Реферат

Утв.

Консульт.

Пров.

Разраб.

*Пацей Н.В.*

*Коренькова А.А.*

*Демьянов В.Р.*

*Парамонов А.И.*

Ф.И.О.

Подпись

Дата

1

*Реферат*

Лист

1

*БГТУ 00.00.ПЗ*

Лит.

Листов

*74417027, 2023*

Abstract

Н. контр.

Утв.

Н. контр.

Пров.

Разраб.

*Пацей Н.В.*

*Коренькова А.А.*

*Демьянов В.Р.*

*Парамонов А.И.*

Ф.И.О.

Подпись

Дата

1

*Abstract*

Лист

1

*БГТУ 00.00.ПЗ*

Лит.

Листов

*74417027, 2023*

Консульт.

**Содержание**

[Введение 7](#_Toc135954982)

[1 Аналитический обзор литературы и постановка задачи 8](#_Toc135954983)

[1.1 Бизнес-процессы склада 8](#_Toc135954984)

[1.2 Размещение и хранение товара на складе 9](#_Toc135954985)

[1.3 Программное средство NetSuite WMS 10](#_Toc135954986)

[1.4 Программное средство Sortly 11](#_Toc135954987)

[1.5 Программное средство Maropost 12](#_Toc135954988)

[1.6 Патентный поиск 13](#_Toc135954989)

[1.7 Вывод по разделу 14](#_Toc135954990)

[2 Проектирование веб-приложения 16](#_Toc135954991)

[2.1 Проектирование базы данных 16](#_Toc135954992)

[2.2 Архитектура веб-приложения 22](#_Toc135954993)

[2.3 Архитектура серверной части веб-приложения 23](#_Toc135954994)

[2.4 Проектирование REST API 24](#_Toc135954995)

[2.5 Архитектура клиентской части веб-приложения 27](#_Toc135954996)

[2.6 Проектирование пользовательского интерфейса 28](#_Toc135954997)

[2.6 Вывод по разделу 28](#_Toc135954998)

[3 Разработка веб-приложения 29](#_Toc135954999)

[3.1 Серверная часть приложения 29](#_Toc135955000)

[3.1.1 Проект WMS.Database 29](#_Toc135955001)

[3.1.2 Проект WMS.Core 33](#_Toc135955002)

[3.1.3 Проект WMS.WebApi 36](#_Toc135955003)

[3.1.4 Проект WMS.Database.Migrations 39](#_Toc135955004)

[3.2 Клиентская часть приложения 39](#_Toc135955005)

[3.2.1 Структура модуля 40](#_Toc135955006)

[3.2.2 AppModule 41](#_Toc135955007)

[3.2.3 SharedModule 41](#_Toc135955008)

[3.2.4 DomainModule 42](#_Toc135955009)

[3.2.5 RoutingModule 43](#_Toc135955010)

[3.2.6 CoreModule 44](#_Toc135955011)

[3.2.7 Angular компоненты 44](#_Toc135955012)

[3.2.8 Angular сервисы 47](#_Toc135955013)

[3.3 Вывод по разделу 48](#_Toc135955014)

[4 Тестирование веб-приложения 49](#_Toc135955015)

[4.1 Тестирование *REST API* 49](#_Toc135955016)

[4.1.1 Тестирование аутентификации *REST API* 49](#_Toc135955017)

[4.1.2 Тестирование авторизации *REST API* 52](#_Toc135955018)

[4.1.3 Тестирование добавления стеллажей в зону склада 53](#_Toc135955019)

[4.2 Тестирование пользовательского интерфейса 57](#_Toc135955020)

[4.2.1 Тестирование сообщений об ошибках при заполнении полей форм 57](#_Toc135955021)

[4.2.2 Тестирование всплывающих сообщений для ошибок 57](#_Toc135955022)

[4.3 Вывод по разделу 58](#_Toc135955023)

[5 Методика установки и использования 59](#_Toc135955024)

[6 Технико-экономическое обоснование проекта 60](#_Toc135955025)

[6.1 Общая характеристика разрабатываемого программного средства 60](#_Toc135955026)

[6.2 Исходные данные для проведения расчётов и маркетинговый анализ 60](#_Toc135955027)

[6.3 Обоснование цены программного средства 61](#_Toc135955028)

[6.3.1 Расчет затрат рабочего времени на разработку программного средства 62](#_Toc135955029)

[6.3.2 Расчёт основной заработной платы 62](#_Toc135955030)

[6.3.3 Расчёт дополнительной заработной платы 63](#_Toc135955031)

[6.3.4 Расчёт отчислений в Фонд социальной защиты населения и по обязательному страхованию 63](#_Toc135955032)

[6.3.5 Расчёт суммы прочих прямых затрат 64](#_Toc135955033)

[6.3.6 Расчёт суммы накладных расходов 64](#_Toc135955034)

[6.3.7 Сумма расходов на разработку программного средства 64](#_Toc135955035)

[6.3.8 Расходы на сопровождение и адаптацию 65](#_Toc135955036)

[6.3.9 Расчёт полной себестоимости 65](#_Toc135955037)

[6.3.10 Определение цены, оценка эффективности 65](#_Toc135955038)

[6.4 Вывод по разделу 66](#_Toc135955039)

[Заключение 68](#_Toc135955040)

[Список использованных источников 69](#_Toc135955041)

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Демьянов В.Р.*

.

Пров.

*Парамонов А.И.*

Н. контр.

*Коренькова А.А.*

Утв.

*Пацей Н.В.*

Содержание

Лит.

Листов

1

*74417027, 2023*

Введение

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Демьянов В.Р.*

Пров.

*Парамонов А.И.*

Н. контр.

*Коренькова А.А.*

Утв.

*Пацей Н.В.*

Введение

Лит.

Листов

1

*74417027, 2023*

Управление складом – это множество процессов, таких как определение местоположения, приёмка, размещение, перемещение, отгрузка, инвентаризация товаров, контролирование работы сотрудников склада и многих других.

Большинство таких процессов являются рутинными, имеют определённый алгоритм действий. Чем больше склад, тем больше времени, а соответственно денег, они затрачивают. Современные технологии позволяют автоматизировать и оптимизировать такие процессы. Так как сегодня почти у каждого есть персональный компьютер или смартфон, где по умолчанию установлен веб-браузер, то хорошим и доступным решением является веб-приложение.

Цель дипломного проекта состоит в том, чтобы разработать веб-приложение, которое позволит, используя популярные веб-браузеры, осуществлять управление складом со стеллажным способом укладки.

Для достижения поставленной цели были сформулированы следующие задачи:

* провести анализ информационных источников;
* выбрать информационные технологии (языки программирования, платформы, алгоритмы, методы) для разработки;
* рассчитать экономические показатели;
* спроектировать структуру базы данных;
* спроектировать архитектуру веб-приложения;
* разработать REST API, который будет включать методы доступа для: товаров, пользователей, задач, комментариев, аутентификации, клиентов, словарей, адресов;
* разработать сервис для отправки уведомлений об изменении статуса задачи, регистрации пользователя, приближающейся просрочки выполнения задач;
* разработать регулярно запускаемые задачи для удаления отгруженных товаров со склада, срок хранения истории о которых истёк; уведомления о просроченных задачах;
* продумать и разработать дизайн пользовательского интерфейса;
* разработать клиентское приложение, которое будет использовать REST API, которое будет включать модули для: товаров; задач; словарей; адресов; панели администратора для управления пользователями и клиентами; аутентификации;
* протестировать веб-приложение;
* написать руководство пользователя;
* написать руководство по развёртыванию.

Консульт.

1 Аналитический обзор литературы и постановка задачи

ФИО

Подпись

Дата

Лист

1

БГТУ 01.00.ПЗ

Разраб.

*Демьянов В.Р.*

Пров.

*Парамонов А.И.*

Н. контр.

*Коренькова А.А.*

Утв.

*Пацей Н.В.*

1 Обзор литературы и аналогов

Лит.

Листов

8

*74417027, 2023*

## **1.1 Бизнес-процессы склада**

Бизнес-процессы склада – это регламентированные операции, которые должны выполняться в четкой последовательности. Эффективный процесс управления складом возможен только при условии перехода от административной системы управления, зависящей от человеческого фактора, к автоматизированной, лишенной этого недостатка [1].

Управление складом включает в себя множество бизнес-процессов, в зависимости от классификации склада их количество и логика может отличаться, но общими для всех можно выделить следующие: приёмка на хранение, перемещение, отгрузка, получение информации, инвентаризация товаров; управление персоналом; управление структурой хранения склада.

Приемка товара на хранение включает в себя: разгрузку автомобиля с товаром, приемку, подписание с клиентом акта приема-передачи товара на хранение, перемещение товара в зону постоянного хранения.

Перемещение товара между местами хранения необходимо для оптимизации используемого складского пространства, а также для сокращения времени подборки товара.

Процесс отгрузки товара включает: подбор товара, перемещение товара в зону погрузочно-разгрузочных работ, погрузку в автомобиль, подписание с клиентом акта приема-передачи.

Инвентаризация необходима для сверки наличия и количества товаров на складе с данными, хранящимися в системе.

Режим получения информации позволяет узнать текущее состояние любого объекта на складе.

Управление персоналом важный процесс, который позволяет снизить текущие (операционные) расходы на содержание склада и повысить эффективность работы всего предприятия. Система ставит задачи персоналу и контролирует их выполнение. Измерение ключевых показателей эффективности работы склада повышает эффективность работы, позволяет измерять показатели эффективности, проверять выполнение и результативность работы, формировать форму отчётности, настраивать систему мотивации и нормы оплаты труда.

Управление структурой хранения склада позволяет оптимизировать используемое складское пространство.

## **1.2 Размещение и хранение товара на складе**

Метод размещения товаров на складе выбирают в зависимости от задач, назначения грузов, выбранного способа хранения, необходимости максимального использования объема склада при рациональном расположении секций, предохранения товаров от повреждения, доступности любой ячейки склада для механизмов на крупных складах, быстрого нахождения необходимых товаров [2].

На складе все имеет свое место и только на этом месте должно находиться. В идеале служащий должен находить товар по названию и адресу, даже не зная его внешнего вида.

Независимо от того, крупный склад или небольшой, обязательно следует ввести единую адресную систему размещения товаров, иначе неизбежны потери товаров, пересортица и убытки. Это важно для обеспечения увеличения оборота, исключения ошибок в размещении товаров и быстрого нахождения их даже новыми сотрудниками после короткого инструктажа. Каждому месту хранения присваивается код (адрес), обозначающий номер зоны хранения, номер стеллажа (штабеля), номер вертикальной секции и номер полки. Адрес может иметь 4-5 и более знаков. Программным путем обеспечивают автоматическое указание адресов в ярлыках, чеках, спецификациях наличия, ведомостях инвентаризации. Ведомости инвентаризации и комплектовочные листы для отбора товаров печатают с сортировкой по адресам.

На складе существуют разные виды укладки товаров. Для затаренных и штучных товаров обычно применяют штабельный и стеллажный способы укладки.

Для хранения товаров, затаренных в мешки, кипы, кули, ящики, бочки применяют штабельную укладку.

Формируя штабель, обеспечивают его устойчивость, допустимую высоту и свободный доступ к товарам. Высота штабеля определяется свойствами товара и его упаковки, возможностями штабелера, предельной нагрузкой на 1 кв. м пола, высотой склада.

Штабельная укладка применяется в трех вариантах: прямая, в перекрестную клетку, в обратную клетку.

При прямой укладке, чаще применяемой для штабелирования ящиков и бочек одинакового размера, каждый ящик ставится строго и ровно на ящик в нижнем ряду. Повышение устойчивости штабеля обеспечивает прямая пирамидальная укладка – в каждом верхнем ряду на одно место меньше, и каждое верхнее место устанавливается на два нижних.

В перекрестную клетку укладывают ящики различных размеров. При этом верхние ящики укладывают поперек нижних.

В обратную клетку укладывают, как правило, товары, затаренные в мешки, – верхний ряд мешков размещают на нижнем в обратном порядке.

При укладке товаров в штабеля следят за тем, чтобы в складе обеспечивались нормальная циркуляция воздуха, санитарные и противопожарные требования – штабеля размещают не ближе 0,5 м от стен и 1,5 м от отопительных приборов. Между штабелями оставляют проходы шириной около 1,5 м.

Штабельное хранение товаров, уложенных на стоечные и ящичные поддоны, позволяет рациональнее использовать помещения и применять механизмы.

При стеллажном способе хранения товары на поддонах, распакованные товары, а также товары в индивидуальной упаковке укладывают в ячейки стеллажей.

Стеллажное хранение товаров на поддонах весьма удобно – при помощи штабелеров поддоны укладывают на полках, расположенных на любой доступной механизмам высоте. На нижних полках можно хранить товары, отбор которых выполняют вручную, на верхних – товары, отгружаемые целиком на поддоне.

## **1.3 Программное средство NetSuite WMS**

Одним лучших примеров системы управления складом является продукт от Oracle – NetSuite WMS [3] (рисунок 1.1).

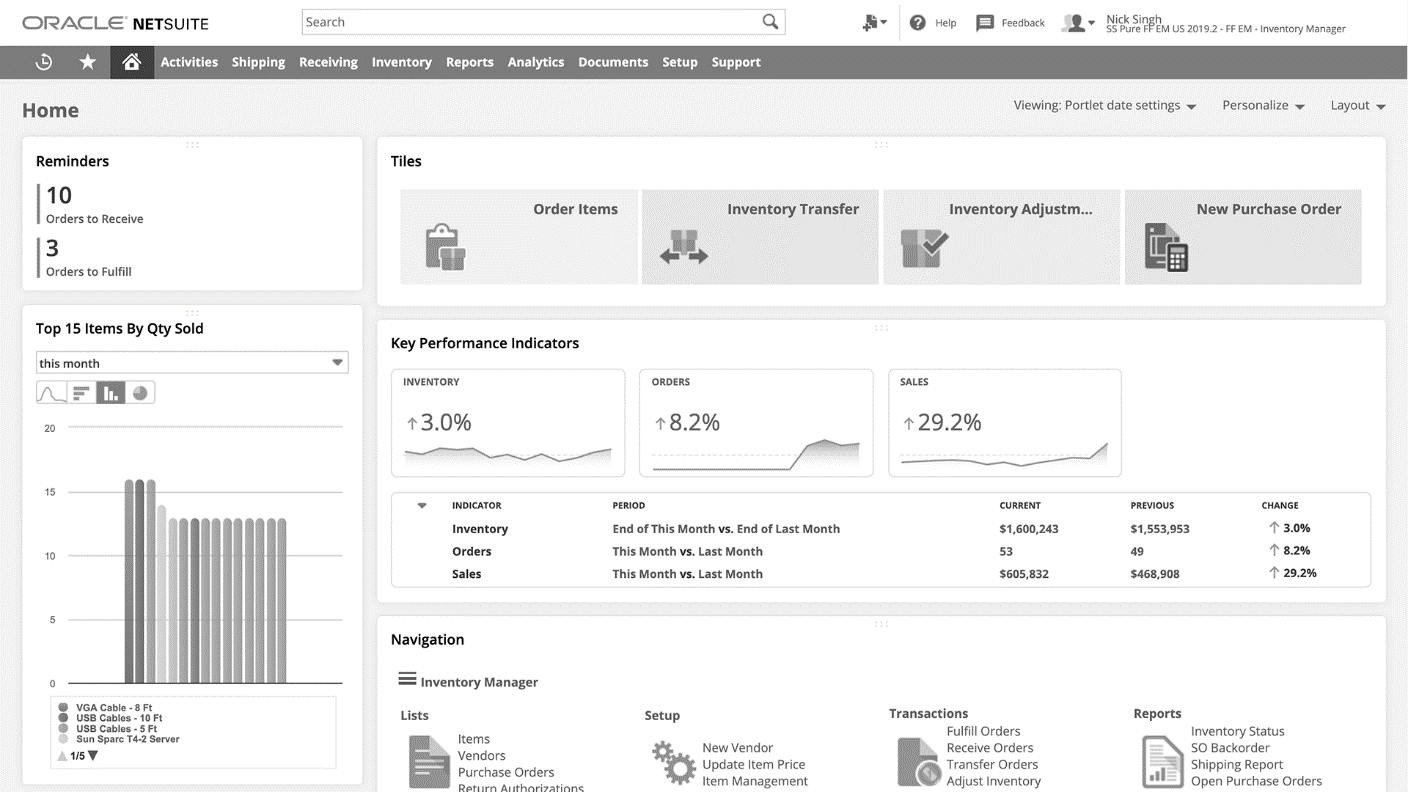


Рисунок 1.1 – Программное средство NetSuite WMS

Информационная система помогает оптимизировать складские и производственные операции. Это помогает пользователям выполнять основные функции склада, такие как получение, хранение и отгрузка товаров.

Особенности инструмента:

* управление задачами;
* мобильное сканирование штрих-кода;
* квитанция об авторизации;
* определение стратегии складывания и сбора;
* данные в реальном времени, управление заказами, отслеживание штрих-кода, выставление счетов, управление рабочей силой и нагрузкой и т.д.

Плюсы:

* простой и удобный интерфейс;
* надёжная компания (Oracle);
* сторонние плагины;
* беспроводное складирование;

К минусам можно отнести высокую цену и неэффективность при использовании в качестве ERP.

## **1.4 Программное средство Sortly**

Sortly [4], несомненно, является одним из лучших визуальных программ для управления складом на основе фотографий для малого бизнеса. С этим программным обеспечением вы получаете полностью настраиваемую систему, которая может отслеживать любой предмет или относящуюся к нему информацию в разных местах.

Пользователи Sortly могут добавлять настраиваемые поля, квитанции и несколько изображений к каждому элементу, чтобы упростить отслеживание активов, независимо от того, на каком складе находится товар. Пользователи также могут создавать и печатать QR-этикетки и штрих-коды, которые можно сканировать с помощью Sortly.

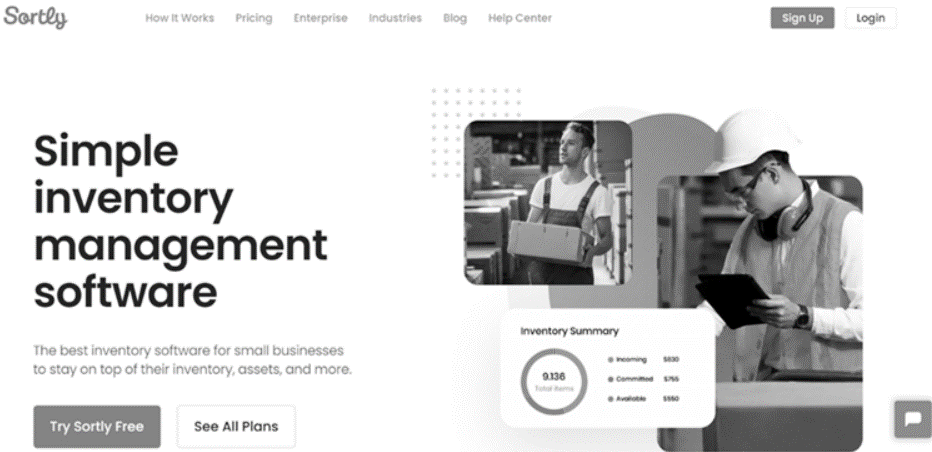


Рисунок 3.2 – Программное средство Sortly

Особенности инструмента:

* сканирование и обновление товаров с помощью QR-меток и штрих-кодов;
* создание автоматического оповещения для отслеживания уровня запасов;
* назначение ролей пользователей и управление правами доступа;
* отслеживание инвентаря и активности пользователей;
* создание собственных отчётов в формате CSV и PDF.

Плюсы:

* прост в использовании, никаких технических знаний не требуется;
* кроссплатформенное программное обеспечение, которое работает практически на всех типах устройств;
* чистая и всеобъемлющая пользовательская панель;
* легко создаваемые персонализированные QR-метки и штрих-коды;
* доступна бесплатная версия.

Минусы:

* нет интеграций;
* будет предоставляться только справочный центр и поддержка по электронной почте, если не купить индивидуальный план.

## **1.5 Программное средство Maropost**

Maropost [5] – это платформа, которая может не только автоматизировать процесс управления запасами, но и упростить управление вашими заказами. Платформа отличается от своих конкурентов тем, что упрощает централизованное управление несколькими интернет-магазинами.

Проще говоря, вы можете положиться на эту платформу для управления несколькими магазинами с единой панели управления. По сути, вы видите всю свою империю онлайн-рынка с высоты птичьего полета. Вы можете легко управлять всеми списками продуктов, заказами и информацией, относящейся к клиентам, с единой интуитивно понятной платформы.

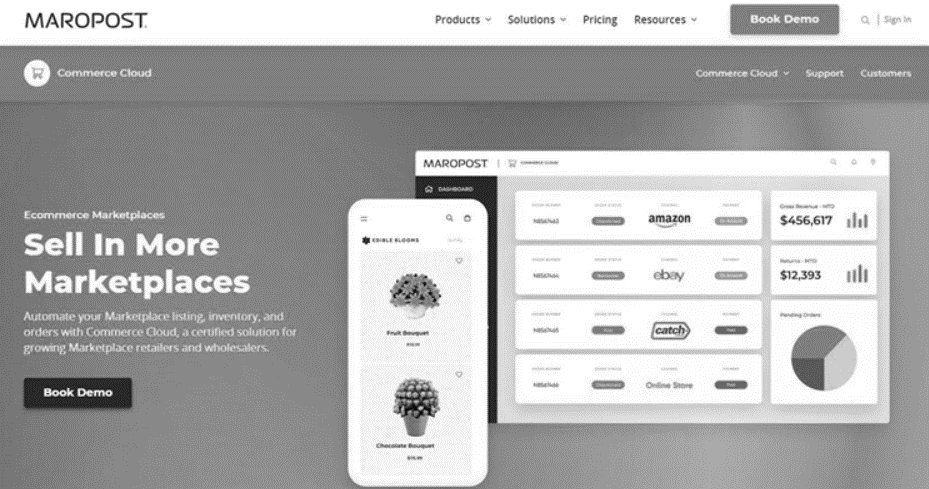


Рисунок 3.3 – Программное средство Maropost

Особенности инструмента:

* управление несколькими каналами продаж из одного месте;
* создание индивидуального интернет-магазина;
* оптимизация управления клиентами;
* установка правил ценообразования;
* синхронизация данных отслеживания доставки.

Плюсы:

* удобный интерфейс;
* централизованное решение для нескольких функций, связанных с электронной коммерцией;
* подробный список каналов и инвентаря;
* гибкое ценообразование.

Из минусов программное средство может не подходить для малого бизнеса.

## **1.6 Патентный поиск**

При проведении дипломного проекта основной целью проведения патентного поиска является определение уровня развития технологии и новизны существующего технического решения.

