МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту на тему:**

Программное средство «Система управления складом»

Выполнил студент Демьянов Владислав Русланович

(Ф.И.О.)

Руководитель проекта ст. препод. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты ст. препод. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер ст. препод. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск, 2022

**Содержание**

[Введение 5](#_Toc122353104)

[1 Постановка задачи 6](#_Toc122353105)

[1.1 Актуальность решаемой задачи 6](#_Toc122353106)

[1.2 Обзор и анализ прототипов 7](#_Toc122353107)

[1.3 Средства разработки 9](#_Toc122353108)

[2 Проектирование программного средства 11](#_Toc122353109)

[2.1 Проектирование базы данных 11](#_Toc122353110)

[2.2 Разработка архитектуры программного средства 12](#_Toc122353111)

[3 Разработка программного средства 13](#_Toc122353112)

[3.1 Разработка Backend-приложения 13](#_Toc122353113)

[3.2 Разработка Frontend-приложения 16](#_Toc122353114)

[4 Тестирование программного средства 19](#_Toc122353115)

[5 Руководство пользователя 21](#_Toc122353116)

[5.1 Миграции 21](#_Toc122353117)

[5.2 Развёртывание на платформе Docker 21](#_Toc122353118)

[5.3 Руководство использования интерфейса API Swagger 23](#_Toc122353119)

[5.4 Руководство использования пользовательского интерфейса 23](#_Toc122353120)

[Заключение 30](#_Toc122353121)

[Список использованных источников 31](#_Toc122353122)

[ПРИЛОЖЕНИЕ А 32](#_Toc122353123)

[ПРИЛОЖЕНИЕ Б 33](#_Toc122353124)

[ПРИЛОЖЕНИЕ В 34](#_Toc122353125)

[ПРИЛОЖЕНИЕ Г 35](#_Toc122353126)

[ПРИЛОЖЕНИЕ Д 36](#_Toc122353127)

# **Введение**

Основная задача выполнения курсового проектирования – это разработать программное средство «Система управления складом».

Программное средство должно:

* быть логически завершённым;
* иметь понятный и удобный пользовательский интерфейс;
* иметь поддерживаемый код;
* иметь расширяемую архитектуру;
* иметь нормализованную структуру базы данных;
* иметь руководство пользователя.

Тема курсового проекта выбрана по причине актуальности автоматизации бизнес процессов. В каждом бизнесе есть рутинные процессы, которые возможно ускорить и сделать более удобными для людей, используя информационную систему.

В целевую аудиторию входят в первую очередь сотрудники склада: руководители склада, операторы, администраторы, аудиторы и другие работники склада. Но также в неё входят клиенты склада. Наличие такой информационной системы позволит сделать бизнес и его процессы более прозрачными, что должно увеличить доверие у клиентов. В качестве типа информационной системы выбрано веб-приложение, это связано с доступностью браузера на всех устройствах, причём по умолчанию.

В пояснительной записке будет приведена более подробная информация в соответствии со списком ниже:

* постановка задачи, включающая актуальность, обзор прототипов, средства разработки;
* проектирование программного средства, включающее проектирование базы данных;
* разработки программного средства;
* тестирование;
* руководство пользователя.

# **1 Постановка задачи**

## **1.1 Актуальность решаемой задачи**

Системы автоматизации бизнеса становятся все более актуальными и востребованными. Они позволяют быстро и оперативно решать текущие задачи, адаптируются к изменениям в компании, минимизируют издержки и повышают эффективность работы.

На практике с понятием автоматизации знакомы немногие. Большинство компаний ведут свою деятельность стихийно, не рассматривая ее как систему, которой можно и нужно управлять. В ходе работы возникают ситуации, которые не влекут за собой каких-либо улучшений, а лишь усложняют производственные процессы, приводят к ненужным тратам и расходам.

В каждом бизнесе есть рутинные процессы, которые возможно ускорить и сделать более удобными для людей. Сейчас почти у каждого есть персональный компьютер и смартфон, по умолчанию на этих устройствах установлен браузер. Поэтому вместо неудобных журналов, бумажек, папок с документами, которые ещё и как правило находятся у одного человека, намного удобнее использовать приведённые выше электронные аналоги. Так, например, в информационной системе можно хранить, обрабатывать и отображать информацию о товарах, арендаторах, сотрудниках. Как в любом деле, где задачи решаются группой людей, которые находятся в финансовых отношениях, важно знать кто и чем сейчас занимается, какую пользу принёс сотрудник, какие решил задачи, с этим могут помочь различные списки задач. На складе могут быть различные события, о которых нужно уведомить сотрудников, например, просрочка задачи или перенос товара в другое место хранения. Таких процессов, которые способна решить и оптимизировать информационная система существует большое количество.

Если обобщить, то автоматизация бизнес-процессов компании позволяет:

* заказчикам, получать в нужном количестве высококачественный продукт, своевременное и качественное обслуживание;
* работникам, повысить эффективность работы, достигать индивидуальных запланированных показателей, разграничивать уровни ответственности и минимизировать конфликтные ситуации внутри коллектива;
* работодателю, оценивать и анализировать результаты деятельности предприятия в целом и каждого из сотрудников, а также контролировать и влиять на конечный результат бизнес-процесса.

Соответственно разрабатываемая информационная система будет актуальна для руководителей склада, операторов, администраторов, аудиторов и другие работников склада. Но также будет актуальна и для клиентов (арендаторов) склада, так как наличие открытой информационной системе позволит сделать бизнес и его процессы более прозрачными, что должно увеличить доверие у клиентов.

## **1.2 Обзор и анализ прототипов**

Система управления складом – это программа, предназначенная для автоматизации управления процессами склада и работы складского комплекса.

Система управления складом интегрируется с учётными программами предприятия, обеспечивает полную прозрачность данных в цепочке поставок всей компании.

В основные бизнес-процессы склада входят: приёмка товара, учёт, товара, оптимизация процесса хранения, управление персоналом.

Учёт товара. Система для склада ведёт учёт товара, предоставляет пользователям актуальную информацию о его количестве, позволяет минимизировать затаривание склада ненужными остатками. Экономия складского пространства и бесперебойная работа логистики склада достигается за счёт оптимального распределения и хранения запасов.

Оптимизация процесса хранения. Система предоставляет пользователям возможность моделирования эффективных схем хранения различных товаров, учитывающих их характеристики, например, такие как, вес товара или его спрос (скорость оборачиваемости товарных запасов). Это позволяет организовать процесс хранения таким образом, что более востребованные или тяжёлые товары будут располагаться ближе к зоне отгрузки или товар, отгружаемый вместе, будет храниться рядом друг с другом, за счёт чего времени для его обработки потребуется меньше. Учёт многочисленных факторов хранения обеспечивает эффективную работу складской логистики.

Управление персоналом. Централизованное управление складом посредством складской системы сокращается необходимость в содержании персонала в большом количестве. Оптимизация рабочего фонда становиться возможной, в том числе, за счёт сокращения частоты инвентаризации товара. Программа позволяет производить инвентаризацию товара без вмешательства в повседневную работу склада. Сокращение расходов на оплату труда позволяет снизить текущие (операционные) расходы на содержание склада и повысить эффективность работы всего предприятия. 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.

Документооборот. Система управления складом способна автоматизировать все процессы, устраняя необходимость ведения бумажного документооборота, требующего значительных ресурсов. Пользователи могут иметь общий доступ к базе данных, обеспечить работников необходимой информацией для быстрой и качественной работы.

Управление складом и контроль. Для предприятий, которым необходимы расширенные возможности контроля, программа предлагает отслеживать товар по различным характеристикам: серийные номера, сроки годности, товарные коды и т.п.

На рынке представлены различные типы систем управления складом, есть простые коробочные решения, а есть сложные комплексные системы управления складом, подходящие для крупных складов.

NetSuite WMS. Одним лучших примеров системы управления складом является продукт от Oracle – NetSuite WMS [1] (рисунок 1.1).

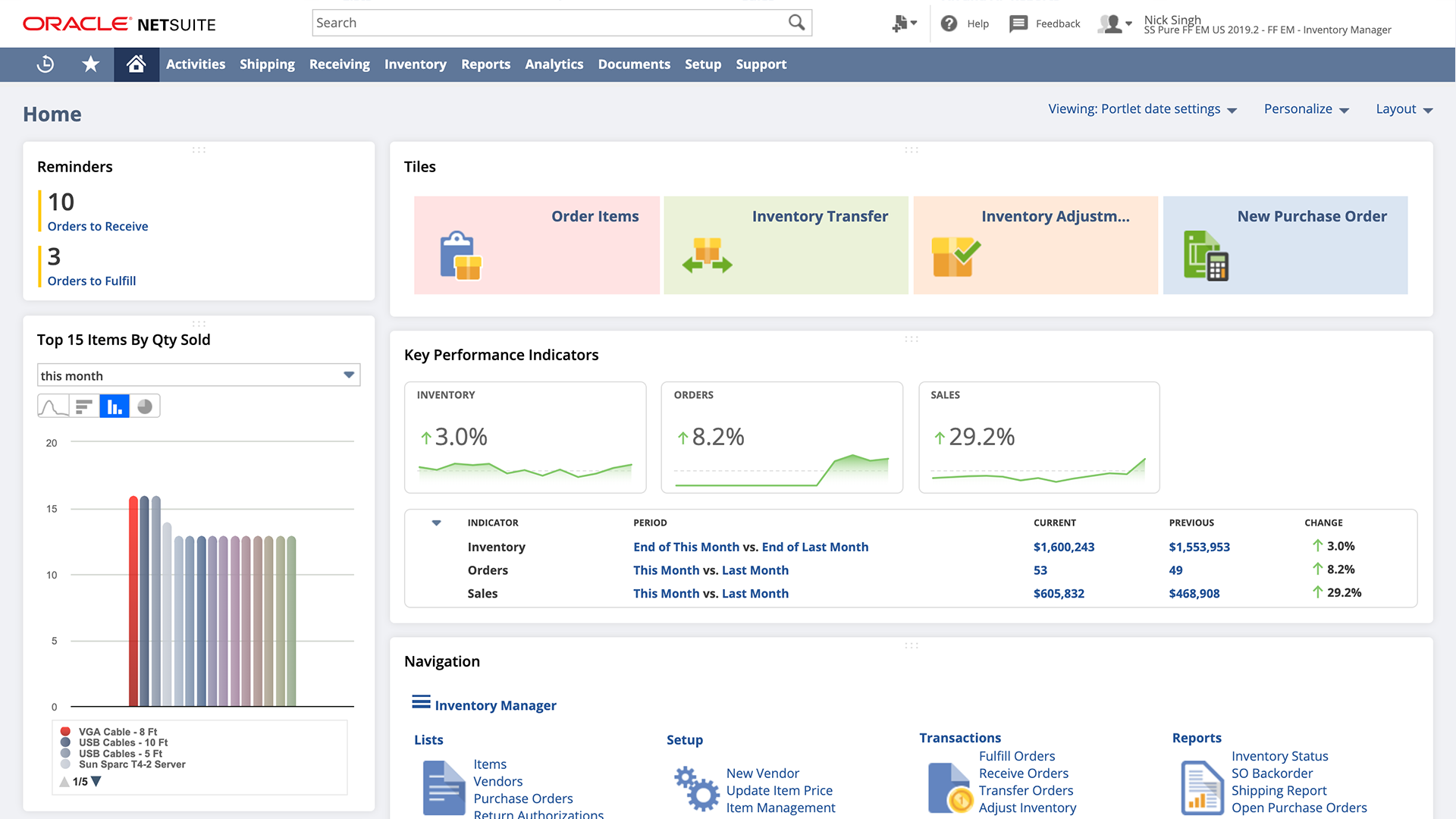


Рисунок 1.1 – NetSuite WMS

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.

Исходя из проведённого исследования прототипов, можно сделать вывод, что для разработки системы управления складом необходимы большие ресурсы, так как на складе происходит большое количество процессов и конкурировать придётся с такими гигантами как Oracle.

Соответственно, стоит задача реализовать хотя бы самые простые и базовые бизнес процессы склада, такие как: получение, хранение, отгрузка, перемещение товаров; управление задачами; управление персоналом.

## **1.3 Средства разработки**

Для разработки серверной части приложения была выбрана платформа .NET версии 6, её фреймворк ASP.NET и ORM Entity Framework Core [2].

В качестве СУБД используется MS SQL Server 19 [3].

Для разработки клиентской части приложения был выбран JavaScript Framework Angular [4] версии 15, TypeScript [5] версии 4.8.4. Также используется библиотека RxJS [6] версии 7.5.7. Для упрощения создания элементов UI, используется библиотека с готовыми компонентами Angular Material [7] версии 15.0.1.

Для публикации приложения выбрана платформа Docker [8].

Для документации API используется набор инструментов Swagger.

.NET (до 5 версии .NET Core) – кроссплатформенная программная среда с открытым исходным кодом, разработанная компанией Microsoft. .NET основана на .NET Framework, платформа .NET отличается от неё модульностью, кроссплатформенностью, возможностью применения облачных технологий, и тем, что в ней произошло разделение между библиотекой CoreFX и средой выполнения CoreCLR.

.NET – модульная платформа. Каждый её компонент обновляется через менеджер пакетов NuGet, а значит можно обновлять её модули по отдельности, в то время как .NET Framework обновляется целиком. Каждое приложения может работать с разыми модулями и не зависит от единого обновления платформы (определение с курса лекций по объектно-ориентированному программированию Пацей Н.В. [9]).

CoreFX – это библиотека, интегрированная в .NET. Среди её компонентов: System.Collections, System.IO, System.Xml.

