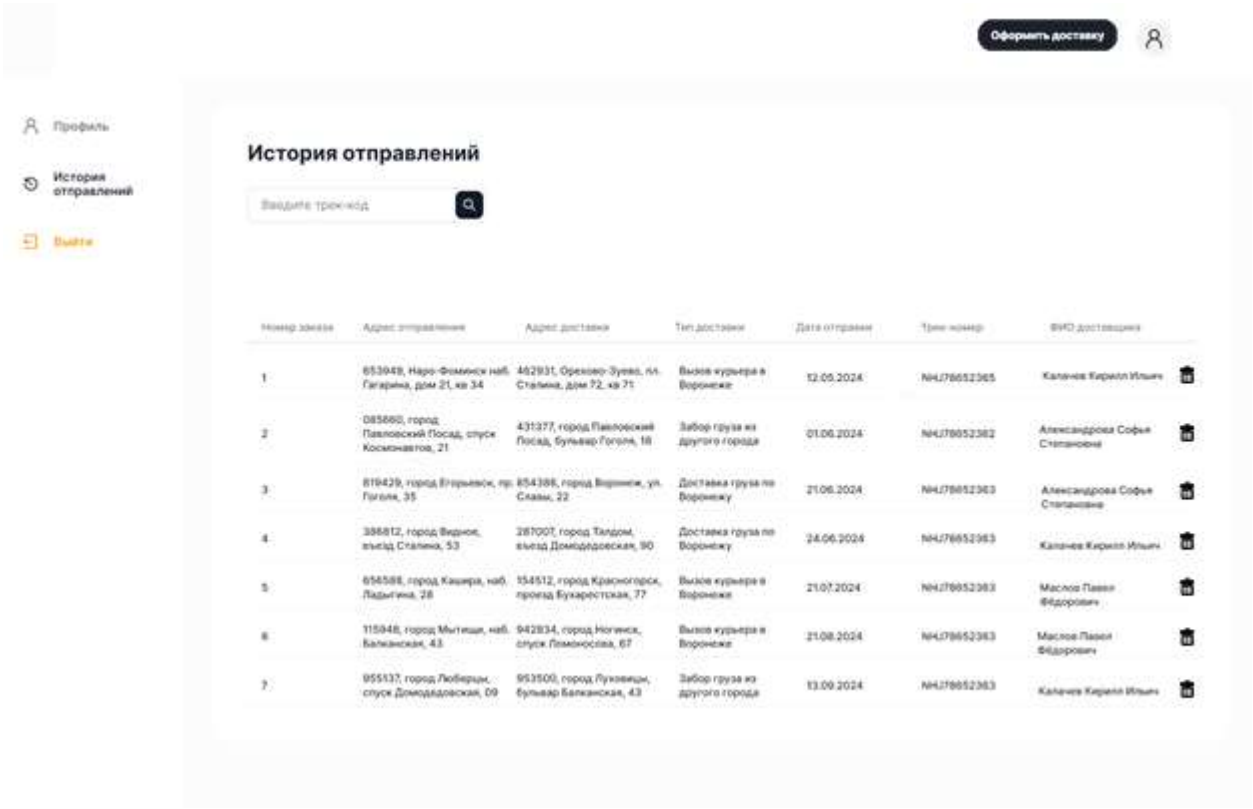

Рисунок 29 – Страница истории отправлений

Если пользователь хочет произвести поиск заказа, он может ввести трек-номер в соответствующее поле и нажав кнопку поиска, отобразить заказ с вписанным трек-номером. Пример поиска показан на рисунке 30.