Процесс поиска определяет, как решалась поставленная задача до появления технических решений, защищенных авторскими свидетельствами, патентами и каковы перспективы их развития.

В результате патентного поиска были обнаружены следующие патенты, представленные в таблицах ниже 1.1, 1.2.

Таблица 1.1 – Описание патента №1

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Номер регистрации | Дата публикации | Правообладатель |
| Система управления складом APWMS | 2017660864 | 28.09.2017 | ООО «Айпитон» |

Программное средство [6] отличается возможностью гибкой настройки контура управления товародвижением; графическим редактором топологий; гибкой системой отчётности (полный контроль за производительностью сотрудников, утилизации склада и всех операций, совершённых в WMS); модулем печати отчётов и этикеток из MS Reporting Services.

Технологии программирования: Transact-SQL, PHP, HTML, JavaScript, C#.

Таблица 1.2 – Описание патента №2

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Номер регистрации | Дата публикации | Правообладатель |
| wmsMLC Pro | 2014616326 | 20.07.2014 | ООО «Информационные технологии МОЛКОМ» |

Программа [7] состоит из трех компонентов: серверная, клиентская и терминальная части. К серверной части программы относятся следующие приложения: Сервис обслуживания клиентов (SDCL); Сервис экспорта и печати (EPS); Сервис расписаний (RS); Сервисы интеграции (SI, API).

Технологии программирования: .Net 4.0, C#.

Таблица 1.3 – Описание патента №3

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Номер регистрации | Дата публикации | Правообладатель |
| LEAD WMS FX | 2022680694 | 03.11.2022 | Блинов Дмитрий Валерьевич |

Система [8] обеспечивает автоматизацию управления складами и складскими комплексами. Обеспечивает поддержку процессов от планирования прибытия транспорта до регистрации доставки в точки отгрузки, включая приёмку, размещение, хранение, пополнение, отбор, комплектацию, консолидацию, инвентаризацию, отгрузку и других. Имеет высокую производительность и гибкую систему конфигурирования посредством правил.

Технологии программирования: C# (.NET 6), C++, JavaScript, PG/SQL.

Таблица 1.4 – Описание патента №4

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Номер регистрации | Дата публикации | Правообладатель |
| AXELOT WCS | 2020661464 | 24.08.2020 | ООО «Акселот-Л» |

Программа предназначена для управления автоматизированными линиями, состоящими из различного автоматического оборудования. Поддержка оборудования разных классов: конвейеры, паллетайзеры, краны-штабеллеры. Планирование технологического процесса по заданию WMS: движения объектов по технологическим линиям, загрузки оборудования и балансировка. Мониторинг тех. процессов и оповещение WMS о ходе исполнения и статусах задач, объектов и технологических линий. Управление технологическим процессом: выдача команд на исполнение, отслеживание движения объектов, состояния оборудования, обработка коллизий, предотвращения авариных ситуаций.

Технологии программирования: C#.

## **1.7 Вывод по разделу**

В результате анализа литературы, исследования существующих аналогов создаваемого программного средства для управления складом и патентного поиска были определены наиболее востребованные функции, плюсы и минусы подобных систем.

По полученным результатам исследования сформированы требования к программному средству:

* программное средство должно представлять собой веб-приложение;
* серверная часть должна быть разработана с использованием ASP.NET;
* должен быть асинхронный UI с использованием Angular;
* данные должны храниться в SQL базе данных;
* приложение должно быть развёрнуто на Docker;
* необходимо рассчитать экономические показатели;
* приложение должно поддерживать авторизацию на основе ролей: администратор, работник, аудитор;
* приложение должно поддерживать функции отгрузки и добавления товаров;
* система должна сама удалять отгруженные товары по истечению срока хранения таких товаров, который можно задать в настройках;
* должна быть возможность поиска товара в системе по названию или с использованием фильтров по адресу;
* приложение должно предоставлять функции управления пользователями;
* пользователь должен иметь возможность работать с задачами;
* система должна уведомлять пользователей о регистрации, просроченных задачах за несколько дней и изменении статуса задач;
* приложение должно предоставлять информацию об адресной системе (количество свободных/занятых мест, структура адресной системы и т.д.);
* пользовательский интерфейс должен быть удобным, быстрым и логически завершённым;
* приложение должно быть протестировано;
* должно быть подробное руководство пользователя и руководство по развёртыванию.

Консульт.

2 Проектирование веб-приложения

ФИО

Подпись

Лист

1

БГТУ 02.00.ПЗ

Разраб.

*Демьянов В.Р.*

Провер.

*Парамонов А.И.*

Н. контр.

*Коренькова А.А.*

Утверд.

*Пацей Н.В.*

2 Проектирование веб-приложения

Лит.

Листов

13

*74417027, 2023*

**2.1 Проектирование базы данных**

Для хранения данных в приложении будет использоваться реляционная база данных под управлением СУБД Microsoft SQL Server.

Логическая схема спроектированной базы данных представлена на рисунке 2.1.

Консульт.

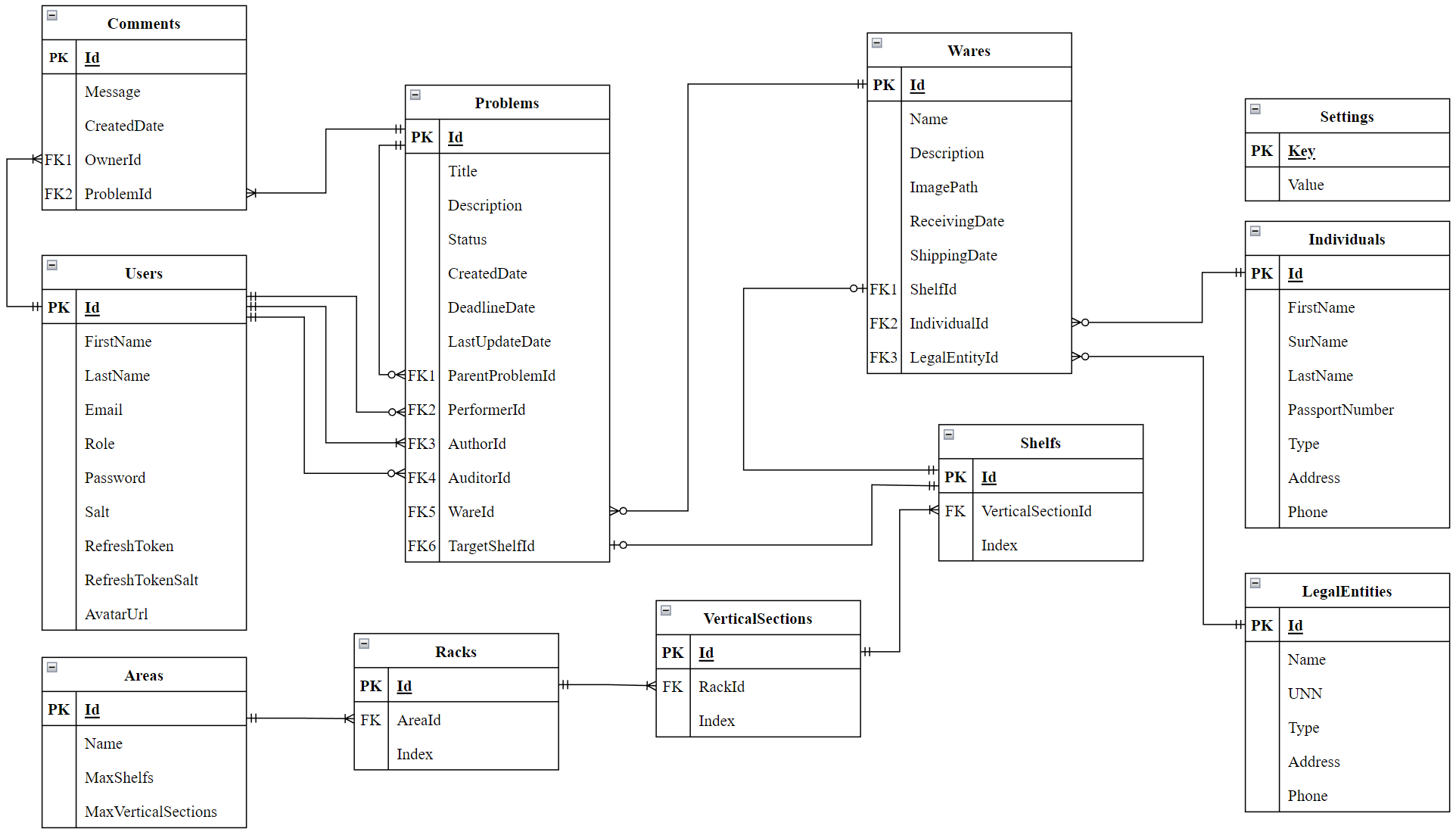


Рисунок 2.1 – Диаграмма базы данных

База данных состоит из 12 основных таблиц (бизнес сущностей), таблицы содержащей историю миграций и 11 таблиц Hangfire.

Для регулярного запуска задач, например, проверка просроченных задач склада или удаление отгруженных товаров, будет использоваться Hangfire.

Hangfire – многопоточный и масштабируемый планировщик задач с открытым исходным кодом, построенный по клиент-серверной архитектуре на стеке технологий .NET (в первую очередь Task Parallel Library и Reflection). Hangfire осуществляет промежуточное хранение задач и другой информации, связанной с обработкой, в базе данных. Суть его работы заключается в том, что процесс-клиент добавляет задачу в базу данных, процесс-сервер периодически опрашивает базу данных и выполняет задачи.

Несколько таблиц, которые использует Hangfire, представлены на рисунке 2.2.

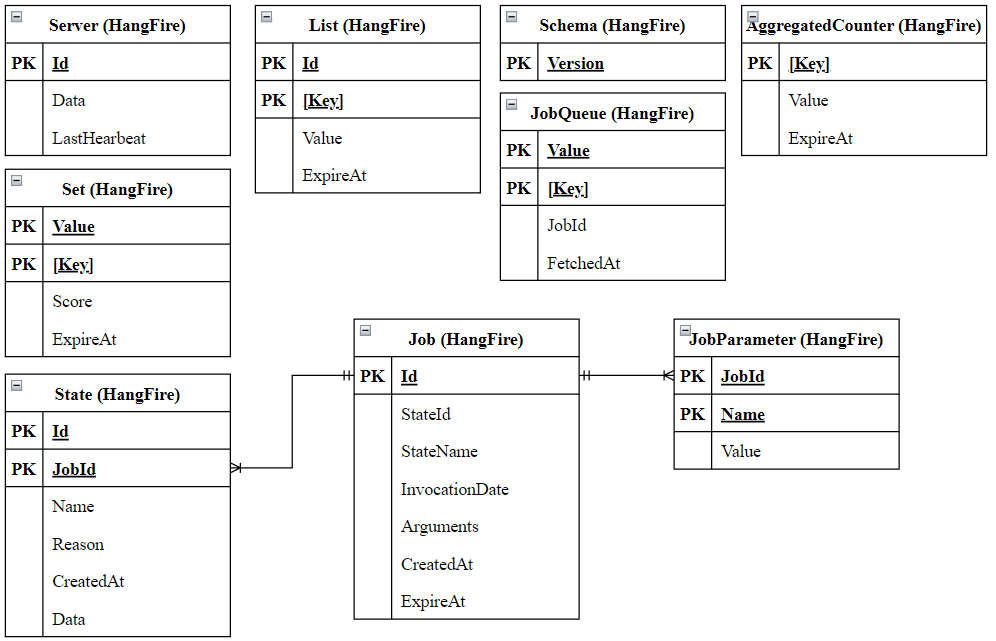


Рисунок 2.2 – Диаграмма базы данных (таблицы Hangfire)

Подробную информацию о таблицах хранилища Hangfire можно изучить в официальной документации [8].

Главным объектом, ради которого собственно существует склад является товар. Соответствующей товару сущностью в базе данный является таблица Wares. Описание таблицы Wares представлено в таблице 2.1.

Таблица 2.1 – Описание таблицы Wares

|  |  |  |
| --- | --- | --- |
| Название колонки | Ограничение целостности | Описание |
| Id | PK, INT, NOT NULL | Уникальный идентификатор. |
| Name | NVARCHAR, NOT NULL | Название товара. |
| Description | NVARCHAR, NULL | Описание товара. |
| ImagePath | NVARCHAR, NULL | URL по которому хранится картинка. |
| ReceivingDate | DATETIMEOFFSET(7), NOT NULL | Дата добавления товара на склад. |

Окончание таблицы 2.1

|  |  |  |
| --- | --- | --- |
| Название колонки | Ограничение целостности | Описание |
| ShippingDate | DATETIMEOFFSET(7), NULL | Дата отгрузки товара со склада. |
| ShelfId | FK, INT, NULL | Внешний ключ для связи с полкой на которой хранится товар. Когда товар отгружен это поле будет пустым, такой товар будет храниться в базе данных до тех пор, пока количество прошедших дней с даты выгрузки не будет больше или равно количеству дней на хранение отгруженных товаров, указанных в настройках. |
| IndividualId | FK, INT, NULL | Внешний ключ для связи с физическим лицом, владеющим этим товаром. |
| LegalEntityId | FK, INT, NULL | Внешний ключ для связи с юридическим лицом, владеющим этим товаром. |

Важным и необходимым объектом после товара на складе является адресная структура. На её организацию в базе данных выделены таблицы: Areas, Racks, VerticalSections, Shelfs.

Склад может делиться на зоны, в базе данных они представлены таблицей Areas. Описание таблицы Areas представлено в таблице 2.2.

Таблица 2.2 – Описание таблицы Areas

|  |  |  |
| --- | --- | --- |
| Название колонки | Ограничение целостности | Описание |
| Id | PK, INT, NOT NULL | Уникальный идентификатор. |
| Name | NVARCHAR, NOT NULL, UNIQUE | Название зоны склада. |
| MaxShelfs | INT, NOT NULL | Максимальное количество полок. |
| MaxVerticalSections | INT, NOT NULL | Максимальное количество вертикальных секций. |

Зона включает в себя стеллажи. Они представлены в базе данных таблицей Racks. Описание таблицы Racks представлено в таблице 2.3.

Таблица 2.3 – Описание таблицы Racks

|  |  |  |
| --- | --- | --- |
| Название колонки | Ограничение целостности | Описание |
| Id | PK, INT, NOT NULL | Уникальный идентификатор. |
| AreaId | FK, INT, NOT NULL | Внешний ключ для связи с зоной. |
| Index | INT, NOT NULL | Порядковый индекс внутри зоны. |

На колонки AreaId и Index создан уникальный, некластеризованный индекс.

Стеллажи состоят из вертикальных секций. Они представлены в базе данных таблицей VerticalSections. Описание таблицы VerticalSections представлено в таблице 2.4.

Таблица 2.4 – Описание таблицы VerticalSections

|  |  |  |
| --- | --- | --- |
| Название колонки | Ограничение целостности | Описание |
| Id | PK, INT, NOT NULL | Уникальный идентификатор. |
| RackId | FK, INT, NOT NULL | Внешний ключ для связи со стеллажом. |
| Index | INT, NOT NULL | Порядковый индекс внутри стеллажа. |

На колонки RackId и Index создан уникальный, некластеризованный индекс.

Вертикальные секции состоят из полок. Они представлены в базе данных таблицей Shelfs. Описание таблицы Shelfs представлено в таблице 2.5.

Таблица 2.5 – Описание таблицы Shelfs

|  |  |  |
| --- | --- | --- |
| Название колонки | Ограничение целостности | Описание |
| Id | PK, INT, NOT NULL | Уникальный идентификатор. |
| VerticalSectionId | FK, INT, NOT NULL | Внешний ключ для связи с вертикальной секцией. |
| Index | INT, NOT NULL | Порядковый индекс внутри вертикальной секции. |

На колонки VerticalSectionId и Index создан уникальный, некластеризованный индекс.

Приложением пользуются люди, сотрудники склада, они проходят авторизацию, выполняют какие-то действия над товарами и другими объектами. Пользователи приложения представлены таблицей Users. Описание таблицы Users представлено в таблице 2.6.

Таблица 2.6 – Описание таблицы Users

|  |  |  |
| --- | --- | --- |
| Название колонки | Ограничение целостности | Описание |
| Id | PK, INT, NOT NULL | Уникальный идентификатор. |
| FirstName | NVARCHAR, NOT NULL | Имя пользователя. |
| LastName | NVARCHAR, NOT NULL | Фамилия пользователя. |
| Email | NVARCHAR, UNIQUE, NOT NULL | Электронная почта пользователя. |
| Role | INT, NOT NULL | Роль пользователя. |

Окончание таблицы 2.6

|  |  |  |
| --- | --- | --- |
| Название колонки | Ограничение целостности | Описание |
| Password | NVARCHAR, NOT NULL | Хеш пароля пользователя. |
| Salt | NVARCHAR, NOT NULL | Соль для вычисления хеша пароля. |
| RefreshToken | NVARCHAR, NULL | Хеш рефреш-токена. |
| RefreshTokenSalt | NVARCHAR, NULL | Соль для вычисления хеша рефреш-токена. |
| AvatarUrl | NVARCHAR, NULL | Url по которому храниться аватар пользователя. |

Товарами могут владеть юридические и физические лица. В базе данных это таблицы Individuals и LegalEntities.

Описание Individuals представлено в таблице 2.7.

Таблица 2.7 – Описание таблицы Individuals

|  |  |  |
| --- | --- | --- |
| Название колонки | Ограничение целостности | Описание |
| Id | PK, INT, NOT NULL | Уникальный идентификатор. |
| FirstName | NVARCHAR, NOT NULL | Имя пользователя. |
| LastName | NVARCHAR, NOT NULL | Фамилия пользователя. |
| SurName | NVARCHAR, NOT NULL | Отчество пользователя. |
| PassportNumber | NVARCHAR, NOT NULL | Номер паспорта пользователя. |
| Type | NVARCHAR, NOT NULL | Тип паспорта пользователя. |
| Address | NVARCHAR, NOT NULL | Адрес пользователя. |
| Phone | NVARCHAR, NOT NULL | Телефонный номер пользователя. |

Описание LegalEntities представлено в таблице 2.8.

Таблица 2.8 – Описание таблицы LegalEntities

|  |  |  |
| --- | --- | --- |
| Название колонки | Ограничение целостности | Описание |
| Id | PK, INT, NOT NULL | Уникальный идентификатор. |

Окончание таблицы 2.8

|  |  |  |
| --- | --- | --- |
| Название колонки | Ограничение целостности | Описание |
| Name | NVARCHAR, NOT NULL | Наименование юридического лица. |
| UNN | NVARCHAR, NOT NULL | UNN номер юридического лица. |
| Address | NVARCHAR, NOT NULL | Адрес пользователя. |
| Phone | NVARCHAR, NOT NULL | Телефонный номер пользователя. |

Настройки системы хранятся в таблице Settings. Она описана в таблице 2.9.

Таблица 2.9 – Описание таблицы Settings

|  |  |  |
| --- | --- | --- |
| Название колонки | Ограничение целостности | Описание |
| Key | PK, NVARCHAR, NOT NULL | Уникальный идентификатор. Название настройки. |
| Value | NVARCHAR, NULL | Значение настройки. |

Задачи склада представлены таблицей Problems. Она описана в таблице 2.10.

Таблица 2.10 – Описание таблицы Problems

|  |  |  |
| --- | --- | --- |
| Название колонки | Ограничение целостности | Описание |
| Id | PK, INT, NOT NULL | Уникальный идентификатор. |
| Title | NVARCHAR, NOT NULL | Название задачи. |
| Description | NVARCHAR, NULL | Описание задачи. |
| Status | INT, NULL | Статус задачи. |
| CreatedDate | DATETIMEOFFSET(7), NOT NULL | Дата создания задачи. |
| DeadlineDate | DATETIMEOFFSET(7), NULL | Дата крайнего срока выполнения задачи. |
| LastUpdateDate | DATETIMEOFFSET(7), NULL | Дата последнего обновления задачи. |
| ParentProblemId | FK, INT, NULL | Внешний ключ для связи с родительской задачей. |
| PerformerId | FK, INT, NULL | Внешний ключ для связи с пользователем, который является исполнителем. |
| AuthorId | FK, INT, NULL | Внешний ключ для связи с пользователем, который является автором. |
| WareId | FK, INT, NULL | Внешний ключ для связи с задачей. |

Окончание таблицы 2.10

|  |  |  |
| --- | --- | --- |
| Название колонки | Ограничение целостности | Описание |
| TargetShelfId | FK, INT, NULL | Внешний ключ для связи с целевой полкой, куда будет перемещаться товар. |

Пользователи могу оставлять комментарии под задачей. Для их хранения в базе данных определена таблица Comments. Таблица Comments описана в таблице 2.11.

Таблица 2.11 – Описание таблицы LegalEntities

|  |  |  |
| --- | --- | --- |
| Название колонки | Ограничение целостности | Описание |
| Id | PK, INT, NOT NULL | Уникальный идентификатор. |
| Message | NVARCHAR, NULL | Текст комментария. |
| CreatedDate | DATETIMEOFFSET(7), NULL | Дата создания комментария. |
| OwnerId | FK, INT, NOT NULL | Внешний ключ для связи с пользователем, который является автором комментария. |
| ProblemId | FK, INT, NOT NULL | Внешний ключ для связи с задачей. |

Один из словарей для хранения единиц измерений представлен таблицей UnitsOfMeasurement, она описана в таблице 2.12.

Таблица 2.12 – Описание таблицы Settings

|  |  |  |
| --- | --- | --- |
| Название колонки | Ограничение целостности | Описание |
| Id | PK, INT, NOT NULL | Уникальный идентификатор. |
| Value | NVARCHAR, NOT NULL | Значение единицы измерения. |

Информацию о таблице с историей миграций можно найти в документации по Entity Framework [9].

**2.2 Архитектура веб-приложения**

Для разрабатываемого веб-приложения была спроектирована диаграмма развёртывания, представленная на рисунке 2.3.