CoreCLR – это среда выполнения, включающая в себя RyuJIT (JIT-компилятор), встроенный сборщик мусора и другие компоненты.

ASP.NET – программная платформа, разработанная Microsoft и предназначенная для разработки веб-приложений. Является развитием технологии OWIN (The Open Web Interface for .NET). Katana – OWIN-совместимый хост, разработанный Microsoft.

ASP.NET приложения могут работать под IIS (Windows) или под веб-сервером Kastrel (кроссплатформенный вариант).

ASP.NET поддерживает следующие типы приложений: MVC UI, Web API, Web Pages. Web Forms не поддерживается. В курсовой работе используется Web API.

ORM Entity Framework Core – среда объектно-ориентированного сопоставления (ORM) с открытым исходным кодом.

СУБД MS SQL Server – система управления реляционными базами данных, разработанная корпорацией Microsoft. Основной используемый язык запросов – Transact-SQL. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов SQL с расширениями.

Angular – фреймворк от Google для создания SPA веб-приложений, на языках программирования TypeScript, JavaScript.

TypeScript – строго типизированный язык программирования, основанный и компилируемый в JavaScript.

RxJS – это библиотека для реактивного программирования, которая позволяет удобно организовать работу с событиями и асинхронным кодом, писать сложную логику декларативно.

Angular Material – это реализация спецификации Google Material Design. Представляет собой набор готовых UI компонентов.

Docker – платформа для разработки, доставки и эксплуатации приложений. Основное назначение – упростить развёртывание приложения.

Docker – это Docker Desktop (Engine) вместе с Docker Hub.

Docker Hub – облачный сервис для распространения контейнеров.

Docker Engine – механизм, (сервис и приложение) для создания и функционирования контейнеров (определение с курса лекций Смелова В.В по программированию интернет сервисов [10]).

Swagger – спецификация для описания Web API, ориентирована на REST, машинно-независимая от языков программирования, официальное название Open API.

Также в приложении используется открытый веб-протокол OData. Он позволяет выполнять операции с ресурсами и получать ответы в форматах XML, JSON.

WebAPI ASP.NET поддерживает как третью, так и четвёртую версию протокола.

Для валидации, строготипизированных правил проверки данных пришедших от клиента на сервер, используется библиотека FluentValidation. Этот инструмент основан на способе реализации объектно-ориентированного API – текучем интерфейсе. Данный метод нацелен на повышение читабельности исходного кода программы.

Текучий интерфейс – это способ реализации объектно-ориентированного API, при котором методы возвращают тот же интерфейс, на котором были вызваны.

В следующем разделе будет приведено описание проектирования приложения с использованием выше перечисленных технологий.

# **2 Проектирование программного средства**

## **2.1 Проектирование базы данных**

Для хранения данных в приложении используется реляционная база данных MS SQL Server.

Логическая схема базы данных, спроектированной в ходе разработки приложения представлена на рисунке 2.1.

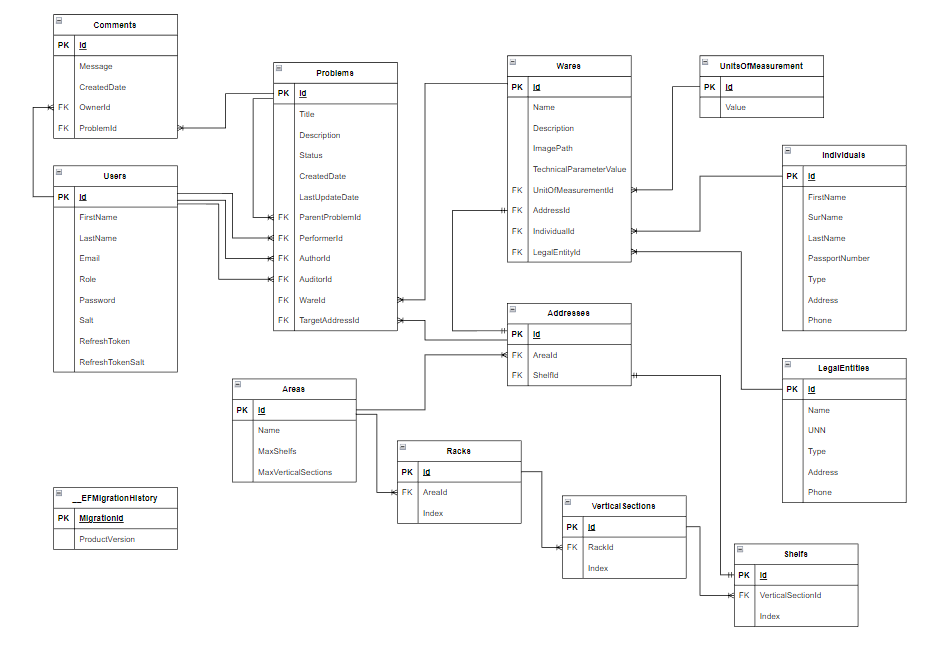


Рисунок 2.1 – Диаграмма базы данных

База данных состоит из 13 таблиц, одна из которых сгенерирована Entity Framework.

Основными сущностями в базе данных являются Users, Problems и Wares.

Таблица Users содержит данные о сотрудниках склада, именно они будут в основном взаимодействовать с системой.

Таблица Wares содержит данные товаров, которые находятся на складе.

Таблица Problems содержит информацию о задачах, это могут независимы от товаров задачи или наоборот связанными с ними. Например, может быть такая задача: перенести товар их зоны А в зону Б.

Чтобы определить где находиться товар, нужно знать его адрес. Для этого есть таблица Addresses. Он включает зону хранения (таблица Areas), номер стелажа (таблица Racks), номер вертикальной секции (таблица VerticalSections) и номер полки (таблица Shelfs).

Товар обычно принадлежит кому-то, как правило это либо физические лица, либо юридические, для них соответственно есть таблицы Individual и LegalEntities.

И последняя таблица это Comments. Она содержит комментарии, которые пользователи могут оставлять под задачами.

Для разработки базы данных был изучен курс лекций Блиновой Е.А. [11].

## **2.2 Разработка архитектуры программного средства**

Разработанная информационная система является веб-приложением. Это означает, что в нём есть как минимум два компонента – клиент и сервер. Для развёртывания была использована платформа Docker. Каждый компонент помещался в отдельный контейнер, соответственно получилось три Docker-контейнера для сервера базы данных, для HTTP сервера представленное ASP.NET и контейнер для Angular приложения. Диаграмму развёртывания можно увидеть на рисунке 2.2.

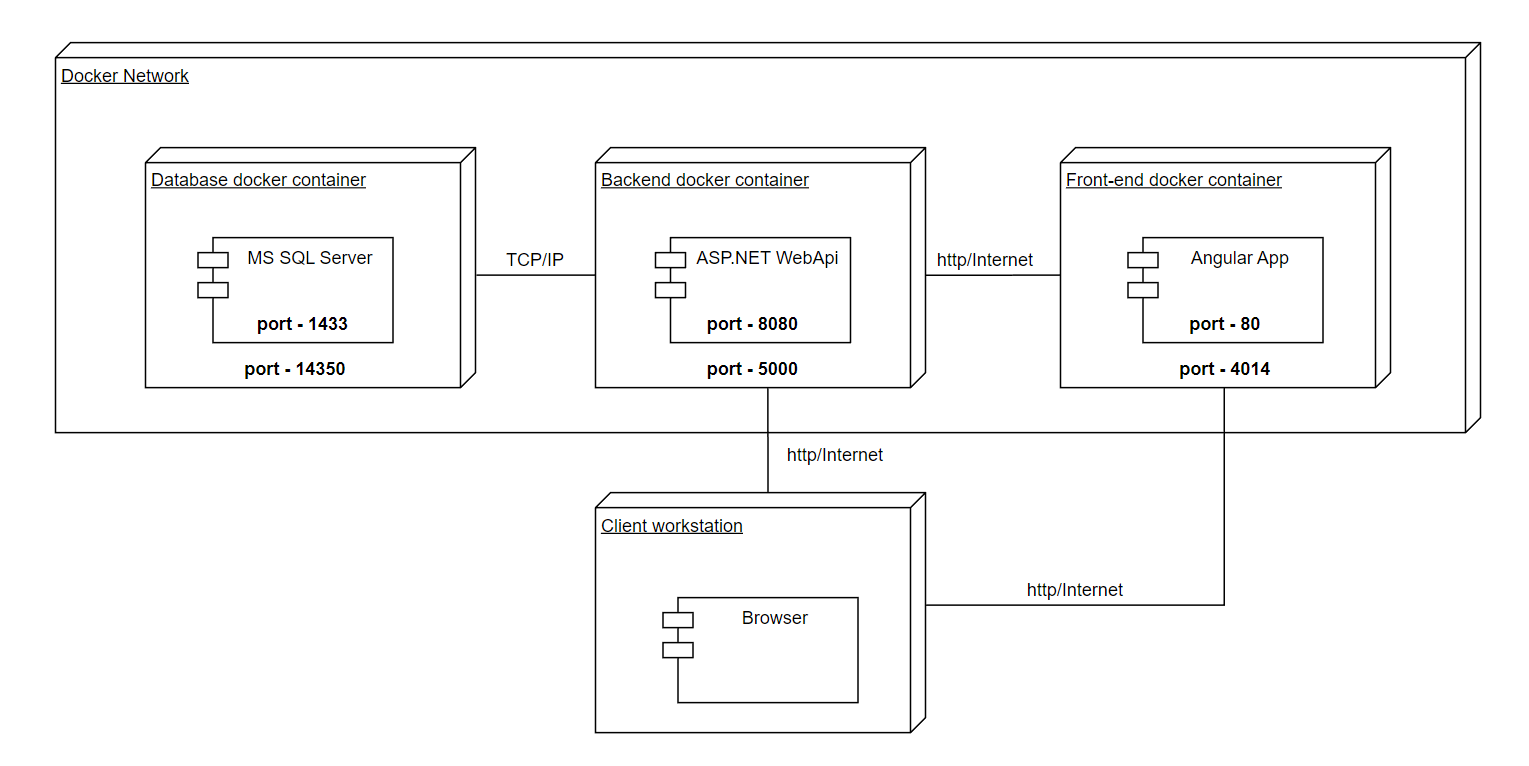


Рисунок 2.2 – Диаграмма развёртывания

Из диаграммы понятно, что 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/IP.

Для разработки диаграммы развёртывания были использованы материалы курса Парамонова А.И по предмету проектирование информационных систем [12].

Далее будет приведено более подробное описание архитектуры backend-приложения и frontend-приложения.

# **3 Разработка программного средства**

## **3.1 Разработка Backend-приложения**

Решение состоит из четырёх проектов: WMS.Core, WMS.Database, WMS.Database.Migrations и WMS.WebApi (рисунок 3.1).

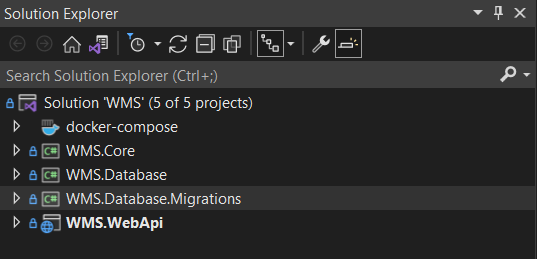


Рисунок 3.1 – Решение Backend-приложения

Проект WMS.WebApi – это ASP.NET приложение типа WebAPI. Оно содержит классы-контроллеры и зависит от проектов WMS.Database, WMS.Database.Migrations и WMS.WebApi. Из которых импортируются сервисы, сущности и другие вспомогательные классы. Проект представлен на рисунке 3.2.

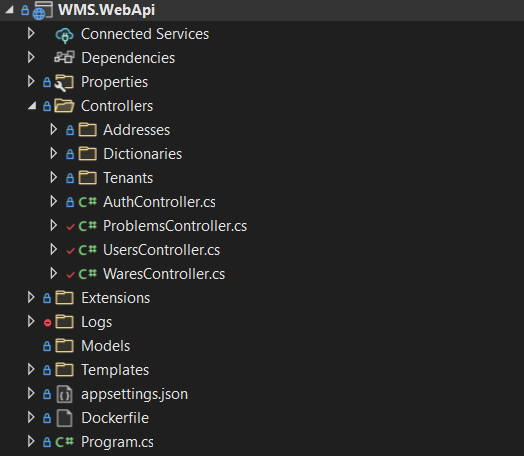


Рисунок 3.2 – Проект WMS.WebApi

Проект WMS.Migrations – это библиотека классов, она содержит классы миграции (рисунок 3.3).

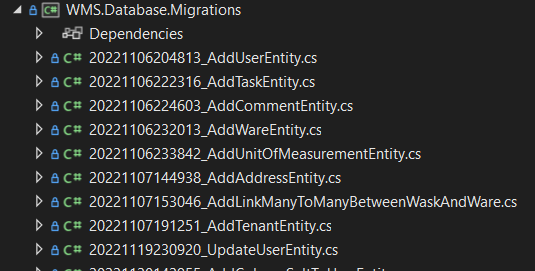


Рисунок 3.3 – Проект WMS.Database.Migrations

О том, как добавлять миграции и обновлять ими базу данных есть описание в руководстве пользователя.

Проект WMS.Database – это также библиотека классов, она содержит контекст базы данных и сами сущности (рисунок 3.4).

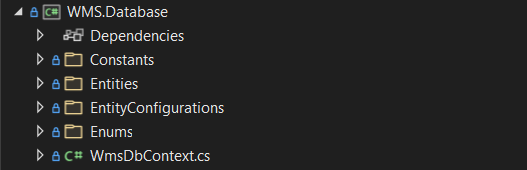


Рисунок 3.4 – Проект WMS.Database

В проекте также есть классы содержащие константы для настройки среды и OData.

