Министерство образования Республики Беларусь

Учреждение образования  
 БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
 ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Рефакторинг и оптимизация программного кода

Отчет

по результатам выполнения лабораторных работ  
и заданий к практическим занятиям

|  |  |  |
| --- | --- | --- |
| Проверила | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | А.В. Шелест |
| зачтено | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (дата защиты) |  |
| Выполнила | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | А.Е. Дорохова  гр. 214371 |

Минск, 2025

**СОДЕРЖАНИЕ**

[Ссылки на репозитории 3](#_heading=h.k89drr64ojjr)

[1 Архитектура программного средства 4](#_heading=h.8hf957j2sd37)

[1.1 Диаграмма вариантов использования 4](#_heading=h.sii3vksnil9j)

[1.2 Нотация моделирования C4-модель. 5](#_heading=h.6qpixqfx1vs)

[1.3 Система дизайна пользовательского интерфейса 7](#_heading=h.pcysc852e9v4)

[1.4 Описание спроектированной архитектуры по уровням   
Clean Architecture 9](#_heading=h.oh712wpcmet)

[2 Проектирование пользовательского интерфейса ПС 11](#_heading=h.f14tsmyw8sey)

[3 Реализация клиентской части ПС 14](#_heading=h.1bgvmjczmgj6)

[4 Спроектировать схему бд и представить описание ее сущностей и их атрибутов 17](#_heading=h.fxtl2ktbgztk)

[5 Представить детали реализации пс через UML-диаграммы 19](#_heading=h.g8riwq5f3g9d)

[5.1 Описание статических аспектов программных объектов. 19](#_heading=h.uc1r6nmz27eh)

[5.2 Описание динамических аспектов поведения программных объектов 22](#_heading=h.uab7rxfisk2w)

[6 Документация к ПС с open api 25](#_heading=h.o3gl90qpxggs)

[7 Реализация системы аутентификации и авторизации пользователей ПС и механизмов обеспечения безопасности данных 28](#_heading=h.g9hyom1ygdln)

[8 Unit- и интеграционные тесты 31](#_heading=h.rohg5edm3ok5)

[9 Описание процесса развертывания ПС 35](#_heading=h.ohmrf31vzsv7)

[10 Разработка руководства пользователя 37](#_heading=h.z8adg43r627k)

# **Ссылки на репозитории**

<https://github.com/hhaze17/DorakhavaAE_214371_RIOPK_Front>

<https://github.com/hhaze17/DorakhavaAE_214371_RIOPK_Server>

<https://github.com/hhaze17/DorakhavaAE_214371_RIOPK_Spec>

# **1 АРХИТЕКТУРА ПРОГРАММНОГО СРЕДСТВА**

## **1.1 Диаграмма вариантов использования**

Диаграмма вариантов использования является ключевым инструментом визуализации взаимодействия различных пользовательских групп с программой. Она охватывает разнообразные потребности и интересы, предоставляя общий обзор функциональности программы. В данной диаграмме представлены основные функции программного средства. Участниками системы выступают сотрудник (например, мерчендайзер) и клиент магазина, каждый из которых обладает определённым набором действий.

В рамках проекта особое внимание уделяется коммерческой зоне –пространству и процессам, непосредственно связанным с выбором, продажей и выкладкой товаров, перемещением товаров в рамках коммерческих зон, а также взаимодействием с клиентом, посредством выбора зоны выдачи заказа.

Cотрудник имеет доступ к полному управлению товарной базой, включая добавление, редактирование и удаление товаров, может управлять заказами и торговыми зонами, перемещать товары между ними. Ему также доступны заказы клиентов, которые сотрудник обрабатывает, и просмотр собственных задач, например, новых заказов или отслеживание низких запасов товаров в коммерческой зоне.

Клиент магазина может просмотреть каталог товаров или найти товар по поиску, добавить его в корзину, после чего сделать заказ, непосредственно выбрав зону выдачи, и отследить статус заказа.

Диаграмма вариантов использования представлена на рисунке 1.1.

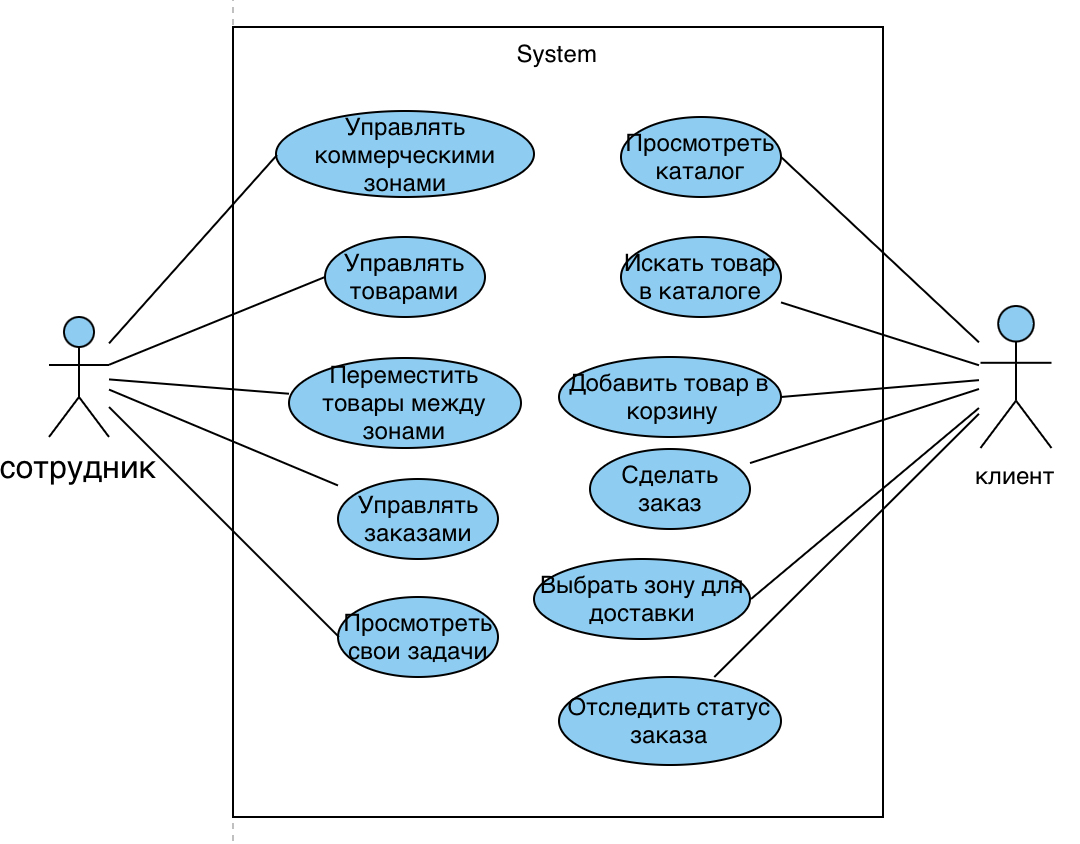


Рисунок 1.1 – Диаграмма вариантов использования

Диаграмма отражает логичную иерархию пользовательского взаимодействия с программной системой.

## **1.2 Нотация моделирования C4-модель.**

На рисунке 1.2 представлена диаграмма контекста системы, отражающая основные внешние взаимодействия программного средства с пользователями и базой данных. Пользователи (сотрудник и клиент) получают доступ к функционалу системы через веб-интерфейс. Приложение обрабатывает их действия и взаимодействует с базой данных для выполнения операций.

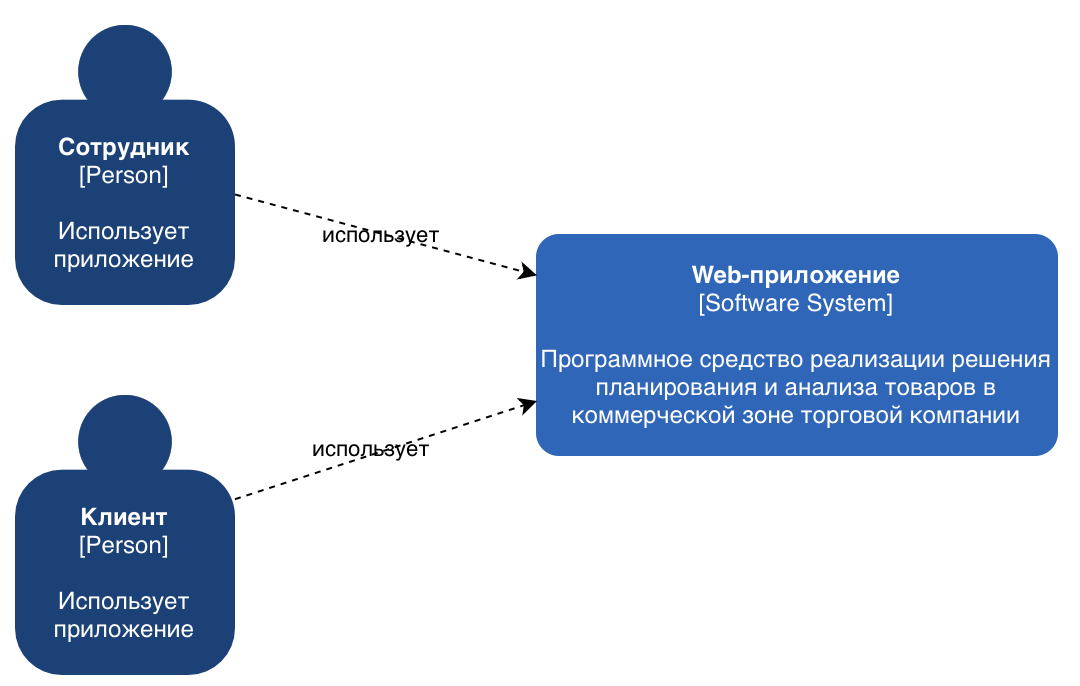


Рисунок 1.2 – Контекстный уровень представления архитектуры

На рисунке 1.3 показана диаграмма контейнеров, в которой детализирована архитектура приложения на уровне логических блоков. Система состоит из трёх основных контейнеров: фронтенда на *React.js*, бэкенда на *Express.ts* и базы данных *MongoDB*, а также *API*. Контейнеры взаимодействуют через *HTTP*-запросы.

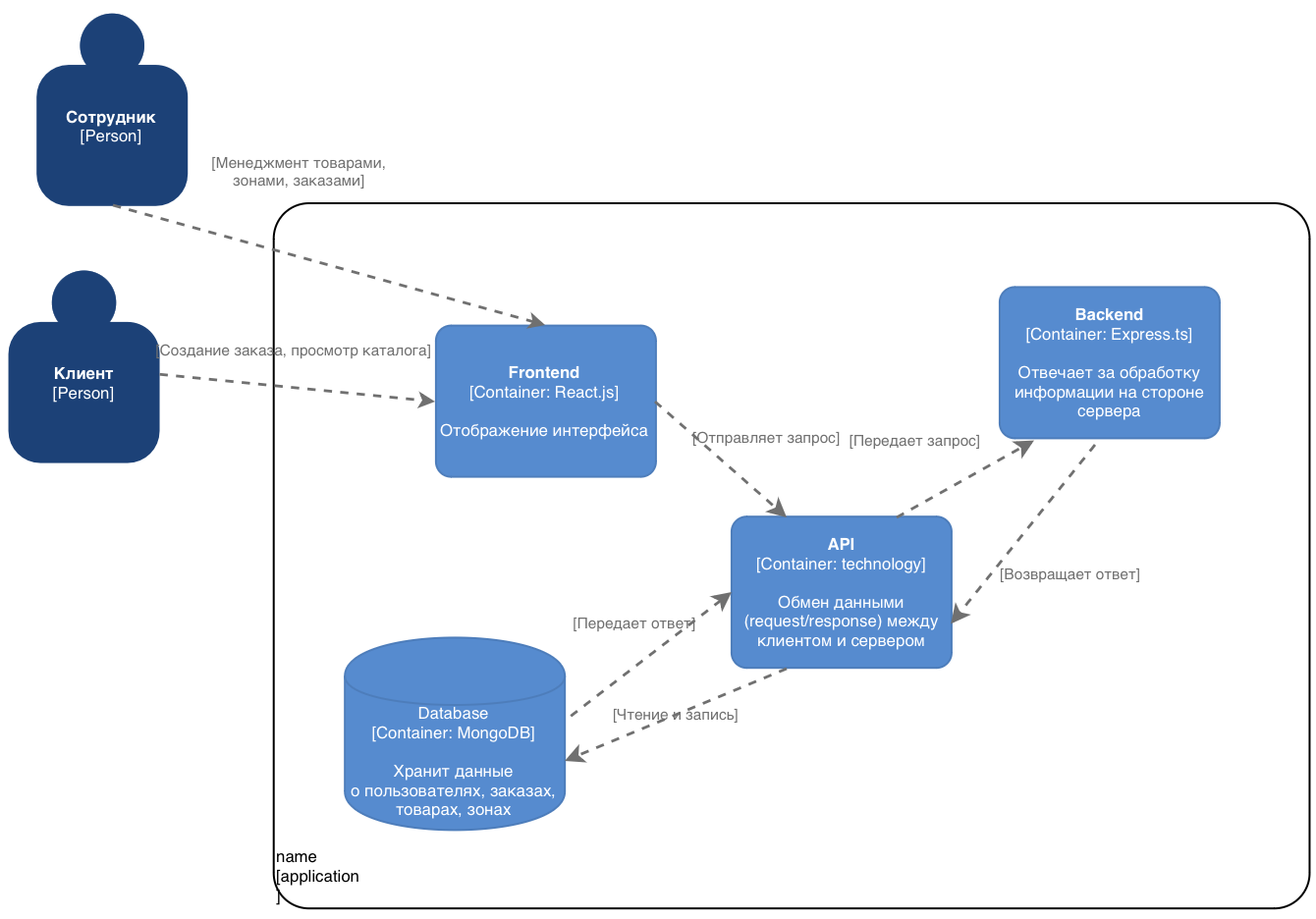


Рисунок 1.3 – Контейнерный уровень представления архитектуры

На рисунке 1.4 приведена диаграмма компонентов, демонстрирующая структуру основных функциональных модулей системы. Для бэкенда выделены контроллеры, отвечающие за авторизацию, работу с товарами, зонами и заказами. Компонент фронтенда реализован как модуль, который обеспечивает доступ к бизнес-логике через *REST API*.