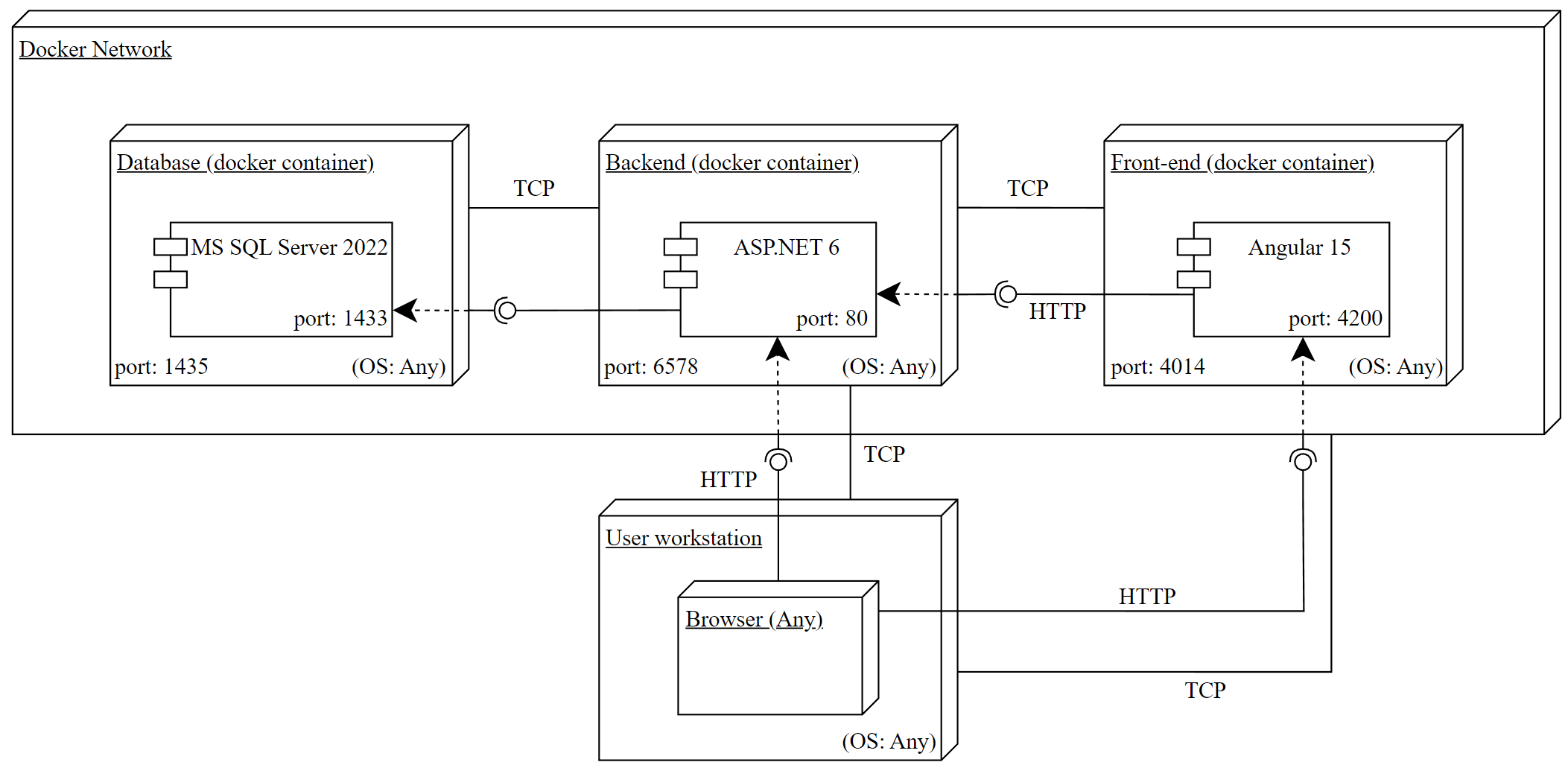


Рисунок 2.3 – Диаграмма развёртывания

Приложение разворачивается в трёх Docker-контейнерах. Контейнер с MS SQL Server 2022, контейнер с ASP.NET 6, контейнер с клиентским приложением на Angular 15.

У каждого контейнера есть внутренние и внешние порты. Backend, Frontend и браузер взаимодействуют по протоколу HTTP, а Backend-приложение и сервер базы данных по TCP.

Для разработки диаграммы развёртывания были использованы материалы курса Парамонова А.И по предмету проектирование информационных систем [10].

**2.3 Архитектура серверной части веб-приложения**

Схема архитектуры серверной части веб-приложения представлена на рисунке 2.4.

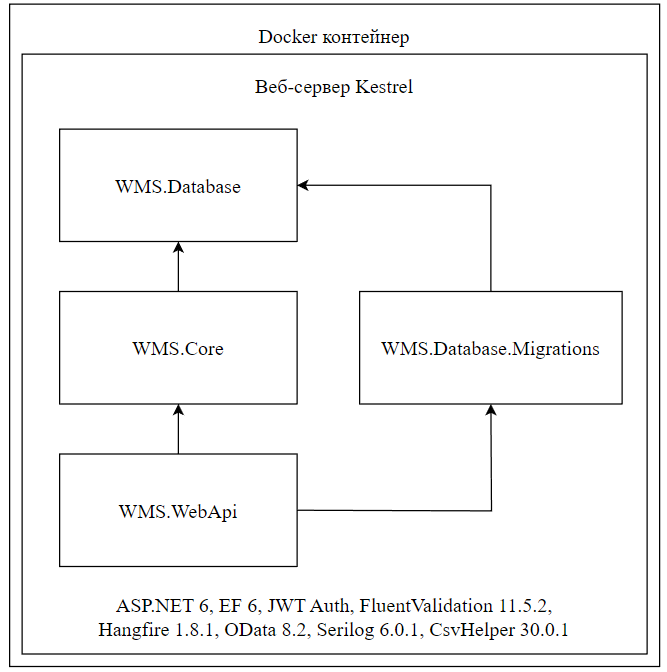


Рисунок 2.4 – Схема архитектуры серверной части веб-приложения

Архитектура серверной части веб-приложения состоит из четырёх проектов: WMS.Core, WMS.Database, WMS.Database.Migrations и WMS.WebApi.

Проект WMS.WebApi – это ASP.NET приложение типа WebAPI, содержит классы-контроллеры и зависит от проектов WMS.Database, WMS.Database.Migrations, WMS.Core. Из которых импортируются сервисы, сущности и другие вспомогательные классы.

Проект WMS.Migrations – это библиотека классов, содержит классы миграции.

Проект WMS.Database – это также библиотека классов, содержит класс контекста базы данных, классы сущностей, константы для настройки среды и OData, классы конфигурации сущностей.

Проект WMS.Core – содержит сервисы, валидаторы, пользовательские исключения и другие классы, которые реализуют бизнес логику.

**2.4 Проектирование REST API**

Для доступа клиента к серверу спроектировано REST API. Были спроектированы 13 контроллеров и 57 действий, последние описаны в таблице 2.13.

Таблица 2.13 – Описание действий REST API

|  |  |  |  |
| --- | --- | --- | --- |
| URI | Метод | OData | Действие |
| /api/Areas | GET | + | Получить список всех зон. |
| /api/Areas/$count | GET | Получить количество всех зон. |
| /api/Auth/Login | POST | - | Вход в систему. |
| /api/Auth/Refresh | POST | Обновить JWT-токены. |

Продолжение таблицы 2.13

|  |  |  |  |
| --- | --- | --- | --- |
| URI | Метод | OData | Действие |
| /api/Comments | GET | + | Получить список всех комментариев. |
| /api/Comments | POST | Создать новый комментарий. |
| /api/Comments/$count | GET | Получить количество всех комментариев. |
| /api/LegalEntities | GET | + | Получить список всех юридических лиц. |
| /api/LegalEntities | POST | Создать новое юридическое лицо. |
| /api/LegalEntities/$count | GET | Получить количество всех юридических лиц. |
| /api/ LegalEntities({key}) | DELETE | Удалить юридическое лицо. |
| /api/ LegalEntities/{key} |
| /api/ LegalEntities({key}) | PUT | Обновить юридическое лицо. |
| /api/ LegalEntities/{key} |
| /api/Problems | GET | + | Получить список всех задач. |
| /api/Problems | POST | Создать новую задачу. |
| /api/Problems/$count | GET | Получить количество всех задач. |
| /api/Problems({key}) | PUT | Обновить задачу. |
| /api/Problems/{key} |
| /api/Problems({key}) | DELETE | Удалить задачу. |
| /api/Problems/{key} |
| /api/Problems/{problemId}/UpdateStatus | PUT | Обновить статус задачи. |
| /api/Problems/{problemId}/Assign | PUT | Назначить задачу. |
| /api/Racks | POST | + | Сгенерировать стеллаж с вертикальными секциями и полками. |
| /api/Racks | GET | Получить список всех стеллажей. |
| /api/Racks({key}) | DELETE | Удалить стеллаж. |
| /api/Racks({key}) | DELETE | + | Удалить стеллаж. |
| /api/Racks({key}) | GET | Получить стеллаж. |
| /api/Racks/{key} |
| /api/Racks/$count | GET | Получить количество всех стеллажей. |
| /api/Shelfs | GET | + | Получить список всех полок. |

Окончание таблицы 2.13

|  |  |  |  |
| --- | --- | --- | --- |
| URI | Метод | OData | Действие |
| /api/Shelfs/$count | GET | + | Получить количество всех полок. |
| /api/Shelfs({key}) | GET | Получить полку. |
| /api/Shelfs/{key} |
| /api/SystemSettings | GET | - | Получить список всех системных настроек. |
| /api/SystemSettings | PUT | Обновить системные настройки. |
| /api/UnitOfMeasurements | GET | + | Получить список всех единиц измерения. |
| /api/UnitOfMeasurements/$count | GET | Получить количество всех единиц измерения. |
| /api/UnitOfMeasurements | POST | Создать новую единицу измерения. |
| /api/UnitOfMeasurements({key}) | DELETE | Удалить единицу измерения. |
| /api/UnitOfMeasurements/{key} |
| /api/UnitOfMeasurements({key}) | PUT | Обновить единицу измерения. |
| /api/UnitOfMeasurements/{key} |
| /api/Users | GET | - | Получить список всех пользователей. |
| /api/Users | POST | Создать нового пользователя. |
| /api/Users/{userId} | DELETE | Удалить пользователя. |
| /api/Users/{userId} | PUT | Обновить пользователя. |
| /api/Users/setPassword | PUT | Обновить пароль. |
| /api/VerticalSections | GET | + | Получить список всех вертикальных секций. |
| /api/VerticalSections/$count | GET | Получить количество всех вертикальных секций. |
| /api/Wares | POST | + | Создать новый товар. |
| /api/Wares | GET | Получить список всех товаров |
| /api/Wares/{wareId}/SoftDelete | PUT | Пометить товар удалённым. |
| /api/Wares/{wareId}/Restore | PUT | Восстановить товар. |
| /api/Wares/$count | GET | Получить количество всех товаров. |
| /api/Wares({key}) | PUT | + | Обновить товар. |
| /api/Wares/{key} |

Более подробную информацию о действиях, приведённых в таблице 2.13, можно посмотреть в документации API Swagger.

**2.5 Архитектура клиентской части веб-приложения**

Схема архитектура клиентской части веб-приложения представлена на рисунке 2.5.

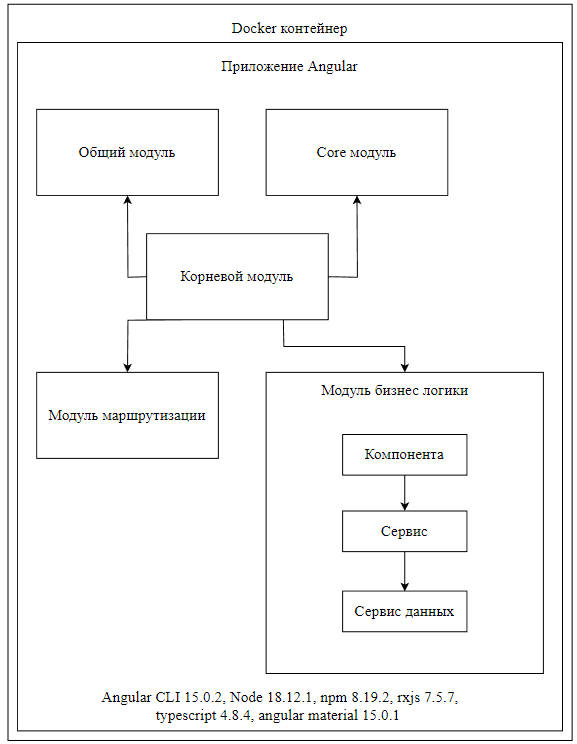


Рисунок 2.5 – Схема архитектуры клиентской части веб-приложения

Для разработки клиентской части веб-приложения был выбран JavaScript-фреймворк Angular.

Программное средство состоит из модулей и должно иметь корневой модуль. Корневой модуль – это место, где Angular начинает упорядочивать дерево приложения. Модель может импортировать компоненты, которые являются Provider, например, сервисы, хелперы и т.д. Для этого модуль по умолчанию – сингелтонг.

Ниже представлено краткое описание пакетов, представленных на рисунке 2.5:

* Core модуль является основным модулем в приложении, он содержит сингелтон сервисы и одноразовые компоненты, например layout;
* общий модуль используется для экспорта компонент, которые делятся между различными модулями (например, диалог подтверждение);
* модуль бизнес логики содержит сервисы, константы, модели и компоненты;
* сервис описывает логику компоненты;
* сервис данных реализует коммуникацию между хранилищем данных и компонентой.

**2.6 Проектирование пользовательского интерфейса**

Интерфейс приложения можно поделить на модули, которые в свою очередь будут включать компоненты. В данном разделе в таблице 2.14 представлено описание спроектированных модулей.

Таблица 2.14 – Описание модулей пользовательского интерфейса

|  |  |
| --- | --- |
| Название | Описание |
| access | Модуль предназначен для интерфейса, связанного с доступом к приложению. Например, он будет содержать компоненту для входа в систему. |
| addresses | Модуль должен содержать компоненты для адресной структуры склада. Главной компонентой будет иерархия состоящая из зон, стеллажей, вертикальных секций и полок. |
| admin-panel | Модуль предназначен для интерфейса, связанного с административными ресурсами. Например, список пользователей. |
| core | Модуль должен содержать такие базовые компоненты пользовательского интерфейса, как layout. |
| dictionaries | Модуль предназначен для интерфейса словарей. |
| home | Модуль должен содержать компоненты интерфейса домашней страницы. |
| problems | Модуль должен содержать компоненты интерфейса, связанные с задачами. Например, доска задач, форма создания/редактирования задачи, комментарии задачи и т.д. |
| shared | Модуль должен содержать общие для всех компоненты интерфейса, например, модульное окно подтверждения действия. |
| system-settings | Модуль должен содержать компоненты интерфейса настроек системы. |
| wares | Модуль должен содержать компоненты интерфейса товаров: список товаров, фильтр, поиск, формы создания/редактирования и т.д. |

Некоторые компоненты модулей будут более подробно описаны в разделе посвященном разработке веб-приложения.

**2.6 Вывод по разделу**

Для проекта была выбрана клиент-серверная архитектура. В качестве сервера базы данных был выбран MS SQL Server. Спроектировано 12 таблиц базы данных. Спроектирован REST API, который будет разработан на основе ASP.NET Web API и будет иметь трёхслойную архитектуру. Клиентское приложение будет разработано на основе JavaScript-фреймворка Angular с модульной архитектурой.

3 Разработка веб-приложения

ФИО

Подпись

Дата

Лист

1

БГТУ 03.00.ПЗ

Разраб.

*Демьянов В.Р.*

Провер.

*Парамонов А.И.*

Н. контр.

Коренькова А.А.

Утверд.

*Пацей Н.В.*

Лит.

Листов

20

*74417027, 2023*

## **3.1 Серверная часть приложения**

3 Разработка веб-приложения

Консульт.

Для разработки серверной части приложения использована N-уровневая архитектура. Уровень – это способ распределения ответственности и управления зависимостями. Каждый уровень несёт определённую ответственность. В более высоком уровне могут использоваться компоненты из более низкого уровня, но не наоборот. В соответствии с данным архитектурным паттерном принято выделять три основных уровня: доступ к данным, бизнес-логика, представление. На рисунке 3.1 представлено решение разработанного приложения.

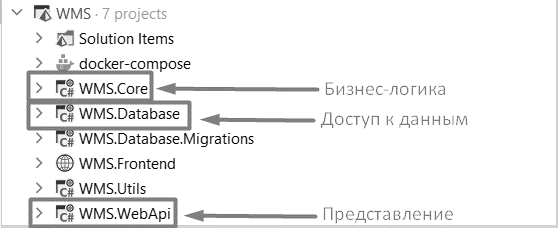


Рисунок 3.1 – Структура решения

Из рисунка 3.1 понятно, что в решении проекты WMS.Core, WMS.Database, WMS.WebApi соответствуют уровням: бизнес-логика, доступ к данным, представление. Кроме основных трёх проектов к серверной части приложения также относятся проекты WMS.Database.Migrations и WMS.Utils.

### **3.1.1 Проект WMS.Database**

Проект предназначен для организации доступа и хранении данных из базы данных. Проект является библиотекой классов, которую использует уровень, реализующий бизнес логику.

В проекте можно выделить три основных группы классов: классы сущностей и класс контекста базы данных; классы конфигурации сущностей; классы, реализующие инициализацию базы данных.

Первая группа классов (классы сущностей и класс контекста базы данных) представлена на рисунке 3.2 в виде диаграммы классов.

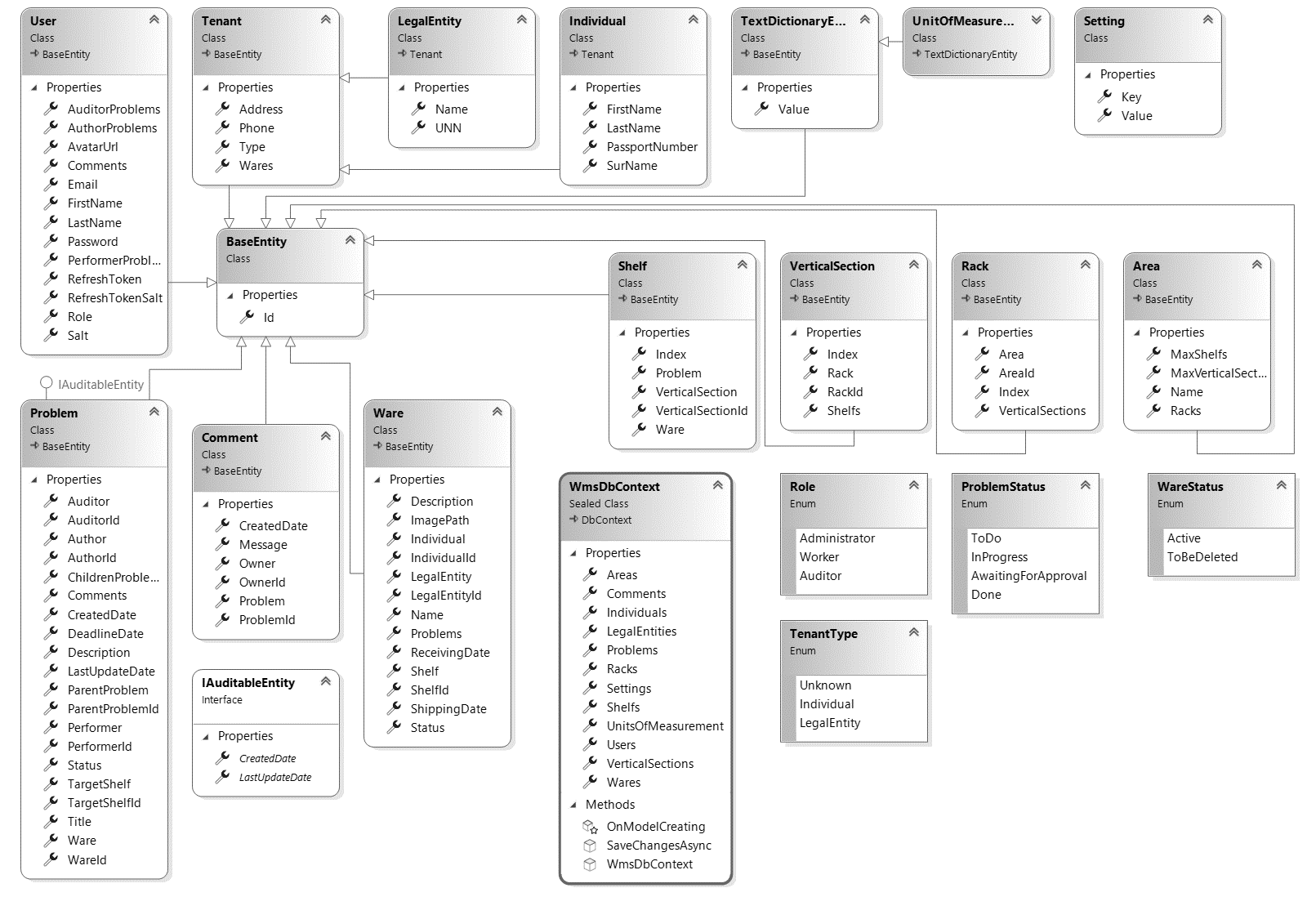


Рисунок 3.2 – Диаграмма классов сущностей и контекста базы данных

Все классы сущностей наследуются от класса BaseEntity, за исключением класса сущности Setting. Главным назначение этого класса является определение первичного ключа.

Всего в проекте 13 классов сущностей.

На рисунке 3.2 можно также увидеть перечисления и интерфейс IAuditableEntity. Данный интерфейс необходим для отслеживания изменений сущности. Например, при вызове метода контекста базы данных SaveChanges для сущностей, унаследованных от интерфейса IAuditableEntity, будет устанавливаться дата последнего изменения.

Класс контекста базы данных WmsDbContext – это основной класс, который координирует функциональные возможности EF для определённой модели данных. Этот класс является производным от класса Microsoft.EntityFrameworkCore.DbContext. Производный класс DbContext указывает сущности, которые включаются в модель данных. Код данного класса представлен в приложении А.

Вторая группа классов (классы конфигурации сущностей) представлена на рисунке 3.3 в виде диаграммы классов.