Кроме контекста и сущностей в проекте также есть классы конфигурации сущностей, это сделано, чтобы не утяжелять класс контекста. Такие классы реализуют обобщённый интерфейс IEntityTypeConfiguration, а точнее метод Configure, в него передаётся экземпляр класса EntityTypeBuilder.

Пример класса конфигурации представлен в листинге 3.1. В нём происходит конфигурация сущности Address. Поле AreaId должно быть обязательным, а также настраивается связь один к одному с сущностью Shelf.

|  |
| --- |
| public class AddressEntityConfiguration : IEntityTypeConfiguration<Address>  {  public void Configure(EntityTypeBuilder<Address> builder)  {  \_ = builder.Property(x => x.AreaId)  .IsRequired();  \_ = builder.HasOne(x => x.Shelf)  .WithOne(x => x.Address)  .OnDelete(DeleteBehavior.NoAction);  }  } |

Листинг 3.1 – Класс конфигурации сущности

Последний проект WMS.Core – содержит сервисы, валидаторы, пользовательские исключения и другие классы, которые реализуют бизнес логику (рисунок 3.5).

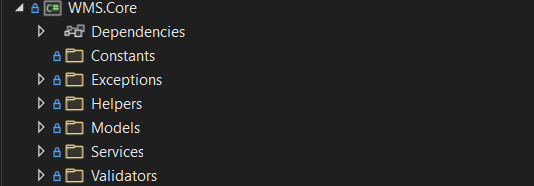


Рисунок 3.5 – Проект WMS.Core

Проект также содержит классы валидаторы. Для этого используется библиотека FluentValidation [13]. Чтобы реализовать такой класс нужно сделать наследование от обобщающего класса AbstractValidator. После чего станет доступен подход скользящего интерфейса. Пример такого класса доступен в приложении А.

Ещё не было сказано, что в проекте WebApi есть, как и обычные классы контроллеры, которые наследуются от класса ControllerBase, но также и OData [14] контроллеры.

Для реализации OData контроллера нужно сделать наследование от класса ODataController. Также нужно зарегистрировать EDM модель.

Регистрация EDM модели и настройка OData происходит в классе расширении WmsODataMvcBuilderExtensions, его можно найти в приложении Б.

Далее будет приведена информация о разработке Frontend-приложения на JavaScript фреймворке Angular.

## **3.2 Разработка Frontend-приложения**

Как уже было сказано для разработки клиентской части приложения был выбран JavaScript фреймворк Angular. Была реализована архитектура, которая проиллюстрирована на рисунке 3.6.

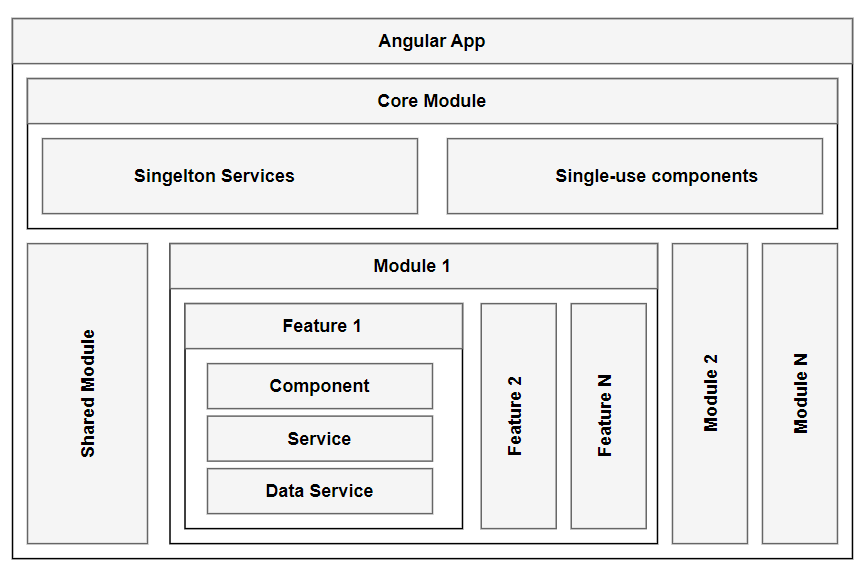


Рисунок 3.6 – Архитектура клиентской части приложения

Приложение состоит из модулей. Модуль представляет собой класс с декоратором @NgModule. Декоратор предоставляет метаданные, которые Angular использует для организации структуры приложения. Приложение должно иметь корневой модуль. Корневой модуль – это место, где Angular начинает упорядочивать дерево приложения. Модель может импортировать компоненты, которые являются Provider (это всё, что помечено декоратором @Injectable), например, сервисы, хелперы и т.д. Для этого модуль по умолчанию – сингелтонг.

Так в приложении есть:

* core module, который является основным модулем в приложении, он содержит сингелтон сервисы и одноразовые компоненты, например layout;
* shared module, который использует для экспорта компонент, которые делятся между различными модулями (например диалог подтверждение);
* feature module, который содержит сервисы, константы, модели и компоненты;
* класс service, который описывает логику компонентыl
* класс data service, который реализует коммуникацию между хранилищем данных и компонентой.

Пример модуля представлен в листинге 3.2.

|  |
| --- |
| @NgModule({    declarations: [LayoutComponent],    imports: [CommonModule, AppRoutingModule, HttpClientModule],    providers: providers,  })  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 3.2 – Angular модуль

Как можно видеть в листинге 3.2, декоратор @NgModule принимает один объект, имеющий свойства declarations, imports, providers также ещё есть exports.

Declarations – компоненты, директивы и фильтры корневого модуля.

Exports – компоненты, сервисы, директивы и фильтры доступные другим модулям;

Imports – другие модули, используемые в корневом модуле;

Providers – сервисы приложения;

Компонент – часть интерфейса приложения с собственной логикой. Для того чтобы класс стал компонентом, нужно пометить его декоратором @Component.

Пример компонента представлен в листинге 3.3.

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

Листинг 3.3 – 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.

Сервисы – классы для предоставления данных компонентам. Сервисы относятся к провайдерам, соответственно нужно пометить класс декоратором @Injectable. Пример сервиса представлен в листинге 3.4.

|  |
| --- |
| @Injectable({    providedIn: 'root'  })  export class AreasService {    constructor(private readonly http: HttpClient) { }    public getAll(): Observable<IArea[]> {      return this.http.get<ODataValue<IArea>>(ApiEndpoints.Areas)        .pipe(map((odataValue: ODataValue<IArea>) => odataValue.value));    }  } |

Листинг 3.4 – Сервис

Сервис представленный в листинге 3.4 служит для отправки запросов на сервер. Для этого он внедряет сервис HttpClient. Затем он используется в методе getAll (GET-запрос) и create (POST-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 0, 1 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 просто отменить его, не дожидаясь ответа. Это большое преимущество над теми же Promise.

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

Пример модуля реализующий ленивую загрузку можно найти в приложении В.

Кроме всего выше перечисленного в приложении также реализованы Guards для защиты путей и Interceptor для внедрения в запрос JWT-токенов.

# **4 Тестирование программного средства**

Тестировать приложение будем вручную. Для этого будем использовать Swagger и пользовательский интерфейс.

Один из условий для зон склада, является то, что они имеют определённую вместимость. Соответственно больше чем доступно добавить стелажей пользователь не может. Пример с попыткой перегрузить секцию склада показан на рисунке 4.1.

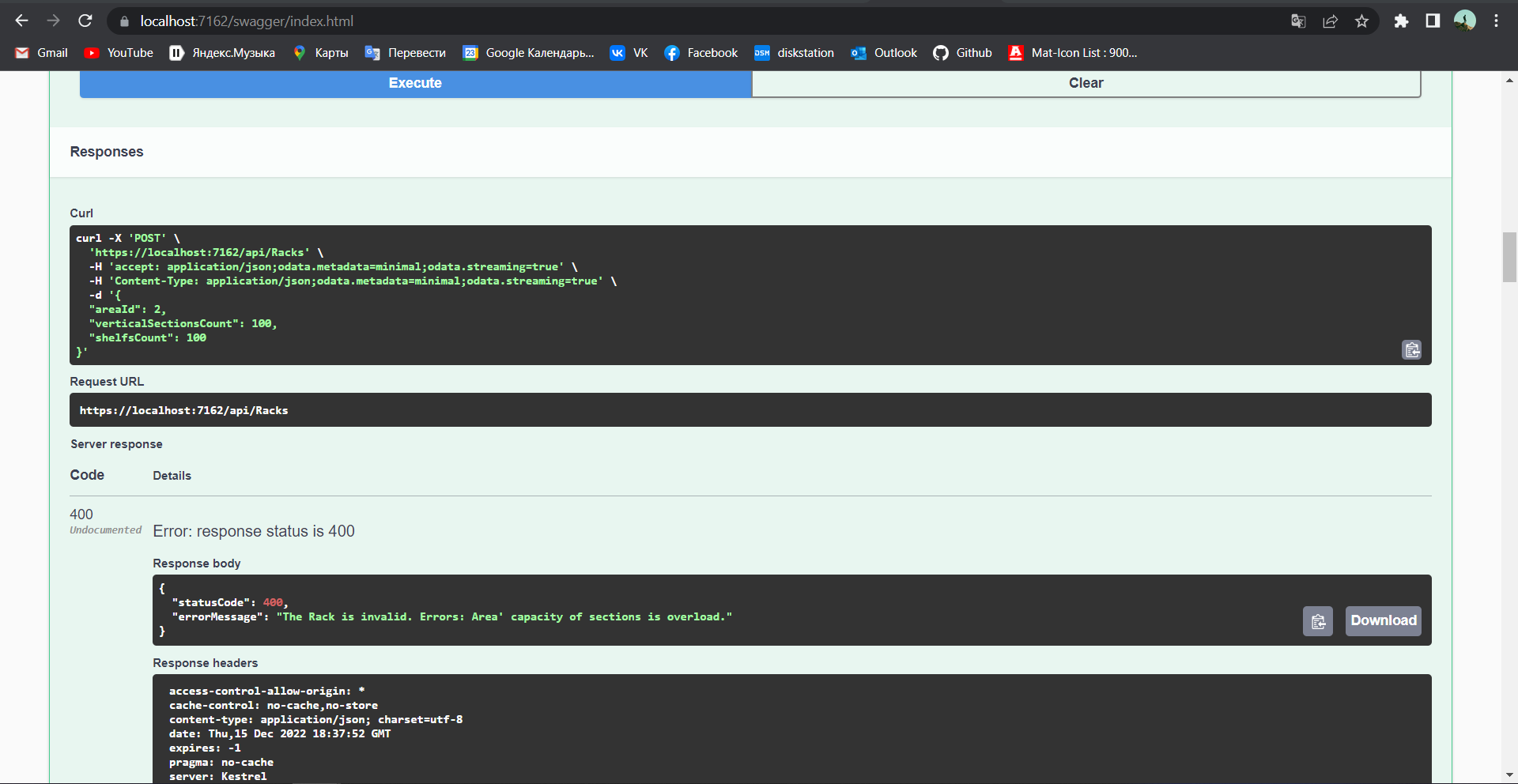


Рисунок 4.1 – Перегруз зоны склада

Это был пример негативного тестирования.

Теперь с той же функцией проведём позитивное тестирование. Для это попробуем добавить в зону склада стелаж с одной секцией, в которой будут две полки, результат запроса можно увидеть на рисунок 4.2.

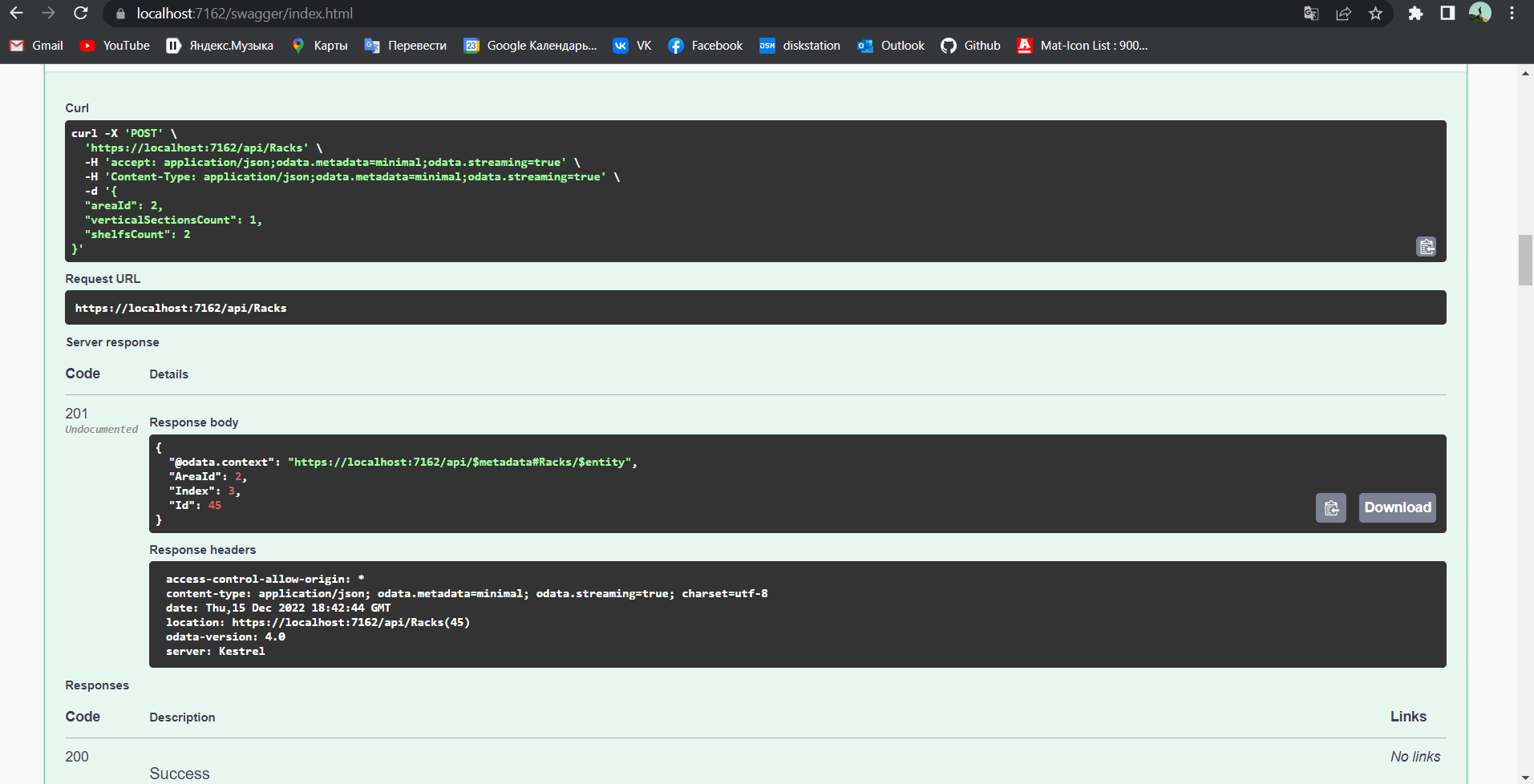


Рисунок 4.2 – Позитивный сценарий с добавление стелажа

Теперь проведём тестирование через интерфейс пользователя. Также выполним позитивный и негативный тест для формы входа.