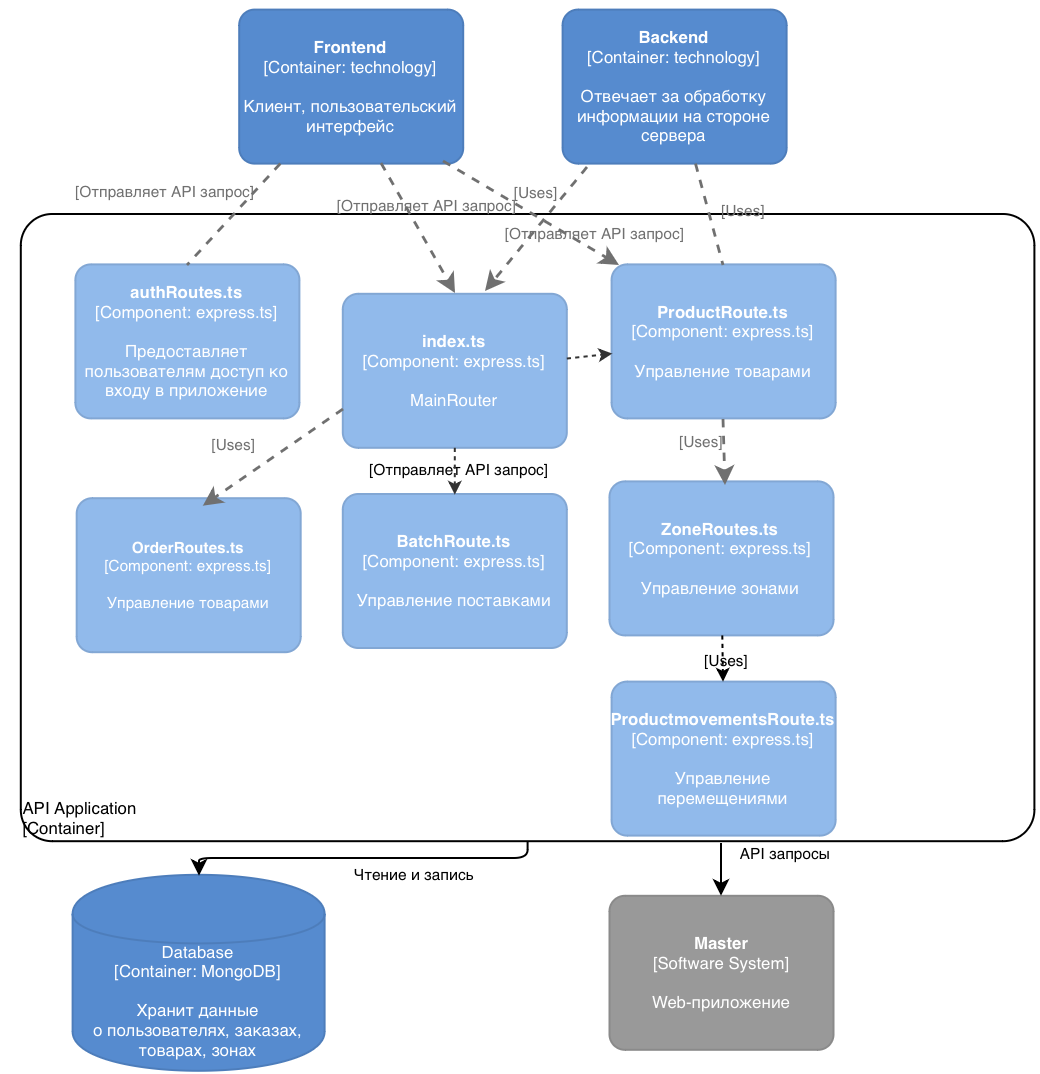


Рисунок 1.4 – Компонентный уровень представления архитектуры

На рисунке 1.5 представлен кодовый уровень архитектуры, иллюстрирующий взаимодействие роутеров в системе. К основному *index.ts* подключены остальные ключевые роутеры, реализующие бизнес-логику.

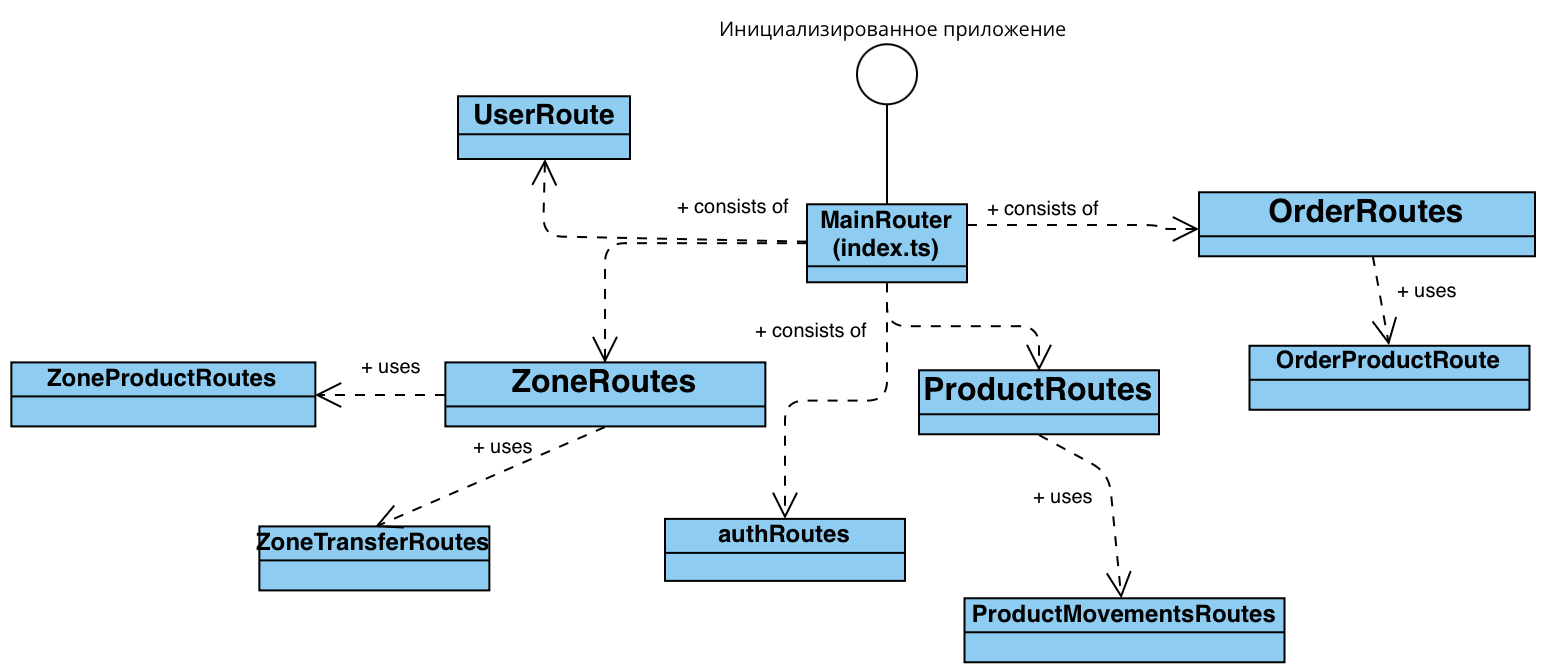


Рисунок 1.5 – Кодовый уровень представления архитектуры

Данная диаграмма отражает структуру маршрутов (*routes*) на уровне серверной части приложения, реализованной на *TypeScript* с использованием *Express.js.* Архитектура построена модульно: каждый функциональный блок (товары, заказы, зоны, пользователи) имеет отдельный модуль маршрутов.

## **1.3 Система дизайна пользовательского интерфейса**

Система дизайна пользовательского интерфейса представляет собой совокупность визуальных и функциональных стандартов, применяемых при разработке интерфейса программного средства, с целью обеспечения удобства, единообразия, доступности и эффективности взаимодействия пользователя с системой.

На рисунке 1.6 изображена разработанная система дизайна, которая включает основные элементы пользовательского интерфейса. Эта система определяет структуру и внешний вид статусов уведомлений, иконки, элементов навигации, цветовую палитру, шрифт и другие детали интерфейса.

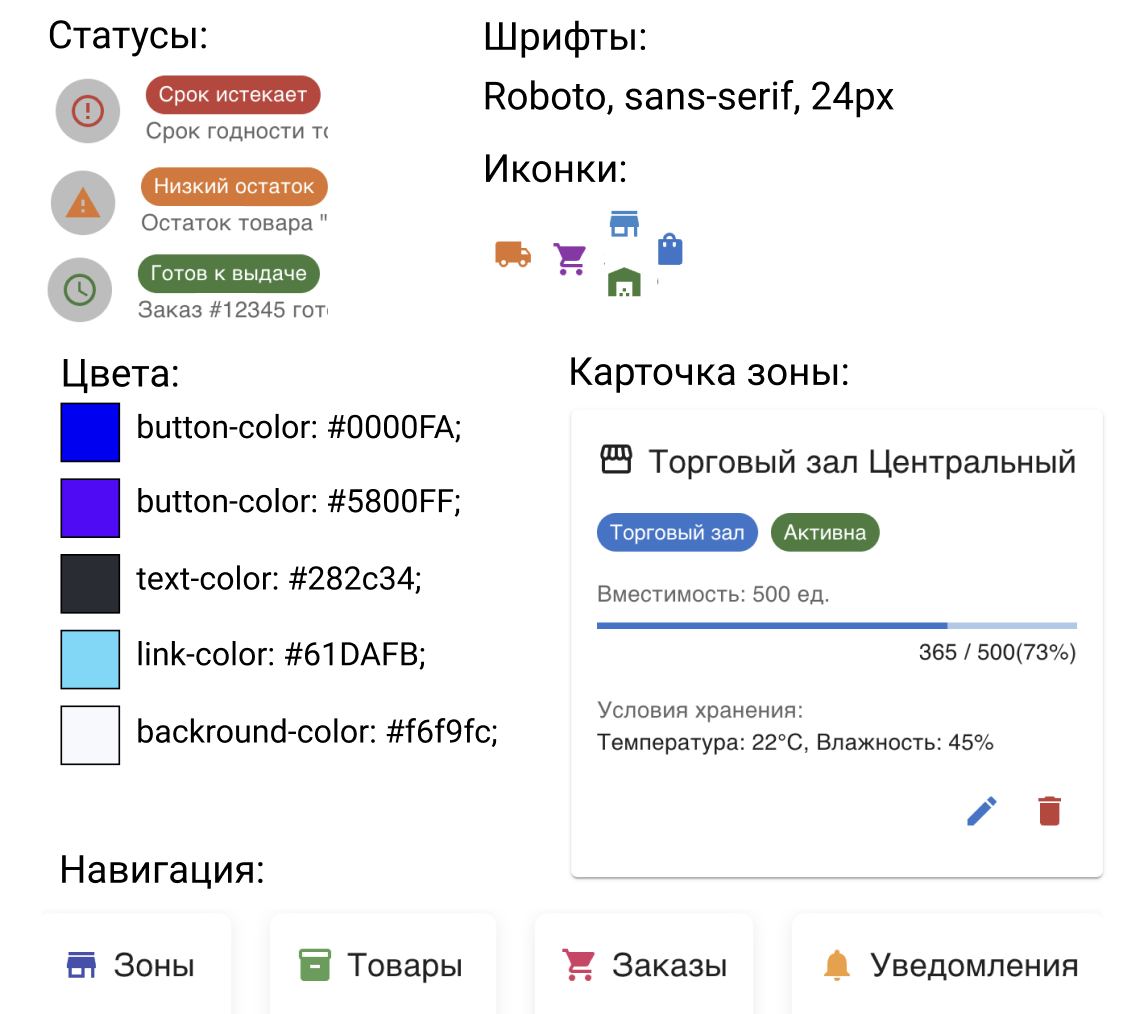
****

Рисунок 1.6 – Система дизайна пользовательского интерфейса программного средства

Результат внедрения системы дизайна – создание профессионального, удобного и масштабируемого интерфейса, обеспечивающего эффективную работу сотрудников магазина и клиентов.

## **1.4 Описание спроектированной архитектуры по уровням Clean Architecture**

Архитектура приложения следует принципам *Clean Architecture*, обеспечивая разделение ответственности, масштабируемость и удобство поддержки. Она условно разделена на четыре уровня:

1 *Entities* (Сущности)

На этом уровне находятся бизнес-сущности и бизнес-логика.

Примеры сущностей:

1 *Product* – модель товара со свойствами *Name, BaseCost, MarkupPercent, FinalPrice, CreatedAt.*

2 *Zone* – модель зоны*.*

3 *User* – сущность пользователя с уровнем доступа и хешированным паролем.

4 *ProductMovements* – модель перемещений товаров между коммерческими зонами.

5 *Orders* – модель заказов пользователя.

2 *Use Cases* (Интеракторы / Приложение)

Реализация прикладной логики. Эти классы управляют потоками данных между сущностями и внешними слоями. Примеры: расчёт конечной цены заказа из нескольких товаров, проверка логина и хеша пароля.

3 *Interface Adapters* (Инфраструктура / контроллеры / репозитории). Включает контроллеры *ProductController, ProductMovementController, OrderController, AuthController, ZoneController*).

4 *Frameworks & Drivers* (Инфраструктурный уровень)

На этом уровне располагаются фреймворки и инструменты, с которыми работает система.

Состав:

1 *Express.js* – *Node.js-*фреймворк, используемый для построения *HTTP* сервера и обработки маршрутов. Используется в *MainRouter*, *ProductRoutes*, *ZoneRoutes* и др.

2 *TypeScript* – Надстройка над *JavaScript*, обеспечивающая статическую типизацию и поддержку современных языковых конструкций.Используется во всем серверном коде.

3 *MongoDB (и Mongoose)* – *NoSQL*-база данных, используемая для хранения информации о пользователях, товарах, зонах, заказах и перемещениях. Используется через *Mongoose* — *ORM/ODM* для *MongoDB*.

4 *React.js* – Фронтенд-библиотека для создания интерфейса клиента (вне представленной диаграммы, но входит в систему в целом).Используется для рендеринга интерфейса сотрудника и клиента.

5 *JWT (jsonwebtoken)* – Библиотека для создания и верификации токенов авторизации. Используется в *authRoutes* и *middleware* проверки доступа.

6 *bcrypt* – Библиотека для хеширования паролей. Используется в *UserRoute* для безопасной регистрации и входа.

7 *Dotenv* – Загрузка переменных окружения из .*env* файла (например, строки подключения к *MongoDB*).Используется при инициализации приложения.

# **2 ПРОЕКТИРОВАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА ПС**

Логика действий пользователя в программном средстве. Чтобы лучше понять, как пользователи будут взаимодействовать с программным средством, была создана диаграмма *User-flow*, которая отражает основные этапы действий пользователя. Эта диаграмма позволяет визуально представить последовательность шагов, которые пользователь выполняет при использовании программы.

Пользователь с ролью «Сотрудник» после входа в аккаунт попадает на дашборд. С этого экрана он может воспользоваться меню навигации в хедере или перейти в разделы, обеспечивающие выполнение ключевых бизнес-процессов:

1 Управление заказами – обработка и отслеживание заказов.

2 Управление зонами – настройка и мониторинг зон.

3 Перейти к задачам – задачи сотрудника, отслеживание сроков годности, инвентаря, необработанных заказов.

На рисунке 2.1 также графически выделены крупные процессы: вход в личный кабинет, управление товарами, управление зонами, товары в зонах, а также заказы и панель сотрудника. Эти процессы отражают целевую логику бизнес-задач, решаемых сотрудником при работе с системой.