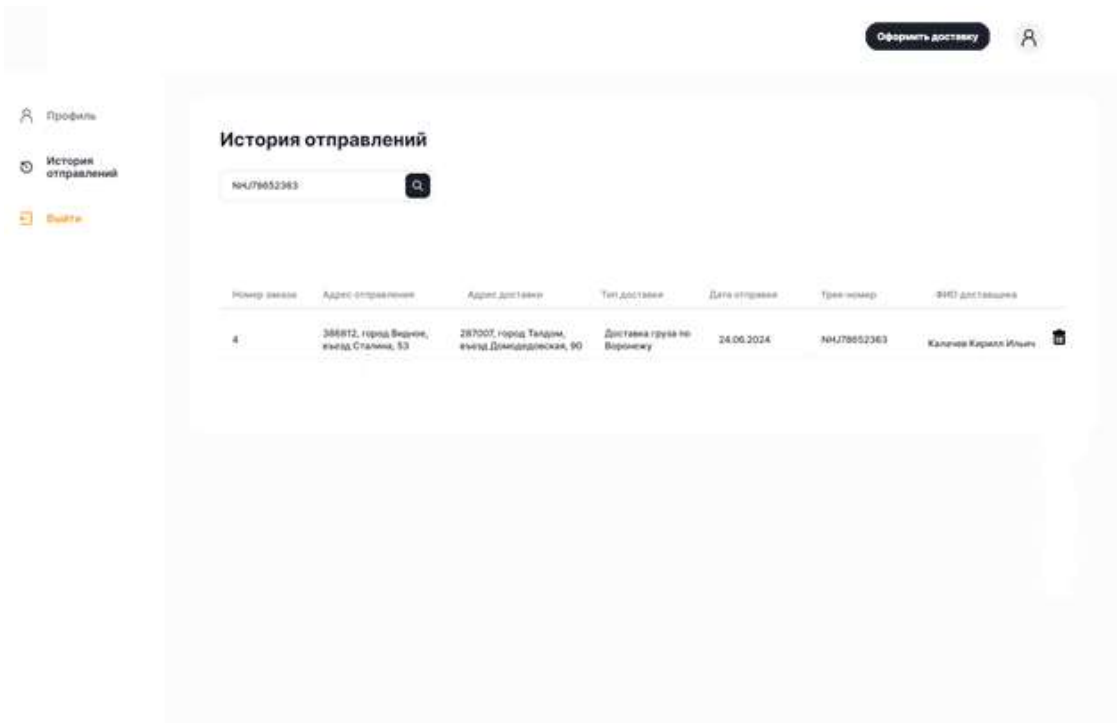

Рисунок 30 – Пример поиска по трек-номеру

Пользователь может удалить элемент таблицы, нажав на соответствующую кнопку. Пример страницы с удаленным полем изображен на рисунке 31.