Введём неправильный пароль (рисунок 4.3).

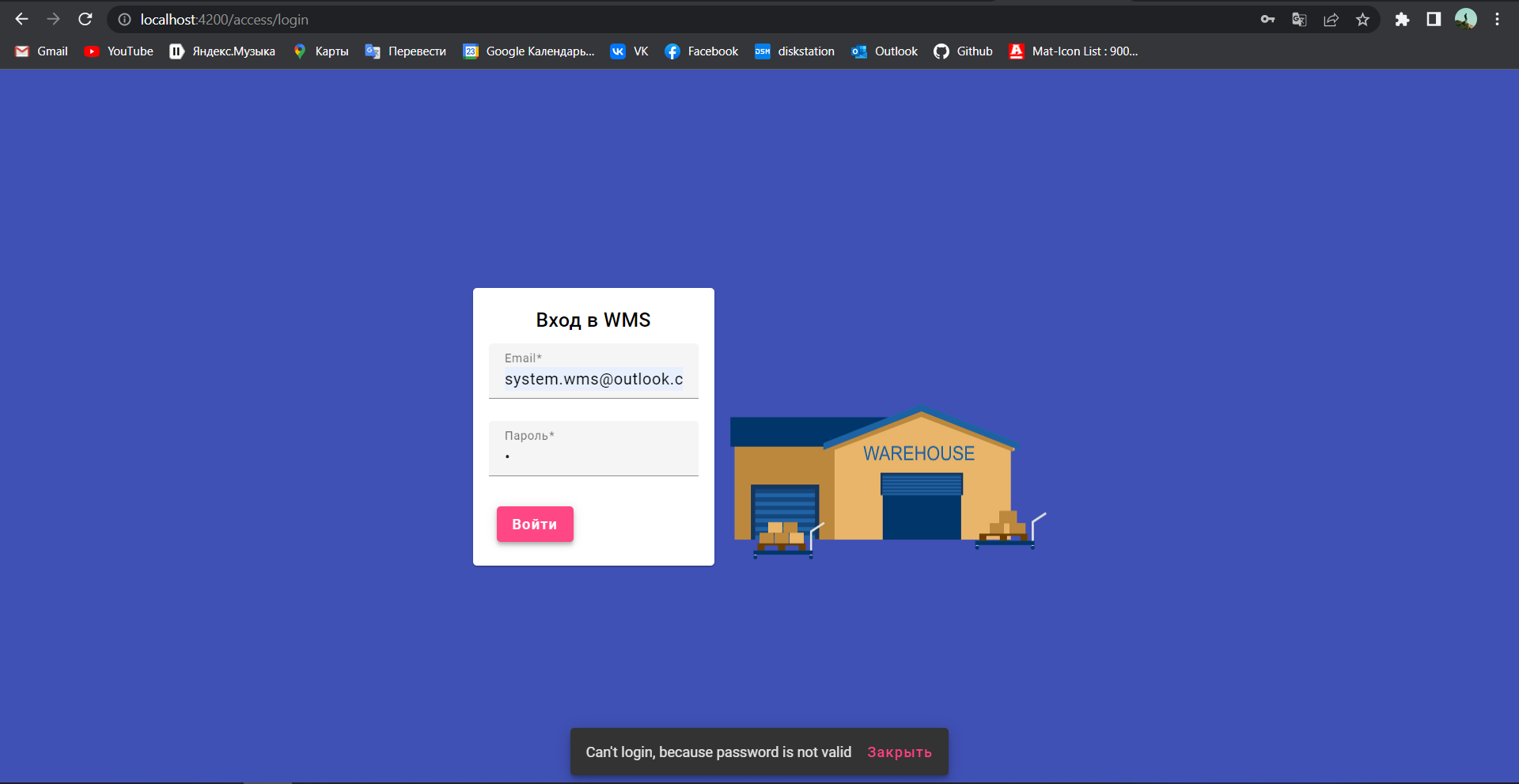


Рисунок 4.3 – Неверный пароль

Теперь введём правильный пароль (рисунок 4.4).

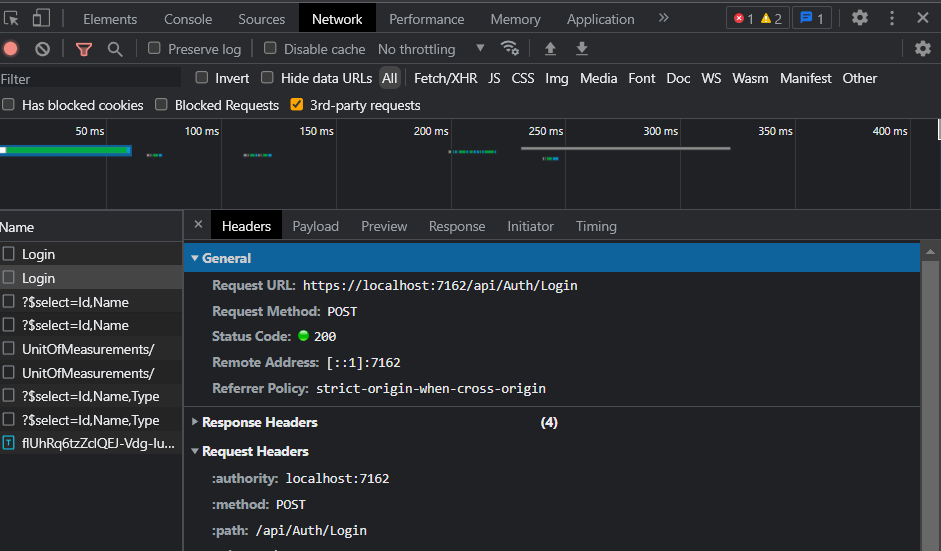


Рисунок 4.4 – Успешный вход

В результате было сделано минимальное тестирование функционала системы. Надо сказать, что в системе присутствует двойная валидация данных, как на стороне клиента, так и на стороне сервера с использование библиотеки FluentValidation. Поэтому сделать мутацию данных пользователю достаточно проблематично, одна ещё есть над чем работать.

# **5 Руководство пользователя**

## **5.1 Миграции**

Чтобы добавить миграции нужно выполнить следующую команду (листинг 5.1).

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

Листинг 5.1 – Добавление новой миграции

Применение миграций к базе данных (листинг 5.2).

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

Листинг 5.2 – Применение миграций к базе данных

Параметр --project указывает проект где хранятся классы миграций.

Параметр --startup-project указывает запускаемый проект.

## **5.2 Развёртывание на платформе Docker**

Как уже упоминалось ранее приложение будет развёрнуто на три контейнера: сервер базы данных, клиентская часть на Angular и серверная часть на ASP.NET.

Создание контейнера с ASP.NET. Для создания контейнера с ASP.NET первым шагом с помощью Visual Studio генерируем Dockerfile (приложение Г).

В нём происходит следующее:

* используется базовый образ aspnet;
* устанавливается каталог приложения docker по умолчанию;
* устанавливается порт контейнера по умолчанию;
* добавляется .net SDK для сборки приложения;
* копируется всё необходимое с хоста в каталог docker;
* устанавливается точка входа WMS.WebApi.dll

Использование Docker-Compose. Это инструмент для определения и запуска мультиконтейнерных приложений Docker. Для настройки служб приложения используется файл YAML.

В нём нужно указать:

* название образа;
* путь к используемому Dockerfile, описанному ранее;
* внутренний и внешний порт.

Часть YAML файла представлена в листинге 5.3.

|  |
| --- |
| wms.webapi:  image: ${DOCKER\_REGISTRY-}wmswebapi  build:  context: .  dockerfile: WMS.WebApi/Dockerfile |

Листинг 5.3 – Настройка запуска контейнера с ASP.NET

Настройка портов представлена в листинге 5.4.

|  |
| --- |
| wms.webapi:  environment:  ports:  - "5000:8080"  - "5001:443" |

Листинг 5.4 – Настройка портов контейнера с ASP.NET

Теперь после запуска приложения, оно создаст образ в Docker, приложение станет доступно на порту 5001.

Теперь нужно развернуть клиентскую часть приложения на Angular. Для это нужно сделать следующие шаги:

* создать файл nginx.conf, который связан с сервером Nginx, он используется как обратный прокси-сервер, балансировка нагрузки и т.д. (листинг представлен в приложении Д);
* создать Dockerfile, для создания образа приложения и добавления всех зависимостей и конфигураций (последовательности команд не сильно отличается от Dockerfile описанного ранее)
* добавляем сервис в созданный ранее Docker-Compose.

Настройка запуска контейнера с клиентским приложением представлен в листинге 5.5.

|  |
| --- |
| wms.frontend:  image: ${DOCKER\_REGISTRY-}wmsfrontend:v1  container\_name: WMS.Frontend  build:  context: ./WMS.Frontend  dockerfile: Dockerfile  ports:  - "4014:80" |

Листинг 5.5 – Настройка запуска клиентского приложения

Для последнего контейнера с MS SQL Server, нужно скачать уже готовый образ, затем добавить настройки его запуска в Docker-Compose, описанный ранее.

И наконец для запуска нужно выполнить команду docker-compose up.

## **5.3 Руководство использования интерфейса API Swagger**

Разработанное API имеет документацию, описанную с помощью набора инструментов Swagger [15]. Перейти к ней можно по маршруту /swagger/index.html, на рисунке 5.1 представлен интерфейс, который увидит пользователь после перехода.

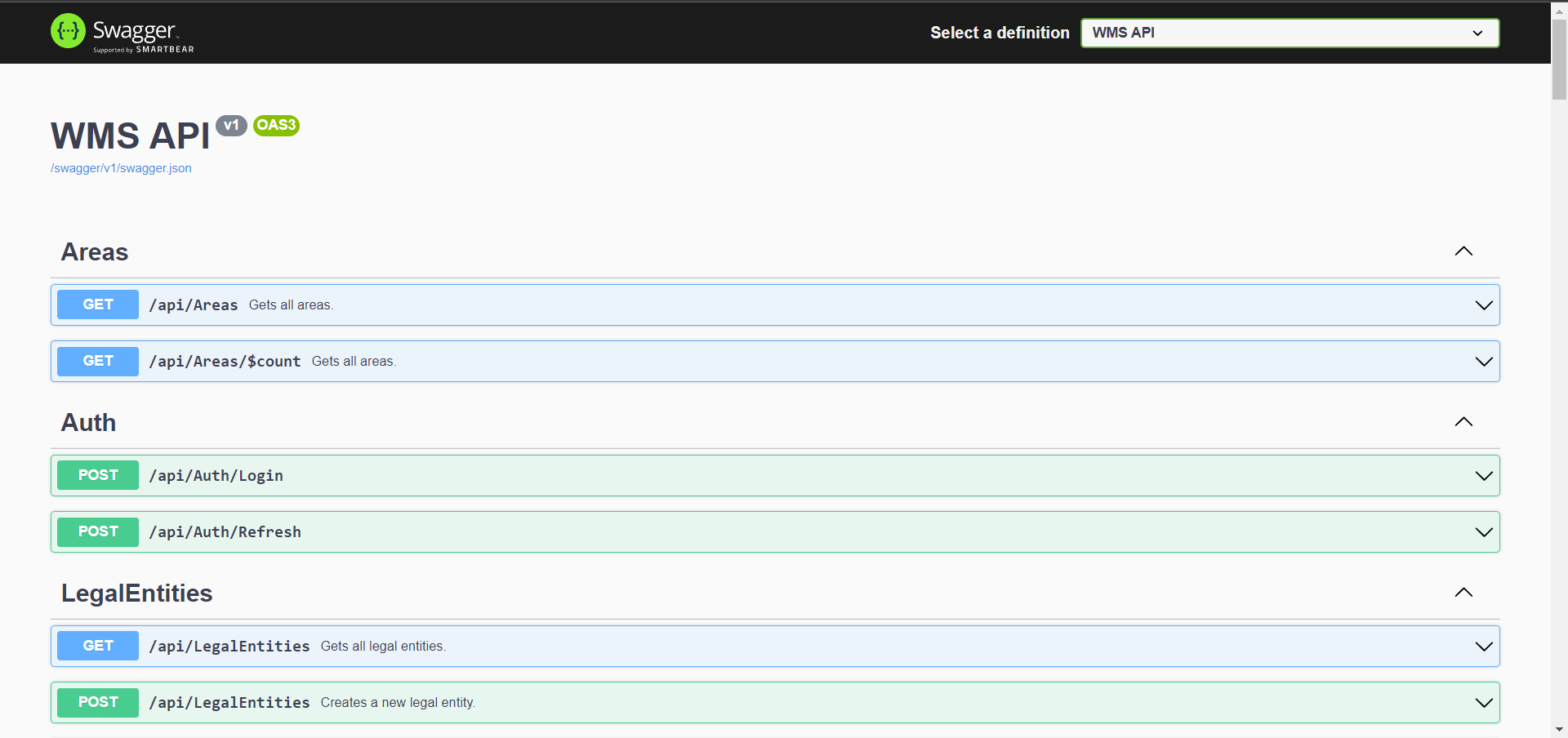


Рисунок 5.1 – Документация API

Данная документация содержит информацию об конечных точках и сущностях.