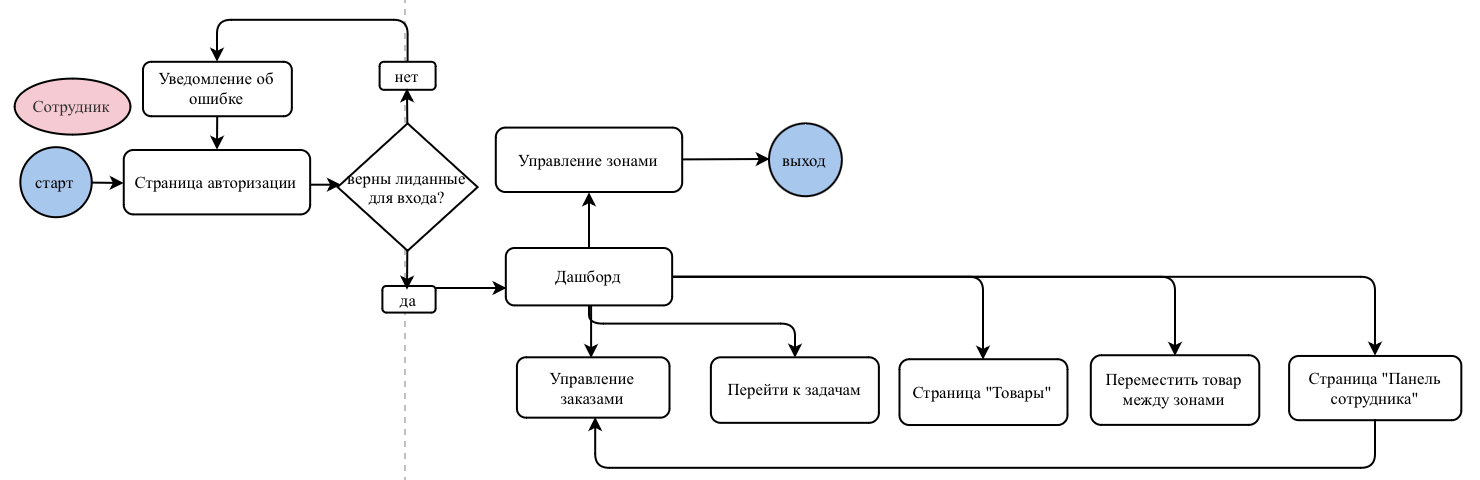


Рисунок 2.1 – *User-flow* диаграмма логики действий пользователя «Сотрудник»

Пользователь с ролью «Клиент» после успешного входа в систему попадает на главную страницу, с которой получает доступ к основным функциям:

1 Каталог товаров – с возможностью добавления товара в корзину и выбора зоны наличия товара, а в дальнейшем оформления заказа.

2 Мои заказы – активные и завершенные заказы, мониторинг статуса заказа.

Результат представлен на рисунке 2.2

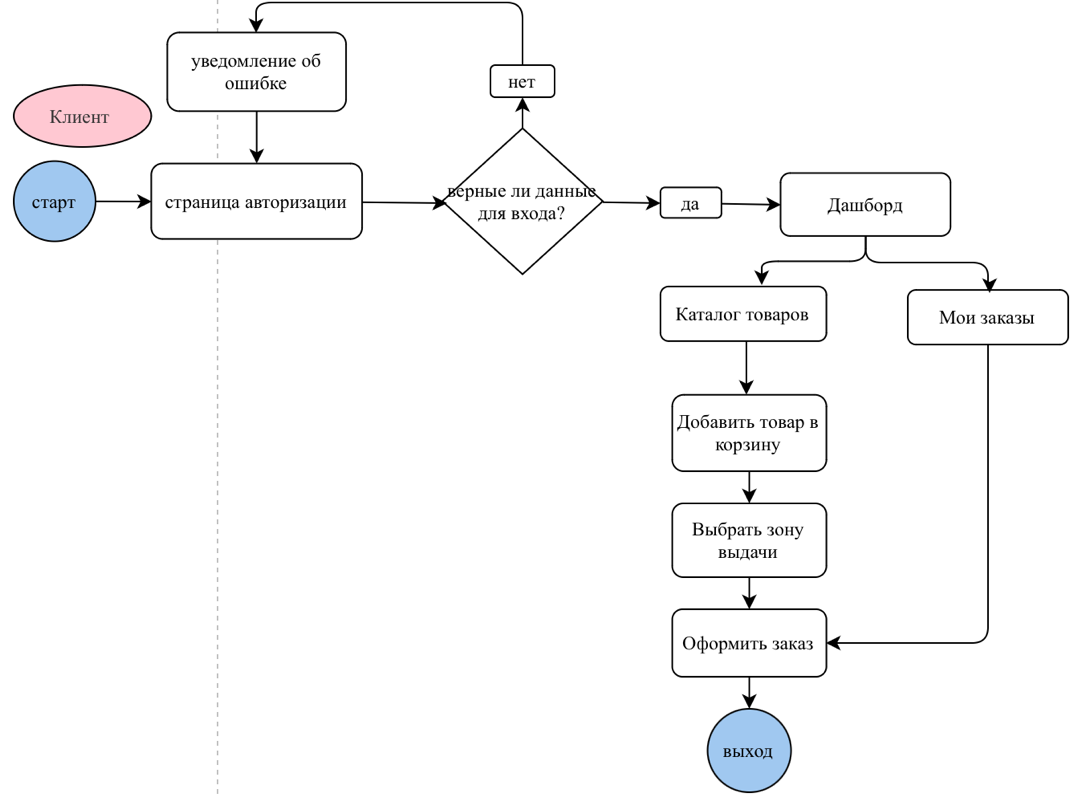


Рисунок 2.2 – *User-flow* диаграмма логики действий пользователя «Клиент»

На схеме визуально выделены основные бизнес-процессы: просмотр каталога товаров, создание заказа с возможностью выбора зоны доставки, просмотр заказов, что позволяет отразить реальную логику взаимодействия клиента с системой.

# **3 РЕАЛИЗАЦИЯ КЛИЕНТСКОЙ ЧАСТИ ПС**

На данном этапе была реализована визуальная часть программной системы в соответствии с архитектурой и функциональными требованиями, определёнными в предыдущих заданиях. Интерфейс обеспечивает удобную навигацию и доступ ко всем ключевым функциям: авторизация, регистрация, работа с товарами и заказами, зонами.

В качестве основных инструментов разработки пользовательского интерфейса программного средства выбраны *JavaScript* и библиотека *React*.

*JavaScript* является основным языком программирования для разработки интерактивных пользовательских интерфейсов в веб-приложениях. Его использование позволяет реализовать динамическое поведение элементов интерфейса, обработку событий, взаимодействие с сервером через *API* и обновление *DOM*-структуры страницы без полной перезагрузки.

На рисунке 3.1 продемонстрирован дизайн интерфейса сотрудника.

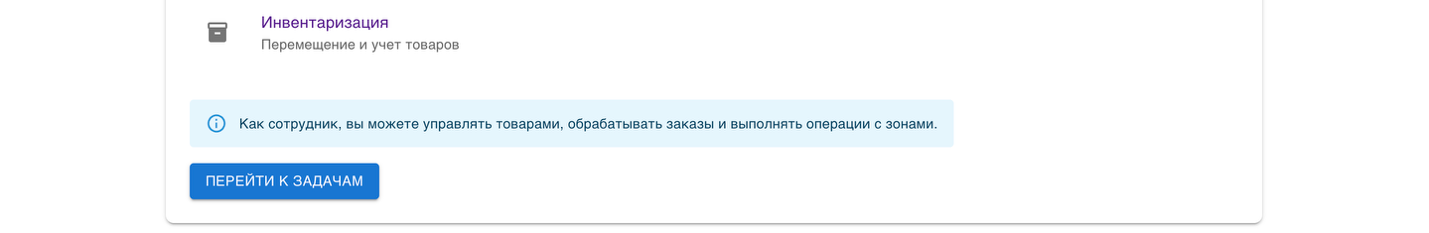
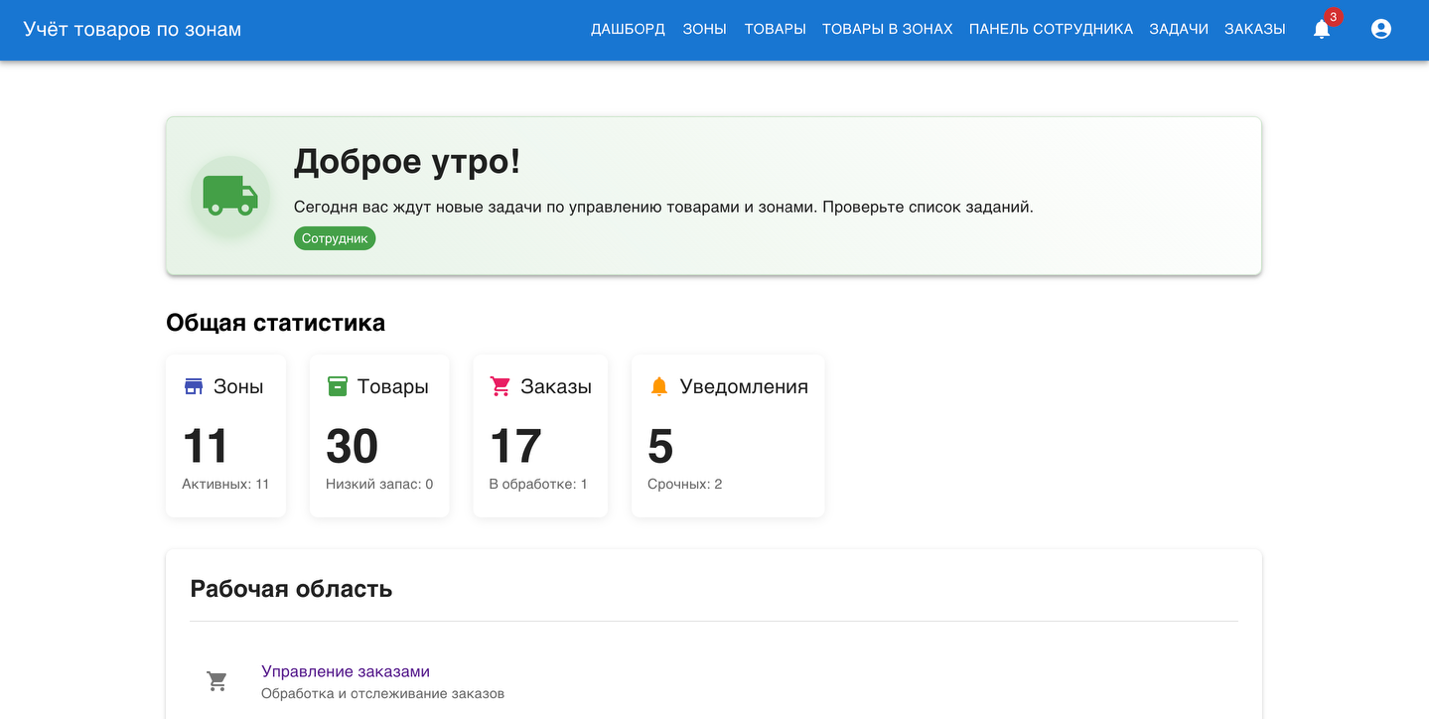


Рисунок 3.1 – Скриншот интерфейса Сотрудника

На рисунке 3.2 продемонстрирована страница управления зонами.



Рисунок 3.2 – Управление зонами

На рисунке 3.3 продемонстрирована страница управления заказами. Здесь есть возможность отследить активные и завершенные заказы, а также отфильтровать их по статусу.

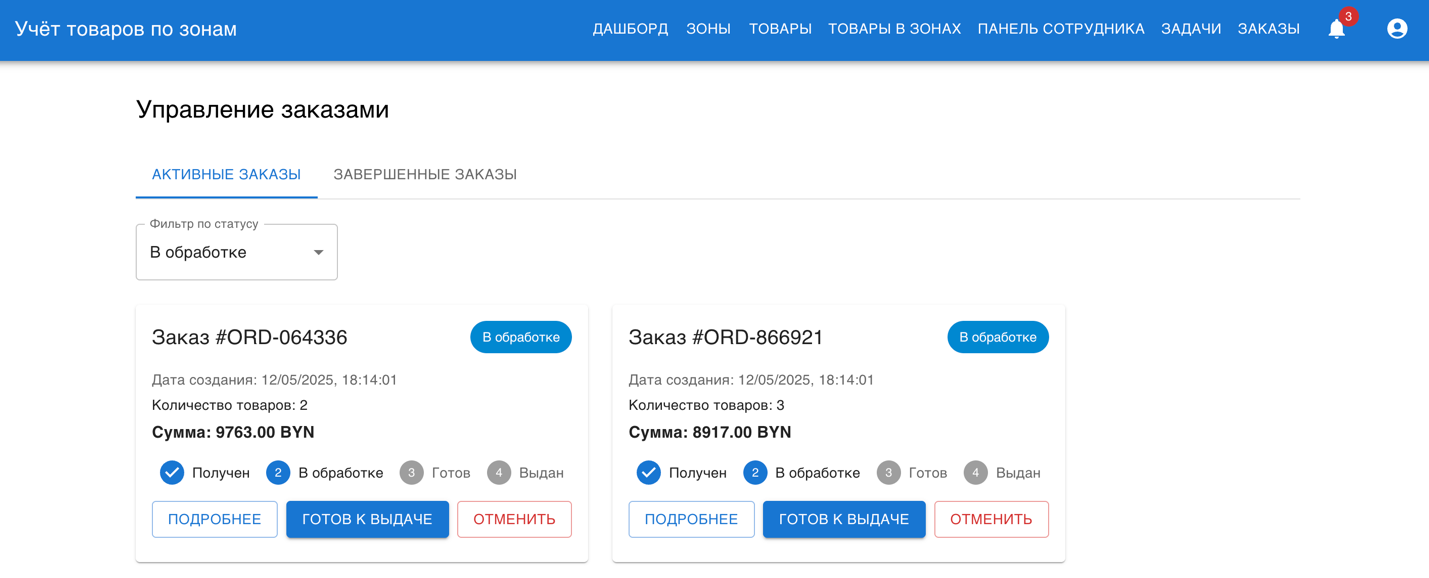


Рисунок 3.3 – Управление заказами

Преимущества выбора *React*: быстрая разработка за счет переиспользуемых компонентов; удобная организация проекта и масштабируемость; поддержка современных подходов; подходит как для простых интерфейсов, так и для сложных бизнес-приложений.

# **4 СПРОЕКТИРОВАТЬ СХЕМУ БД И ПРЕДСТАВИТЬ ОПИСАНИЕ ЕЕ СУЩНОСТЕЙ И ИХ АТРИБУТОВ**

На этапе физического проектирования была создана схема, представленная на рисунке 4.1.