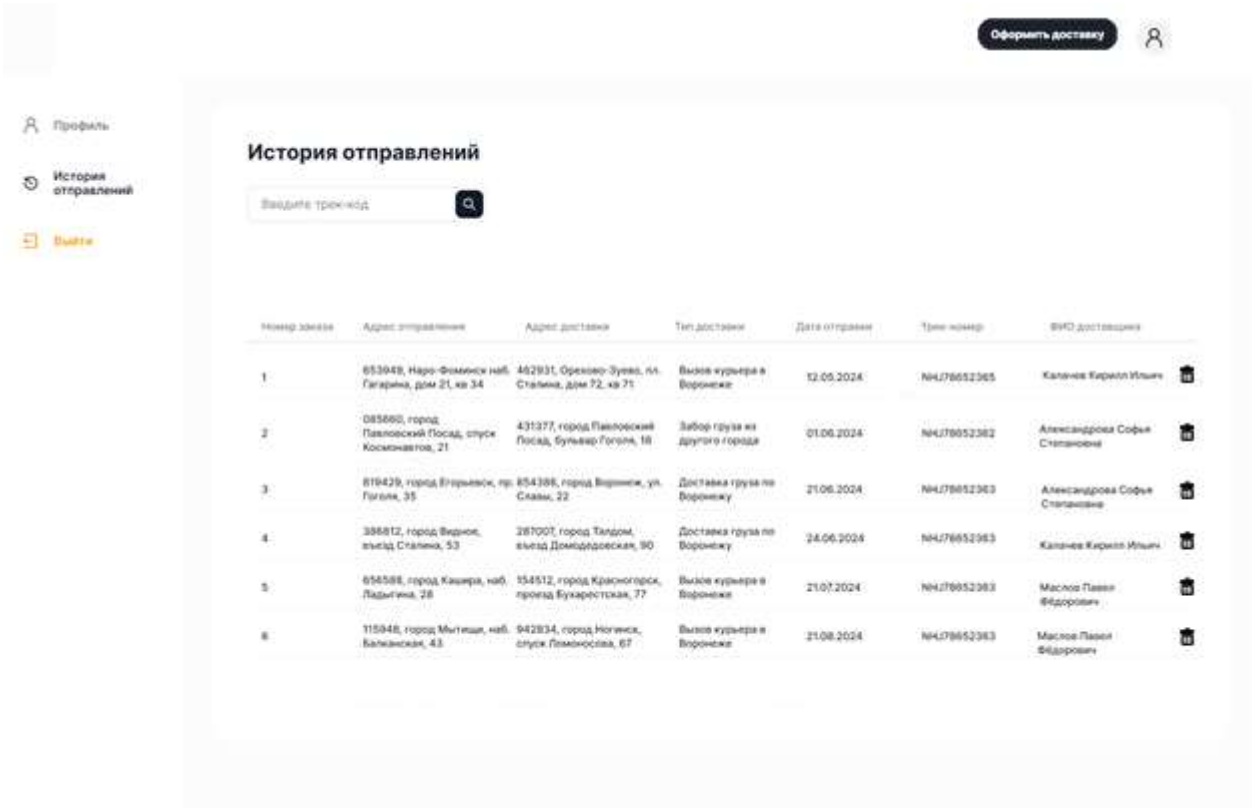

Рисунок 31 – Пример страницы с учетом удаленного поля

Заключение

В заключении выпускной квалификационной работы на тему «Разработка веб-ориентированной информационной системы учета оборудования транспортного предприятия» можно отметить, что цель исследования – разработка веб-ориентированной информационной системы учета оборудования транспортного предприятия была достигнута.

В ходе выполнения дипломной работы были выполнены все поставленные задачи, а именно:

— анализ предприятия и выявление недостатков;

— изучение основных технологий и инструментов для реализации учета оборудования;

— разработка и реализация веб-ориентированной информационной системы учета оборудования транспортного предприятия ООО «Бизнес-Курьер».

Разработанная веб-ориентированная информационная система позволяет эффективно производить учет данных оборудования.

Разработанная информационная система успешно внедрена и функционирует в компании ООО «Бизнес-Курьер». В дальнейшем планируется его техническое сопровождение.

В заключение, следует упомянуть, что разработанная веб-ориентированная ИС для учета оборудования обладает потенциалом для будущего развития и совершенствования.

Внедряя новые технологии, расширения функциональности и адаптации к изменяющимся требованиям рынка приложение будет сохранять свою актуальность и оставаться важным инструментом в области логистики.

Успешная реализация перспективных направлений развития обеспечит информационной системе конкурентные преимущества и способствует повышению эффективности при учете оборудования.

В итоге, завершение всех этапов проекта, позволило реализовать веб-приложение, соответствующее современным стандартам и требованиям. Разработанная информационная система будет мощным инструментом для учета данных транспортного предприятия.

Исследование методов и технологий разработки информационной системы учета оборудования показало, необходимость разработки интуитивно понятного интерфейса и автоматизации системы.

Разработанный пользовательский интерфейс информационной системы выполнен с использованием сочетаний цветов, позволяющих акцентировать внимание на основном контенте.

Исследовательские методы включают в себя анализ технической литературы, моделирование процессов и проектирование программного продукта. Для разработки веб-ориентированной информационной системы применяются современные технологии и инструменты программирования.

Список используемых источников

1. MDN Web Docs. [Электронный ресурс]. // Официальный сайт Django. // Режим доступа: https://developer.mozilla.org/ru/docs/Web/JavaScript

2. Next.js documentation [Электронный ресурс]. // Официальный сайт Django. // Режим доступа: https://nextjs.org/docs

3. Node.js v22.3.0 documentation [Электронный ресурс]. // Официальный сайт Django. // Режим доступа: https://nodejs.org/docs/latest/api/

4. Алексеев А. П. Введение в Web-дизайн. Учебное пособие. — М.: Солон-Пресс, 2021. — 184 с.