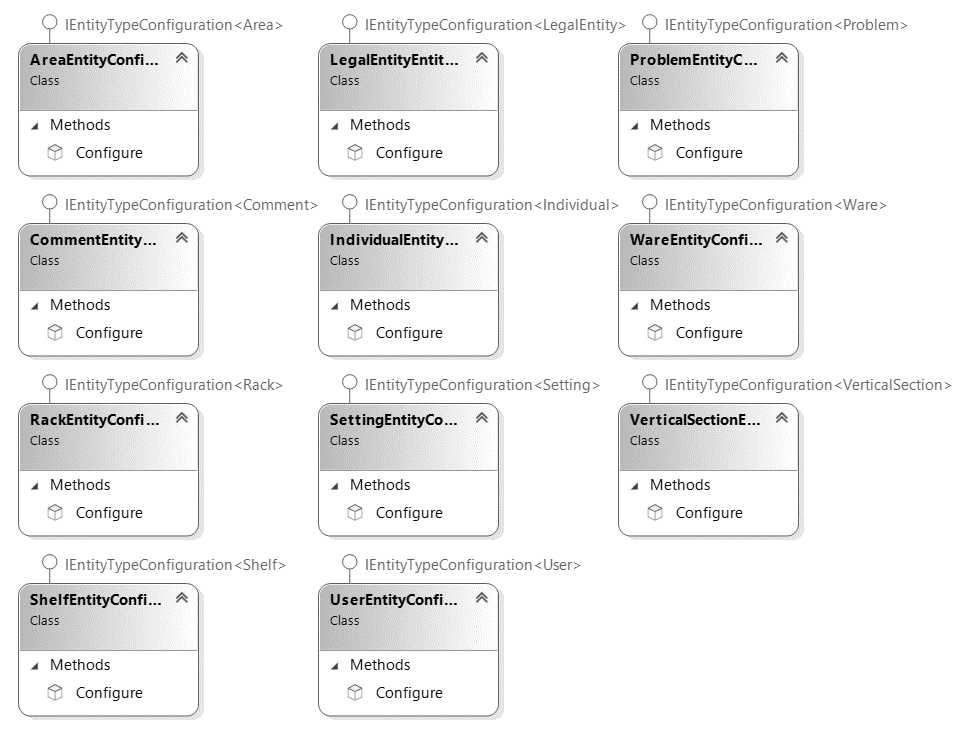


Рисунок 3.3 – Диаграмма классов конфигураций сущностей

Конфигурации сущностей базы данных определены с использованием FluentAPI. Для чистоты кода конфигурация сущностей вынесена в отдельные классы. Пример такого класса приведён в листинге 3.1.

|  |
| --- |
| public class ShelfEntityConfiguration : IEntityTypeConfiguration<Shelf>  {  public void Configure(EntityTypeBuilder<Shelf> builder)  {  \_ = builder.Property(x => x.VerticalSectionId)  .IsRequired();  \_ = builder.HasIndex(x => new { x.Index, x.VerticalSectionId })  .IsUnique();  }  } |

Листинг 3.1 – Класс конфигурации сущности

Классы конфигурации наследуются от интерфейса IEntityTypeConfiguration и реализуют метод Configure в который передаётся экземпляр класса EntityTypeBuilder.

В листинге 3.1 представлена конфигурация сущности Shelf. Значение в колонке VerticalSectionId настраивается как обязательное. Создаётся уникальный некластеризованный индекс на колонки Index и VerticalSectionId.

Для инициализации некоторых таблиц базы данных были разработаны специальные классы. Они представлены на рисунке 3.4 в виде диаграммы классов.

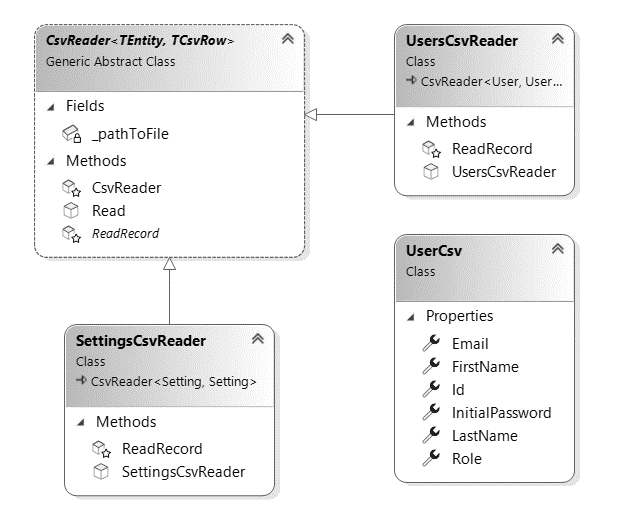


Рисунок 3.4 – Диаграмма классов для инициализации базы данных

Инициализация базы данных происходит посредством чтения данных из csv файлов и передачи этих данных в метод HasData экземпляра класса EntityTypeBuilder. Для этого реализован абстрактный класс CsvReader от которого наследуются классы уже для каждой конкретной сущности. Производные классы переопределяют и реализуют абстрактный метод ReadRecord. Пример производного класса приведён в листинге 3.2.

|  |
| --- |
| public class SettingsCsvReader : CsvReader<Setting, Setting>  {  public SettingsCsvReader(string relativePathToFile)  : base(relativePathToFile) {}  protected override Setting ReadRecord(Setting row, int recordNumber)  {  return new Setting()  {  Key = row.Key,  Value = row.Value,  };  }  } |

Листинг 3.2 – Производный класс от CsvReader

Код класса CsvReader можно найти в приложении Б.

Таким образом происходит инициализация двух таблиц базы данных: Users и Settings. Таблица Users инициализируется первый пользователем, который будет добавлять в систему последующих. Таблица Settings инициализируется настройками с указанием количества дней до уведомления о просроченных задачах и количеством дней хранения отгруженных товаров.

### **3.1.2 Проект WMS.Core**

В проекте реализована бизнес-логика приложения. Проект является библиотекой классов, используется уровнем представления.

Проект содержит классы сервисы, валидаторы, модели представлений, пользовательские исключения.

Классы сервисы определяют бизнес-логику приложения. Они взаимодействуют с уровнем доступа к данным и уровнем представления, перемещая данные между ними.

Всего в проекте 17 классов сервисов, интерфейсы, которые они реализуют представлены на рисунке 3.5 в виде диаграммы классов.

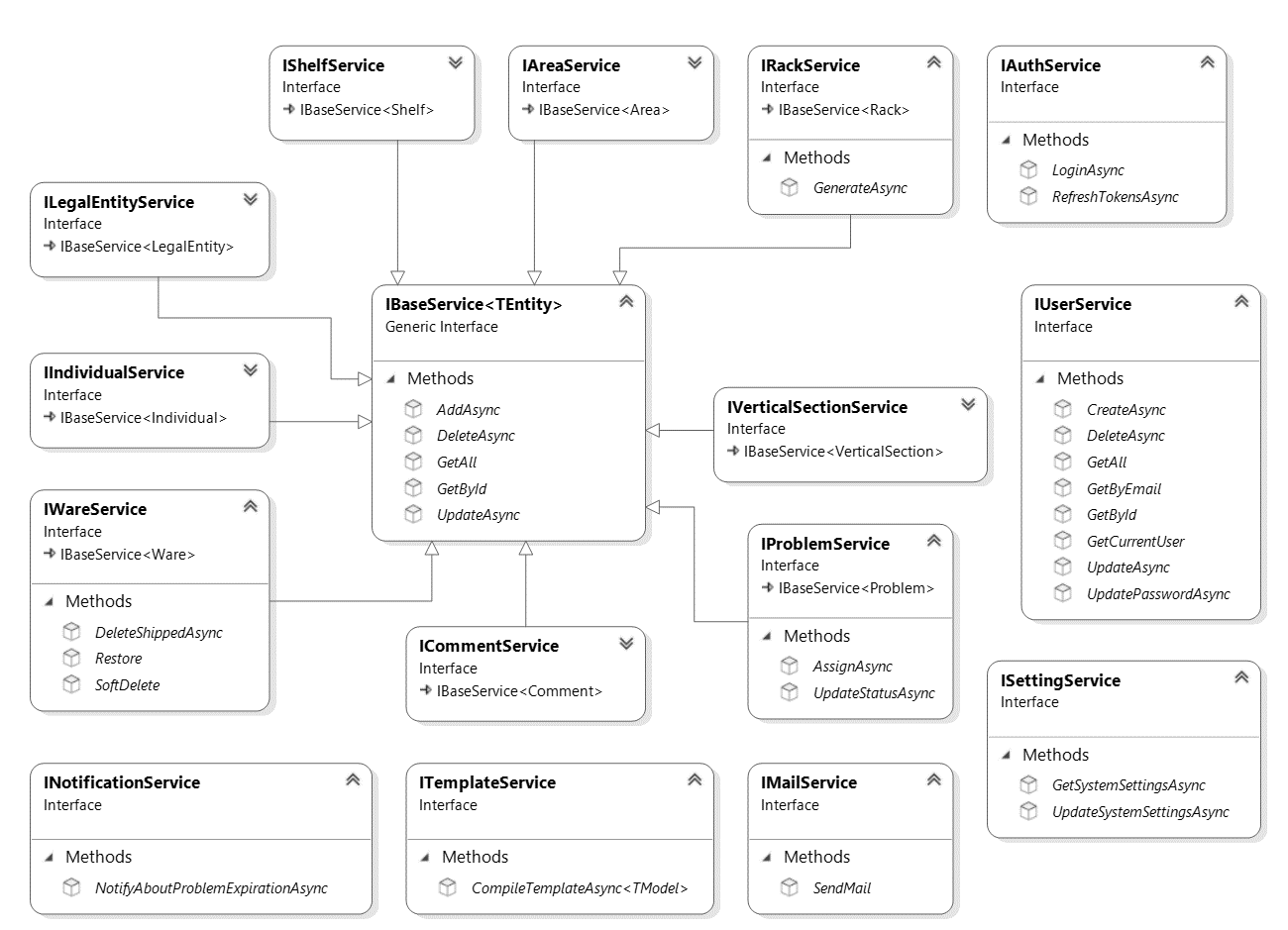


Рисунок 3.5 – Диаграмма классов для интерфейсов сервисов

Почти каждый сервис имеет доступ к базе данных посредством класса контекста из проекта WMS.Database. Сервисы, которые содержат реализацию CRUD операций, наследуются от абстрактного класса BaseService, который предоставляет стандартную реализацию таких операций.

Пример одного метода сервиса, реализующего бизнес логику, представлен в листинге 3.3.

|  |
| --- |
| public class WareService : BaseService<Ware>, IWareService  {  private readonly WareValidator \_wareValidator;  private readonly IUserService \_userService;  private readonly ISettingService \_settingService;  public WareService(  WmsDbContext dbContext,  WareValidator wareValidator,  IUserService userService,  ISettingService settingService) : base(dbContext)  {  this.\_wareValidator = wareValidator;  this.\_userService = userService;  this.\_settingService = settingService;  }  . . .  public async Task SoftDelete(int wareId)  {  var ware = await this.DbSet  .FirstOrDefaultAsync(x => x.Id == wareId);  if (ware == null)  {  throw new EntityNotFoundException("Can't soft delete ware, because it doesn't exist.");  }  ware.Status = WareStatus.ToBeDeleted;  ware.ShippingDate = DateTimeOffset.Now;  ware.ShelfId = null;  \_ = await this.DbContext.SaveChangesAsync();  }    . . .  } |

Листинг 3.3 – Класс сервис, реализующий бизнес логику

Реализацию сервиса BaseService можно найти в приложении В.

Для проверки пришедших от пользователя данных используется библиотека FluentValidation [11]. Диаграмма классов валидаторов представлена на рисунке 3.6.

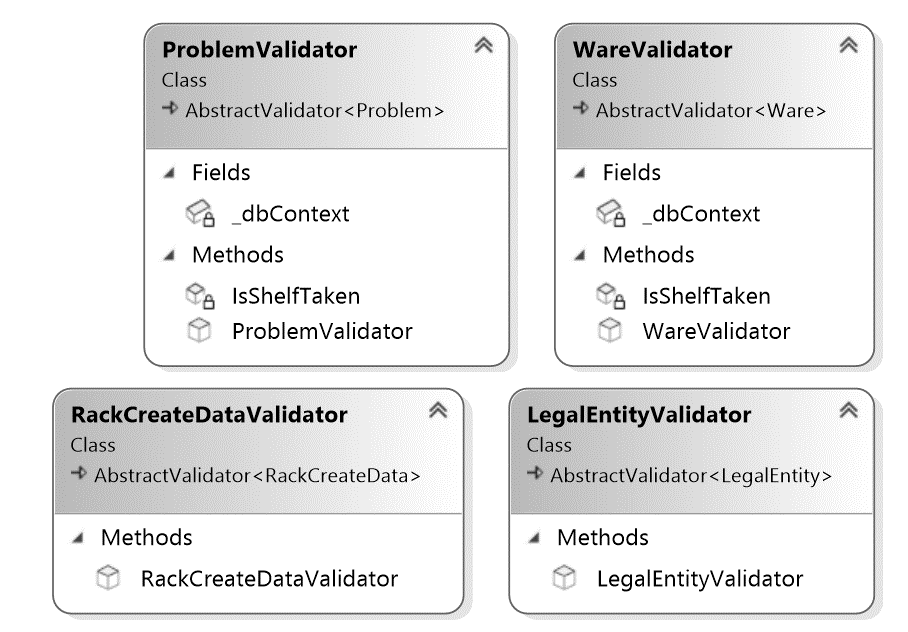


Рисунок 3.6 – Диаграмма классов валидаторов

В библиотеке FluentValidation есть возможность настроить автоматический вызов валидации, но был сделан выбор в пользу ручного, для более точечного контролирования процесса валидации.

Чтобы реализовать класс валидатор необходимо сделать наследование от класса AbstractValidator. После чего станет доступен «текучий» интерфейс. Текучий интерфейс – это способ реализации объектно-ориентированного API, при котором метод возвращают тот же интерфейс, на котором были вызваны. Пример такого валидатора представлен в приложении Г.

В приложении кроме встроенных типов исключений используются пользовательские. Всего таких классов 4, они представлены в таблице 3.1.

Таблица 3.1 – Описание пользовательских исключений

|  |  |  |
| --- | --- | --- |
| Название класса | Описание | HTTP код |
| ApiOperationFailedException | Ошибка клиента. | 400 |
| AuthenticationFailedException | Отсутствуют учётные данные. | 401 |
| AuthorizationFailedException | Недостаточно прав. | 403 |
| EntityNotFoundException | Ресурс не найден. | 404 |

В проекте присутствуют модели представления. Одним из минусов разработанной архитектуры является то, что для большей части сущностей нет модели представления, это было сделано для ускорения процесса разработки. Но для критически важных для безопасности сущностей были разработаны такие модели, а для некоторых сущностей при настройке OData EDM моделей были скрыты некоторые свойства.

Для реализации моделей представлений был выбран тип record, который появился в C# 9. Ключевая особенности records состоит в том, что они могут представлять неизменяемый тип. Это необходимо для моделей представлений, так если бы свойство DTO изменились, то данные перестали бы быть точными. Пример такого типа представлен в листинге 3.4.

|  |
| --- |
| public record SystemSettings  {  public int ProblemExpirationNotificationDays { get; init; }  public int ShippedWaresStorageDays { get; init; }  } |

Листинг 3.4 – Модель представления

Важно отметить, что применение ключевого слова record не гарантирует неизменяемость объектов record. Чтобы сделать его действительно неизменяемым, для свойств вместо обычных сеттеров надо использовать модификатор init.

### **3.1.3 Проект WMS.WebApi**

Проект является уровнем представления данных. Это первый уровень, с которым взаимодействует клиент. Для этого было разработано REST API.

Проект включает 14 контроллеров. Большая часть из них поддерживает открытый протокол OData. Описание маршрутов можно найти в разделе 2.

Для REST-контроллеров были реализованы классы, которые являются производными от класса ControllerBase. Для реализации OData контроллеров, были реализованы классы производные от класса ODataController, а также зарегистрированы EDM модели.

Entity Data Model – это расширенная версия Entity-Relationship, которая определяет концептуальную модель данных с использованием различных методов моделирования.

Регистрацию EDM моделей и настройка OData происходит в классе расширении WmsODataMvcBuilderExtensions, его можно найти в приложении Г.

Класс Program является входной точкой в приложение. В нём происходит конфигурация приложения, регистрация сервисов в контейнере, установка компонент для обработки запроса (middleware), настройка логгирования в приложении.

Для логирования в приложении используется библиотека Serilog. Логи приложения пишутся в текстовый документ по определённому шаблону, который представлен в листинге 3.5.

|  |
| --- |
| "outputTemplate": "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Level}][{RequestId}]: {Message}{NewLine}{Exception}" |

Листинг 3.5 – Шаблон записи в файле логов

Полную конфигурацию логирования можно найти в приложении Д.

В проекте была реализована пользовательская обработка ошибок, она представлена в листинге 3.6.

|  |
| --- |
| app.UseExceptionHandler(app => app.Run(async context =>  {  var exception=context.Features.Get<IExceptionHandlerFeature>()?.Error;  context.Response.StatusCode = exception switch  {  ApiOperationFailedException => (int)HttpStatusCode.BadRequest,  EntityNotFoundException => (int)HttpStatusCode.NotFound,  AuthenticationFailedException => (int)HttpStatusCode.Unauthorized,  AuthorizationFailedException => (int)HttpStatusCode.Forbidden,  \_ => (int)HttpStatusCode.InternalServerError,  };  await context.Response.WriteAsJsonAsync(new  {  StatusCode = context.Response.StatusCode,  ErrorMessage = exception?.Message ?? string.Empty,  });  })); |

Листинг 3.6 – Пользовательская обработка ошибок

Для документирования API была подключена библиотека Swagger, код конфигурации представлен в листинге 3.7.

|  |
| --- |
| builder.Services.AddSwaggerGen(options =>  {  options.SwaggerDoc(  "v1",  new()  {  Title = "WMS API",  Version = "v1",  });  var entryAssembly = Assembly.GetExecutingAssembly();  var xmlFilename = $"{entryAssembly.GetName().Name}.xml";  var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFilename);  options.IncludeXmlComments(xmlPath);  }); |

Листинг 3.7 – Конфигурация Swagger

В приложении есть регулярно запускаемые задачи, для оповещения об истечении срока выполнения задач и удаления отгруженных товаров. Для этого используется библиотека Hangfire.

Регистрация упомянутых выше задач представлена в листинге 3.8.

|  |
| --- |
| \_ = builder.Services.AddHangfire(configuration => \_ = configuration.UseSqlServerStorage(connectionString));  \_ = builder.Services.AddHangfireServer();  JobStorage.Current = new SqlServerStorage(connectionString);  string? cronScheduleExpression = builder.Configuration["HangFire:CronScheduleExpression"];  // Configure Hangfire jobs  RecurringJob.AddOrUpdate<NotificationService>(  "ProblemExpirationJob",  notificationService => notificationService.NotifyAboutProblemExpirationAsync(),  cronScheduleExpression);  RecurringJob.AddOrUpdate<WareService>(  "ShippedWaresStorageExpirationJob",  wareService => wareService.DeleteShippedAsync(),  cronScheduleExpression); |

Листинг 3.7 – Конфигурация Swagger

Для аутентификации и авторизации был выбран механизм с использованием JWT-токенов. С помощью метода AddJwtBearer в приложение добавляется конфигурация токена, которая представлена в листинге 3.8.

|  |
| --- |
| builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)  .AddJwtBearer(options =>  {  options.TokenValidationParameters = new TokenValidationParameters  {  ValidateIssuer = true,  ValidIssuer = builder.Configuration["AuthOptions:Issuer"],  ValidateAudience = true,  ValidAudience = builder.Configuration["AuthOptions:Audience"],  ValidateIssuerSigningKey = true,  IssuerSigningKey = new SymmetricSecurityKey(key),  RequireExpirationTime = true,  ValidateLifetime = true,  ClockSkew = TimeSpan.Zero,  };  }); |

Листинг 3.8 – Конфигурация токена

Для конфигурации токена применяется объект JwtBearerOptions.

Для защиты конечной точки применяется атрибут Authorize. Пример такой конечной точки представлен в листинге 3.9.

|  |
| --- |
| /// <summary>  /// Deletes user by id.  /// </summary>  /// <param name="userId">User Id.</param>  [HttpDelete("{userId:int}")]  [Authorize(Roles = nameof(Role.Administrator))]  public async Task<ActionResult> Delete(int userId)  {  await this.\_userService.DeleteAsync(userId);  return this.NoContent();  } |

Листинг 3.9 – Защищённая конечная точка

Из листинга 3.9 можно понять, что удалять пользователей могут только пользователи, прошедшие аутентификацию и авторизированные в роли администратора.

### **3.1.4 Проект WMS.Database.Migrations**

Проект является библиотекой классов и предназначен для хранения классов миграций.

Инструкцию по добавлению и применению миграций можно найти в разделе 5.

## **3.2 Клиентская часть приложения**

В разделе 2 была спроектирована модульная архитектура. Помочь реализовать такую архитектуру позволяет JavaScript фреймворк Angular.

Несмотря на то, что в веб-разработке уже используется модульная архитектура, которая стала возможной с появлением ES6-импортов, Angular-модули добавляют в систему дополнительный уровень логической группировки.

Важно, чтобы структура модулей как можно лучше соответствовала решаемым с их помощью задачам. Для этого в Angular существуют различные типы модулей:

* DomainModule, реализующие ключевые задачи приложения;
* RoutingModule, предназначенные для маршрутизации;
* ServiceModule, предназначенные для сервисов;
* CoreModule, предназначенные для объявления глобальных сервисов;
* SharedModule, предназначенные для экспорта компонент для совместного использования;
* WidgetModule, предназначенные для экспорта директив, компонентов, модулей, которые можно переиспользовать в других проектах;
* AppModule, корневой модуль.

Это основные типы модулей, которые рекомендуются сообществом Angular, но в зависимости от сложности проекта можно выделить и другие.

В разработанном приложении используются не все типы из тех, которые представлены выше, это сделано намерено для ускорения разработки.

### **3.2.1 Структура модуля**

Модули Angular состоят из набора свойств и одного хука жизненного цикла. В листинге 3.10 приведён интерфейс модуля.

|  |
| --- |
| export interface NgModule {  providers?: Provider[];  declarations?: Array<Type<any> | any[]>;  imports?: Array<Type<any> | ModuleWithProviders<{}> | any[]>;  exports?: Array<Type<any> | any[]>;  entryComponents?: Array<Type<any> | any[]>;  bootstrap?: Array<Type<any> | any[]>;  id?: string;  schemas?: Array<SchemaMetadata | any[]>;  jit?: true;  } |

Листинг 3.10 – Интерфейс Angular модуля

Свойство providers используется для определения сервисов, которые были украшены декоратором Injectable, что делает их доступными через Angular DI.

Свойство declarations это самое простое и наиболее часто используемое свойство. Оно используется для импорта компонентов, директив, каналов.

Свойство imports используется для импорта других модулей.

По умолчанию всё, что определено в модуле, является частным. Свойство exports – это массив, который позволяет объявлениям в модуле быть доступными из модулей, которые импортируют модуль, в котором они определены.

Свойство entryComponents указывает список компонентов, скомпилированных при загрузке модуля. Эти компоненты не являются компонентами, определёнными в шаблоне, но обычно загружаются принудительно, например, с помощью ViewContainerRef.createComponent(). Примером являются компоненты маршрутизации, но платформа автоматически добавляет их в entryComponents.

Свойство bootstrap также указывает компоненты, скомпилированные при загрузке модуля, и автоматически добавляет их в entryComponents.

Свойство идентификатор указывает имя, которое идентифицирует модуль.

Свойство schemas указывает значения подобные: NO\_ERRORS\_SCHEMA или CUSTOM\_ELEMENTS\_SCHEMA.

Свойство jit заставит фреймворк всегда компилировать модуль с использованием JIT, а не AOT.

### **3.2.2 AppModule**

AppModule, также часто называется корневым модулем, – это модуль, отвечающий за загрузку всего приложения. Это означает, что в нём следует импортировать все модули и провайдеры, необходимые для запуска приложения.