## **5.4 Руководство использования пользовательского интерфейса**

При первом переходе в приложение пользователя встречает страница аутентификации. Не аутентифицированным пользователям доступна только она, как на рисунке 5.2.

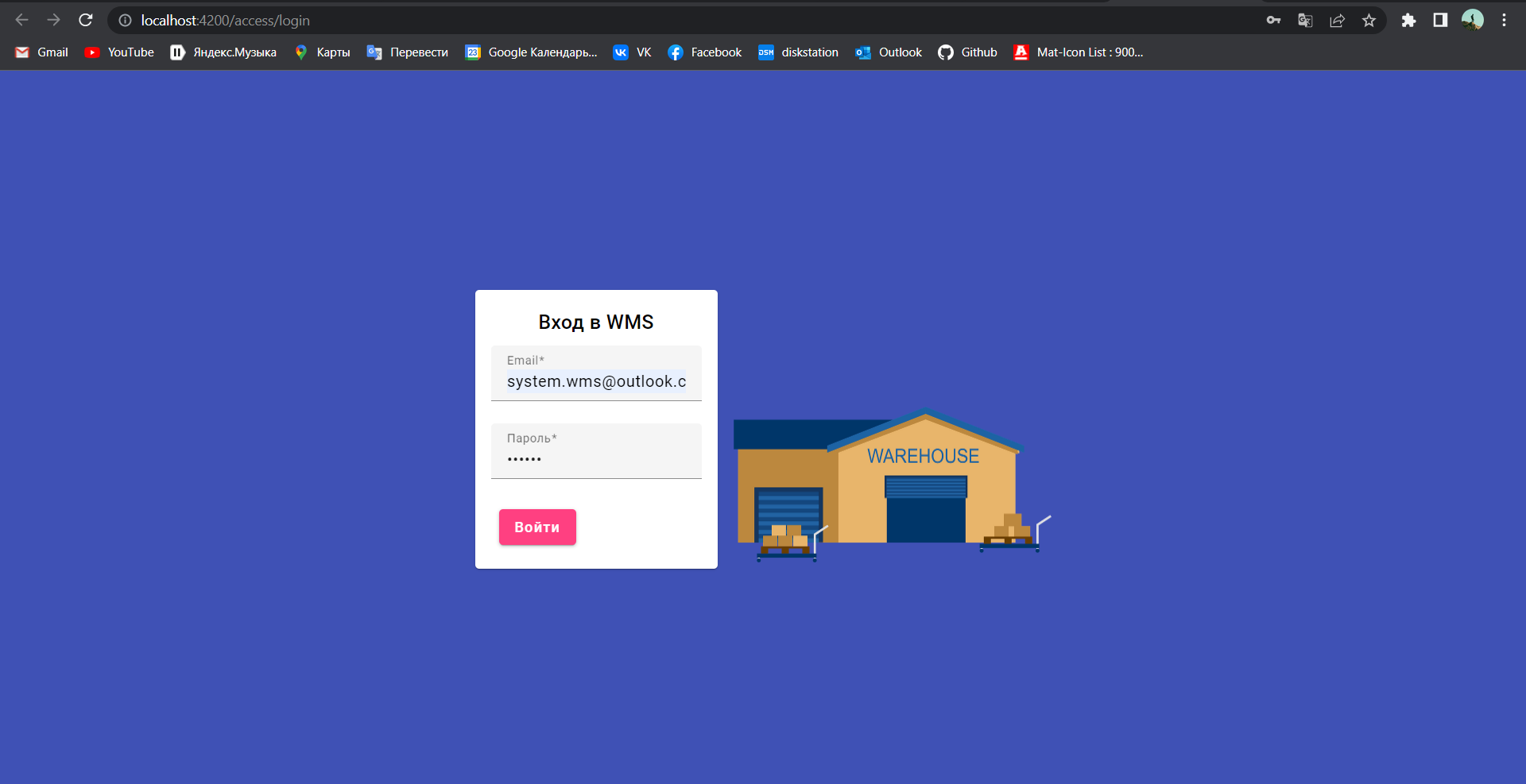


Рисунок 5.2 – Страница входа

Зарегистрироваться в системе сам пользователь не может, для этого нужен администратор, он регистрирует пользователя, после чего данные для входа приходят пользователю на почту (рисунок 5.3).

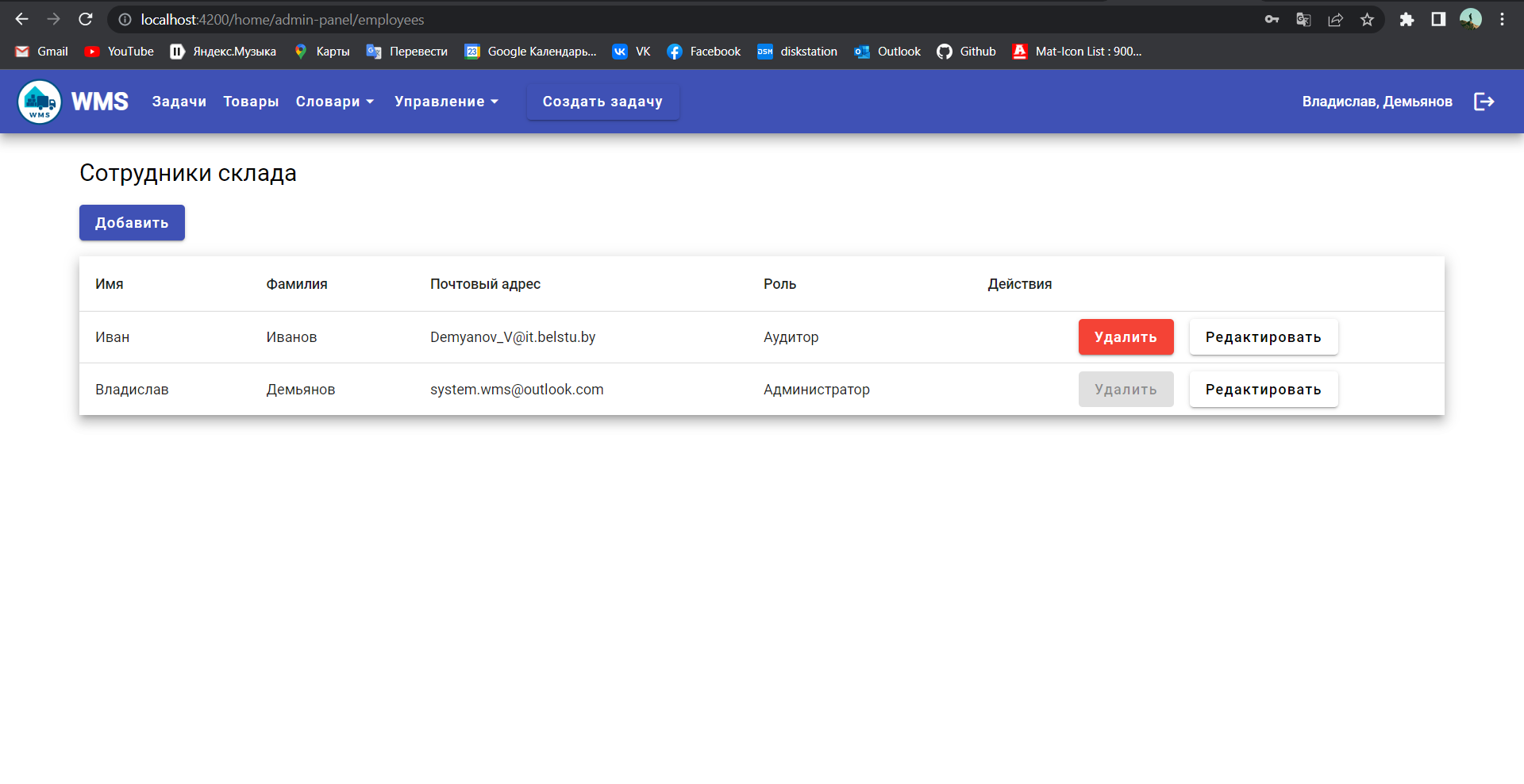


Рисунок 5.3 – Управление пользователями

На рисунке 5.4 представлено сообщение, которое получит пользователь на почту после его регистрации.

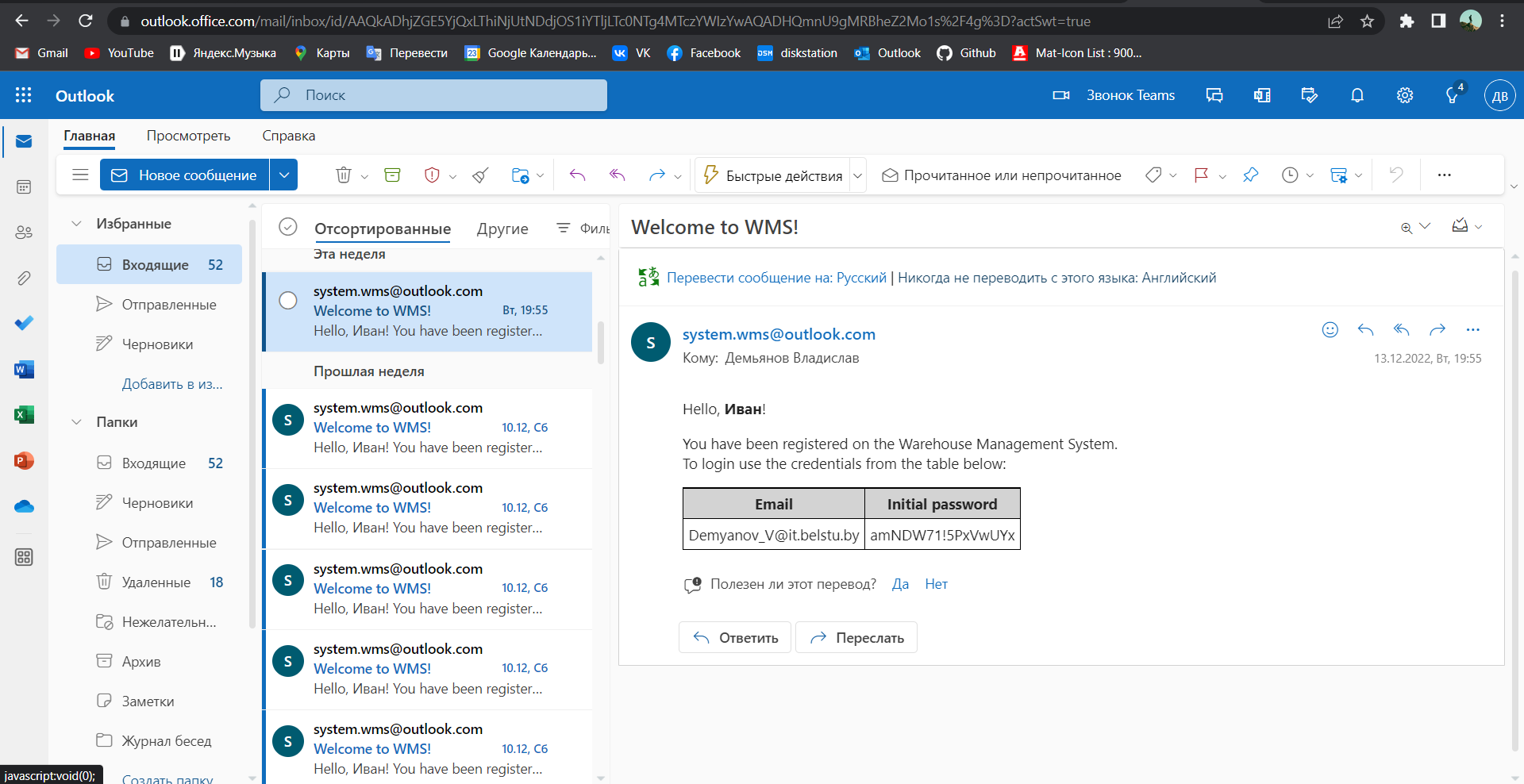


Рисунок 5.4 – Данные для входа

В приложении у пользователя может быть три роли администратор, аудитор и рабочий. В зависимости от роли ему доступны определённые функции.

Не формально в приложении есть ещё роль гость, но кроме как проходить аутентификацию он больше ничего не может.

Диаграмма использования представлена на рисунке 5.5 (для её построения был использован инструмент diagrams.net [16]).