5. Диков А. В. Клиентские технологии веб-дизайна. HTML5 и CSS3. Учебное пособие для вузов. — М.: Лань, 2023. — 188 с.

6. Документация Django. [Электронный ресурс]. // Официальный сайт Django. // Режим доступа: https://docs.djangoproject.com/ru/

7. Документация Python. [Электронный ресурс]. // Официальный сайт Django. // Режим доступа: https://docs.python.org/3/

8. Документация React. [Электронный ресурс]. // Официальный сайт Django. // Режим доступа: https://ru.legacy.reactjs.org/

9. Жемчужников Д. Г. Web-дизайн. Уровень 1. — М.: Просвещение-Союз, 2023. — 144 с.

10. Жемчужников Д. Г. Web-дизайн. Уровень 2. — М.: Просвещение-Союз, 2023. — 144 с.

11. Кангин В. В. Интернет. Языки HTML и JavaScript. — М.: ТНТ, 2021. — 488 с.

12. Кириченко А. В. Справочник HTML. Кратко, быстро, под рукой. — М.: Наука и техника, 2023. — 288 с.

13. Нагаева И. А., Фролов А. Б., Кузнецов И. А. Основы web-дизайна. Методика проектирования. — М.: Директ-Медиа, 2021. — 184 с.

14. Немцова Т. И., Казанкова Т. В., Шнякин А. В. Компьютерная графика и web-дизайн. Учебное пособие. — М.: Форум, 2023. — 400 с.

15. Никулин В. В. Разработка серверной части веб-ресурса. Учебное пособие для вузов. — М.: Лань, 2023. — 132 с.

16. Полуэктова Н. Р. Разработка веб-приложений. — М.: Юрайт, 2024. — 205 с.

17. Современный учебник JavaScript. [Электронный ресурс]. // Официальный сайт Django. // Режим доступа: https://learn.javascript.ru/

18. Татро К., Макинтайр П. Создаем динамические веб-сайты на PHP. — СПб.: Питер, 2021. — 544 с.

19. Ткаченко О. Н., Капустина О. Г., Макарова Т. В.. Основы информационных технологий в рекламе. — М.: Юнити-Дана, 2022. — 271 с.

20. Тузовский А. Ф. Проектирование и разработка web-приложений. — М.: Юрайт, 2023. — 220 с.

21. Уэйншенк С. 100 главных принципов дизайна. 2-е издание. Как удержать внимание. — СПб.: Питер, 2021. — 265 с.

22. Хрусталев А. А. Дубовик Е. В. Справочник CSS3. Кратко, быстро, под рукой. — М.: Наука и техника, 2021. — 304 с.

Листинг программы

В файле views.py:

```
from rest_framework import viewsets
from .models import Deliveryman, Order
from .serializers import DeliverymanSerializer, OrderSerializer

class DeliverymanViewSet(viewsets.ModelViewSet):
    queryset = Deliveryman.objects.all()
    serializer_class = DeliverymanSerializer

class OrderViewSet(viewsets.ModelViewSet):
    queryset = Order.objects.all()
    serializer_class = OrderSerializer
```