Этот модуль обычно импортирует основные модули Angular, общие модули, доменные модули, сервисные модули и AppRoutingModule (корневой модуль маршрутизации).

Кроме того, он загружает корневой компонент, обычно называемы AppComponent. Код разработанного корневого модуля представлен в листинге 3.11.

|  |
| --- |
| @NgModule({  declarations: [AppComponent],  imports: [  BrowserModule,  AppRoutingModule,  AccessModule,  CoreModule,  BrowserAnimationsModule,  ],  bootstrap: [AppComponent],  providers: [  { provide: HTTP\_INTERCEPTORS, useClass: JwtInterceptor, multi: true },  { provide: HTTP\_INTERCEPTORS, useClass: AuthErrorInterceptor, multi: true },  ],  })  export class AppModule {} |

Листинг 3.11 – Код корневого Angular модуля

В корневом модуле импортирует только то, что необходимо для загрузки приложения. Поэтому всё, что должно быть загружено отложено, не должно быть импортировано в этот модуль. В случаем с разработанным приложением были импортированы AccessModule и CoreModule. AccessModule содержит форму входа, которая будет первым, что увидит пользователь при первом запуске приложения.

### **3.2.3 SharedModule**

SharedModule – это модуль, отвечающий за размещение всех общих объектов, которые будут предоставляться каждому модулю приложения.

SharedModule обычно состоит из сущностей, которые являются общими для разных модулей внутри проекта, но обычно не требуется вне его.

Если появляются сервисы или компоненты, которые можно повторно использовать в разных командах и проектах, и которые в идеале не меняются очень часто, то для них следует создать библиотеку Angular.

Код разработанного SharedModule представлен в листинге 3.12.

|  |
| --- |
| @NgModule({  declarations: [  ConfirmDialogComponent,  AddressPickerComponent,  AddressPickerDialogComponent,  ],  imports: [  CommonModule,  MatModule,  ReactiveFormsModule,  ScrollingModule,  ],  exports: [  ConfirmDialogComponent,  AddressPickerComponent,  MatModule,  ReactiveFormsModule,  CommonModule,  ScrollingModule,  ],  })  export class SharedModule { } |

Листинг 3.12 – Angular SharedModule

Из листинга 3.12 видно, что общим для всех компонент является, например, компонент ConfirmDialogComponent.

### **3.2.4 DomainModule**

DomainModule – это модули, относящиеся к бизнес-функциям приложения. В разработанном приложении таких модулей всего 8: AccessModule, AddressesModule, AdminPanelModule, DictionariesModule, HomeModule, ProblemsModule, SystemSettingsModule, WaresModule.

Один из таких модулей представлен в листинге 3.13.

|  |
| --- |
| @NgModule({  declarations: [LoginComponent],  imports: [  SharedModule,  AccessRoutingModule,  ],  })  export class AccessModule {  } |

Листинг 3.13 – Angular DomainModule

В зависимости от сложности доменного модуля, он может быть разбил на более мелки модули домена.

Это имеет смысл, когда каждый модуль представляет собой сложный набор компонентов или если в будущем определённый компонент будет всё более и более сложным.

Доменные модули должны экспортировать ничего, кроме верхнего компонента.

### **3.2.5 RoutingModule**

Модули маршрутизации отвечают за объявление маршрутов модуля домена (включая корневой модуль), передачу конфигурации в модуль, а также за определение сервисов защиты.

Для корневого модуля маршруты определяются с использованием свойства forRoot, в то время как остальные должны использовать свойство forChild. Код разработанного модуля маршрутизации для корневого модуля представлен в листинге 3.14.

|  |
| --- |
| const routes: Routes = [{  path: AppRoute.Access,  component: LayoutComponent,  children: [{  path: AppRoute.Default,  loadChildren: () => import('./access/access.module')  .then(m => m.AccessModule),  }],  },{  path: AppRoute.Default,  component: LayoutComponent,  children: [{  path: AppRoute.Home,  loadChildren: () => import('./home/home.module')  .then(m => m.HomeModule),  canActivate: [AuthGuard],  }, {  path: '\*\*',  redirectTo: AppRoute.Home,  }],  },  ];  @NgModule({  imports: [RouterModule.forRoot(routes)],  exports: [RouterModule]  })  export class AppRoutingModule { } |

Листинг 3.14 – Angular RoutingModule для корневого модуля

В приложении реализована ленивая загрузка модулей. По умолчанию NgModules загружаются с готовностью. Это значит, что как приложение загрузится и все NgModules, нужны они сразу или нет. Для больших приложений с большим количеством маршрутов лучше реализовать отложенную загрузка – шаблон проектирования, по принципам которого NgModules загружается по мере необходимости. Ленивая загрузка помогает уменьшить начальные размеры пакетов, что, в свою очередь, помогает сократить время загрузки.

### **3.2.6 CoreModule**

CoreModule – это модуль, который размещает в себе общие для других модулей singleton сервисы.

Инжектор зависимостей Angular создаёт новый экземпляр сервисов для каждого лениво загруженного модуля. Так как мы хотим гарантировать, что singleton сервисы будут созданы в виде одного экземпляра, то необходимо разместить их в CoreModule.

Ещё одна часть приложения, которая должна находится в CoreModule – это компоненты уровня приложения, например, layout.

Код разработанного Core модуля представлен в листинге 3.15.

|  |
| --- |
| @NgModule({  declarations: [LayoutComponent],  imports: [CommonModule, AppRoutingModule, HttpClientModule],  providers: [AuthenticationDataService, AuthenticationService, PermissionsService, AuthGuard, PublicGuard],  })  export class CoreModule {  constructor(@Optional() @SkipSelf() parentModule: CoreModule) {  if (parentModule) {  throw new Error(  'CoreModule has already been loaded. Import Core modules in the AppModule only.',  );  }  }  } |

Листинг 3.15 – Angular CoreModule

Из листинга 3.15 можно увидеть, что в нём объявляется компонент реализующий layout приложения, сервисы для аутентификации и Guards.

### **3.2.7 Angular компоненты**

Компонент – часть интерфейса приложения с собственной логикой.

Компоненты складываются в модули и затем могут быть переиспользованы в разных частях приложения.

Для того чтобы класс стал компонентом, нужно пометить его декоратором @Component.

Код одного небольшого компонента представлен в листинге 3.16.

|  |
| --- |
| @Component({  selector: 'app-wares',  templateUrl: './wares.component.html',  styleUrls: ['./wares.component.scss']  })  export class WaresComponent {} |

Листинг 3.16 – Angular компонента

Этот декоратор принимает объект, который имеет следующие свойства:

* selector, содержит название компонента;
* templateUrl или template, содержит HTML-разметку в виде пути к файлу HTML или строки;
* providers, список сервисов, поставляющих данные для компонента;
* styles, содержит массив путей к CSS-файлам.

Чтобы использовать созданный компонент нужно добавить его в declarations описанный ранее. И затем можно его использовать в качестве тега в другом компоненте.

Всего в приложении разработано 32 компонента, их описание представлено в таблице 3.1.

Таблица 3.1 – Описание компонент клиентского приложения

|  |  |
| --- | --- |
| Название | Описание |
| LoginComponent | Форма входа в приложение. |
| AddressFormAreaComponent | Компонента для отображения информации о зоне склада. |
| AddressFormRackComponent | Компонента для отображения информации о стеллаже склада. |
| AddressFormShelfComponent | Компонента для отображения информации о полке склада. |
| AddressFormVerticalSectionComponent | Компонента для отображения информации о вертикальной секции склада. |
| AddressToolbarComponent | Компонента для панели инструментов адресной системы склада. |
| AddressTreeComponent | Компонента для отображения адресной системы склада. |
| AddressComponent | Компонента, являющаяся разметкой для остальных компонент адресной системы склада. |
| EmployeesComponent | Компонента, отображающая список сотрудников склада. |
| IndividualsComponent | Компонента, отображающая список физических лиц. |
| LegalEntitiesComponent | Компонента, отображающая список юридических лиц. |

Окончание таблицы 3.1

|  |  |
| --- | --- |
| Название | Описание |
| DictionariesPageComponent | Компонента, отображающая словари. |
| HeaderComponent | Навигация приложения. |
| HomePageComponent | Домашняя страница приложения. |
| ProblemAssignDialogComponent | Модальное окно для назначения исполнителя задачи. |
| ProblemCardComponent | Компонента для отображения информации задачи в виде карточки. |
| ProblemChildrenComponent | Компонента, отображающая дочерние задачи. |
| ProblemCommentsComponent | Компонента, отображающая список комментариев. |
| ProblemDialogComponent | Модальное окно для создания/редактирования задачи. |
| ProblemInfoComponent | Компонента, отображающая информацию о задаче. |
| ProblemsComponent | Компонента для отображения доски задач. |
| ConfirmDialogComponent | Модальное окно подтверждения действия. |
| AddressPickerDialogComponent | Модальное окно для выбора адреса. |
| AddressPickerComponent | Пользовательский элемент управления для выбора адреса. |
| SystemSettingBottomSheetComponent | Компонент для отображения системных настроек. |
| WaresComponent | Макет для компонент товаров. |
| WareRestoreDialogComponent | Модальное окно для восстановления товара. |
| WareFilterDialogComponent | Модальное окно для фильтров задач. |
| WareFormComponent | Форма создания/редактирования/просмотра товара. |
| WareNavigationComponent | Список товаров. |
| WareToolbarComponent | Панель инструментов для управления списком товаров. |
| AppComponent | Корневой компонент. |

Связать компоненты между собой, чтобы передать данные от родительского компонента к дочернему или подписаться на событие можно через свойства компонент. Для этого нужно пометить такие свойства декораторами @Input или @Output.

@Input и @Output хорошо работают в простых случаях, но, когда мы хотим связать входящие данные и связать исходящие события с компонентом-контейнером, это может стать кошмаром для управления.

Нам нужно добавить много @Input и @Output на многие уровни компонента. Одним из решений является реализация паттерна шины событий.

Концепция шины событий очень проста. Нужно создать глобально доступны сервис. Затем можно отправлять события в шину, и, если какой-либо прослушиватель был зарегистрирован для этого имени события, он выполнит функцию обратного вызова. Реализация одной такой шины событий представлена в приложении Е.

### **3.2.8 Angular сервисы**

Сервисы – это классы провайдеры, которые предоставляют данные компонентам. Для того, чтобы сделать класс провайдером нудно пометить его декоратором @Injectable.

Сервисы бывают разных типов. В приложении были разработаны: DataService, ComponentService, EventBusService.

DataService – это сервисы, которые взаимодействует с источником данных. Это может быть REST API или LocalStorage.

ComponentService – это сервисы, которые реализуют логику компоненты.

EventBustService – это реализация паттерна шины событий.

Пример DataService представлен в листинге 3.16.

|  |
| --- |
| @Injectable()  export class EmployeesDataService {  constructor(private readonly http: HttpClient) { }  public getAll = (): Observable<IEmployee[]> =>  this.http.get<IEmployee[]>(ApiEndpoints.Users);  public delete(id: number): Observable<void> {  return this.http.delete<void>(`${ApiEndpoints.Users}${id}`);  }  public create(employee: IEmployee): Observable<IEmployee> {  return this.http.post<IEmployee>(ApiEndpoints.Users, employee);  }  public update(id: number, employeeUpdateData: IEmployee): Observable<void> {  return this.http.put<void>(`${ApiEndpoints.Users}${id}`, employeeUpdateData);  }  } |

Листинг 3.16 – Angular DataService

Сервис представленный в листинге 3.16 служит для отправки запросов на сервер. Для этого он внедряет сервис HttpClient. Затем он используется в методе getAll (GET-запрос), create (POST-запрос), update (PUT-запрос).

Методы, определённые в HttpClient являются асинхронными и возвращают Observable, который выдаёт запрошенные данные при получении ответа. Observable или наблюдаемый объект, позволяет подписаться на него и обрабатывать ноль, одно или более событий, которые производит этот объект. Причём можно отменить подписку. Это значит, что мы можем сделать запрос и, если вдруг нам он больше не нужен отменить его, не дожидаясь ответа. Это большое преимущество над теми же Promise.

## **3.3 Вывод по разделу**

В соответствии со спроектированной ранее архитектурой, разработаны:

* база данных с использованием подхода Code First;
* серверная часть веб-приложения;
* клиентская часть веб-приложения.

База данных разработана под управлением СУБД MS SQL Server. Но подход Code First и Entity Framework позволяют в будущем абстрагироваться от конкретной СУБД.

Серверная часть разработана с использованием стека технологий .NET 6, а именно: ASP.NET, Class Library, Entity Framework и других библиотек.

Серверная часть разработана по принципу N-tier архитектуры и включает 5 проектов. Реализовано 57 конечных точек HTTP API.

Клиентская часть разработана с использованием JavaScript фреймворка Angular 15. Была реализована модульная архитектура, которая включает 10 модулей и 32 компоненты.

4 Тестирование веб-приложения

ФИО

Подпись

Дата

Лист

1

БГТУ 04.00.ПЗ

Разраб.

*Демьянов В.Р.*

Провер.

*Парамонов А.И.*

Н. контр.

Коренькова А.А.

Утверд.

*Пацей Н.В.*

Лит.

Листов

10

*74417027, 2023*

Тестирование ПО – проверка соответствия между реальным и ожидаемым поведением программы. Это важный этап жизненного цикла разработки ПО, в зависимости от используемой модели разработки процесс тестирования может отличаться.

4 Тестирование веб-приложения

Консульт.

В тестировании ПО принято выделять уровни:

* компонентное тестирование;
* интеграционное тестирование;
* системное тестирование;
* приёмочное тестирование.

В идеальном варианте ПО должно пройти все эти уровни, чтобы снизить риски появления дефектов на этапе эксплуатации и поддержке. Но исходя из установленных сроков выполнения дипломного проекта, было принято решение проводить только системное тестирование, причём ручным способом.

Далее будут приведены результаты выполненного тестирования. Оно будет разделено на две части: тестирование *REST API* и пользовательского интерфейса.

Для тестирования будут использованы техники позитивного и негативного тестирования, а также анализа граничных значений.

Разработанное ПО имеет систему логирования и обработки ошибок, что поможет при проведении тестирования и выявления дефектов.

## **4.1 Тестирование *REST API***

Для тестирования, разработанного *API*, будет использоваться инструмент *Postman*. При тестировании будут учитываться полученные *HTTP* ответы от сервера. Будут проверяться: полученный статус код, тело ответа, ожидаемое поведение.

### **4.1.1 Тестирование аутентификации *REST API***

Аутентификация *REST API* реализована с использованием *JWT*-токенов.

*JSON Web Token* (*JWT*) – это *JSON* объект, который определён в открытом стандарте *RFC* 7519 [14]. Он считается одним из безопасных способов передачи информации между двумя участниками. Для его создания необходимо определить заголовок (*header*) с общей информацией по токену, полезные данные (*payload*), такие как *Id* пользователя, его роль и т.д. и подписи (*signature*).

Простыми словами, *JWT* – это лишь строка в следующем формате: «*header*.*payload*.*signature*».

Предположим, что мы хотим добавить новый товар на склад. Приложение использует JWT для проверки аутентификации следующим образом:

1. Пользователь делает запрос к серверу на получение *JWT*.
2. Сервер, если данные прошли валидацию, отправляет *JWT* клиенту.
3. Когда пользователь делает запрос к *API* приложения, он добавляет к нему полученный ранее *JWT*.
4. Сервер проверяет на валидность переданный пользователем *JWT*.
5. Сервер возвращает ошибку или запрошенный ресурс.

Теперь зная процесс аутентификации в приложении перейдём к тестированию. Будем использовать технику позитивного тестирования. Входные данные для тестирования представлены в таблице 4.1.

Таблица 4.1 – Входные данные для позитивного тестирования аутентификации

|  |  |  |  |
| --- | --- | --- | --- |
| Конечная точка | *HTTP* метод | Входные данные | Ожидаемый результат |
| *api*/*Auth*/*Login* | *POST* | Адрес электронной почты и пароль пользователя, который существует в системе | *JWT, 200 статус-код* |
| *api*/*Wares* | *POST* | Данные товара и полученный от сервера *JWT* в заголовке запроса | Данные созданного товара, 201 статус-код |

Первым шагом сделаем запрос на получение JWT от сервера, данный шаг показан на рисунке 4.1.

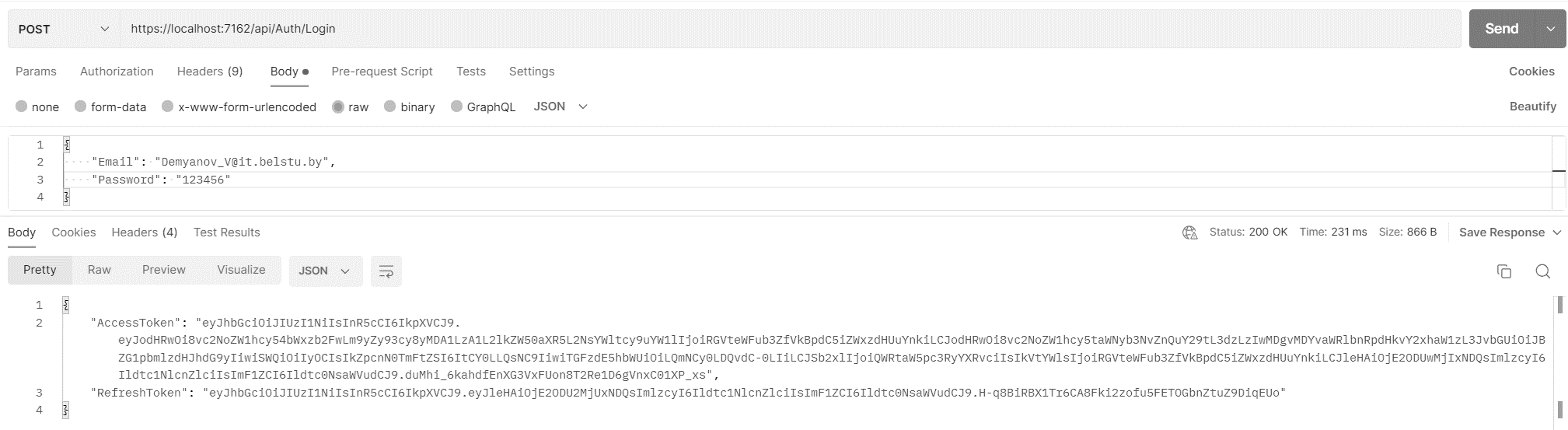


Рисунок 4.1 – Запрос на получение JWT для позитивного теста аутентификации

Запрос выполнен успешно, мы получили от сервера сгенерированный JWT и 200 статус-код. Полученный ответ соответствуют ожидаемому результату, приведённому выше в таблице 4.1.

Вторым шагом сделаем запрос на создание товара, используя полученный *access token*, данный шаг показан на рисунке 4.2.

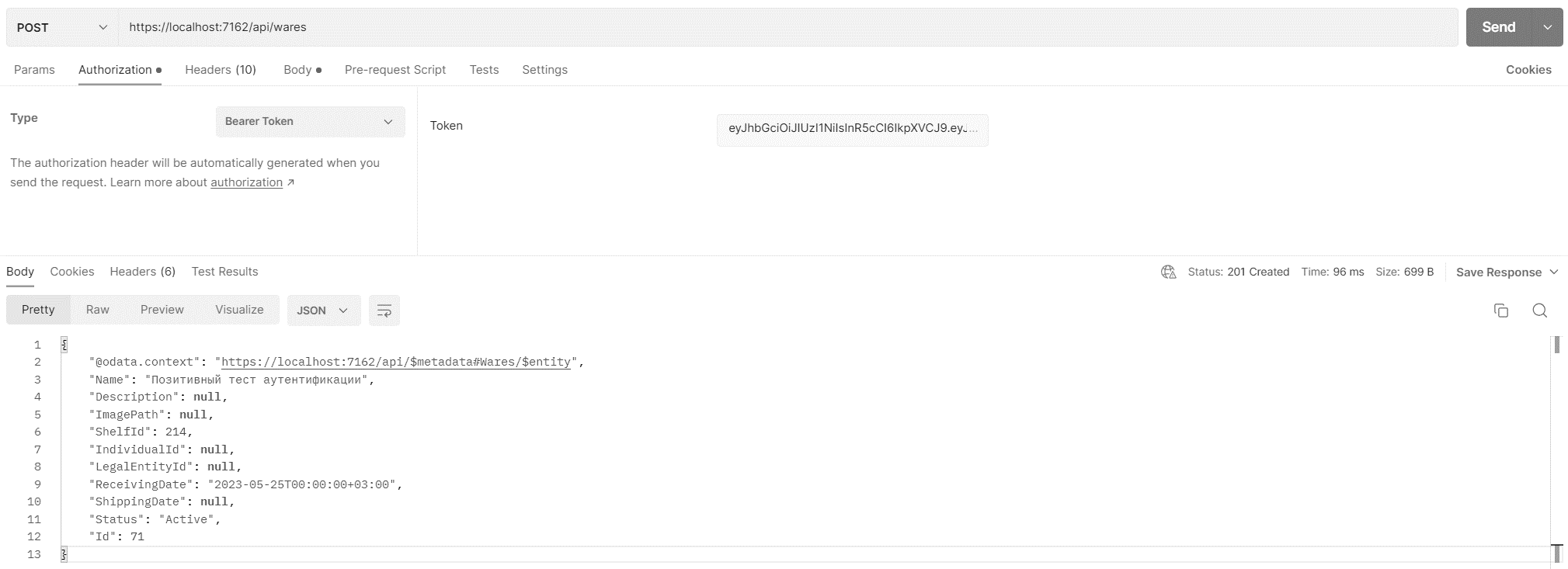


Рисунок 4.2 – Запрос на создание товара для позитивного теста аутентификации

Запрос выполнены успешно, мы получили от сервера ответ со статусом 201 и созданным товаром. Полученный ответ соответствует ожидаемому результату, приведённому в таблице 4.1.

В результате выполнения приведённых выше запросов, можно сделать вывод, что аутентификация программного средства прошла позитивное тестирование.

Теперь проведём негативное тестирование аутентификации программного средства. Входные данные для тестирования представлены в таблице 4.2.

Таблица 4.2 – Входные данные для негативного тестирования аутентификации

|  |  |  |  |
| --- | --- | --- | --- |
| Конечная точка | *HTTP* метод | Входные данные | Ожидаемый результат |
| *api*/*Wares* | *POST* | Данные товара и просроченный *JWT* в заголовке запроса | 401 статус-код |

Для негативного тестирования мы возьмём ранее полученный *JWT*, но к моменту негативного тестирования он уже является просроченным. Результат запроса представлен на рисунке 4.3.