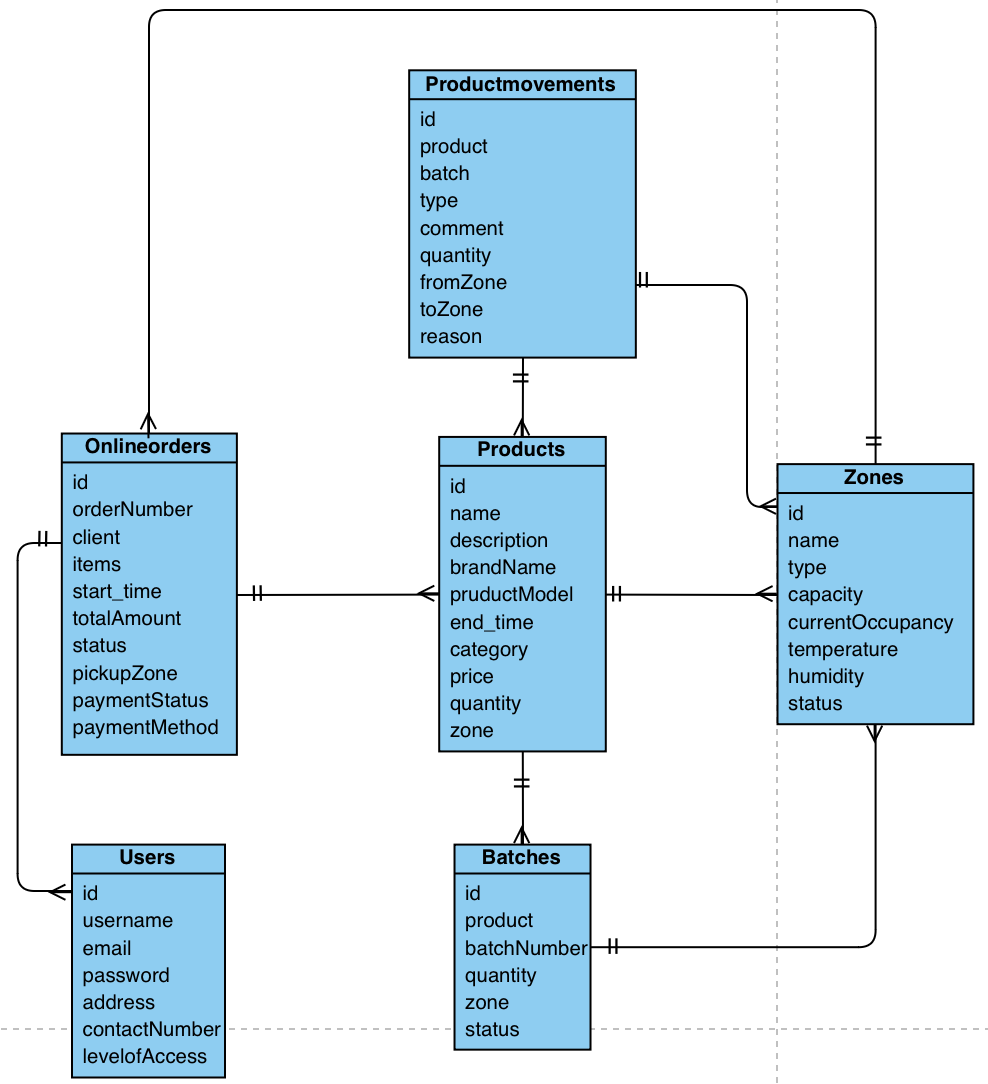


Рисунок 4.1 – Физическая схема БД

Текстовое описание сущностей базы данных представлено в виде таблицы 4.1.

Таблица 4.1 – Описание сущностей БД

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Назначение |
| *Users* (Пользователи) | | |
| *Id* | *INT (PK)* | Уникальный идентификатор пользователя |
| *Username* | *STRING* | Имя пользователя (уникальное) |
| *PasswordHash* | *STRING* | Хэшированный пароль |
| *email* | *STRING* | Электронная почта |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Назначение |
| *address* | *STRING* | Адрес |
| *contactNumber* | *STRING* | Номер телефона |
| *levelofAccess* | *STRING* | Роль |
| *Products* (Продукты) | | |
| *Id* | *INT (PK)* | Уникальный идентификатор товара |
| *Name* | *STRING* | Название товара |
| *Description* | *STRING* | Описание товара |
| *brandName* | *STRING* | Бренд |
| *productsModel* | *STRING* | Модель |
| *End-time* | *STRING* | Срок годности |
| *Category* | *STRING* | Категория |
| *Price* | *NUMBER* | Цена |
| *Quantity* | *NUMBER* | Количество |
| *zone* | *INT (FK)* | *FK* зона |
| *Zones* (Зоны) | | |
| *Id* | *INT (PK)* | Уникальный идентификатор зоны |
| *Name* | *STRING* | Название зоны |
| *Type* | *STRING* | Тип зоны (склад, продажа) |
| *Capacity* | *NUMBER* | Вместимость |
| *currentOccupancy* | *NUMBER* | Текущая занятость |
| *Temperature* | *NUMBER* | Температура |
| *Humidity* | *NUMBER* | Влажность |
| *status* | *STRING* | Статус (активна, неактивна) |
| *Productmovements* (Перемещения) | | |
| *Id* | *INT (PK)* | Уникальный идентификатор перемещения |
| *Product* | *INT (FK)* | *FK* Товар |
| *Batch* |  | Поставка |
| *Type* | *STRING* | Тип |
| *comment* | *STRING* | Комментарий |
| *Quantity* | *NUMBER* | Количество |
| *fromZone* | *INT (FK)* | *FK зона* |
| *toZone* | *INT (FK)* | *FK зона* |
| *Reason* | *STRING* | Причина |
| *Batches* | | |
| *Id* | *INT (PK)* | Уникальный идентификатор поставки |
| *Product* | *INT (FK)* | *FK товар* |
| *batchNumber* | *STRING* | Номер поставки |
| *Quantity* | *NUMBER* | Количество |
| *Zone* | *INT (FK)* | *FK зона* |
| *Status* | *STRING* | Статус |
| *Onlineorders* (Заказы) | | |
| *Id* | *INT (PK)* | Уникальный идентификатор заказа |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Назначение |
| *orderNumber* | *STRING* | Номер заказа |
| *Client* | *INT (FK)* | *FK клиент* |
| *Items* | *ARRAY* | Товары |
| *Start\_time* | *DATE* | Время заказа |
| *totalAmount* | *NUMBER* | Общая сумма заказа |
| *Status* | *STRING* | Статус заказа |
| *pickupZone* | *DATETIME* | *FK* Зона выдачи |
| *paymentStatus* | *STRING* | Статус оплаты |
| *paymentMethod* | *STRING* | Метод оплаты |

*MongoDB* – документоориентированная система управления базами данных, не требующая описания схемы таблиц. Считается одним из классических примеров *NoSQL*-систем, использует *JSON*-подобные документы и схему базы данных.

# **5 ПРЕДСТАВИТЬ ДЕТАЛИ РЕАЛИЗАЦИИ ПС ЧЕРЕЗ UML-ДИАГРАММЫ**

## **5.1 Описание статических аспектов программных объектов.**

Основными сущностями (моделями) являются *Users* (пользователь), *Products* (товары), *Zones* (зоны), *ProductMovements* (перемещения товаров) и *OnlineOrder* (заказы), *Batches* (Поставки).

*Users* хранит информацию о пользователях системы, включая имя пользователя, хэш пароля и уровень доступа (сотрудник или клиент).

*Products* описывает товары, включая их наименование, описание и конечную цену.

*ProductMovements* сохраняет перемещения товаров между зонами.

*OnlineOrder* хранит данные о заказах клиентов.

*Zones* хранит данные о коммерческих зонах, специфике, их статусе.

*Batches* хранит информацию о поставках товаров.

Для физического представления системы была построена диаграмма компонентов. Данная диаграмма позволяет показать архитектуру системы в целом, а также зависимость между программными компонентами. Диаграмма компонентов представлена на рисунке 5.2.

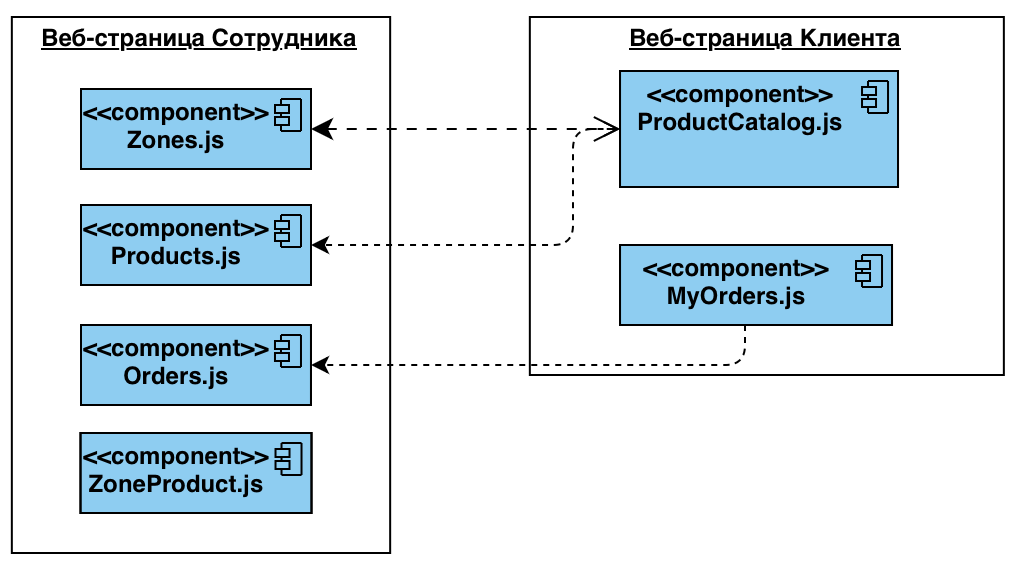


Рисунок 5.2 – Диаграмма компонентов

Основные графические элементы данной диаграммы – это компоненты зоны, товары, заказы со стороны сотрудника и каталог товаров со стороны клиента, а также зависимости между ними.

## **5.2 Описание динамических аспектов поведения программных объектов**

На диаграмме деятельности, представленной на рисунке 5.4, отображён процесс взаимодействия сотрудника с системой при выполнении варианта использования «Переместить товары между зонами». Диаграмма отражает последовательность шагов: от выбора товара до отображения всех товаров в целевой зоне. Этот процесс иллюстрирует логику бизнес-функциональности системы и показывает, как система обеспечивает перемещение товаров между коммерческими зонами торговой компании.

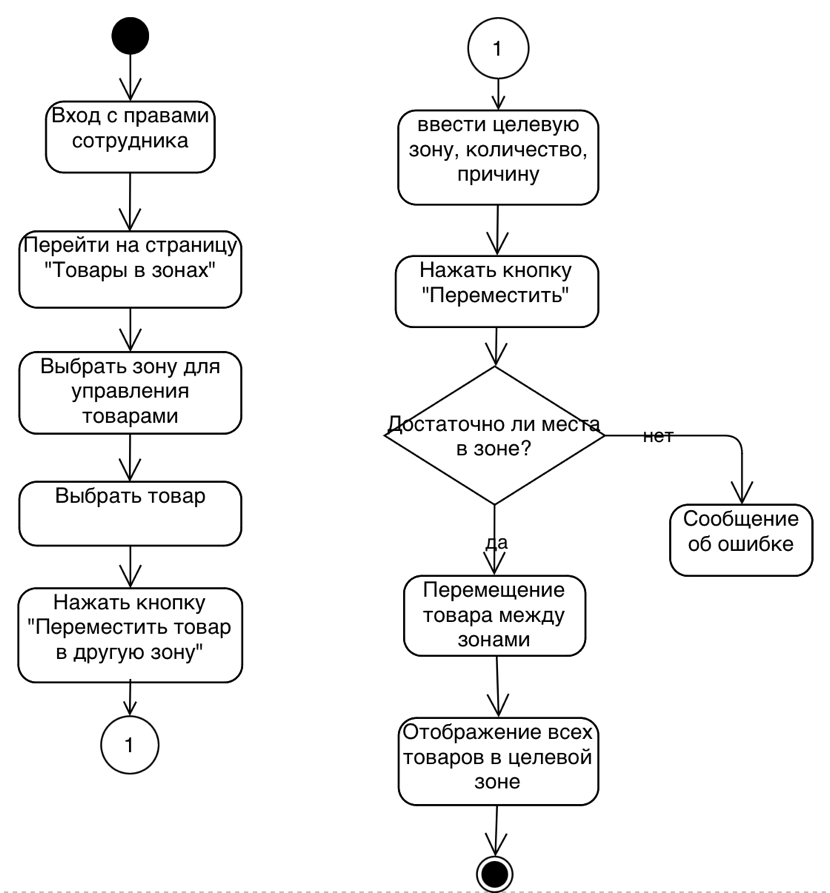


Рисунок 5.4 – Диаграмма деятельности варианта использования «Переместить товары между зонами»

На диаграмме последовательности (рисунок 5.5) для варианта использования «Переместить товары между зонами» был разработан следующий алгоритм. Пользователь переходит на страницу управление товарами. Фронтенд отображает интерфейс страницы. Сотрудник выбирает товар, зону, количество и нажимает на кнопку, тем самым инициируя отправку запроса *POST* на сервер. Сервер передает данные в базу данных, где происходит проверка на наличие доступного места в зоне и сохранение данных о перемещении в таблицу. Успешный ответ отправляется на сервер, фронтенд отображает клиенту итоговую страницу с товарами в целевой зоне.

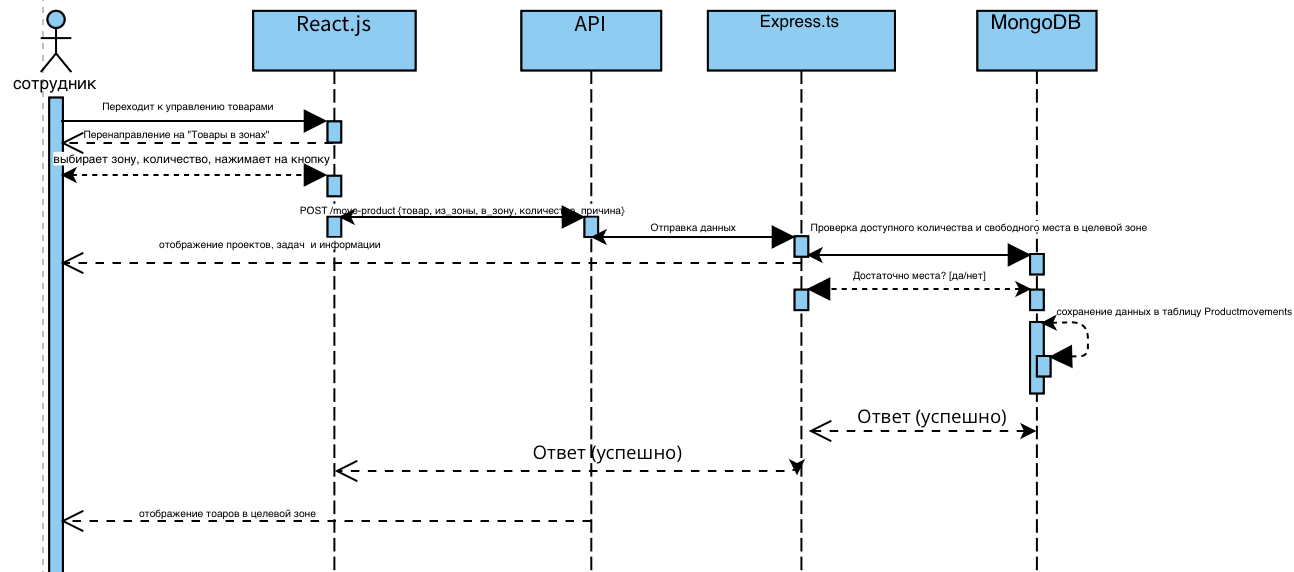


Рисунок 5.5 – Диаграмма последовательности варианта использования «Переместить товары между зонами»

Диаграмма состояния для товара в системе. Сотрудник заносит новый продукт в систему, указывает его зону, перемещает, меняет статус, а система сохраняет и заносит в базу данных. Диаграмма представлена на рисунке 5.6.