В файле models.py:

```
from django.db import models

class Contractor(models.Model):
    first_name = models.CharField(max_length=50,
verbose_name='Имя')
    last_name = models.CharField(max_length=50,
verbose_name='Фамилия')
    surname = models.CharField(max_length=50,
verbose_name='Отчество')
    phone_number = models.CharField(max_length=11,
verbose_name='Номер телефона', blank=True, null=True, default='')
    mail = models.EmailField(blank=True, null=True, default='',
verbose_name='Почта')
    organization = models.CharField(max_length=50, blank=True,
null=True, default='', verbose_name='Организация')
    index_number = models.CharField(max_length=6,
verbose_name='Индекс')
    district = models.CharField(max_length=50,
verbose_name='Район')
    city = models.CharField(max_length=50, verbose_name='Город')
    street = models.CharField(max_length=50, verbose_name='Улица')
    house = models.CharField(max_length=4, verbose_name='Дом')
    apartment = models.CharField(max_length=4,
verbose_name='Квартира', blank=True, null=True, default='')

    class Meta:
        abstract = True

class Deliveryman(BaseDataModel):
    transport = models.ForeignKey("Transport",
on_delete=models.CASCADE)
```

```
    first_name = models.CharField(max_length=50,
verbose_name='Имя')
    last_name = models.CharField(max_length=50,
verbose_name='Фамилия')
    surname = models.CharField(max_length=50,
verbose_name='Отчество')
    phone_number = models.CharField(max_length=11,
verbose_name='Номер телефона', blank=True, null=True, default='')
    mail = models.EmailField(verbose_name='Почта', blank=True,
null=True, default = '')

    class Meta:
        db_table = 'deliverymans'
        verbose_name = 'Курьер'
        verbose_name_plural = 'Курьеры'

class Sender(Contractor):
    inn = models.PositiveIntegerField(verbose_name='ИНН', default
= 000_000_000_000)
    class Meta:
        db_table = 'senders'
        verbose_name = 'Отправитель'

class Recipient(Contractor):
    class Meta:
        db_table = 'recipients'
        verbose_name = 'Получатель'

class Cargo(models.Model):
    content = models.CharField(max_length=30,
verbose_name='Содержимое')
    weight = models.IntegerField(verbose_name='Вес(кг)')
    size = models.CharField(max_length=12,
verbose_name='Габариты(см)')

    class Meta:
        db_table = 'cargos'
        verbose_name = 'Груз'
        verbose_name_plural = 'Грузы'

class Order(BaseDataModel):
    sender = models.ForeignKey("Sender", on_delete=models.PROTECT)
    recipient = models.ForeignKey("Recipient",
on_delete=models.PROTECT)
    deliveryman = models.ForeignKey("Deliveryman",
on_delete=models.PROTECT)
    cargo = models.ForeignKey("Cargo", on_delete=models.PROTECT)
    order_type = models.CharField(max_length=40,
verbose_name='Тип')
    dispatch_time = models.DateField(verbose_name='Дата
исполнения')
```

```
    who_pays = models.CharField(max_length=9,
verbose_name='Оплатит')
    payment_method = models.CharField(max_length=11,
verbose_name='Форма оплаты')
    comment = models.CharField(max_length=200, blank=True,
null=True, default='', verbose_name='Комментарий')
    urgent = models.BooleanField(default=False,
verbose_name='Срочность')
    fill_out_invoice = models.CharField(max_length=9,
verbose_name='Заполнит накладную')
    proxy_for_cargo = models.BooleanField(default=False,
verbose_name='Доверенность на забор груза')
    insurance_cost = models.FloatField(verbose_name='Стоимость
страховки')

    class Meta:
        db_table = 'orders'
        verbose_name = 'Заказ'
        verbose_name_plural = 'Заказы'
        ordering = ['dispatch_time']
```