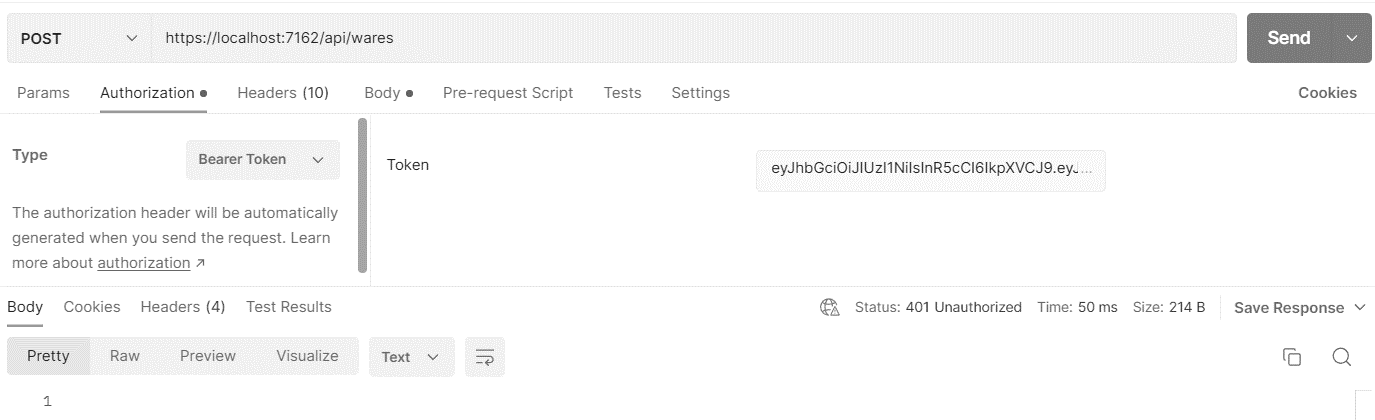


Рисунок 4.3 – Запрос на создание товара для негативного теста аутентификации

В результате мы получили ответ от сервера с 401 статус кодом, что соответствует ожидаемому результату и позволяет сделать вывод об успешном прохождении негативного тестирования аутентификацией программного средства.

По результатам позитивного и негативного тестирования можно сказать, что аутентификация в программном средстве работает конкретно.

### **4.1.2 Тестирование авторизации *REST API***

Авторизация в программном средстве также разработана на основе *JWT* и ролей. Всего было реализовано три роли: администратор, аудитор, пользователь. Для проверки авторизации проведём позитивное и негативное тестирование.

Начнём с позитивного тестирования авторизации. Данные для тестирования приведены в таблице 4.3.

Таблица 4.3 – Входные данные для позитивного тестирования авторизации

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Конечная точка | *HTTP* метод | Требуемая роль | Входные данные | Ожидаемый результат |
| *api*/*Users* | *POST* | Администратор | Данные добавляемого пользователя и *JWT* с ролью администратор в заголовке запроса | 200 статус-код, DTO данные созданного пользователя |

Шаг с получением *JWT* пропущен, так как он был протестирован ранее. Поэтому перейдём сразу к добавлению пользователя для которой требуется роль администратор. Результат запроса представлен на рисунке 4.4.

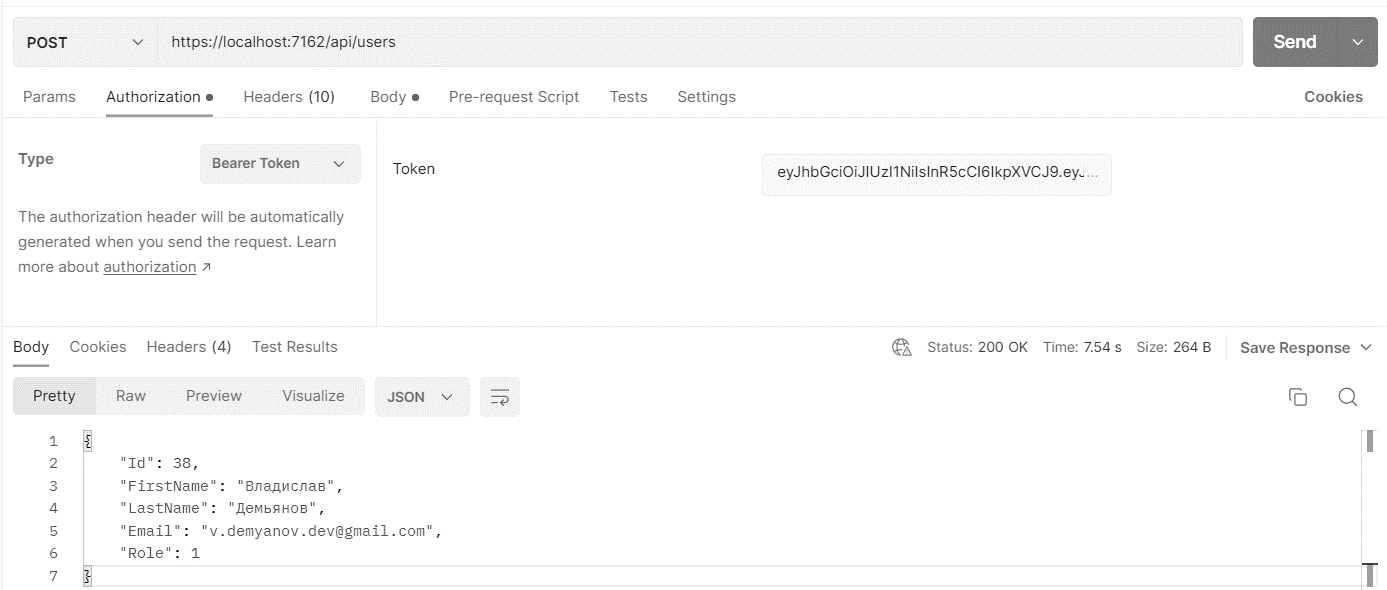


Рисунок 4.4 – Запрос на создание нового пользователя для позитивного теста авторизации

Из рисунка 4.4 можно увидеть, что запрос на добавления пользователя был выполнен и вернул в ответе *DTO* данные созданного пользователя со статус кодом 200. Результаты выполненного запроса соответствуют ожидаемому результату, поэтому можно сказать, что авторизация программного средства прошла позитивное тестирование.

Далее проведём негативное тестирование авторизации. Входные данные для тестирования представлены в таблице 4.4.

Таблица 4.4 – Входные данные для позитивного тестирования авторизации

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Конечная точка | *HTTP* метод | Требуемая роль | Входные данные | Ожидаемый результат |
| *api*/*Users* | *POST* | Администратор | Данные добавляемого пользователя и *JWT* с ролью работник в заголовке запроса | 403 статус-код |

Первым шагом нам нужно получить *JWT* с ролью отличной от роли администратор. Используем для этого пользователя, созданного в предыдущем тесте, который имеет роль работник. Этот шаг не будет показан, так как был протестирован ранее.

Вторым шагом сделаем запрос с полученным *JWT*, результат представлен на рисунке 4.5.

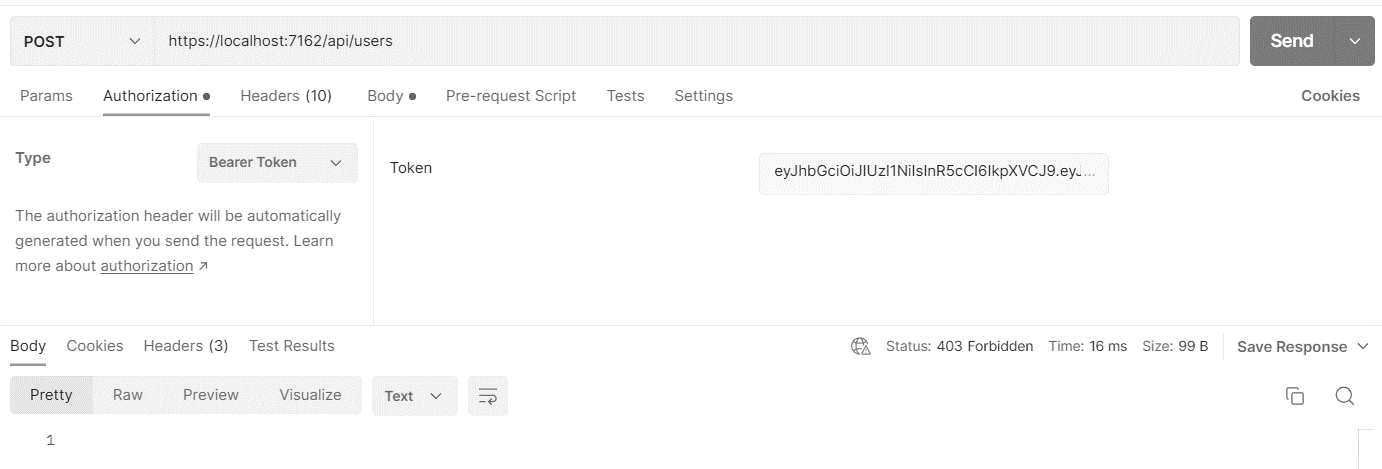


Рисунок 4.5 – Запрос на создание пользователя для негативного теста авторизации

Из рисунка 4.5 можно увидеть, что в результате мы получили от сервера 403 статус код, как и ожидалось. Это означает, что авторизация системы успешно прошла негативное тестирование.

По результатам позитивного и негативного тестирования можно сказать, что авторизация в программном средстве работает конкретно.

### **4.1.3 Тестирование добавления стеллажей в зону склада**

Склад для которого разрабатывается система управления имеет стеллажный способ укладки. Он состоит из зон, зоны в свою очередь состоят из стеллажей, стеллажи состоят из вертикальных секций, а вертикальные секции состоят из полок. Особенность данного склада заключается в том, что все полки на складе будут иметь одинаковый размер. Т.е. каждая зона, можно сказать, будет представлять собой трёхмерную матрицу, каждая ячейка которой будет иметь одинаковые размеры.

Для этого зона имеет ограничения, а именно свойства обозначающие максимальное количество вертикальных секций в зоне и максимальное количество полок в вертикальной секции. При добавлении стеллажей в зону, эти свойства будут учитываться во время проверки введённых пользователем данных, характеризующих добавляемый стеллаж. Для добавления стеллажа пользователю необходимо ввести количество вертикальных секций в стеллаже и количество полок в вертикальной секции стеллажа.

Т.е. система должна проверить не превышает ли количество секций в добавляемом стеллаже количество вертикальных секций уже имеющихся в зоне склада. А также проверить не превышает ли количество полок в добавляемом стеллаже максимальное количество полок в вертикальной секции зоны.

Для того, чтобы это протестировать воспользуемся методом граничных значений. Первым шагом определим граничные значения, они указаны в таблице 4.5.

Таблица 4.5 – Граничные значения для тестирования добавления стеллажа в зону

|  |  |  |  |
| --- | --- | --- | --- |
| Минимальное количество вертикальных секций в стеллаже | Максимальное количество вертикальных секций в зоне склада | Минимальное количество полок в вертикальной секции стеллажа | Максимальное количество полок в вертикальной секции склада |
| 1 | 15 | 0 | 10 |

Ожидаемый результат для позитивных тестов: созданный стеллаж и 201 статус код. Ожидаемый результат для негативных тестов: статус код 400.

Теперь перейдём к тестированию. Для начала проверим минимальное значение количества вертикальных секций в стеллаже: 1. И минимальное количество полок в вертикальной секции стеллажа: 0. Результат представлен на рисунке 4.6.

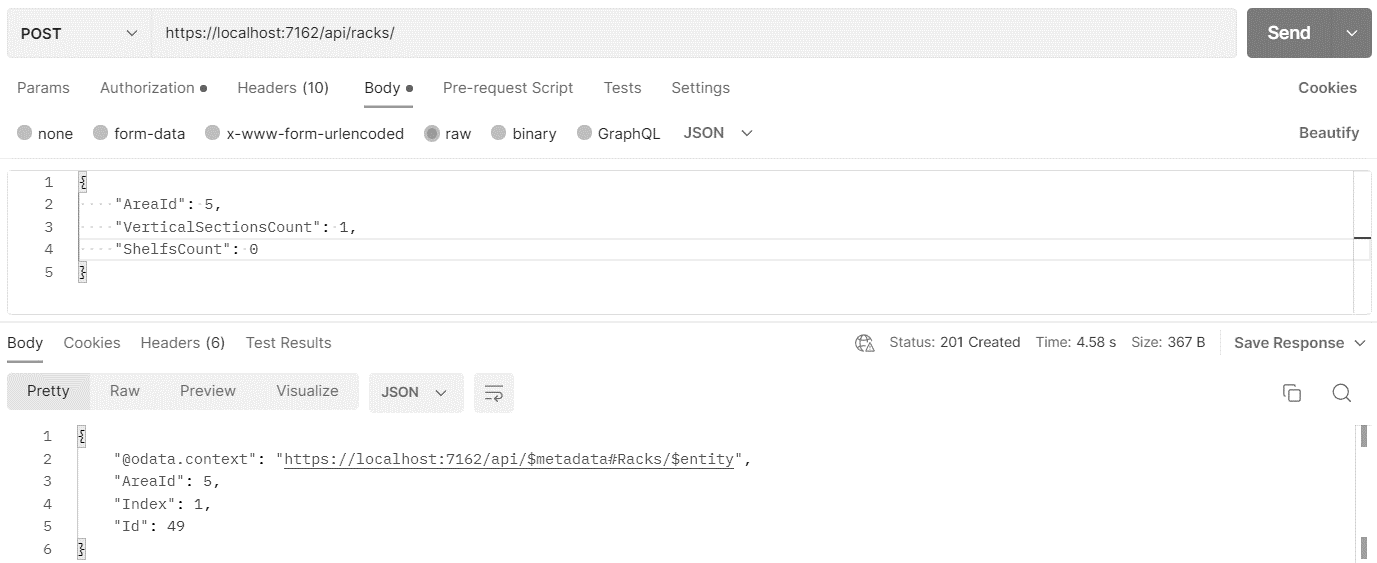


Рисунок 4.6 – Тестирование минимальных значений для добавления стеллажа в зону склада

Из рисунка 4.6 видно, что статус код ответа 201 и тело ответа содержит созданный стеллаж – это означает, что тест пройден.

Следующим шагом протестируем максимальное количество вертикальных секций в зоне склада: 15. И максимальное количество полок в вертикальной секции стеллажа: 10.

Стоит обратить внимание, что для корректности теста нужно взять не 15 секций, а 14, так как в зоне склада уже есть одна вертикальная секция. Результат представлен на рисунке 4.7.

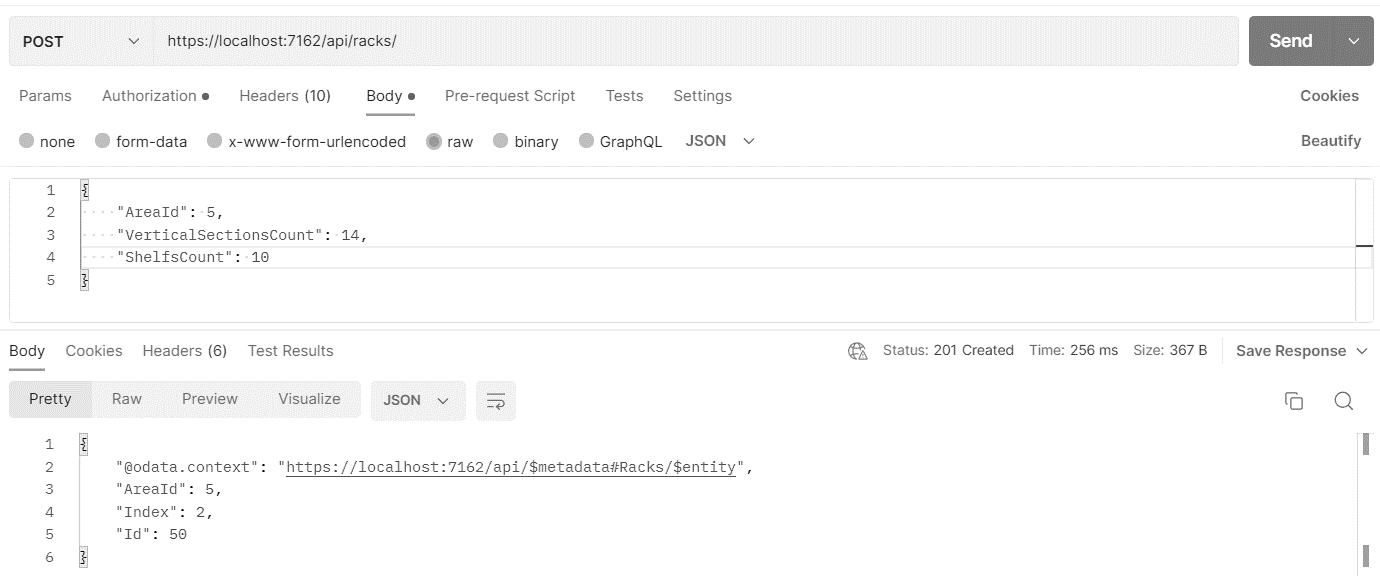


Рисунок 4.7 – Тестирование максимальных значений для добавления стеллажа в зону склада

Из рисунка 4.7 видно, что статус код ответа 201 и тело ответа содержит созданный стеллаж – это означает, что тест пройден.

Следующим шагом протестируем значение, меньше минимального для количества вертикальных секций: 0. Результат приведён на рисунке 4.8.

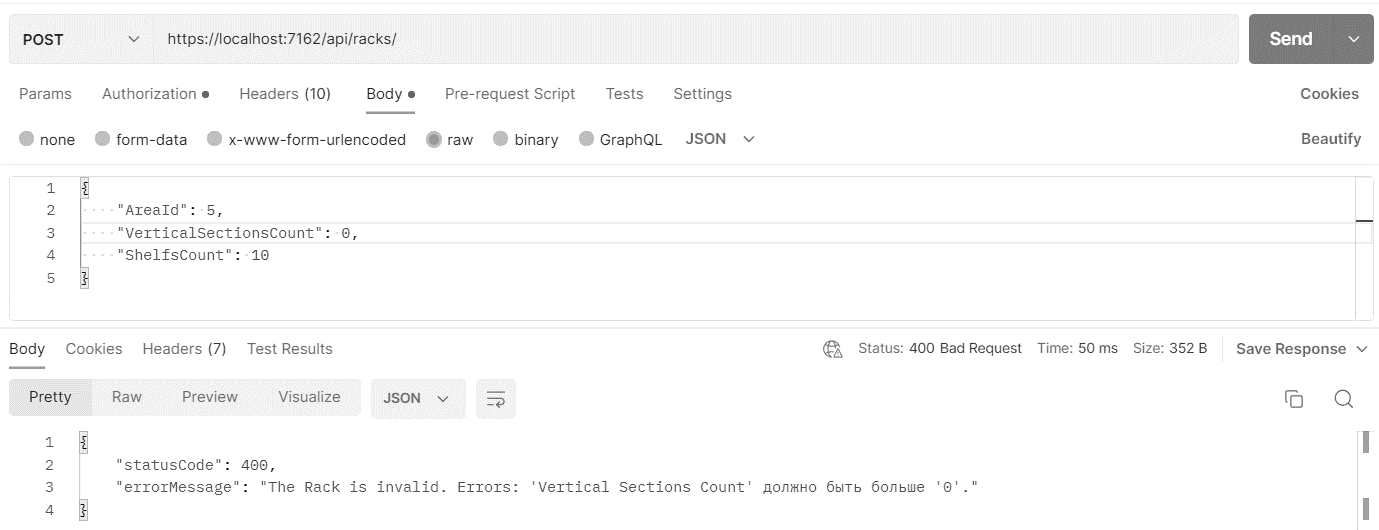


Рисунок 4.8 – Тестирование значения меньше минимального для количества вертикальных секций

Из рисунка 4.8 видно, что статус код ответа 400 – это означает, что тест пройден.

Следующим шагом протестируем значение, меньше минимального для количества полок: -1. Результат представлен на рисунке 4.9.

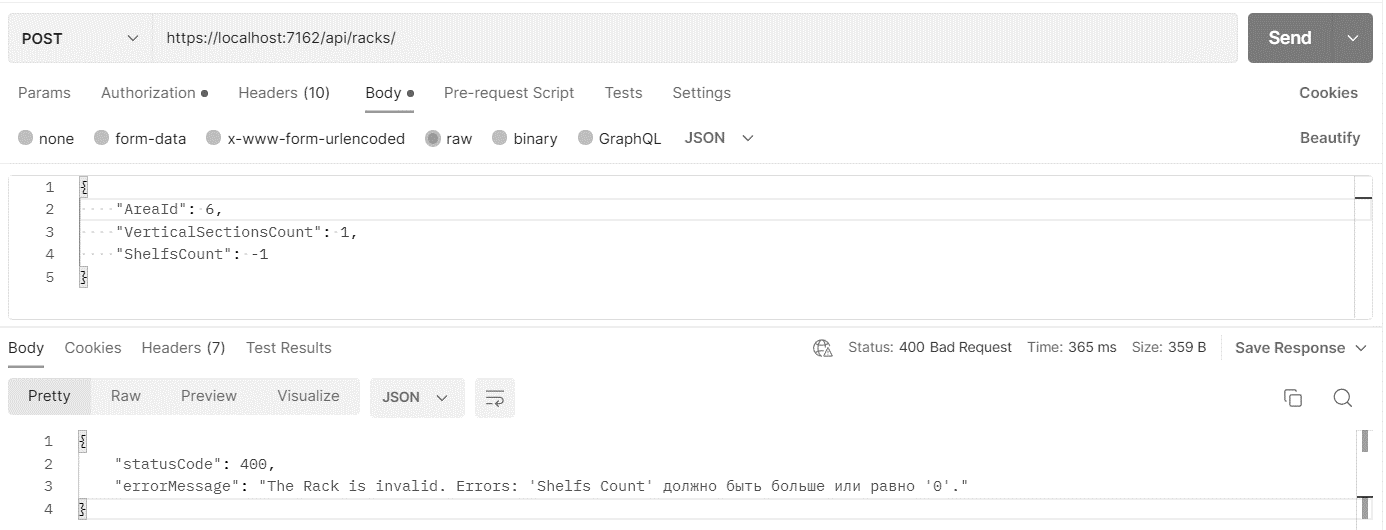


Рисунок 4.9 – Тестирование значения меньше минимального для количества полок в вертикальной секции

Из рисунка 4.9 видно, что статус код ответа 400 – это означает, что тест пройден.

Следующим шагом протестируем значение больше максимального для количества вертикальных секций: 16. И значение, больше максимального для количества полок: 11. Результат представлен на рисунке 4.10.