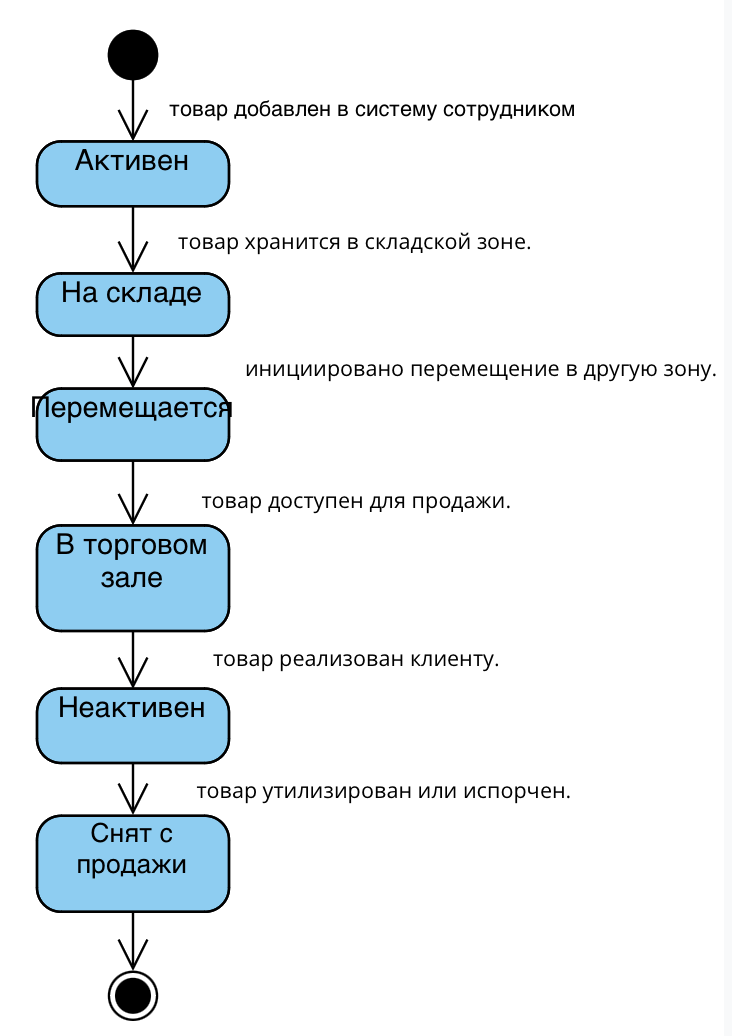


Рисунок 5.6 – Диаграмма состояния товара

Каждое состояние отображает, где находится товар и в каком статусе.

# **6 ДОКУМЕНТАЦИЯ К ПС С OPEN API**

**1 Реализация серверной части программной системы**

Серверная часть программной системы была реализована с использованием платформы *Node.js* и фреймворка *Express.ts*, с применением языка *TypeScript* для обеспечения строгой типизации и повышения надёжности кода. Архитектура проекта построена по принципу разделения ответственности, где логика распределена между слоями маршрутов (контроллеров), моделей, промежуточных обработчиков и сервисов взаимодействия с базой данных. Обмен данными осуществляется через *REST API*.

Реализована базовая система авторизации на основе *JWT*-токенов. При регистрации и входе пользователя пароль хешируется с использованием библиотеки *bcryptjs*, после чего создаётся токен, используемый для доступа к защищённым маршрутам. Маршруты защищаются через *middleware* *authMiddleware*, а также дополнительно проверяются роли пользователей с помощью *checkRole*.

Основные реализованные контроллеры (маршруты):

1 *UserRoutes* – регистрация, аутентификация, получение и обновление профиля пользователя.

2 *ProductRoutes* – *CRUD*-операции для управления товарами (создание, редактирование, удаление, фильтрация).

3 *ZoneRoutes* – управление коммерческими зонами (торговыми и складскими).

4 *ProductMovementsRoutes* – перемещение товаров между зонами, проверка остатка и вместимости.

5 *OrderRoutes* – оформление и просмотр онлайн-заказов пользователями.

Ключевые особенности реализации:

1 Использование *middleware* для обработки аутентификации и авторизации.

2 Хранение данных пользователей, товаров, зон и заказов в коллекциях *MongoDB.*

3 Поддержка разделения прав доступа по ролям: сотрудник, клиент.

4 Обработка ошибок *API* с возвратом статус-кодов и сообщений.

5 Адаптация логики под ролевую модель и реальные бизнес-процессы перемещения товаров.

Хранение и управление данными реализовано с использованием *MongoDB*, к которой осуществляется доступ через библиотеку Mongoose (*ODM*). Все *API*-запросы работают с форматами *JSON* и используют авторизацию по токену.

**2 Документация к *API* с использованием *OpenAPI* (*Swagger*)**

В рамках реализации серверной части программной системы была разработана документация к *REST API* в формате *OpenAPI* 3.0 (также известном как *Swagger*). Это позволило структурировать описание всех эндпоинтов системы, включая их параметры, типы данных, возможные ответы и схемы безопасности.

Документация *API* необходима для:

– стандартизации интерфейсов между фронтендом и бэкендом;

– облегчения тестирования и отладки;

– предоставления разработчикам и тестировщикам наглядного описания доступных операций;

– автоматической генерации запросов в *Swagger UI*.

В качестве инструмента использовалась библиотека *swagger-ui-express* для отображения интерактивной документации на стороне сервера и ручной файл спецификации *swagger.ts*, содержащий описание *API* в виде *JavaScript*-объекта.

После запуска *API*-документация доступна по адресу: [*http://localhost:8081/api-docs*](http://localhost:8081/api-docs)*.*

Для защиты ресурсов *API* применяется схема аутентификации *Bearer Token* (*JWT*). При успешном входе в систему пользователь получает токен, который необходимо передавать в заголовке каждого запроса:

*Authorization: Bearer* <токен>

Пример схемы модели (*Product*)

Product:

type: object

properties:

name:

type: string

description:

type: string

brandName:

type: string

productModel:

type: string

category:

type: string

price:

type: number

quantity:

type: integer

zone:

type: string

*Swagger UI* предоставляет:

– просмотр и описание всех методов *API*;

– ввод параметров запроса в веб-интерфейсе;

– тестирование запросов прямо из браузера;

– отображение схем данных и возможных ошибок.

Примеры доступных эндпоинтов отображены на рисунке 6.1.

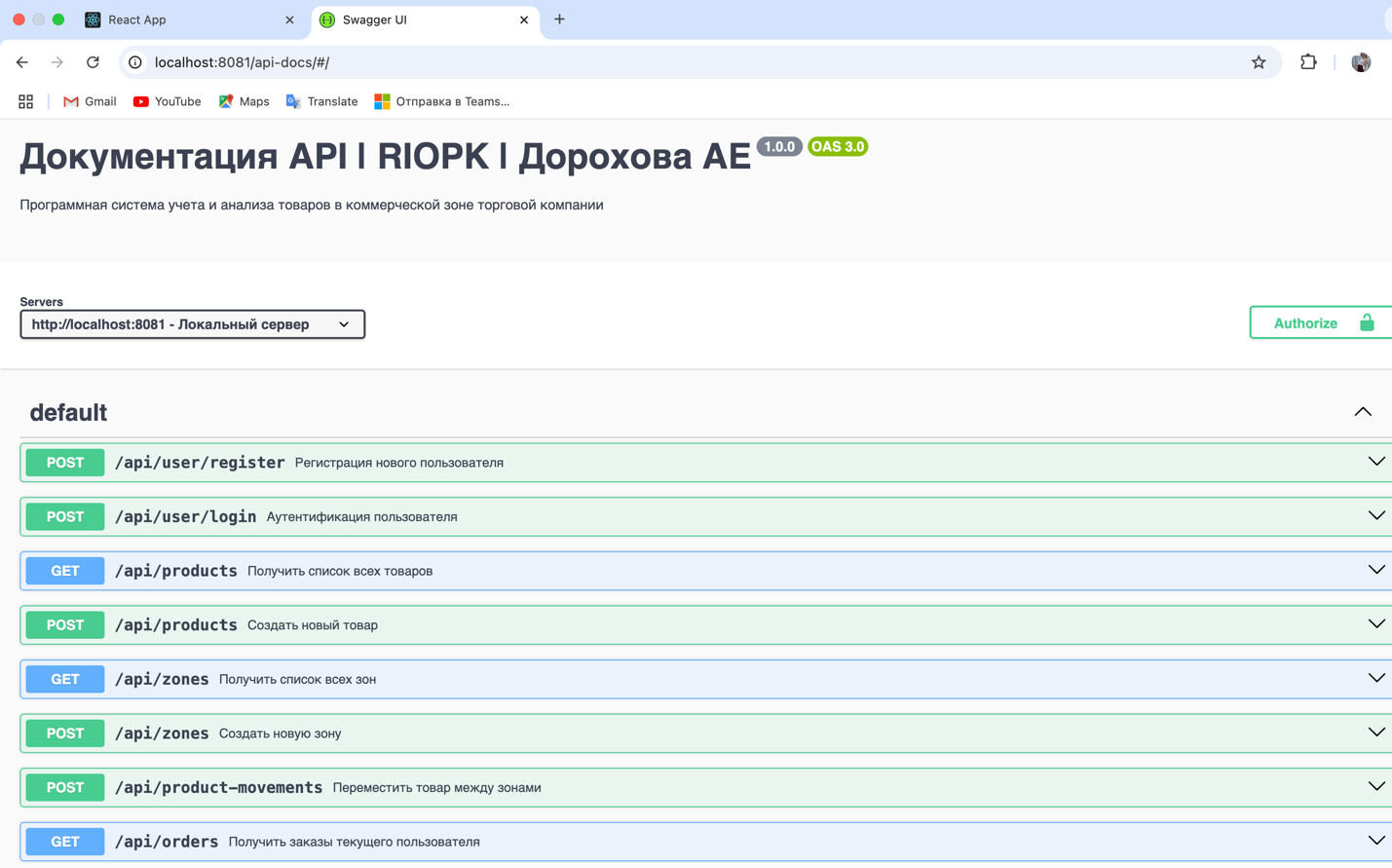


Рисунок 6.1 – Диаграмма состояния товара

Документация *API*, реализованная с помощью *OpenAPI*, значительно повышает прозрачность и удобство взаимодействия с серверной частью системы, сокращает время интеграции и облегчает сопровождение кода.

**3 Метрики качества кода**

Для оценки качества исходного кода программной системы были использованы следующие метрики, отражающие структурные, логические и стилевые характеристики кода. Эти метрики помогают определить уровень читаемости, сопровождённости и надёжности ПО.

1 Длина функций (*Function Length*).

Описание: количество строк кода в каждой функции.

Критерий качества: не более 40 строк.

Результат: большинство функций содержат от 5 до 30 строк, соблюдая принцип единичной ответственности.

2 Дублирование кода (*Code Duplication*).

Описание: наличие повторяющихся фрагментов кода.

Критерий качества: менее 10% дублирования по проекту.

Результат: дублирование сведено к минимуму благодаря использованию переиспользуемых компонентов, *middleware* и схем *Mongoose*.

3 Сложность кода (*Cyclomatic Complexity*).

Описание: число логических путей через код (*if, else, switch* и т. д.).

Критерий качества: до 10 – считается нормальной.

Результат: средняя цикломатическая сложность функций – 4-7, что говорит о разумной логике и отсутствии перегруженности.

4 Покрытие тестами (*Test Coverage*).

Описание: процент строк кода, покрытых автоматическими тестами.

Критерий качества: от 50% для пользовательской логики.

Результат: покрытие интерфейса (*React*-компонентов) достигает ~70% (по данным *Jest*), что обеспечивает проверку основных сценариев взаимодействия пользователя с системой.

5 Следование стилю (*Code Style Consistency*).

Описание: соответствие кода единому стилевому гайдлайну.

Критерий качества: автоматическая проверка средствами линтинга (например, *ESLint*).

Результат: код форматируется при помощи *Prettier*, а *ESLint* помогает соблюдать единый стиль.

6 Наличие документации и типизации.

Описание: использование *JSDoc*/*TypeDoc* и *TypeScript* для самоописания кода.

Результат: все модели, запросы и функции типизированы; в сложных участках добавлены комментарии.

**4 Оценка качества кода ПС**

Проведённая оценка качества кода показала, что программная система отвечает основным требованиям к промышленной разработке: код читаем, структурирован, сопровождаем и хорошо покрыт тестами. Это создаёт предпосылки для надёжной эксплуатации и расширения системы в будущем.

Показатели говорят о высоком качестве кода, модульной структуре и низкой связанности.

В процессе выполнения проекта была успешно реализована серверная часть ПС, покрытая *unit*- и интеграционными тестами. Использование *OpenAPI* упростило документирование и тестирование *API*. Благодаря применению метрик анализа кода, можно утверждать, что проект имеет хорошее качество архитектуры и легко поддаётся сопровождению. Возможности масштабирования, добавления новых фич и адаптации к требованиям заказчика были заложены изначально.

# **7 РЕАЛИЗАЦИЯ СИСТЕМЫ АУТЕНТИФИКАЦИИ И АВТОРИЗАЦИИ ПОЛЬЗОВАТЕЛЕЙ ПС И МЕХАНИЗМОВ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ ДАННЫХ**

В программной системе реализована полноценная система аутентификации, авторизации и защиты пользовательских данных, построенная на основе *JWT (JSON Web Token), bcrypt и middleware-*механизмов *Express.js*. Используемый подход соответствует современным требованиям к безопасности веб-приложений.

Аутентификация реализована с использованием логина и пароля, введённых пользователем. Процесс включает:

– проверку наличия пользователя в базе данных по имени пользователя;

– сравнение введённого пароля с захешированным с использованием bcrypt;

– генерацию *JWT*-токена с использованием секретного ключа, установленного в .env (*JWT\_SECRET*).

Пример генерации токена:

const generateToken = (id: string): string => {

return jwt.sign({ id }, process.env.JWT\_SECRET as string, {

expiresIn: '30d'

});

};

*JWT* передаётся клиенту и сохраняется в *localStorage* или *cookie*, после чего используется в заголовке *Authorization* всех защищённых запросов.

Авторизация и разграничение доступа. Для защиты маршрутов и данных реализованы *middleware*-функции:

1 АuthMiddleware – проверяет наличие и корректность *JWT*-токена, извлекает *ID* пользователя и загружает его из базы данных. Если пользователь не найден или токен недействителен – запрос отклоняется.

2 *СheckRole*([‘*client*, ‘*employee’*]) – проверяет, имеет ли текущий пользователь необходимый уровень доступа к защищённому ресурсу. Поддерживаются уровни: *employee, client*.

Пример использования:

app.get('/api/admin-only', authMiddleware, checkRole(['admin']), handler);

Защита паролей. Все пароли хранятся в захешированном виде с помощью библиотеки *bcryptjs*;

– при регистрации и смене пароля используется соление (*salt*) и повторная генерация хеша;

– ни в одном ответе сервера пароли не передаются – они исключаются с помощью .*select*('-*password'*).

Профиль пользователя и обновление данных. Пользователь может:

– Получить свой профиль (*GET /api/users/profile*);

– Обновить персональные данные и пароль;

– Сменить пароль с обязательной проверкой текущего.