В файле serializer.py:

```
from rest_framework import serializers
from .models import BaseDataModel, Contractor, Transport,
Deliveryman, Sender, Recipient, Cargo, Order

class BaseDataModelSerializer(serializers.ModelSerializer):
    class Meta:
        model = BaseDataModel
        fields = ['date_created', 'date_updated', 'to_remove']

class ContractorSerializer(serializers.ModelSerializer):
    class Meta:
        model = Contractor
        fields = ['first_name', 'last_name', 'surname',
'phone_number', 'mail', 'organization', 'index_number',
'district', 'city', 'street', 'house', 'apartment']

class DeliverymanSerializer(BaseDataModelSerializer):
    transport = TransportSerializer()

    class Meta(BaseDataModelSerializer.Meta):
        model = Deliveryman
        fields = BaseDataModelSerializer.Meta.fields +
['transport', 'first_name', 'last_name', 'surname',
'phone_number', 'mail']

    def create(self, validated_data):
        transport_data = validated_data.pop('transport')
        transport = Transport.objects.create(**transport_data)
```

```
        deliveryman =
Deliveryman.objects.create(transport=transport, **validated_data)
        return deliveryman

    def update(self, instance, validated_data):
        transport_data = validated_data.pop('transport')
        transport = instance.transport

        instance.first_name = validated_data.get('first_name',
instance.first_name)
        instance.last_name = validated_data.get('last_name',
instance.last_name)
        instance.surname = validated_data.get('surname',
instance.surname)
        instance.phone_number = validated_data.get('phone_number',
instance.phone_number)
        instance.mail = validated_data.get('mail', instance.mail)
        instance.save()

        transport.car_type = transport_data.get('car_type',
transport.car_type)
        transport.car_brand = transport_data.get('car_brand',
transport.car_brand)
        transport.car_model = transport_data.get('car_model',
transport.car_model)
        transport.car_color = transport_data.get('car_color',
transport.car_color)
        transport.car_number = transport_data.get('car_number',
transport.car_number)
        transport.car_number_region =
transport_data.get('car_number_region',
transport.car_number_region)
        transport.save()

        return instance

class SenderSerializer(ContractorSerializer):
    class Meta(ContractorSerializer.Meta):
        model = Sender
        fields = ContractorSerializer.Meta.fields + ['inn']

class RecipientSerializer(ContractorSerializer):
    class Meta(ContractorSerializer.Meta):
        model = Recipient

class CargoSerializer(serializers.ModelSerializer):
    class Meta:
        model = Cargo
        fields = ['content', 'weight', 'size', 'number_of_seats',
'invoices_quantity', 'envelopes_quantity', 'packages_quantity']
```

```
class OrderSerializer(BaseDataModelSerializer):
    sender = SenderSerializer()
    recipient = RecipientSerializer()
    deliveryman = DeliverymanSerializer()
    cargo = CargoSerializer()

    class Meta(BaseDataModelSerializer.Meta):
        model = Order
        fields = BaseDataModelSerializer.Meta.fields + ['sender',
'recipient', 'deliveryman', 'cargo', 'order_type',
'dispatch_time', 'who_pays', 'payment_method', 'comment',
'urgent', 'fill_out_invoice', 'proxy_for_cargo', 'insurance_cost']

    def create(self, validated_data):
        sender_data = validated_data.pop('sender')
        recipient_data = validated_data.pop('recipient')
        deliveryman_data = validated_data.pop('deliveryman')
        cargo_data = validated_data.pop('cargo')

        sender = Sender.objects.create(**sender_data)
        recipient = Recipient.objects.create(**recipient_data)
        deliveryman =
Deliveryman.objects.create(**deliveryman_data)
        cargo = Cargo.objects.create(**cargo_data)

        order = Order.objects.create(sender=sender,
recipient=recipient, deliveryman=deliveryman, cargo=cargo,
**validated_data)
        return order

    def update(self, instance, validated_data):
        sender_data = validated_data.pop('sender')
        recipient_data = validated_data.pop('recipient')
        deliveryman_data = validated_data.pop('deliveryman')
        cargo_data = validated_data.pop('cargo')

        sender = instance.sender
        recipient = instance.recipient
        deliveryman = instance.deliveryman
        cargo = instance.cargo

        instance.order_type = validated_data.get('order_type',
instance.order_type)
        instance.dispatch_time =
validated_data.get('dispatch_time', instance.dispatch_time)
        instance.who_pays = validated_data.get('who_pays',
instance.who_pays)
        instance.payment_method =
validated_data.get('payment_method', instance.payment_method)
        instance.comment = validated_data.get('comment',
instance.comment)
```

```
        instance.urgent = validated_data.get('urgent',
instance.urgent)
        instance.fill_out_invoice =
validated_data.get('fill_out_invoice', instance.fill_out_invoice)
        instance.proxy_for_cargo =
validated_data.get('proxy_for_cargo', instance.proxy_for_cargo)
        instance.insurance_cost =
validated_data.get('insurance_cost', instance.insurance_cost)
        instance.save()

        return instance
```