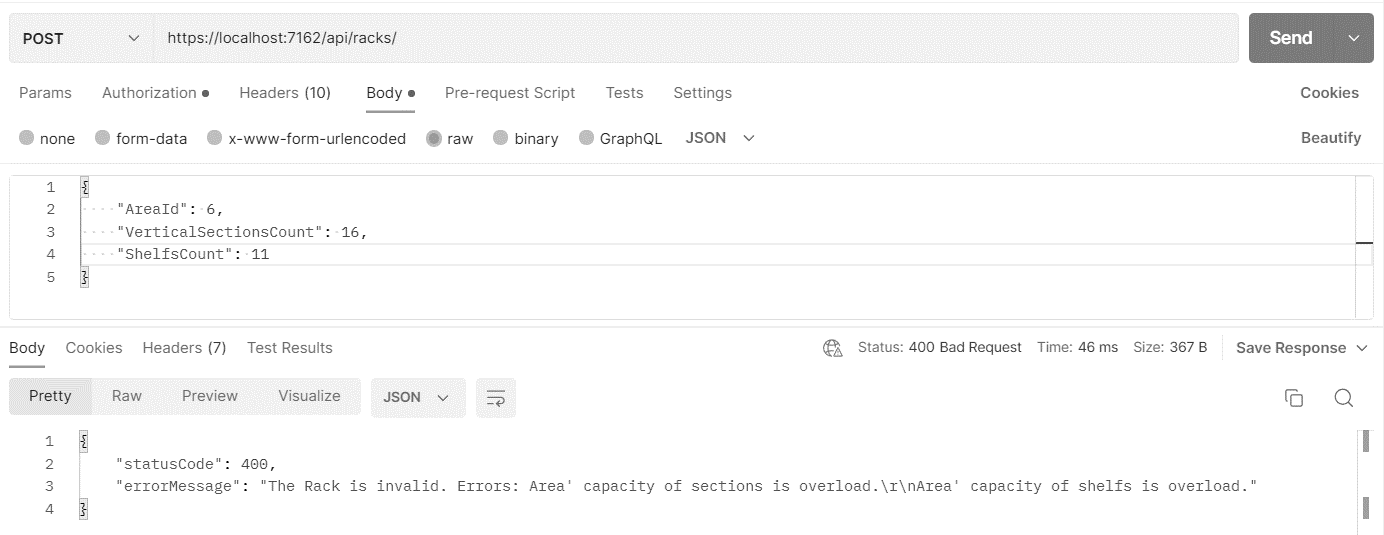


Рисунок 4.10 – Тестирование значения, больше максимального для добавления стеллажа в зону склада

Из рисунка 4.10 видно, что статус код ответа 400 – это означает, что тест пройден.

## **4.2 Тестирование пользовательского интерфейса**

Для тестирования пользовательского интерфейса будет проверяться ожидаемое поведение интерфейса. Также будет обращаться внимание на сообщения об ошибках во всплывающих окнах и при вводе данных поля формы.

### **4.2.1 Тестирование сообщений об ошибках при заполнении полей форм**

В форме на рисунке 4.11 некоторые поля являются обязательными: заголовок, статус, дата создания.

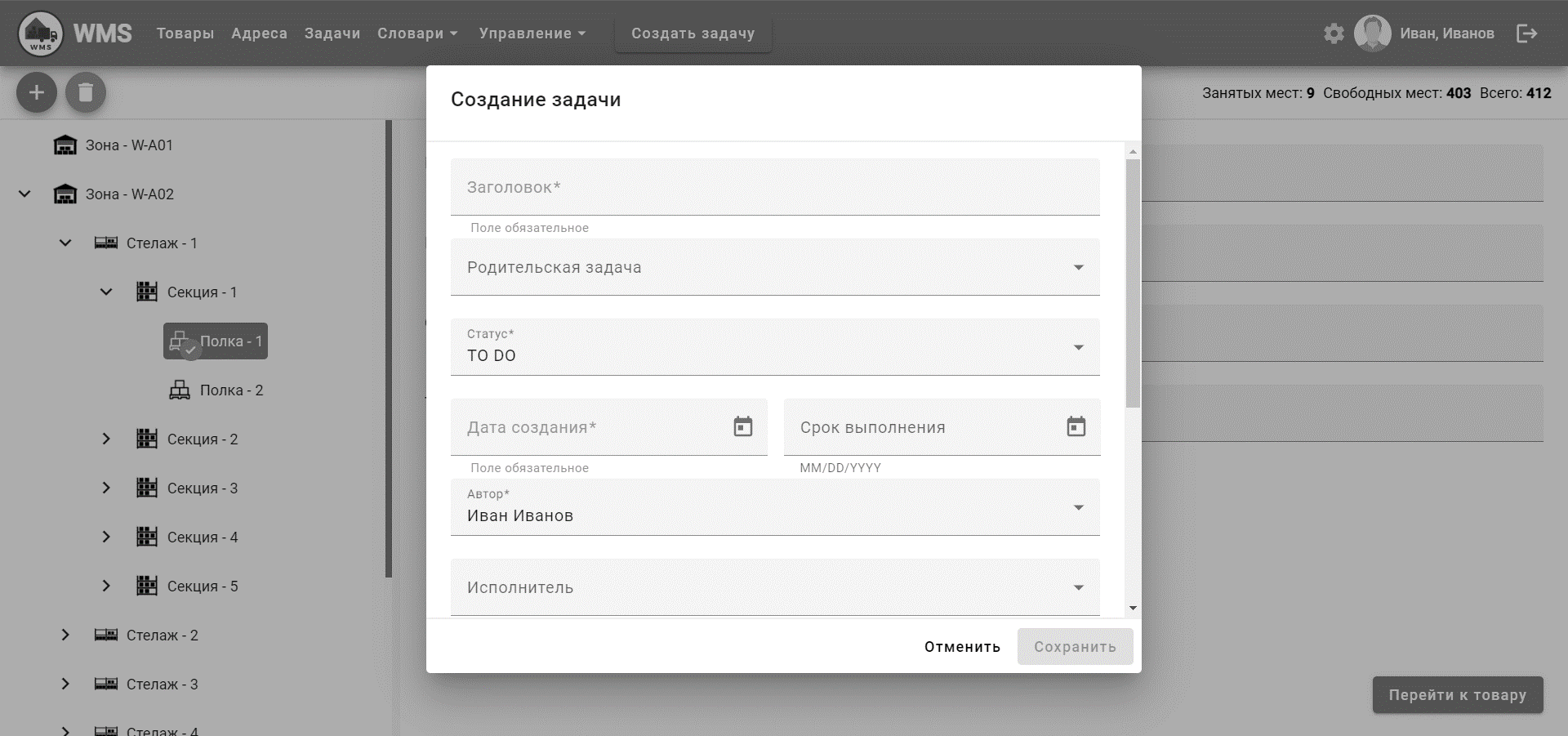


Рисунок 4.11 – Тестирование валидации полей форм

Для таких полей, если пользователь не ввёл значение, должно появиться предупреждение и кнопка для сохранения должна быть неактивна. Из рисунка 4.11 можно сделать вывод, что форма соответствует ожидаемому поведению.

### **4.2.2 Тестирование всплывающих сообщений для ошибок**

Ошибки, которые не обработаны на клиенте, должны быть обработаны на сервере. В таких случаях в приложении они отображаются во всплывающем окне. Пример такой ошибки представлен на рисунке 4.12.

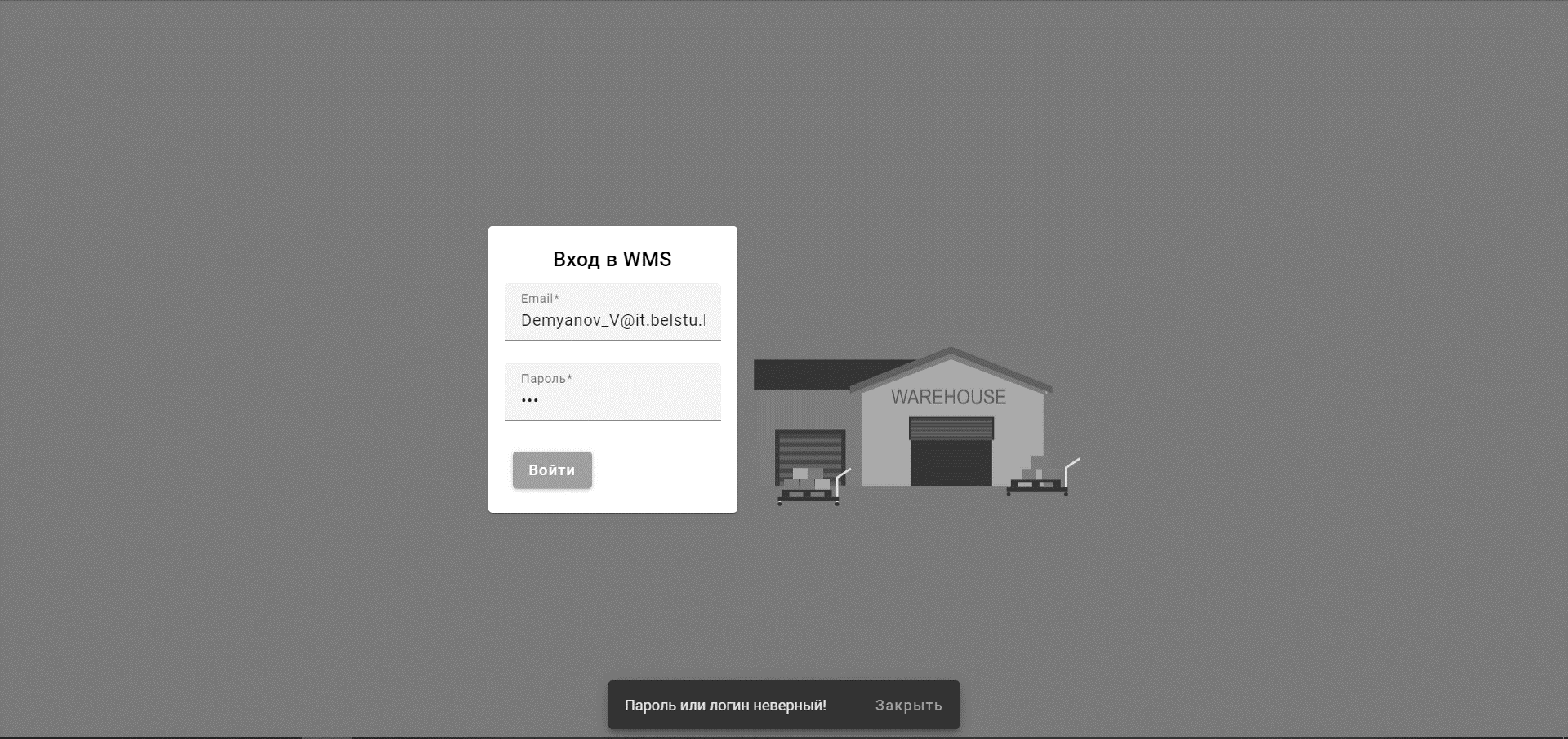


Рисунок 4.12 – Тестирование ошибок во всплывающем окне

На рисунке 4.12 представлено отображение ошибки во всплывающем окне, которое сообщает пользователю о неверно введённых данных при входе в приложение.

## **4.3 Вывод по разделу**

В данном разделе были предоставлены результаты тестирования разработанного программного средства. Было проведено тестирование *REST API* с использованием платформы *Postman* и тестирование пользовательского интерфейса.

Были использованы различные техники тестирования ПО: анализ граничных значений, позитивное и негативное тестирования.

Были протестированы валидационные сообщения в полях форм, а также во всплывающих окнах.

К недостаткам разработанного приложения можно отнести отсутствие модульных и интеграционных тестов, это то, что можно доработать на этапе поддержки. Исправление такого недостатка позволит в будущем облегчить поддержку приложения и помочь стабилизировать его, а также увеличить оценку качества и доверие пользователей.

Полученные результаты проведённого тестирования показали, что программное средство работает корректно и в соответствии с требованиями.

5 Методика установки и использования

ФИО

Подпись

Дата

Лист

1

БГТУ 05.00.ПЗ

Разраб.

*Демьянов В.Р.*

Провер.

*Парамонов А.И.*

Н. контр.

Коренькова А.А.

Утверд.

*Пацей Н.В.*

Лит.

Листов

13

*74417027, 2023*

## **5.1 Руководство по использованию миграций**

5 Методика установки и использования

Консульт.

Чтобы поддерживать синхронизацию схемы базы данных с моделью данных программного средства в *EF* существуют функции миграции.

Во время разработки было принято решение вынести файлы миграции в отдельный проект – библиотеку классов. Рекомендуется соблюдать созданную архитектуру и применять команды для миграции в конфигурации, представленной в данном разделе.

Некоторые команды из данного руководства также будут использоваться в руководстве по развёртыванию приложения.

### **5.1.1 Создание новой миграции**

После обновления модели данных программного средства для синхронизации с базой данных первым шагом необходимо создать новую миграцию. Конфигурация команды для добавления новой миграции представлена в листинге 5.1.

|  |
| --- |
| dotnet ef migrations add MigrationName --project WMS.Database.Migrations --startup-project WMS.WebApi |

Листинг 5.1 – Конфигурация команды для добавления новой миграции

Параметр *--project* указывает проект с файлами миграций.

Параметр *--startup-project* указывает запускаемый проект.

### **5.1.2 Синхронизация схемы базы данных с моделью данных программного средства**

Конфигурации команды для синхронизации схемы базы данных с моделью данных программного средства представлена в листинге 5.2.

|  |
| --- |
| dotnet ef database update --project WMS.Database.Migrations --startup-project WMS.WebApi -- --mode=design-time |

Листинг 5.2 – Конфигурация команды для синхронизации схемы базы данных с моделью данных программного средства

Параметр *--project* указывает проект с файлами миграций.

Параметр *--startup-project* указывает запускаемый проект.

Маркер «--» указывает *dotnet ef* перенаправлять нераспознанные параметры приложению.

### **5.1.3 Генерация *SQL* скрипта с миграциями**

Для применения миграции существуют различные стратегии, некоторые из которых больше подходят для производственных сред, а другие – для жизненного цикла разработки. В предыдущем пункте был рассмотрен вариант больше подходящий для жизненного цикла разработки. Однако для производственной базы данных больше подходит подход с генерацией *SQL* скрипта.

Конфигурация команды для создания скрипта с миграциями для пустой базы данных представлена в листинге 5.3.

|  |
| --- |
| dotnet ef migrations script --project WMS.Database.Migrations --startup-project WMS.WebApi --output ./migrations.sql -- --mode=design-time |

Листинг 5.3 – Конфигурация команды для генерации скрипта с миграциями для пустой базы данных

Параметр *--project* указывает проект с файлами миграций.

Параметр *--startup-project* указывает запускаемый проект.

Маркер «--» указывает *dotnet ef* перенаправлять нераспознанные параметры приложению.

Параметр --*output* указывает путь к генерируемому файлу.

Для того, чтобы сгенерировать *SQL* скрипт, содержащий определённый диапазон миграций, необходимо указать названия миграций из *from* в *to*.

Для отката миграций можно использовать *from* новее, чем *to*.

## **5.2 Руководство развёртывания программного средства в среде разработки**

Первым шагом необходимо развернуть базу данных. Для это на машине или в Docker контейнере должен быть установлен *MS SQL Server 19*. Если это условие выполнено, то создадим пустую базу данных. Скрипт для создания базы данных представлен в листинге 5.4.

|  |
| --- |
| USE master;  GO  CREATE DATABASE WMS;  GO |

Листинг 5.4 – Скрипт для создания базы данных среды разработки

Следующим шагом нужно добавить строку подключения к созданной базе данных в *appsettings.Development.json* или *.NET User Secrets*. Фрагмент из .*NET User Secrets* представлен в листинге 5.5.

|  |
| --- |
| "ConnectionStrings:WMSDatabase": "Server=VDemyanov;Database=WMS;Trusted\_Connection=true;" |

Листинг 5.5 – Фрагмент из .NET User Secrets для конфигурации строки подключения к базе данных среды разработки

Следующим шагом является добавления в *csv* файлы данных для инициализации базы данных. Для корректной работы программного средства необходимо добавить такие данные в файл «*Settings.csv*» и «*Users initial data.csv*», они представлены в листингах 5.6 и 5.7 соответственно.

|  |
| --- |
| Key,Value  ProblemExpirationNotificationDays,1  ShippedWaresStorageDays,1 |

Листинг 5.6 – Данные для инициализации настроек

|  |
| --- |
| Id,FirstName,LastName,Email,Role,InitialPassword  1,Vladislav,Demyanov,system.wms@outlook.com,Administrator,123456 |

Листинг 5.7 – Данные для инициализации пользователей

Следующим шагом необходимо добавить новую миграцию и применить миграции к созданной базе данных одним из способов, описанных в предыдущем подразделе.

Следующим шагом необходимо добавить конфигурацию для *SMTP*. Фрагмент конфигурации из *.NET User Secrets* представлен в листинге 5.8.

|  |
| --- |
| "MailSettings": {  "SystemEmailAddress": "",  "SystemEmailPassword": "",  "SmtpHost": "smtp-mail.outlook.com",  "SmtpPort": 587,  "EnableSsl": true,  "UseDefaultCredentials": false  } |

Листинг 5.8 – Конфигурация SMTP

Теперь программное средство готово к запуску. Для этого нам нужно запустить два приложения *WMS.WebApi* и *WMS.Frontend*.

Конфигурация запуска для *WMS.Frontend* представлена на рисунке 5.1.

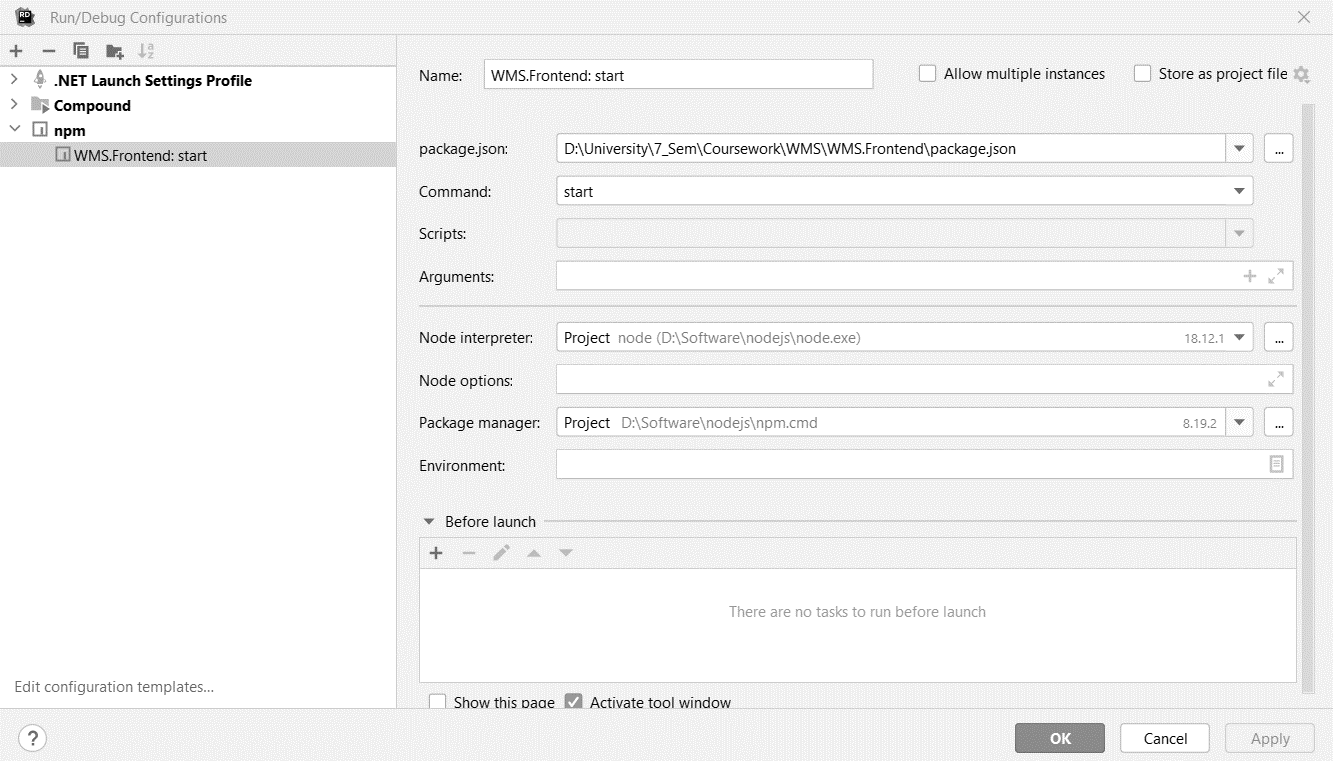


Рисунок 5.1 – Конфигурация запуска приложения WMS.Frontend

Конфигурация запуска приложения *WMS.WebApi* представлена на рисунке 5.2.

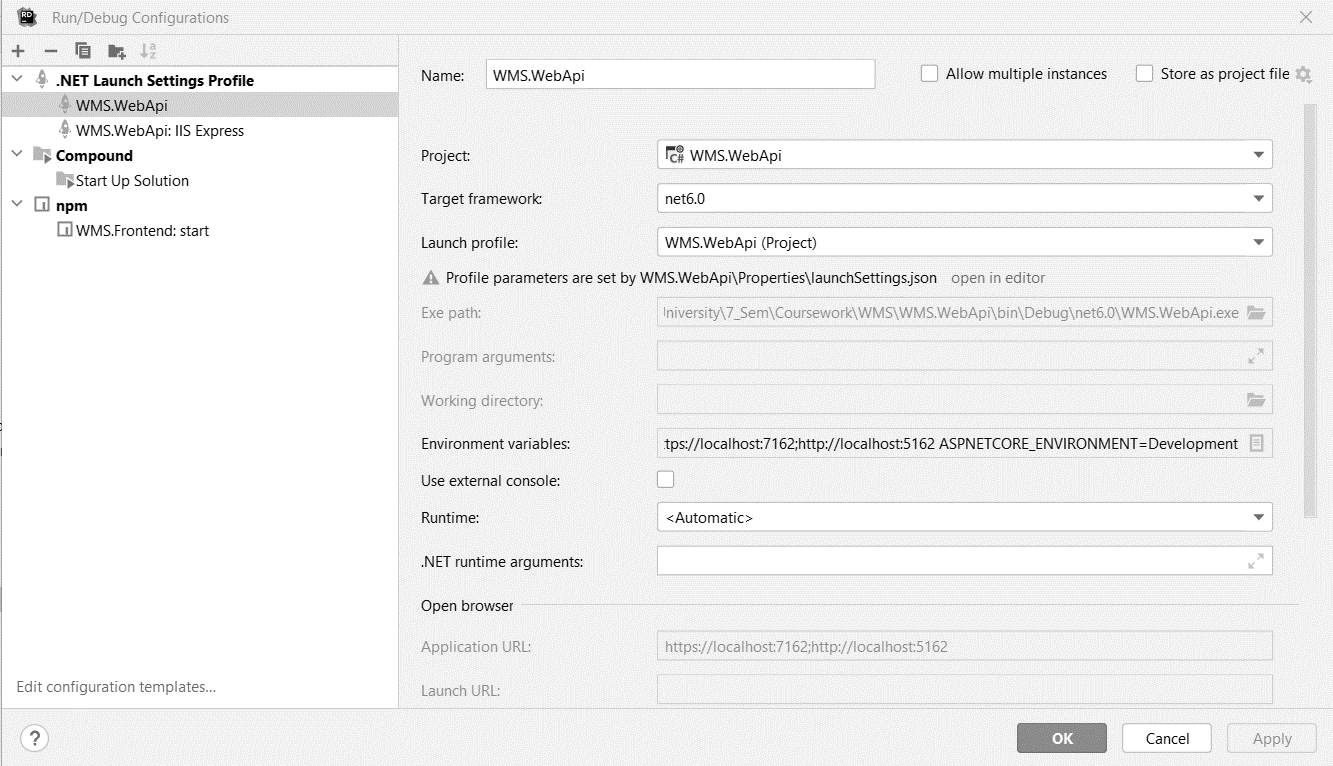


Рисунок 5.2 – Конфигурация запуска приложения WMS.WebApi

В результате будут запущены два приложения *WMS.WebApi* и *WMS.Frontend*.