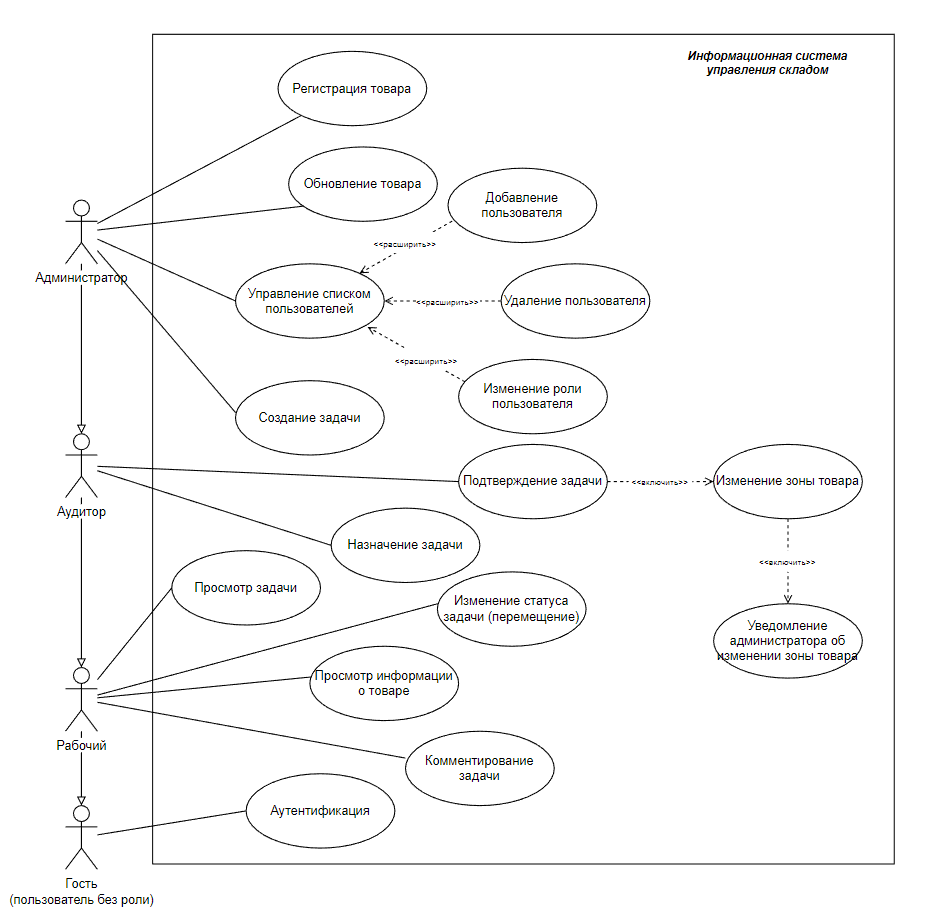


Рисунок 5.5 – Диаграмма использования

Роль администратор обобщает роли: аудитор, рабочий и гость. Данному актёру доступны все варианты использования, которые доступны другим актёрам.

Роль аудитор обобщает роли: рабочий и гость. Данному актёру доступны все варианты использования, актёров с ролями рабочий и гость.

Роль рабочий имеет все варианты использования, которые доступны гостю.

Гость – это пользователь, которому доступна только аутентификация в системе.

Прецеденты системы:

* аутентификация;
* комментирование задачи;
* просмотр информации о товаре;
* изменение статуса задачи (перемещение на доске);
* просмотр задачи;
* назначение задачи;
* подтверждение задачи включает: изменение зоны товара, уведомление администратора об изменении зоны товара
* создание задачи;
* управление списком пользователей расширяется: изменением роли пользователя, удалением пользователя, добавлением пользователя;
* обновление товара;
* регистрация товара.

Теперь давайте пройдём по приложении за пользователя с ролью администратор, так как ему доступны все функции приложения.

На рисунке 5.3 можно было увидеть функции добавления, удаления, обновления пользователей. Все они доступны администратору.

На рисунке 5.6 администратор может просматривать товары, которые находятся на складе.

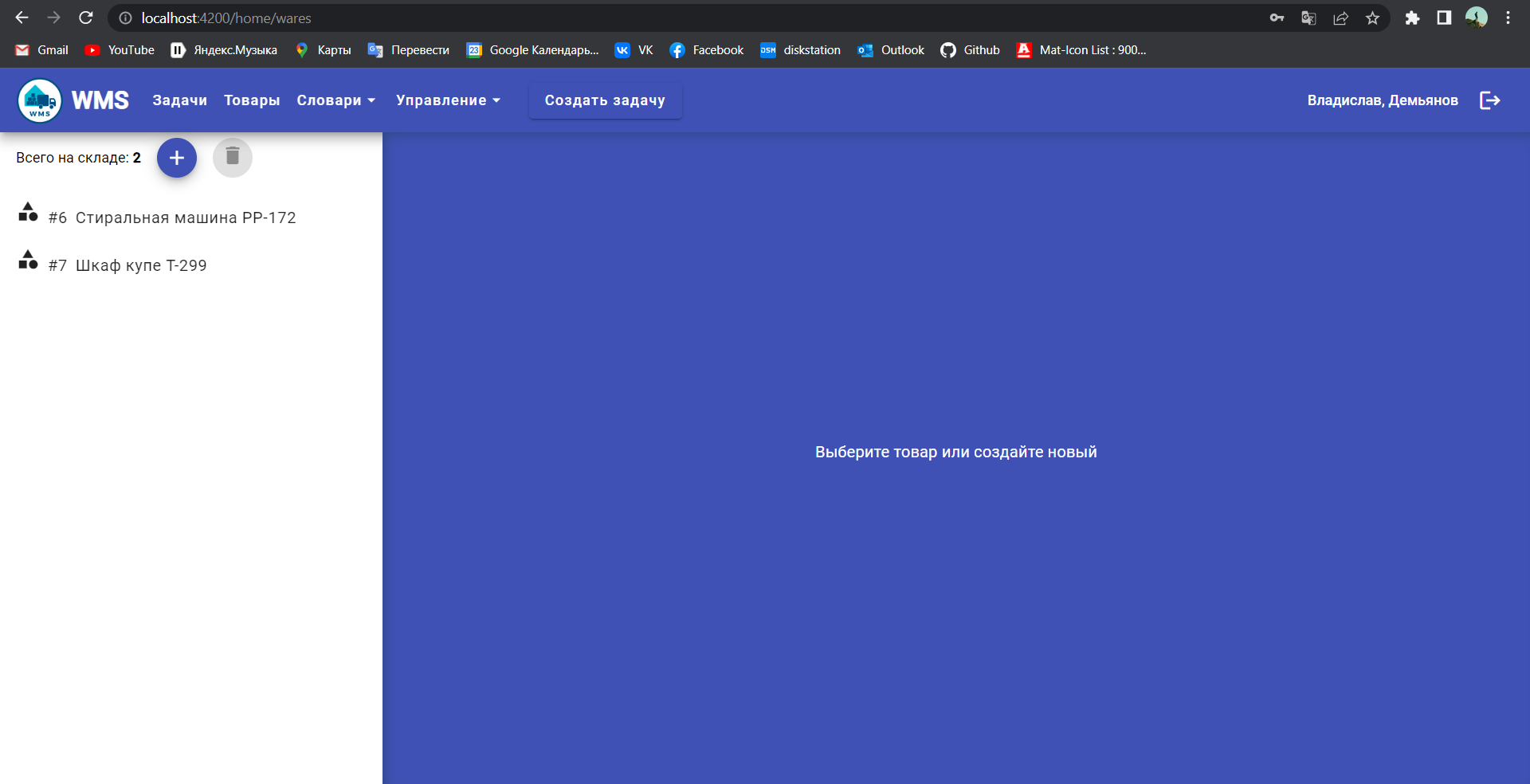


Рисунок 5.6 – Список товаров на складе

Обновление товара представлено на рисунке 5.7.

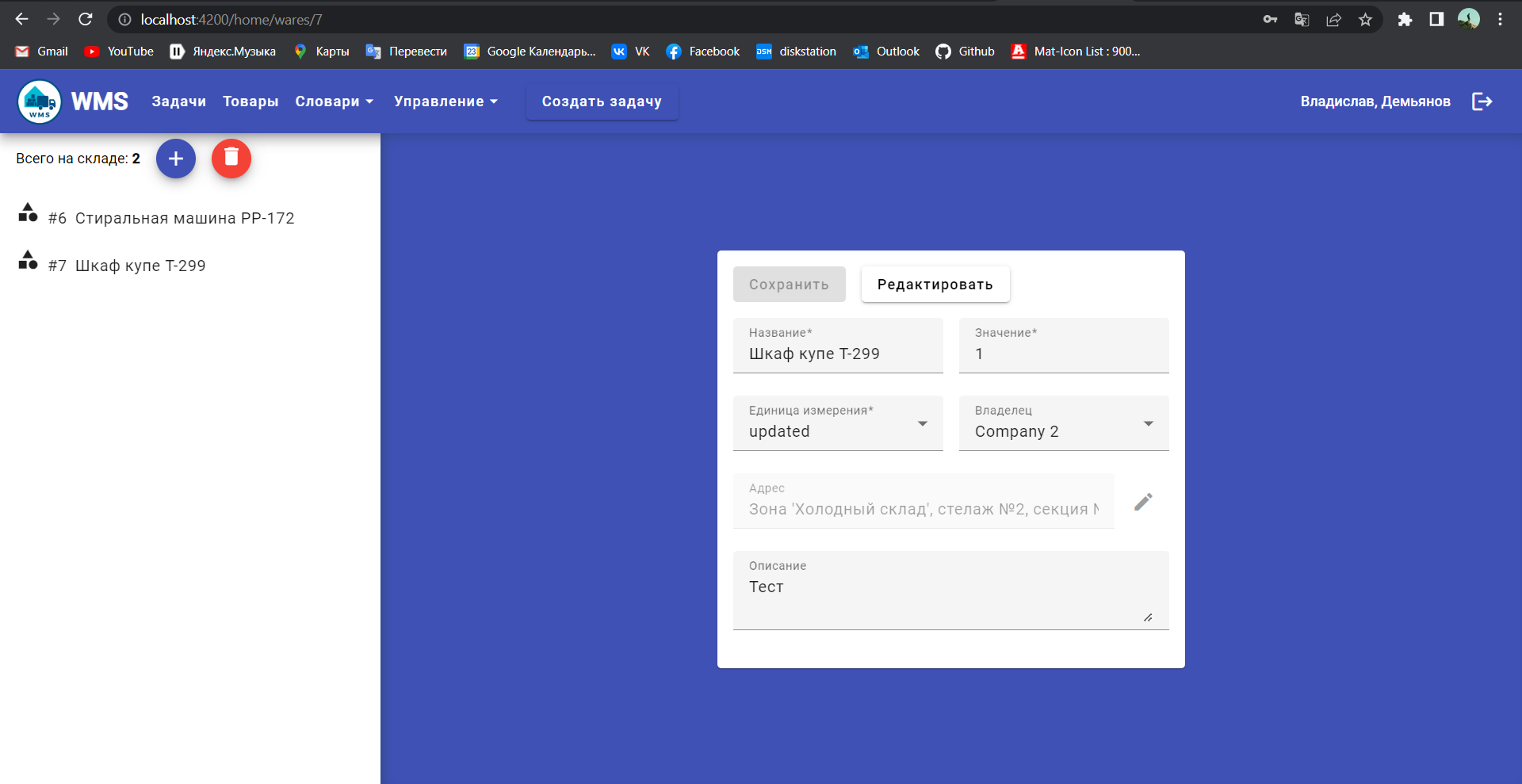


Рисунок 5.7 – Обновление товара

На рисунке 5.8 показан процесс назначение адреса товару. Для того чтобы изменить адрес товара нужно нажать на карандашик, после чего появится всплывающее окно для ввода адреса.