В файле urls.py:

```
from django.contrib import admin
from django.urls import path, include
from equipment_accounting.views import DeliverymanViewSet,
OrderViewSet
from django.conf import settings
from django.conf.urls.static import static
from rest_framework.routers import DefaultRouter

router = DefaultRouter()
router.register(r'deliveryman', DeliverymanViewSet)
router.register(r'orders', OrderViewSet)

urlpatterns = [
    path('', include(router.urls)),
]
```

```
index.js:
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
reportWebVitals();
```

```
App.js:
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Link } from
'react-router-dom';
import DeliverymanList from './components/DeliverymanList';
import OrdersList from './components/OrdersList';
import CreateOrder from './components/CreateOrder';
```

```
function App() {
  return (
    <Router>
      <div className="App">
        <nav>
          <ul>
            <li><Link to="/deliveryman">Курьеры</Link></li>
            <li><Link to="/orders">Заказы</Link></li>
            <li><Link to="/create-order">Создать заказ</Link></li>
          </ul>
        </nav>
        <Routes>
          <Route path="/deliveryman" element={<DeliverymanList
/>}/>
          <Route path="/orders" element={<OrdersList />} />
          <Route path="/create-order" element={<CreateOrder />} />
        </Routes>
      </div>
    </Router>
  );
}

export default App;
```

OrderList.js:

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useTable } from 'react-table';

const OrderList = () => {
  const [orders, setOrders] = useState([]);

  useEffect(() => {
    const fetchOrders = async () => {
      try {
        const response = await
axios.get('http://localhost:8000/api/orders/');
        setOrders(response.data);
      } catch (error) {
        console.error('Ошибка при загрузке заказов:', error);
      }
    };

    fetchOrders();
  }, []);

  const {
    getTableProps,
    getTableBodyProps,
    headerGroups,
```

```
    rows,
    prepareRow,
  } = useTable({ columns, data: orders });

  return (
    <div>
      <h1>История отправлений</h1>
      <table {...getTableProps()}>
        <thead>
          {headerGroups.map(headerGroup => (
            <tr {...headerGroup.getHeaderGroupProps()}>
              {headerGroup.headers.map(column => (
                <th
{...column.getHeaderProps()}>{column.render('Header')}</th>
              ))}
            </tr>
          ))}
        </thead>
        <tbody {...getTableBodyProps()}>
          {rows.map(row => {
            prepareRow(row);
            return (
              <tr {...row.getRowProps()}>
                {row.cells.map(cell => {
                  return <td
{...cell.getCellProps()}>{cell.render('Cell')}</td>;
                })}
              </tr>
            );
          })}
        </tbody>
      </table>
    </div>
  );
};

export default OrderList;
```

Header.js:

```
const Header = () => {
    return (
        <div className="text-center">
            <img
                src="https://cdn.worldvectorlogo.com/logos/react-
2.svg"
                width="100"
                className="img-thumbnail"
                style={{marginTop: "20px"}}
                alt="logo"
            />
            <hr/>
```

```
            <h1>App for project on React + Django</h1>
        </div>)
}

export default Header;

CreateOrder:
import React, { useState } from 'react';
import axios from 'axios';

function CreateOrder() {
    const [order, setOrder] = useState({
        order_type: '',
        dispatch_time: '',

    });

    const handleChange = (event) => {
        setOrder({ ...order, [event.target.name]:
event.target.value });
    };

    const handleSubmit = (event) => {
        event.preventDefault();
        axios.post('/api/orders/', order)
            .then(response => {
                console.log('Заказ успешно создан:',
response.data);
            })
            .catch(error => {
                console.error('Ошибка при создании заказа:',
error);
            });
    };

    return (
        <form onSubmit={handleSubmit}>
            <label>
                Тип заказа:
                <input type="text" name="order_type"
onChange={handleChange} />
            </label>
            <label>
                Дата исполнения:
                <input type="date" name="dispatch_time"
onChange={handleChange} />
            </label>
            <button type="submit">Создать заказ</button>
        </form>
    );
}
```

```
export default CreateOrder;
```