## **5.3 Руководство развёртывания программного средства в производственной среде**

В производственной среде программное средство развёртывается в *Docker Network* в трёх контейнерах: *database*, *api*, *frontend*. Для управления контейнерами используется *Docker Compose*.

### **5.3.1 Руководство по конфигурации развёртывания контейнера frontend**

*Docker* файл для развёртывания *frontend* контейнера представлен в листинге 5.9.

|  |
| --- |
| ### STAGE 1: BUILD  FROM node:18.12.1 AS build  WORKDIR /usr/src/app  COPY package.json package-lock.json ./  RUN npm install  COPY . .  RUN npm run build --prod  #STAGE 2: RUN  FROM nginx:1.17.1-alpine  COPY ./nginx.conf /etc/nginx/nginx.conf  COPY --from=build /usr/src/app/dist/wms-frontend /usr/share/nginx/html |

Листинг 5.9 – Конфигурация Docker файла для контейнера frontend

В первой стадии разворачивается *Node JS* и билдится приложение. Для это выполняются шаги, описанные ниже.

Первым шагом определим родительский образ с помощью инструкции «*FROM node:18.12.1 AS build*». Это будет образ *nodejs* версии 18.12.1.

Вторым шагом определим рабочую папку разворачиваемому контейнеру с помощью инструкции «*WORKDIR /usr/src/app*».

Третьим шагом скопируем файлы *package.json* и *package-lock.json* из локального корневого каталога с помощью инструкции «*COPY package.json package-lock.json ./*».

Четвёртым шагом необходимо установить пакеты из *package.json*. Для этого выполняется инструкция «*RUN npm install*».

Пятым шагом скопируем остальные файлы с исходным кодом с помощью инструкции «*COPY . .*».

Последним шагом запустим билд приложения с помощью инструкции «*RUN npm run build --prod*».

Во второй стадии полученный билд на первой стадии запускается на *NGINX*. Для это выполняются шаги, описанные ниже.

Первым шагом определяется родительский образ с помощью инструкции «*FROM nginx:1.17.1-alpine*». Это будет образ *NGINX* версии 1.17.1.

Вторым шагом копируем файл конфигурации *NGINX* (листинг можно найти в приложении Ж). Для этого выполняется инструкция «*COPY ./nginx.conf /etc/nginx/nginx.conf*».

Третьим шагом сборка приложения копируется на *NGINX* сервер.

### **5.3.2 Руководство по конфигурации развёртывания контейнера *api***

Для развёртывания контейнера *api* был разработан *Docker* файл. Полный листинг Docker файла представлен в приложении И. В этом пункте будет кратко описана последовательность основных инструкций:

* используется базовый образ *aspnet* версии 6.0;
* устанавливается каталог приложения *docker* по умолчанию;
* устанавливается порт контейнера по умолчанию;
* добавляется *.net SDK* версии 6.0 для сборки приложения;
* копируется все необходимое исходные файлы с хоста в каталог *docker*;
* создаётся билд приложения;
* созданный билд публикуется на *Kestrel*;
* устанавливается точка входа *WMS.WebApi.dll*.

### **5.3.3 Руководство по развёртыванию и запуску контейнеров с использованием *Docker Compose***

Для развёртывания и запуска сконфигурированных контейнеров с помощью *Docker Compose* был разработан файл «*docker-compose.yml*». Листинг данного файла представлен в приложении К. С помощью *Docker Compose* мы конфигурируем контейнеры в сеть.

Для развёртывания базы данных используется готовый образ с *MS SQL Server* 2022. Также для конфигурации строки подключения необходимо добавить в решение файл «*database.env*». Пример содержимого файла представлен в листинге 5.10.

|  |
| --- |
| SA\_PASSWORD: "S3cur3P@ssW0rd!"  ACCEPT\_EULA: "Y" |

Листинг 5.10 – Пример содержимого файла конфигурации базы данных

После выполнения всех конфигураций, указанных в этом и предыдущих пунктах подраздела, необходимо выполнить сборку контейнеров. Это можно сделать с помощью команды «*docker-compose build*».

Теперь, когда контейнеры собраны, пришло время их запустить. Для этого необходимо выполнить команду «*docker-compose up*».

После того как контейнер с сервером базы данных был развёрнут, необходимо развернуть саму базу данных. Для этого рекомендуется сгенерировать *SQL* скрипт с миграциями и накатить его на созданную базу данных. Как сгенерировать *SQL* скрипт с миграциями описано ранее.

Также для корректной работы программного средство необходимо не забыть добавить конфигурацию *SMTP*, как это было сделано в руководстве по развёртыванию программного средства в среде разработки.

## **5.4 Руководство по использованию GUI**

df

6 Технико-экономическое обоснование проекта

ФИО

Подпись

Дата

Лист

1

БГТУ 06.00.ПЗ

Разраб.

*Демьянов В.Р.*

Провер.

*Парамонов А.И.*

Н. контр.

Коренькова А.А.

*.*

Утверд.

*Пацей Н.В.*

6 Технико-экономическое обоснование проекта

Лит.

Листов

8

*74417027, 2023*

## **6.1 Общая характеристика разрабатываемого программного средства**

Семенова Л.С.

Консульт.

Конс..

Разработанное программное средство предназначено для управления складом со стеллажным способом укладки. Целевой аудиторией программного средства в первую очередь являются сотрудники склада, а именно администраторы, аудиторы и работники. Под их работу разработана текущая версия пользовательского интерфейса. В будущих версиях планируется доработать пользовательский интерфейс для клиентов склада, на данном этапе для них доступно *REST API*.

Программное средство отличается от аналогов тем, что оно в первую очередь нацелено на управление именно складом со стеллажным способом укладки. Как правило, аналогичные системы управления складом имеют широкий охват функционала, что делает их дорогими для малых складов.

Серверная часть приложения разработана с использованием платформы *ASP.NET* 6, а также ряда других *.NET* технологий и библиотек, таких как *EF* 6, *FluentValidation* 11.5.2, *Hangfire* 1.8.1, *OData* 8.2, *Serilog* 6.0.1, *CsvHelper* 30.0.1.

Клиентская часть приложения разработана с использование *JavaScript* фреймворка *Angular* 15, библиотеки *Angular Material* 15.0.1.

Для хранения данных разработана реляционная база данных под управлением СУБД *MS SQL Server*.

Приложение развёрнуто на платформе *Docker*.

Стратегия монетизации предполагает разовую покупку с последующей поддержкой.

## **6.2 Исходные данные для проведения расчётов и маркетинговый анализ**

Источниками исходных данных для расчётов выступают действующие законы и нормативно-правовые акты. Исходные данные для расчёта приведены в таблице 6.1.

Таблица 6.1 – Исходные данные для расчёта

|  |  |  |
| --- | --- | --- |
| Наименование показателя | Условные  обозначения | Норматив |
| Численность разработчиков, чел | Чр | 1 |
| Норматив дополнительной заработной платы, % | Ндз | 15,00 |
| Ставка отчислений в Фонд социальной защиты населения, % | Нфсзн | 34,00 |

*У*

*Лист*

*Изм.*

Окончание таблицы 6.1

|  |  |  |
| --- | --- | --- |
| Наименование показателю | Условные  обозначения | Норматив |
| Ставка отчислений по обязательному страхованию в БРУСП «Белгосстрах», % | Нбгс | 0,60 |
| Норматив прочих прямых затрат, % | Нпз | 28,00 |
| Норматив накладных расходов, % | Нобп, обх | 60,00 |
| Норматив расходов на сопровождение и адаптацию, % | Нрса | 10,00 |
| Ставка НДС, % | ННДС | 0,00 |
| Налог на прибыль, % | НП | 0,00 |

В ходе проведения маркетингового анализа, была определена стоимость разработки аналогичного программного продукта по управлению складом. Средняя цена разработки *REST API* для такой системы стоит 25 000,00 руб. Средняя цена разработки веб-клиента стоит 10 000,00 руб. Таким образом, общая стоимость разработки программного средства, выбранного в качестве базы сравнения составит 35 000,00 руб.

## **6.3 Обоснование цены программного средства**

Широкое применение вычислительных технологий требует постоянного обновления и совершенствования ПС. Выбор эффективных проектов ПС связан с их экономической оценкой и расчетом экономического эффекта, который может определяться как для разработчика, так и для пользователя.

У разработчика экономический эффект выступает в виде чистой прибыли от реализации ПС, остающейся в распоряжении организации, а у пользователя – в виде экономии трудовых, материальных и финансовых ресурсов, получаемой за счет:

− снижения трудоемкости расчетов и алгоритмизации программирования и отладки программ;

− сокращения расходов на оплату машинного времени и других ресурсов на отладку программных средств;

− снижения расходов на материалы;

− оптимизации программных средств;

− улучшения показателей основной деятельности в результате использования передовых программных средств.

Стоимостная оценка программных средств у разработчиков предполагает определение затрат, что включает следующие статьи:

− заработная плата исполнителей – основная и дополнительная;

− отчисления в фонд социальной защиты населения;

− отчисления по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний;

− прочие прямые затраты;

− накладные расходы.

На основании затрат рассчитывается себестоимость и отпускная цена конечного программного средства.

6.3.1 Расчет затрат рабочего времени на разработку программного средства

В таблице 6.2 в укрупнённом виде указаны все работы, реально выполненные для создания, указанного в дипломной работе программного средства и количество часов, реально потраченных для выполнения этих работ.

Таблица 6.2 – Затраты рабочего времени на разработку ПС

|  |  |
| --- | --- |
| Содержание работ | Затраты рабочего времени, часов |
| Обзор аналогов, прототипов, анализ предметной области | 16,00 |
| Разработка функциональных требований | 32,00 |
| Проектирование архитектуры программного средства | 120,00 |
| Реализация программного средства | 128,00 |
| Тестирование и отладка | 64,00 |
| Всего | 360,00 |

Всего на разработку программного средства ушло 360 часов.

### **6.3.2 Расчёт основной заработной платы**

Для определения величины основной заработной платы, было проведено исследование величин заработных плат *full-stack* веб-разработчиков на *.NET* и любом современном *frontend* фреймворке (*Angular*, *React*, *Vue*) на предприятии, где проходилась преддипломная практика. В итоге было установлено, что средняя месячная заработная плата на позиции *middle* составляет 3 500,00 руб, а часовая ставка соответственно 20,83 руб/час (если разработчик не находится в ожидании активного проекта).

Согласно таблице 6.2, проект разрабатывался одним специалистом на протяжении 360,00 часов. Основная заработная плата будет рассчитываться по формуле 6.1.

|  |  |
| --- | --- |
|  | (6.1) |

где Соз – основная заработная плата, руб.;

Траз – время разработки, месяцев;

Сзп – средняя месячная заработная плата;

Краз – количество разработчиков, человек.

 руб.

Основная заработная плата, рассчитана исходя из данных представленных в таблице 6.1.

### **6.3.3 Расчёт дополнительной заработной платы**

Законодательство о труде предусматривает наличие выплат, которые определяются по нормативу в процентах к основной заработной плате по формуле 6.2.

|  |  |
| --- | --- |
|  | (6.2) |

где Соз – основная заработная плата, руб.;

Ндз – норматив дополнительной заработной платы, %.

 руб.

Исходя из основной заработной платы, а также норматива дополнительной заработной платы, можно рассчитать сумму дополнительной заработной платы.

### **6.3.4 Расчёт отчислений в Фонд социальной защиты населения и по обязательному страхованию**

Отчисления в Фонд социальной защиты населения (ФСЗН) определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей и вычисляются по формуле 6.3.

|  |  |
| --- | --- |
|  | (6.3) |

где Соз – основная заработная плата, руб.;

Сдз – дополнительная заработная плата на конкретное ПС, тыс. руб.;

Нфсзн – норматив отчислений в Фонд социальной защиты населения, %.

Отчисления в БРУСП «Белгосстрах» вычисляются по формуле 6.4.

|  |  |
| --- | --- |
|  | (6.4) |

Вычисление отчисления в Фонд социальной защиты населения:

 руб.

Вычисление отчисления в БРУСП «Белгосстрах»:

 руб.

Таким образом, общие отчисления в БРУСП «Белгосстрах» составили 51,74 руб., а в фонд социальной защиты населения – 2 932,03 руб.

### **6.3.5 Расчёт суммы прочих прямых затрат**

Так как разработка дипломного проекта велась не в коммерческих целях, а в целях обучения, то использовались бесплатные лицензии на ПО, предоставляемые студентам. Однако, при разработке в коммерческих целях такие затраты необходимо учитывать, они представлены в таблице 6.3.

Таблица 6.3

|  |  |
| --- | --- |
| Содержание затрат | Значение, руб. |
| Оплата лицензии интегрированной среды разработки *JetBrains Rider* | 112,79 |
| Оплата лицензии интегрированной среды разработки *JetBrains WebStorm* | 120,36 |
| Оплата хостинга | 300,00 |
| Покупка доменного имени | 91,00 |
| Покупка сертификата безопасности | 61,00 |

Затраты на оплату лицензий *JetBrains Rider*, *WebStorm*, хостинг и домен рассчитаны на определённый период, равный округлённому количеству месяцев, потраченных на разработку проекта. Сюда не входят прямые прочие затраты, которые будут на этапе поддержки.

Сумма прочих затрат составит:

 руб.

### **6.3.6 Расчёт суммы накладных расходов**

Сумма накладныхрасходовСобп,обх – произведение основной заработной платы исполнителей на конкретное программное средство Соз на норматив накладных расходов в целом по организации Нобп,обх, находится по формуле 6.5.

|  |  |  |
| --- | --- | --- |
|  |  | (6.5) |

Сумма накладных расходов составит:

 руб.

### **6.3.7 Сумма расходов на разработку программного средства**

Нужно определить сумму расходов на разработку программного средства Ср. Сумма расходов на разработку программного средства Ср определяется как сумма основной и дополнительной заработных плат исполнителей на конкретное программное средство, отчислений на социальные нужды, суммы прямых прочих затрат и суммы накладных расходов, по формуле 6.6.

|  |  |
| --- | --- |
|  | (6.6) |

Все данные необходимые для вычисления есть, поэтому можно определить сумму расходов на разработку программного средства.

 руб.

Сумма расходов на разработку программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе, и составила 16 791,82 руб.

### **6.3.8 Расходы на сопровождение и адаптацию**

Сумма расходов на сопровождение и адаптацию программного средства Срса определяется как произведение суммы расходов на разработки на норматив расходов на сопровождение и адаптацию Нрса, по формуле 6.7.

|  |  |  |
| --- | --- | --- |
|  |  | (6.7) |

Сумма расходов на сопровождение и адаптацию составит:

 руб.

Все проведенные выше расчеты необходимы для вычисления полной себестоимости проекта.

### **6.3.9 Расчёт полной себестоимости**

Полная себестоимость Сп определяется как сумма двух элементов: суммы расходов на разработку Ср и суммы расходов на сопровождение и адаптацию программного средства Срса.

Полная себестоимость Сп вычисляется по формуле 6.8.

|  |  |  |
| --- | --- | --- |
|  |  | (6.8) |

Полная себестоимость составит:

 руб.

Полная себестоимость программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе.

### **6.3.10 Определение цены, оценка эффективности**

Расчёт отпускной цены будем делать, опираясь среднюю рыночную цену аналогов. Средняя рыночная цена составляет 35 000 руб. Ставка НДС и налог на прибыль не учитываются, так как организация является резидентом ПВТ.

С учёт исходных данных, прибыль от реализации программного средства будем рассчитывать по формуле 6.9.

|  |  |  |
| --- | --- | --- |
|  |  | (6.9) |

где Ппс – прибыль от реализации программного средства, руб.;

Цр –средняя рыночная цена продукта, руб.;

Сп – полная себестоимость программного средства, руб.

Прибыль от реализации программного средства составит:

 руб.

Уровень рентабельности разработанного программного средства определяется по формуле 6.10.

|  |  |  |
| --- | --- | --- |
|  |  | (6.10) |

где Урент – уровень рентабельности программного средства, %;

Сп – полная себестоимость программного средства, руб.;

Ппс – прибыль от реализации программного средства, руб.

Уровень рентабельности составит:

 %.

Рентабельность разработки программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе.

## **6.4 Вывод по разделу**

В таблице 6.4 представлены результаты расчетов для основных показателей данного раздела.

Таблица 6.4 – Результаты расчётов

|  |  |
| --- | --- |
| Наименование показателя | Значение |
| Время разработки, ч. | 360,00 |
| Количество программистов, чел. | 1 |
| Основная заработная плата, руб. | 7 498,80 |
| Дополнительная заработная плата, руб. | 1 124,82 |
| Отчисления в БРУСП «Белгосстрах», руб. | 51,74 |
| Отчисления в Фонд социальной защиты населения, руб. | 2 932,03 |
| Прочие прямые затраты, руб. | 685,15 |
| Накладные расходы, руб. | 4 499,28 |
| Себестоимость разработки программного средства, руб. | 16 791,82 |
| Расходы на сопровождение и адаптацию, руб. | 1 679,18 |
| Полная себестоимость, руб. | 18 471,00 |
| Прибыль от реализации программного средства, руб. | 16 582,99 |
| Рентабельность разработки, % | 89,48 |

Необходимость разработки программного средства обусловлена выявленным спросом на автоматизацию бизнес процессов склада и отсутствием доступных по цене решений для малых и средних складов.

Разработанное программное средство должно позволить управлять складом со стеллажным типом укладки, автоматизируя следующие функции:

* приёмка товара;
* размещение в зонах хранения;
* отгрузка товара;
* инвентаризация;
* учёт товара;
* оптимизация процесса хранения;
* управление персоналом;
* управление задачами склада;
* уведомление сотрудников о событиях склада.

Разработка программного средства осуществлялась одним программистом. Суммарные временные затраты составили 360,00 часов, а полная себестоимость программного средства составила 18 471,00 руб. Опираясь на среднюю цену разработки подобных программных средств на рынке, рентабельность должна составить 89,48%.

Из этого можно сделать вывод, что разработка системы управления складом является непростой и недешёвой работой, а это ещё нужно учитывать, что кроме веб-клиента и *REST API*, разработанных в дипломном проекте, полноценные системы управления складом имеют множество других элементов таких как: система аналитики, отчётности, мобильные клиенты, модули обмена данным и т.д. Цена полноценной системы управления начинается от 100 000,00 руб., а верхняя граница может уходить в миллионы рублей. Не смотря на большую цену, такое программное средство может оправдать себя в долгосрочной перспективе, внедрение даёт бизнесу огромные преимущества, начиная от возможности быстрого принятия решений, на основе предоставляемых данных и до значительного сокращения издержек при выполнении заказов и затрат на их обработку. И бизнесу, который хочет на долго закрепиться на конкурентном рынке, необходимо внедрять такие решения.

В результате разработанное программное средство должно быть конкурентоспособным на рынке.

Заключение

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Демьянов В.Р.*

Провер.

*Парамонов А.И.*

Н. контр.

Коренькова А.А.

Утверд.

*Пацей Н.В.*

Заключение

Лит.

Листов

1

*74417027, 2023*

Консульт.

Конс..

Список использованных источников

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Демьянов В.Р.*

Провер.

*Парамонов А.И.*

Н. контр.

Коренькова А.А

Утверд.

*Пацей Н.В.*

Список использованных источников

Лит.

Листов

1

*74417027, 2023*

1 Бухта [Электронный ресурс]. – Режим доступа: https://www.buhta.ru/sistema-upravleniya-skladom-wms/description-of-system/494-obschee-opisanie-biznes-processov.html – Дата доступа 17.05.2023.

2 Метал Сити [Электронный ресурс]. – Режим доступа: https://www.metalcity.ru/info/articles/hranenie\_tovara\_na\_skladah/ – Дата доступа: 17.05.2023.

3 NetSuite [Электронный ресурс]. – Режим доступа: https://www.netsuite.com/portal/home.shtml – Дата доступа 17.05.2023.

4 Sortly [Электронный ресурс]. – Режим доступа: https://www.sortly.com – Дата доступа 17.05.2023.

5 Maropost [Электронный ресурс]. – Режим доступа: https://www.maropost.com – Дата доступа 17.05.2023.

6 Система управления складом APWMS [Электронный ресурс]. – Режим доступа: https://new.fips.ru/iiss/document.xhtml?faces-redirect=true&id=7905c14c49e46a5d9f0b465adf05c726 – Дата доступа 17.05.2023.

7 wmsMLC Pro [Электронный ресурс]. – Режим доступа: https://new.fips.ru/iiss/document.xhtml?faces-redirect=true&id=10e5437ee8b4c0c0c3a628dda4c7af6f – Дата доступа 17.05.2023.

Консульт.

Конс..

8 Hangfire [Электронный ресурс]. – Режим доступа: https://docs.hangfire.io/en/latest/configuration/using-sql-server.html – Дата доступа 19.05.2023.

9 Entity Framework [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/ef/ – Дата доступа 19.05.2023.

10 Парамонов, А.И. Курс лекций по проектированию информационных систем / А.И. Парамонов – Минск: БГТУ, 2022. – 63 с.

13 Fluent Validation [Электронный ресурс] / Документация. – Режим доступа: https://docs.fluentvalidation.net. – Дата доступа: 21.05.2023.

14 RFC 7519 [Электронный ресурс]. – Режим доступа: https://datatracker.ietf.org/doc/html/rfc7519. – Дата доступа: 25.05.2023.

Консульт.

Конс..