Безопасность данных. Все критичные действия (регистрация, логин, управление данными) защищены *JWT*-механизмами; введены ограничения доступа по ролям; выполняется проверка активности пользователя (*isActive*);

Исключены SQL-инъекции благодаря использованию *Mongoose* (*ODM*);

При возникновении ошибок или подозрений система возвращает соответствующие статусы и сообщения.

Пример защищённого запроса с токеном

*GET /api/orders*

*Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR...*

Пример кода запроса аутентификации:

const response = await axios.post('/api/users/login', {

username: 'admin',

password: 'admin123'

});

localStorage.setItem('token', response.data.token);

Система аутентификации и авторизации построена на надежных криптографических методах (*bcrypt*, *JWT*), обеспечивает гибкое разграничение прав доступа, а также защиту от несанкционированного доступа и утечки данных. Реализованные механизмы позволяют безопасно обрабатывать регистрацию, вход, редактирование профиля и ограничивать доступ в зависимости от уровня прав пользователя.

# **8 UNIT- И ИНТЕГРАЦИОННЫЕ ТЕСТЫ**

*Unit*-тестирование (модульное тестирование) – это методика, при которой тестируется отдельная, изолированная часть программы (обычно – один метод или класс).

Модуль автоматических *unit*-тестов для компонента *Products* в приложении на *React*, написанный с использованием библиотеки *@testing-library/react* и *Jest*. Он проверяет, как компонент отображает, фильтрует и обрабатывает данные, а также как работает логика ролей и ошибок. Технологии описаны в таблице 8.1.

Таблица 8.1 – Используемые технологии

|  |  |
| --- | --- |
| Технология | Назначение |
| *@testing-library/react* | Рендерит компонент и имитирует действия пользователя |
| *jest.mock(...)* | Подменяет реальные модули (*API*, маршруты, *localStorage*) |
| *BrowserRouter* | Обеспечивает маршрутизацию внутри теста (для *React Router*) |
| *waitFor* | Ожидает асинхронные действия, например, загрузку данных |
| *fireEvent* | Симулирует ввод текста, клики и т.п |

Реализованный тест для товаров имитирует ввод текста "*Model X*" в строку поиска и проверяет, что отображается только нужный товар (*Test Product* 1).

test('filters products by search term', async () => {

render(

<BrowserRouter>

<Products />

</BrowserRouter>

);

await waitFor(() => {

expect(screen.queryByRole('progressbar')).not.toBeInTheDocument();

});

// Type in search box

const searchInput = screen.getByPlaceholderText('Поиск товаров...') ||

screen.getByLabelText('Поиск') ||

screen.getByRole('textbox');

fireEvent.change(searchInput, { target: { value: 'Model X' } });

// Check if only matching product is displayed

expect(screen.getByText('Test Product 1')).toBeInTheDocument();

expect(screen.queryByText('Test Product 2')).not.toBeInTheDocument();

expect(screen.queryByText('Test Product 3')).not.toBeInTheDocument();

});

Данный тест выбирает фильтр категории *Electronics* и проверяет, что отображаются только товары из этой категории (1 и 3).

test('filters products by category', async () => {

render(

<BrowserRouter>

<Products />

</BrowserRouter>

);

await waitFor(() => {

expect(screen.queryByRole('progressbar')).not.toBeInTheDocument();

});

// Select category filter

const categorySelect = screen.getByLabelText('Категория') ||

screen.getAllByRole('combobox')[0];

fireEvent.change(categorySelect, { target: { value: 'Electronics' } });

// Check if only Electronics products are displayed

expect(screen.getByText('Test Product 1')).toBeInTheDocument();

expect(screen.queryByText('Test Product 2')).not.toBeInTheDocument();

expect(screen.getByText('Test Product 3')).toBeInTheDocument();

});

Тест-кейсы для проверки уровня базовых пользовательских требований приведены в таблице 8.2.

Таблица 8.2 – Тест-кейсы для проверки уровня базовых пользовательских   
 требований

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Заглавие тест-кейса | Шаги тест-кейса | Ожидаемый результат |
| UC1 | Просмотр каталога товаров | 1. Перейти в раздел «Каталог»  2. Убедиться, что отображается список добавленных товаров | Каталог товаров корректно отображается |
| UC2 | Добавление товаров | 1. Нажать «Добавить товар»  2. Заполнить обязательные поля  3. Нажать «Сохранить» | Товар успешно добавлен, появляется в списке |
| UC3 | Редактирование товаров | 1. Перейти к списку товаров  2. Нажать «Редактировать» у нужного товара  3. Внести изменения  4. Сохранить | Изменения сохранены, данные обновлены |
| UC4 | Удаление товара | 1. Перейти к списку товаров 2. Нажать «Удалить» у нужного товара 3. Подтвердить действие | Товар удалён из системы, список обновлён |
| UC5 | Добавить новую зону | 1. Перейти в раздел «Зоны» 2. Просмотреть список существующих политик | Зона успешно добавлена, появляется в списке |
| UC6 | Переместить товар между зонами | 1. Нажать иконку перемещения на товаре 2. Заполнить поля (зона, количество, причина.) 3. Нажать «Переместить» | Товар перемещен в выбранную зону |
| UC7 | Добавить товар в корзину | 1. Перейти на страницу каталога товаров 2. Выбрать товар  3. Нажать на кнопку «Добавить в корзину» | Товар отображен в корзине, итоговая стоимость посчитана |
| UC8 | Оформить заказ | 1. Нажать на кнопку «Оформить заказ» 2. Заполнить поля 3. Нажать «Оформить заказ» | Заказ отображен на странице Мои заказы |
| UC9 | Просмотр статуса заказа | 1. Перейти в раздел «Мои заказы» 2. Отследить статус заказа | Отображены статусы доставки заказа |

Покрытие интерфейса (*React*-компонентов) достигает ~70%, что обеспечивает проверку основных сценариев взаимодействия пользователя с системой.

# **9 ОПИСАНИЕ ПРОЦЕССА РАЗВЕРТЫВАНИЯ ПС**

Необходимые программы и компоненты. Для успешной установки и запуска веб-приложения *Food Delivery Master* необходимо наличие следующих компонентов:

– операционная система семейства *MacOS* или *Windows*;

– веб-сервер *Express.js*;

– зависимости (библиотеки) для серверной части приложения;

– клиентская часть *React.js*;

– зависимости (библиотеки);

– сервер базы данных *ModngoDB*.

Последовательность установки. Для установки разработанного программного средства необходимо выполнить следующие шаги:

1 Подготовка среды для запуска фронтенда: скачать последнюю версию редактора Visual Studio Code с официального сайта.

2 Установить и настроить *Node.js*.

3 Установить зависимости для серверной части.

4 Установить зависимости для клиентской части.

5 Установить и настроить сервер баз данных *MongoDB*, для удобства можно также использовать *MongoDB Compass*.

6 Установить и настроить разработанное ПС.

Состав дистрибутива. В поставляемый конечному пользователю дистрибутив входят следующие элементы:

– скрипт генерации пустой базы данных;

– ярлык, содержащий ссылку на сайт.

Проверка работоспособности. Для проверки работоспособности веб-приложения необходимо:

– убедиться в том, что сервис *MongoDB* запущен;

– убедиться в том, что сервер веб-приложения запущен;

– убедиться в том, что клиентская часть веб-приложения запущена;

– используя любой интернет браузер перейти на адрес *http://localhost:8081*, где 8081 порт, указанный в файле сервера, или воспользоваться ярлыком.

В случае успеха в окне браузера откроется главная страница веб-приложения.

7 Авторизация в системе: администратор создается автоматически с учётными данными:

Логин: *admin*

Пароль: *123456*

Обычный пользователь создается при самостоятельной регистрации через интерфейс фронтенда.

8 Запуск *Unit*-тестов: тесты написаны с использованием фреймворка *xUnit*. В открывшемся Обозревателе тестов выбрать нужные тесты или запустить все с помощью кнопки «Выполнить все тесты».

После выполнения тестов, напротив каждого теста появится зеленая галочка. В случае неудачного теста отобразится красный крестик с описанием ошибки.

# **10 РАЗРАБОТКА РУКОВОДСТВА ПОЛЬЗОВАТЕЛЯ**

Программное средство предназначено для автоматизации процессов управления товарами, коммерческими зонами и онлайн-заказами в торговой компании. Интерфейс разделён по ролям: сотрудник и клиент, каждая из которых имеет доступ только к соответствующим функциям.

Роль: Сотрудник.

1 Авторизация.

После запуска приложения сотрудник проходит авторизацию, введя логин и пароль (рисунок 10.1). В случае некорректного ввода данных система отобразит сообщение об ошибке.

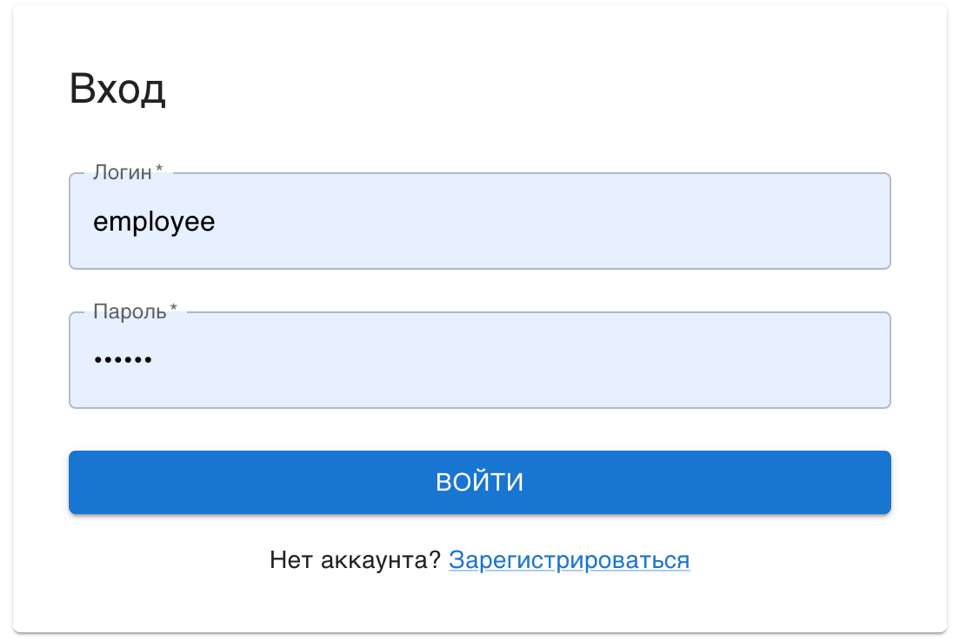


Рисунок 10.1 – Форма ввода логина и пароля

2 Дашборд.

После успешного входа сотруднику доступен основной дашборд с возможностью перехода к остальным разделам.

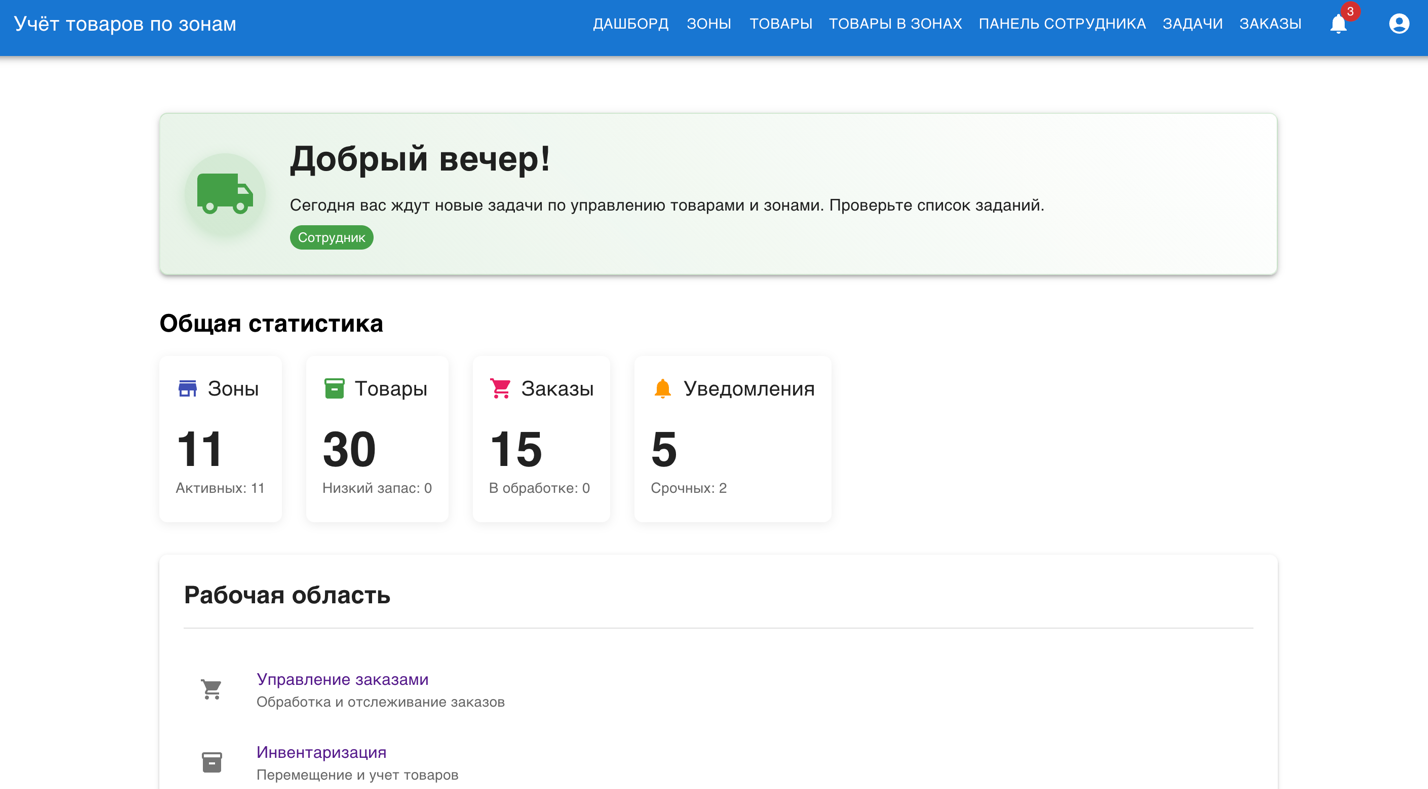


Рисунок 10.2 – Дашборд сотрудника

Страница управление коммерческими зонами (рисунок 10.3) – добавление, редактирование и удаление складов и торговых залов, изменение их параметров (температура, вместимость и т.п.).