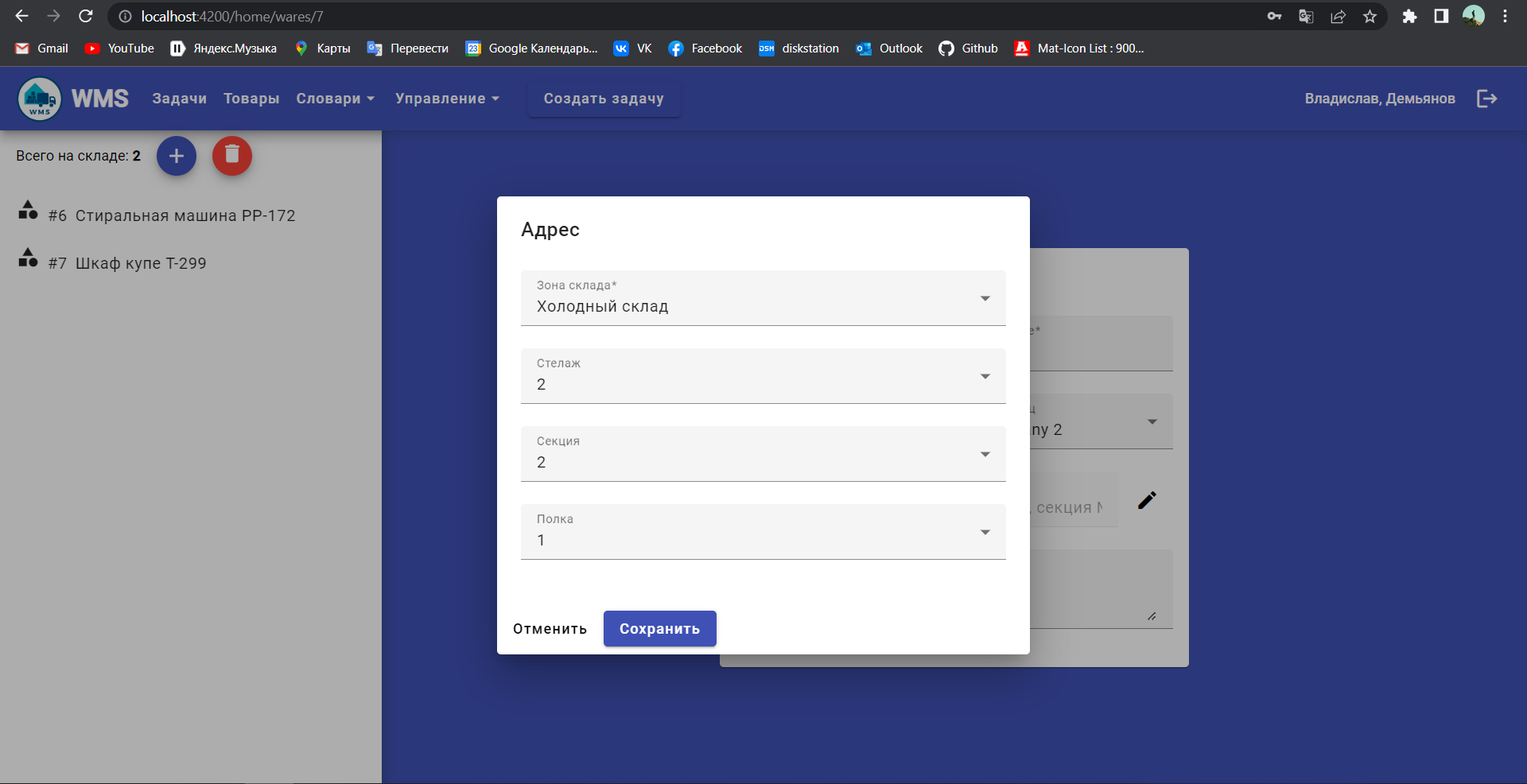


Рисунок 5.8 – Назначение нового адреса товару

Назначение адреса происходит по следующему алгоритму:

* выбрать зону склада;
* подгружаются стелажи, которые находятся в этой зоне;
* выбрать стелаж;
* подгружаются секции, которые находятся в выбранном стелаже;
* выбрать секцию;
* подгружаются полки, которые находятся в секции, причём занятые полки будут заблокированы для выбора;
* выбрать полку;
* нажать на кнопку сохранить.

Чтобы удалить товар пользователь должен выбрать его, для этого нужно ввести либо идентификатор товара в URI или выбрать в его в списке.

Затем нужно нажать на кнопку удалить. Она находится в панели инструментов над списком товаров рядом с кнопкой добавления.

Администратор также может редактировать и добавлять товары.

Так как эти процессы похожи, то ниже представлен процесс только создания нового товара.

Для этого нужно нажать на плюсик в панеле инструментов, описанной ранее.

После это появится форма с незаполненными полями. После их заполнения нужно нажать на кнопку сохранить, если будет что-то неправильно введено, то кнопка сохранить будет заблокирована, а под полями появятся ошибки.

Процесс создания товара показа на рисунке 5.9.

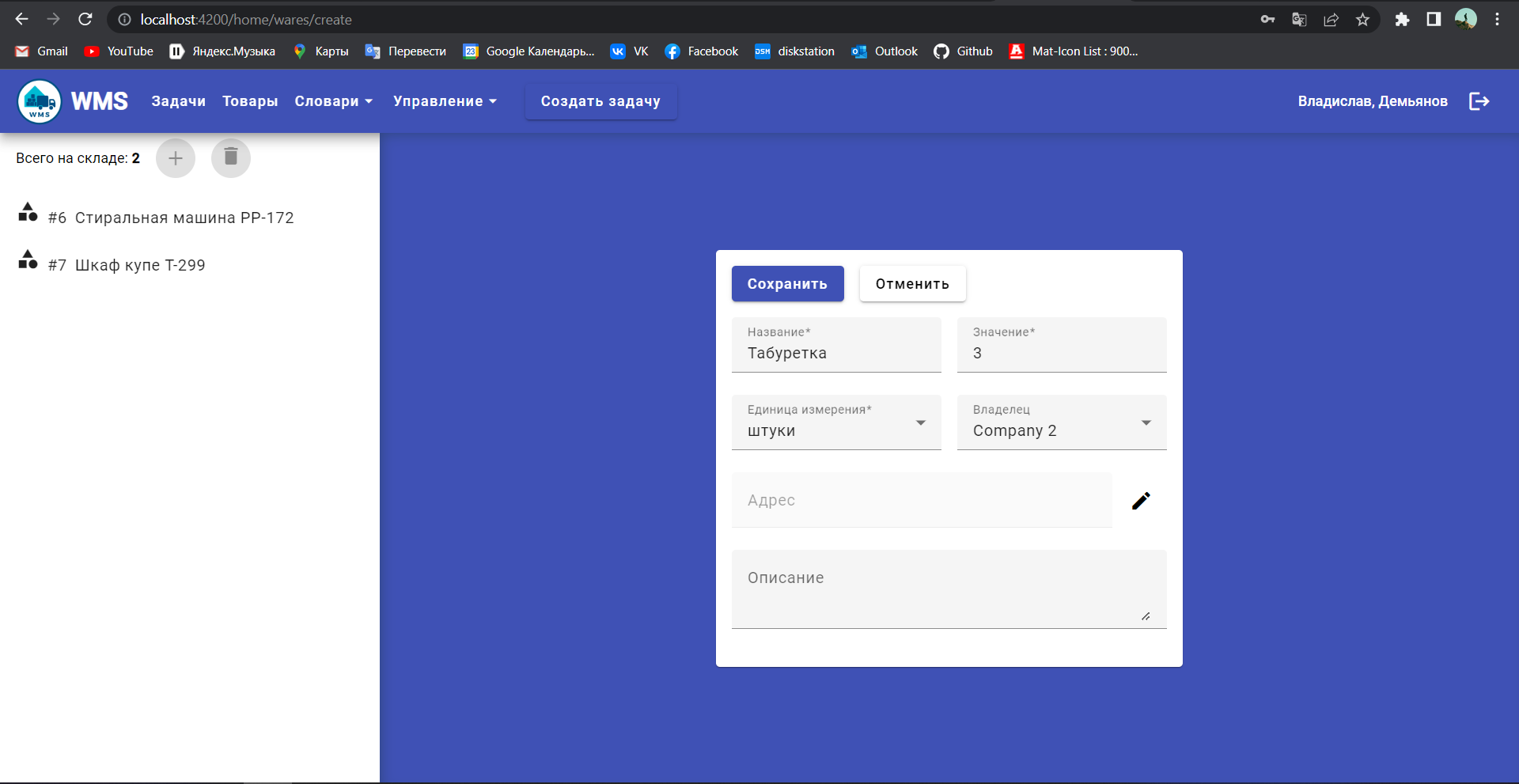


Рисунок 5.9 – Создание товара

Важным функционалом в приложении является управление задачами. Администратор может создавать задачи, перемещать их, обновлять, подтверждать.

На рисунке 5.10 показана доска с задачами.

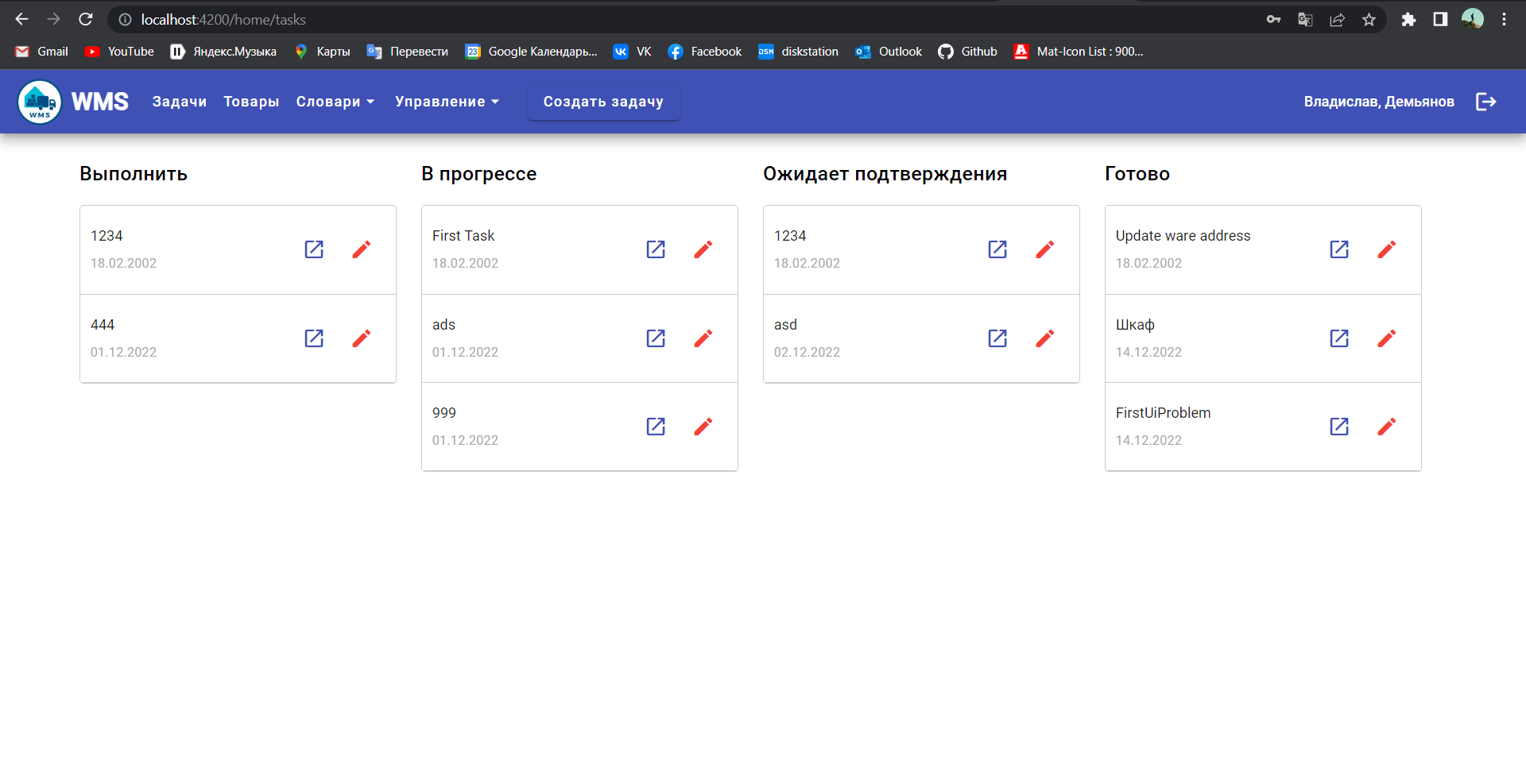


Рисунок 5.10 – Доска с задачами

Для изменения статуса задачи, нужно зажать курсор над задачей, затем перетащить её в другую колонку.

После изменения статуса на почту пользователям, которые как-то связаны с это задачей придёт уведомление.

Уведомление об изменении статуса задачи показано на рисунке 5.11.

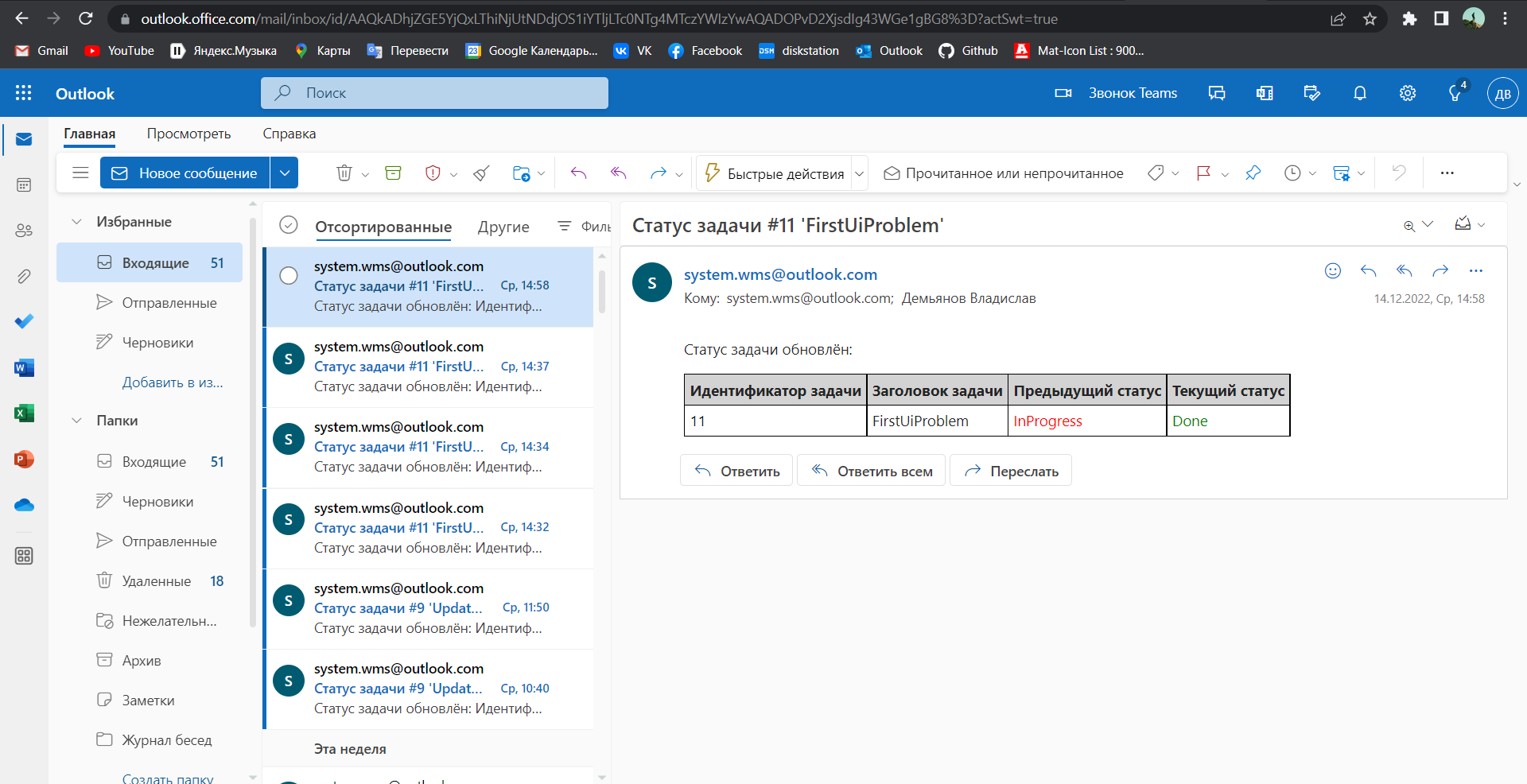


Рисунок 5.11 – Уведомление пользователя об изменении статуса задачи

Создание задачи показано на рисунке 5.12.