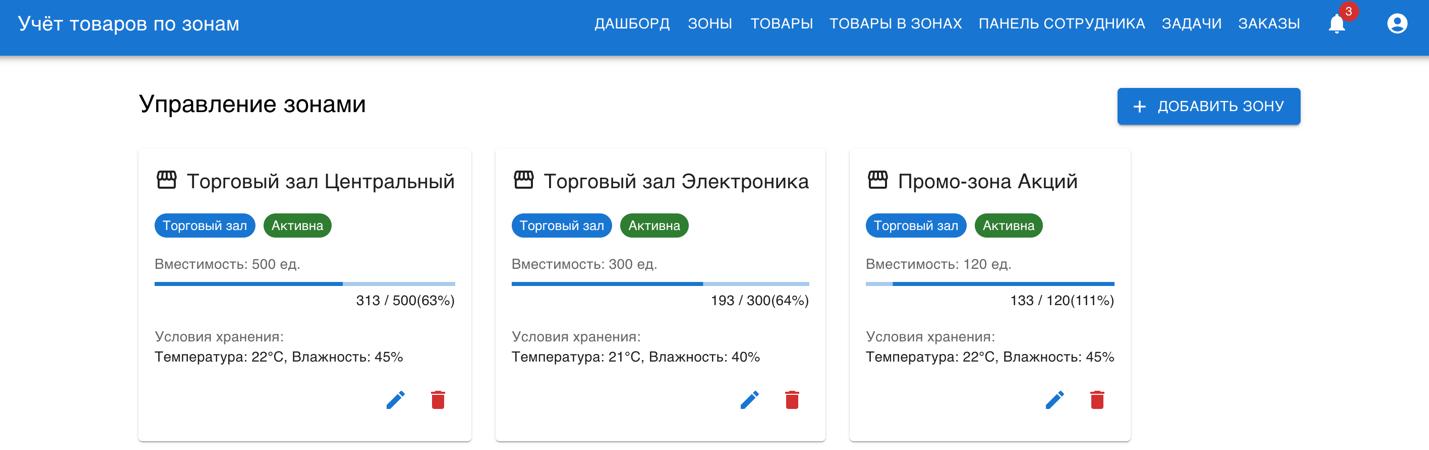


Рисунок 10.3 – Cтраница Зоны

Управление товарами – обновление описаний, количества, зоны хранения и характеристик товаров, представлено на рисунке 10.4.

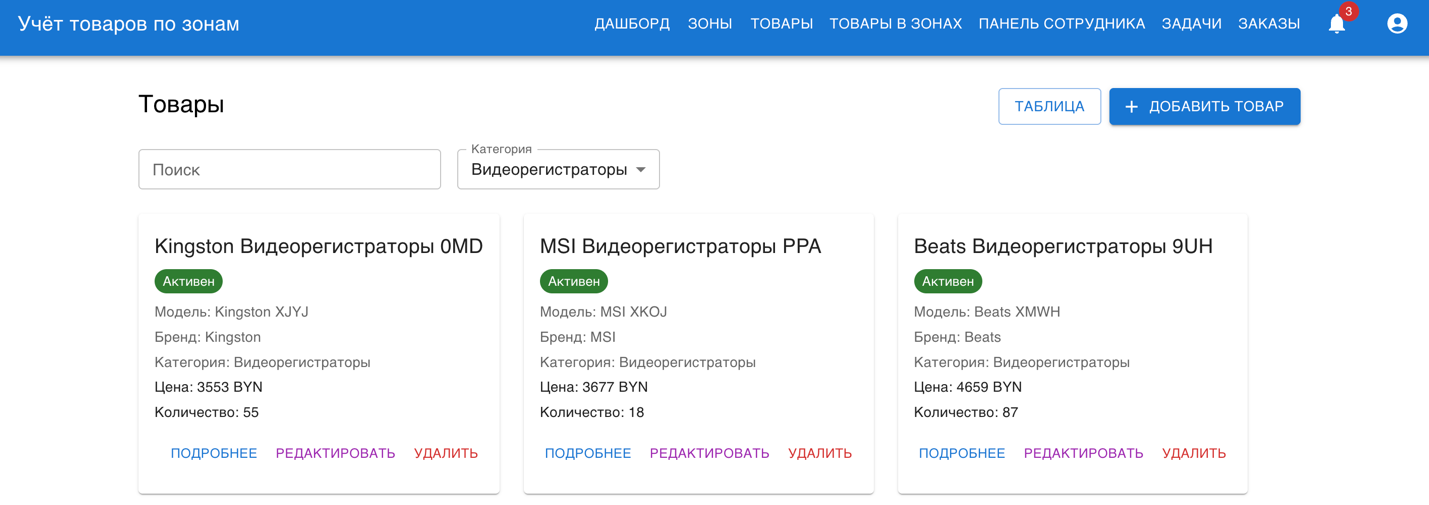


Рисунок 10.4 – Cтраница Товары

Перемещение товаров между зонами – выбор партии, количества и зон перемещения. Действия автоматически регистрируются в системе, отображено на рисунке 10.5. Здесь сотрудник может переместить товар и проверить его нахождение.

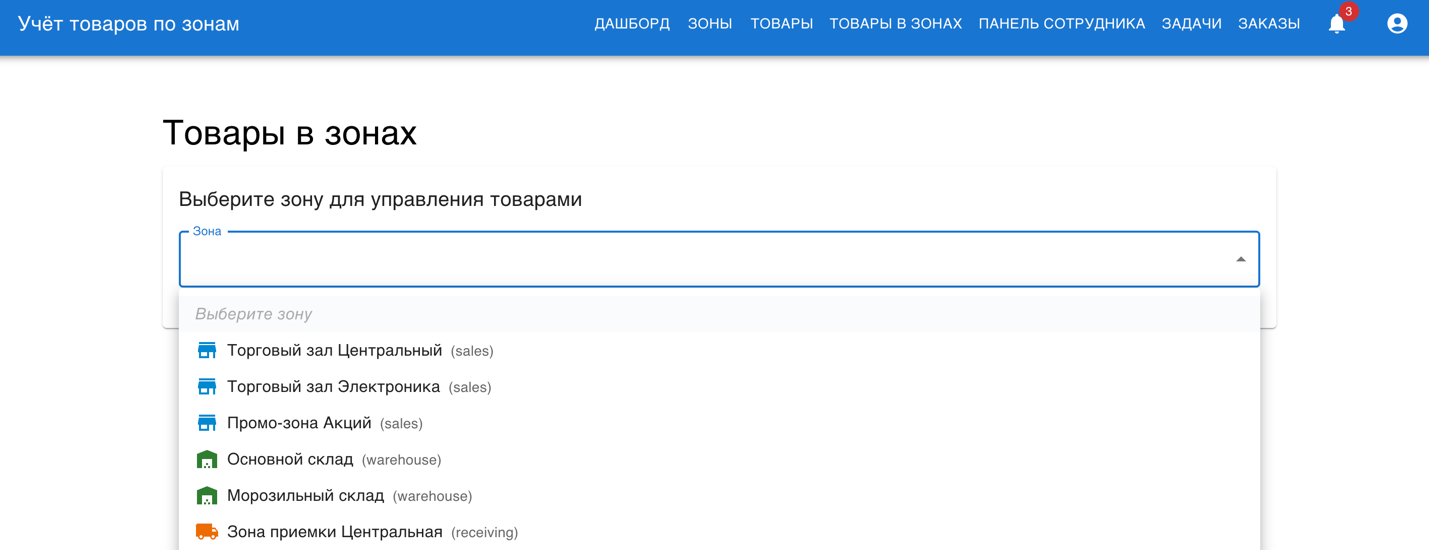


Рисунок 10.5 – Cтраница Товары в зонах

Управление заказами, представленное на рисунке 10.6, – просмотр и подтверждение онлайн-заказов, изменение их статуса.

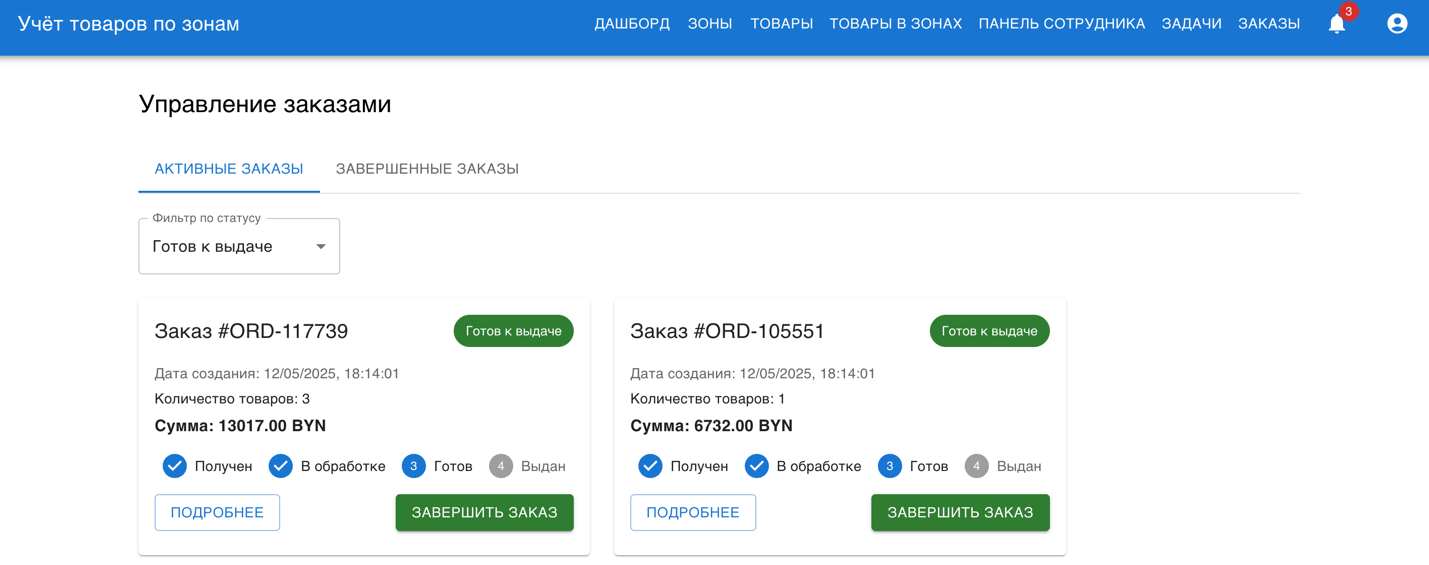


Рисунок 10.6 – Cтраница Заказы

5 Просмотр задач сотрудника – личный список задач, связанных с логистикой и управлением складом. Страница изображена на рисунке 10.7.

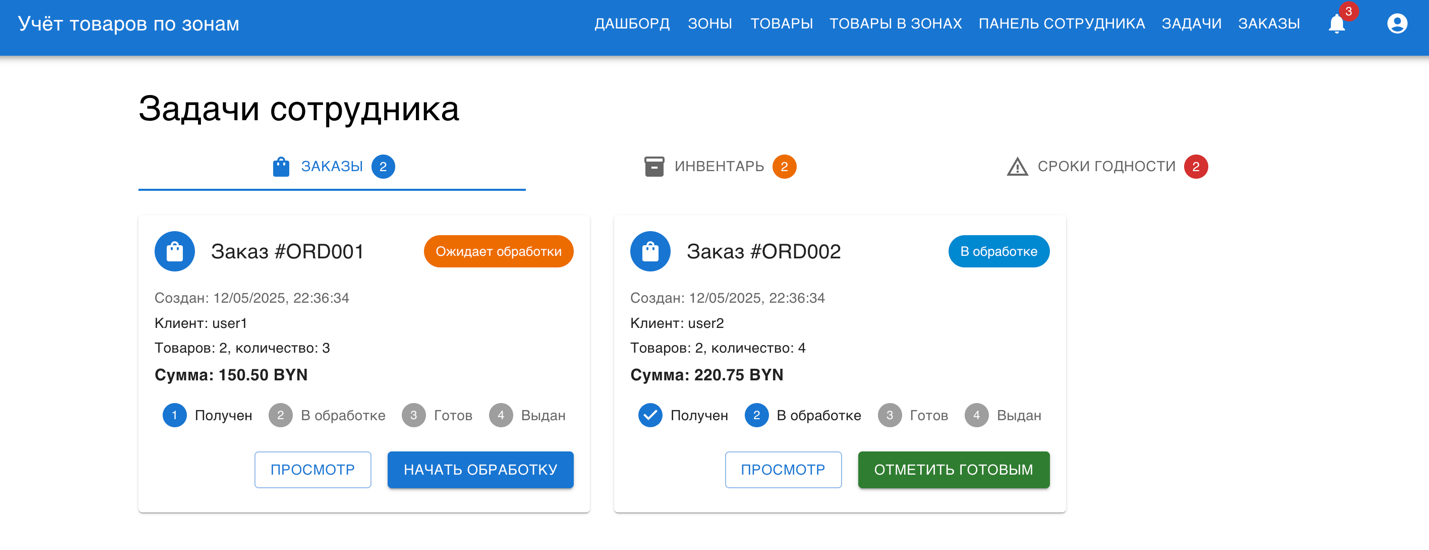


Рисунок 10.6 – Cтраница Задачи

Завершение работы.

По завершении работы сотрудник может выйти из системы с помощью кнопки «Выход».

Роль: Клиент.

Авторизация.

Клиент может пройти авторизацию или зарегистрироваться (рисунок 10.7). При неверных данных выводится уведомление об ошибке.

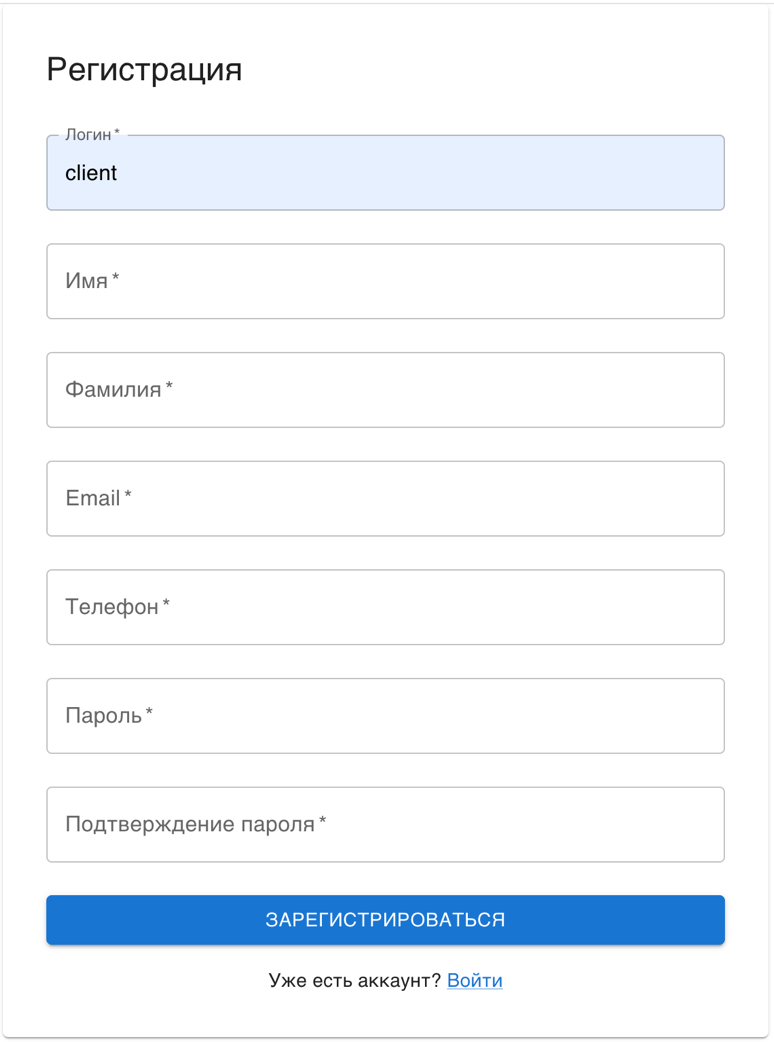


Рисунок 10.7 – Форма регистрации

После авторизации клиент попадает на страницу Дашборд, представленную на рисунке 10.8. С нее он может попасть на другие страницы приложения.

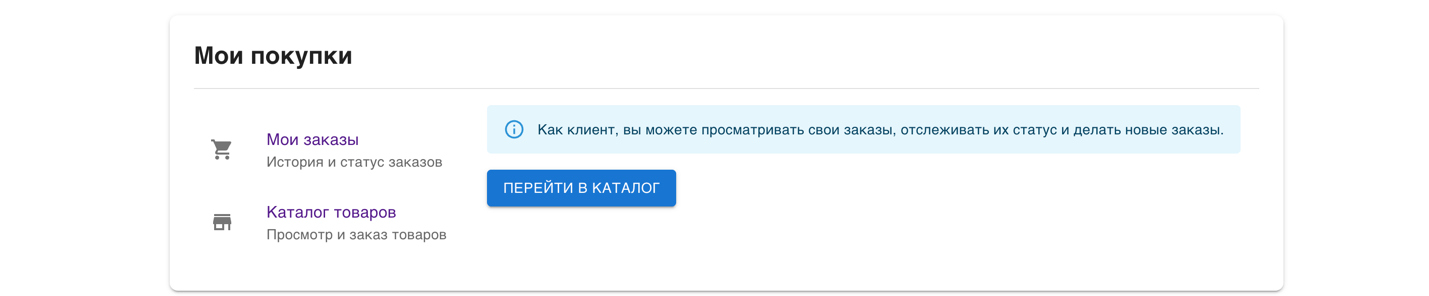
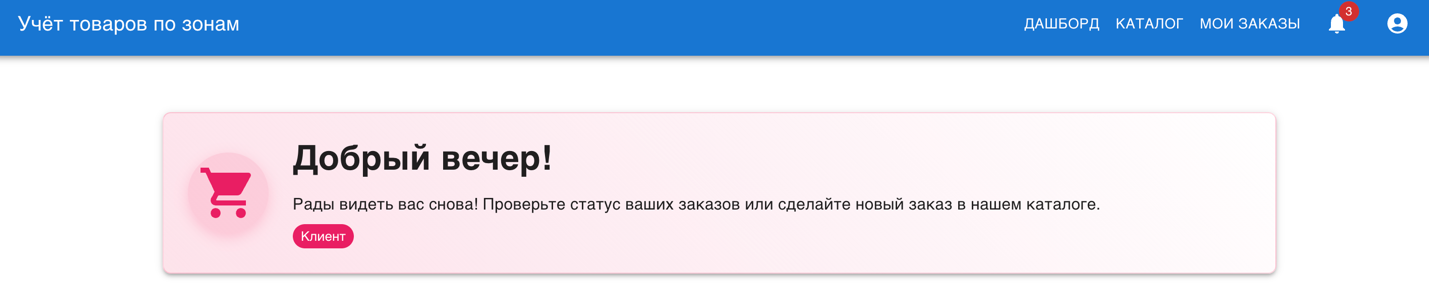


Рисунок 10.8 – Дашборд клиента

Просмотр каталога товаров – пользователь может листать каталог и ознакомиться с товарами. Тут он может выполнить поиск товаров – быстрый поиск по названию, категории или производителю.

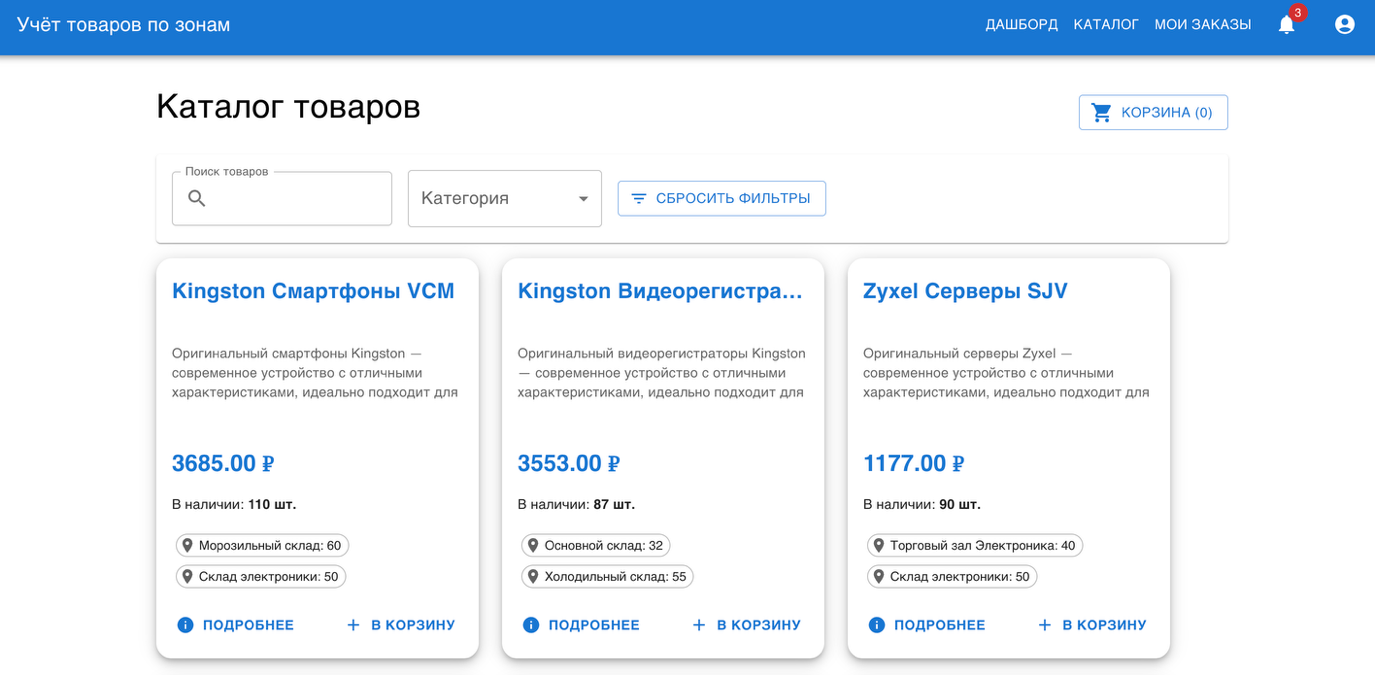


Рисунок 10.9 – Страница каталог товаров

Добавление товара в корзину – клиент выбирает товар и добавляет его в заказ. Выбор зоны получения – при оформлении заказа (рисунок 10.10) указывается ближайшая или желаемая зона выдачи товара (например, зона выдачи заказов №1). Оформление заказа – после добавления всех товаров и указания зоны, заказ подтверждается клиентом.

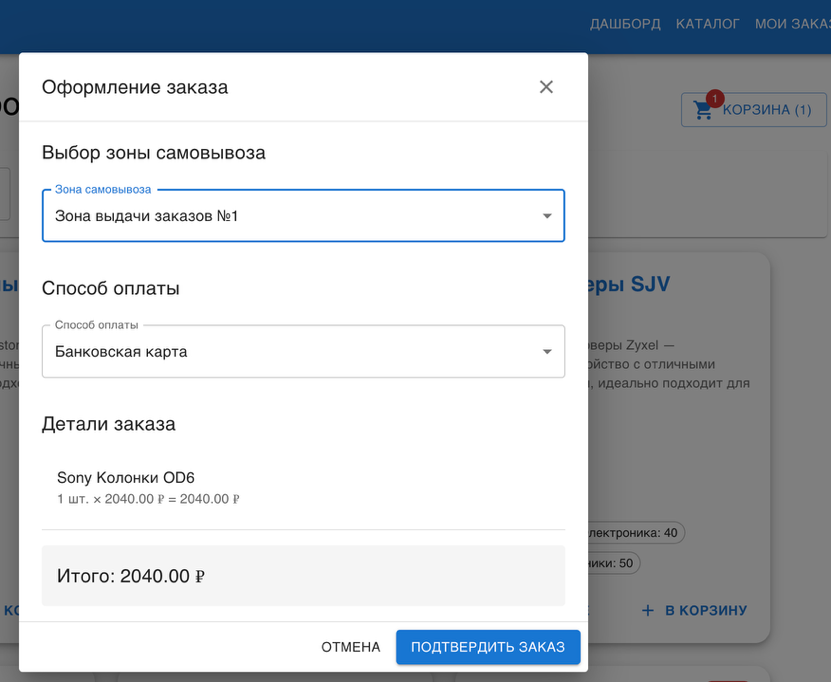


Рисунок 10.10 – Страница оформления заказа

Просмотр и отслеживание заказов – в разделе «Мои заказы» клиент может просмотреть текущий статус всех оформленных заявок, страница представлена на рисунке 10.11.

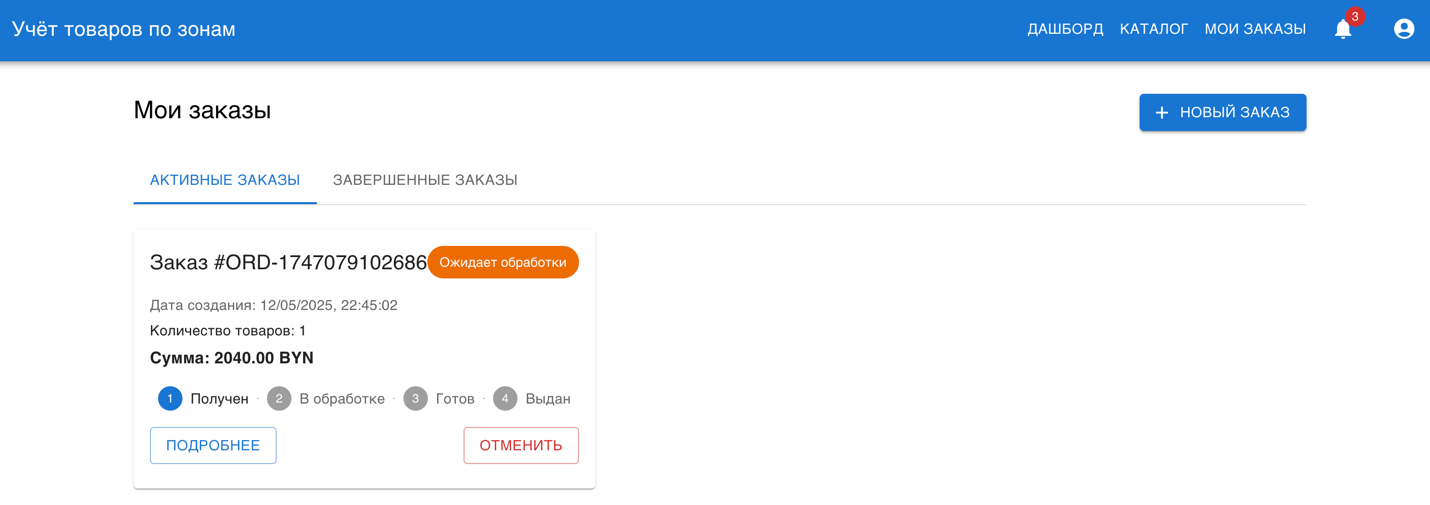


Рисунок 10.11 – Страница Мои заказы

Завершение работы.

Клиент может завершить сеанс, используя кнопку выхода на панели управления.

Навигация по интерфейсу.

Навигация выполнена в виде панели меню с иконками и подписями. Все действия доступны максимум в два клика. Интерфейс адаптирован под разные устройства.

Поддержка. В случае ошибок, сбоя при авторизации или других затруднений рекомендуется:

– проверить интернет-соединение;

– обновить страницу;

– обратиться к администратору системы.

В таблице 10.1 описаны функции и задачи веб-приложения.

Таблица 10.1 – Описание функций и задач

|  |  |  |
| --- | --- | --- |
| Функции | Задачи | Описание |
| Управление пользователями | Регистрация, авторизация | Обеспечивает доступ к системе, разграничение прав (сотрудник/клиент), аутентификация через логин и пароль |
| Управление товарами | Добавление, редактирование, удаление товаров | Сотрудник может создавать, изменять и удалять товары, указывая характеристики: наименование, номер партии, срок годности и т.д. |

Продолжение таблицы 10.1

|  |  |  |
| --- | --- | --- |
| Управление заказами | Обработка и изменение статуса заказа | Сотрудник может манипулировать заказами клиентов |
| Управление зонами | Выбор зоны выдачи, отслеживание количества товара в определенной зоне | Сотрудник и клиент могут отслеживать количество товара в зоне, выбирать зоны для выдачи заказа |
| Просмотр созданных заказов | Просмотр и отслеживание заказов | Отображает список всех ранее созданных заказов и их статусы |

В таблице 10.2 описаны операции обработки данных для задач.

Таблица 10.2 – Описание реализуемых операций

|  |  |
| --- | --- |
| Операция | Описание |
| Регистрация пользователя | Позволяет создать новую учетную запись с ролью «пользователь» |
| Авторизация | Проверяет логин и пароль, позволяет войти в систему |
| Добавление товара | Сотрудник создает новый товар с указанием всех необходимых характеристик |
| Редактирование товара | Обновление информации о товаре: цена, срок годности, номер партии и др. |
| Удаление товара | Полное удаление товара из базы данных |
| Добавление зоны | Сотрудник создает новую зону с указанием всех необходимых характеристик |
| Редактирование зоны | Обновление информации о зоне: вместимость, название, номер партии и др. |
| Удаление зоны | Полное удаление зоны из базы данных |
| Просмотр созданных заказов расчетов | Отображение списка всех ранее созданных заказов и их статусы |

В рамках проектирования и разработки программной системы были определены основные функции и задачи, направленные на обеспечение эффективной работы с товарами, зонами и заказами. Функциональность системы ориентирована на пользователей с различными ролями – от сотрудника до обычного пользователя, что позволяет гибко управлять доступом и предоставлять необходимый инструментарий в зависимости от уровня полномочий.

**Вывод**

В ходе выполнения проекта было разработано программное средство реализации решения планирования и анализа товаров в коммерческой зоне торговой компании. Цель проекта заключалась в создании удобного и функционального решения, позволяющего автоматизировать ключевые бизнес-процессы торговой компании.

На этапе анализа и проектирования была выбрана архитектура с разделением на фронтенд и бэкенд. В качестве технологий использовались *React.js* для клиентской части и *Express.ts* для серверной части. Реализована авторизация и аутентификация пользователей с разграничением прав доступа.

Функционал включает в себя: работу с товарами (добавление, редактирование, удаление), работу с зонами (добавление, редактирование, удаление), перемещение товаров между зонами, создание и управление заказами. Каждый из этих элементов прошел тестирование – как модульное (*unit*-тесты), так и интеграционное. В рамках тестирования была проведена оценка качества кода с использованием метрик, что позволило выявить и устранить потенциальные слабые места в логике приложения.

Документация к *API* создана с использованием спецификации *OpenAPI*, что облегчает взаимодействие сторонних клиентов с системой.

Также разработано подробное руководство пользователя с учётом разных ролей (сотрудник и клиент), где описаны доступные функции, интерфейсы и порядок действий. Это позволяет легко вводить в эксплуатацию новых сотрудников и снижает порог входа в систему.

В результате была создана полнофункциональная и расширяемая система, ориентированная на практическое применение в условиях реальной торговой компании. Проект продемонстрировал успешную реализацию поставленных целей и может служить основой для дальнейшего масштабирования или адаптации под нужды конкретной организации.