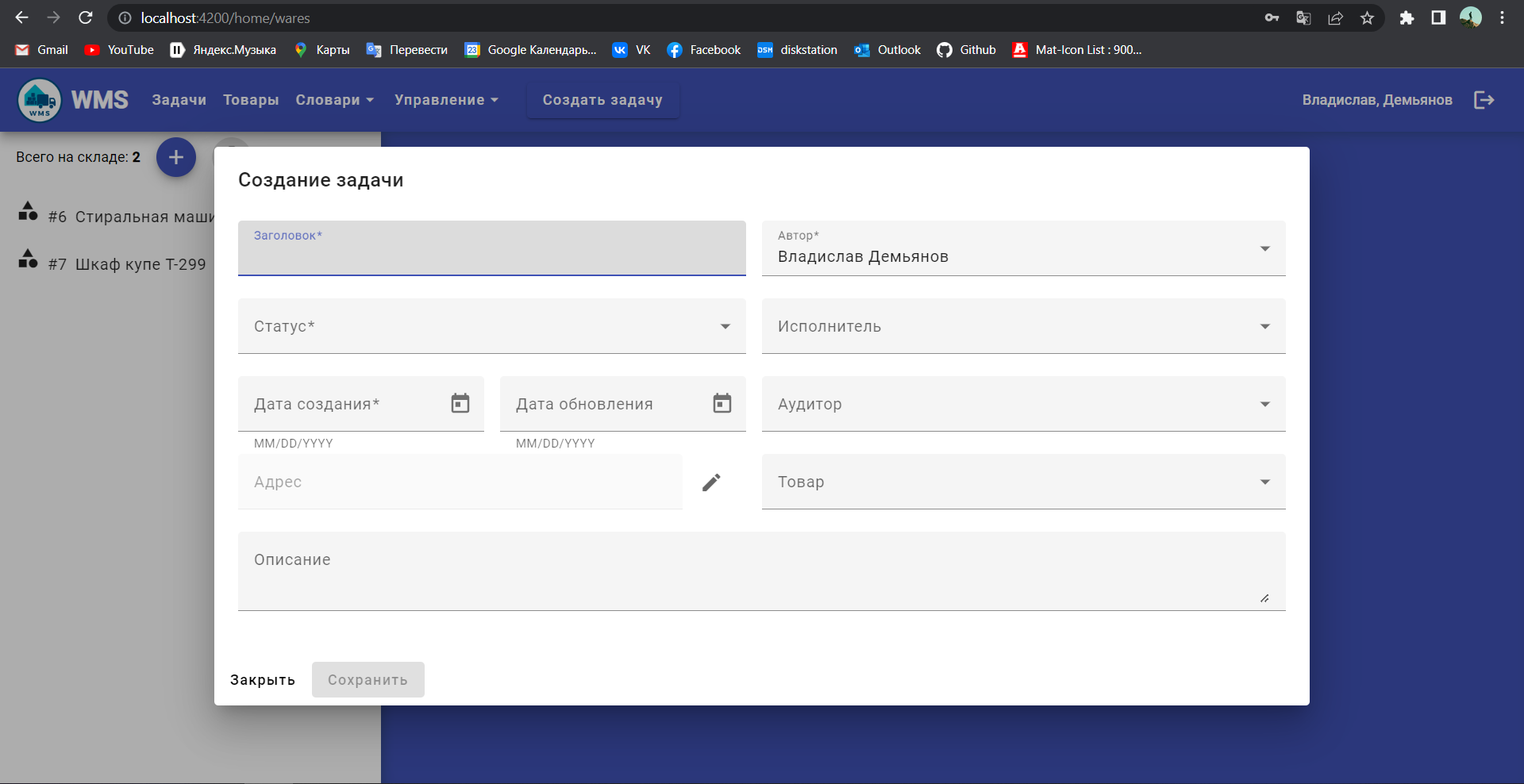


Рисунок 5.12 – Создание задачи

Для создания задачи заполните обязательные поля формы, они помечены звёздочками, а затем нажмите на кнопку сохранить.

Для обновления задачи нужно перейти на доску с задачами и нажать на карандашик.

Для просмотра задачи нужно нажать на иконку, расположенную рядом с карандашиком.

Ещё стоит отметить, что если пользователь создаёт задачу на перемещение, то изменение адреса произойдёт только после изменения статуса задачи на готово.

# **Заключение**

В результате выполнения курсовой работы было разработано Web-приложение. Оно представляет собой систему управления складом.

В рамках работы над проектом был проведён обзор аналогичных решений, изучен теоретический материал, определён основной функционал приложения, выбран технологический стек, спроектирована архитектура приложения и структура базы данных, разработан дизайн интерфейса разработано руководство пользователя, проведено тестирование приложения.

В итоге приложение состоит из базы данных которая содержит 12 связанных таблиц; серверного приложения на ASP.NET, которое предоставляет API для клиента; клиентского приложения на Angular, с которого делаются запросы к серверу, упомянутого выше.

Приложение развёрнуто на платформе Docker и распределяется на три контейнера.

Функционально приложение позволяет:

* аутентифицироваться и авторизироваться;
* поддерживать роли администратора, работника, аудитора;
* управлять пользователями: добавлять, удалять, изменять роль;
* ввод информацию о новом товаре;
* обновление информации о товаре;
* создание задачи;
* назначение задачи и ей одобрение;
* просмотр процессов склада, товаров, данных для инвентаризации;
* комментирование задач;
* перемещать товары;
* уведомлять администратора о перемещении товара в новую зону.

Исходя из полученных результатов можно сделать вывод, что приложение разработано в соответствии с техническим заданием, условия которого выполнены.

# **Список использованных источников**

1 NetSuite [Электронный ресурс]. – Режим доступа: [https://www.netsuite.com/portal/home.shtml – Дата доступа 15.12.2022](https://www.netsuite.com/portal/home.shtml%20–%20Дата%20доступа%2015.12.2022).

2 Entity Framework 6 [Электронный ресурс] / Документация. – Режим доступа: https://professorweb.ru/my/entity-framework/6/level1/. – Дата доступа: 15.12.2022.

3 SQL Server 2019 | Microsoft [Электронный ресурс]. – Режим доступа: <https://www.microsoft.com/en-us/sql-server/sql-server-2019> – Дата доступа 15.12.2022.

4 Angular [Электронный ресурс]. – Режим доступа: <https://angular.io/> – Дата доступа: 15.12.2022.

5 TypeScript: JavaScript With Syntax For Types [Электронный ресурс]. – Режим доступа: <https://www.typescriptlang.org/> – Дата доступа: 15.12.2022.

6 RxJs [Электронный ресурс] / Документация. – Режим доступа: https://rxjs.dev. – Дата доступа: 15.12.2022.

7 Angular Material [Электронный ресурс] / Документация. – Режим доступа: https://material.angular.io. – Дата доступа: 15.12.2022.

8 Docker [Электронный ресурс]. – Режим доступа: <https://www.docker.com/> – Дата доступа: 15.12.2022.

9 Пацей, Н.В. Курс лекций по языку программирования С# / Н.В. Пацей. – Минск: БГТУ, 2021. – 175 с.

10 Смелов, В.В. Курс лекций по программированию веб-сервисов / В.В. Смелов. – Минск: БГТУ, 2022. – 32 с.

11 Блинова Е.А. Курс лекций по базам данных / Е.А. Блинова – Минск: БГТУ, 2022. – 78 с.

12 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: 15.12.2022.

14 OData [Электронный ресурс] / Документация. – Режим доступа: https://www.odata.org. – Дата доступа: 15.12.2022.

15 Swagger.io [Электронный ресурс]. – Режим доступа: <https://swagger.io/>. – Дата доступа: 15.12.2022.

16 Diagrams.net [Электронный ресурс]. – Режим доступа: https://app.diagrams.net. – Дата доступа: 15.12.2022.

# ПРИЛОЖЕНИЕ А

Класс валидатор

|  |
| --- |
| public class RackCreateDataValidator : AbstractValidator<RackCreateData>  {  public RackCreateDataValidator(WmsDbContext dbContext)  {  this.RuleFor(rackCreateData => rackCreateData.AreaId)  .Must((rackCreateData, areaId) => dbContext.Areas.Any(area => area.Id == areaId))  .WithMessage("The area with such id has not been found.");  this.RuleFor(rackCreateData => rackCreateData.VerticalSectionsCount)  .Must((rackCreateData, count) =>  {  var area = dbContext.Areas  .Include(area => area.Racks)  .ThenInclude(rack => rack.VerticalSections)  .FirstOrDefault(area => area.Id == rackCreateData.AreaId);  var currentSectionsCount = area.Racks.Select(rack => rack.VerticalSections.Count).Sum();  return currentSectionsCount + count <= area.MaxVerticalSections;  })  .WithMessage("Area' capacity of sections is overload.");  this.RuleFor(rackCreateData => rackCreateData.ShelfsCount)  .Must((rackCreateData, count) =>  {  var area = dbContext.Areas.FirstOrDefault(area => area.Id == rackCreateData.AreaId);  if (area == null)  {  return false;  }  return count <= area.MaxShelfs;  })  .WithMessage("Area' capacity of shelfs is overload.");  }  } |

# ПРИЛОЖЕНИЕ Б

Настройка OData

|  |
| --- |
| public static class WmsODataMvcBuilderExtensions  {  public static IMvcBuilder AddWmsOData(this IMvcBuilder app) =>  app.AddOData(options => options  .Select()  .Filter()  .OrderBy()  .Count()  .Expand()  .AddRouteComponents("api", GetEdmModel()));  private static IEdmModel GetEdmModel()  {  var modelBuilder = new ODataConventionModelBuilder();  \_ = modelBuilder.EntitySet<LegalEntity>("LegalEntities");  \_ = modelBuilder.EntitySet<Rack>("Racks");  \_ = modelBuilder.EntitySet<Ware>("Wares");  \_ = modelBuilder.EntitySet<Area>("Areas");  \_ = modelBuilder.EntitySet<VerticalSection>("VerticalSections");  \_ = modelBuilder.EntitySet<Shelf>("Shelfs");  \_ = modelBuilder.EntitySet<Problem>("Problems");  \_ = modelBuilder.EntitySet<Comment>("Comments");  return modelBuilder.GetEdmModel();  }  } |

# ПРИЛОЖЕНИЕ В

Ленивая загрузка модулей

|  |
| --- |
| 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 { } |

# ПРИЛОЖЕНИЕ Г

Dockerfile для серверного приложения

|  |
| --- |
| FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base  WORKDIR /app  EXPOSE 8080  EXPOSE 443  FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build  WORKDIR /src  COPY ["WMS.WebApi/WMS.WebApi.csproj", "WMS.WebApi/"]  COPY ["WMS.Database/WMS.Database.csproj", "WMS.Database/"]  COPY ["WMS.Core/WMS.Core.csproj", "WMS.Core/"]  COPY ["WMS.Database.Migrations/WMS.Database.Migrations.csproj", "WMS.Database.Migrations/"]  RUN dotnet restore "WMS.WebApi/WMS.WebApi.csproj"  COPY . .  WORKDIR "/src/WMS.WebApi"  RUN dotnet build "WMS.WebApi.csproj" -c Release -o /app/build  FROM build AS publish  RUN dotnet publish "WMS.WebApi.csproj" -c Release -o /app/publish  FROM base AS final  WORKDIR /app  COPY --from=publish /app/publish .  ENTRYPOINT ["dotnet", "WMS.WebApi.dll"] |

# ПРИЛОЖЕНИЕ Д

Настройка прокси сервера

|  |
| --- |
| server {      listen 80;      sendfile on;      default\_type application/octet-stream;      gzip on;      gzip\_http\_version 1.1;      gzip\_disable "MSIE [1–6]\.";      gzip\_min\_length 256;      gzip\_vary on;      gzip\_proxied expired no-cache no-store private auth;      gzip\_types text/plain text/css application/json application/javascript application/x-javascript text/xml application/xml application/xml+rss text/javascript;      gzip\_comp\_level 9;      root /usr/share/nginx/html;      location / {        try\_files $uri $uri/ /index.html =404;      }  } |