МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-40 05 01 «Программное обеспечение информационных технологий»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Компьютерные языки разметки»

Тема Веб-сайт «Веб-приложение склада»

**Исполнитель**

студент 1 курса 3 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гулешов Т.И

подпись, дата

**Руководитель**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Жиляк Н.А

должность, ученая степень, ученое звание подпись, дата

Допущен к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Жиляк Н.А

подпись дата инициалы и фамилия

Содержание

[Введение 3](#_Toc134778804)

[2. Проектирование веб-приложения 4](#_Toc134778805)

[2.1 Проектирование структуры базы данных 4](#_Toc134778806)

[2.2 Проектирование приложения сервера 11](#_Toc134778807)

[2.3 Проектирование приложения клиента 12](#_Toc134778808)

[2.5 Структурная схема веб-приложения 16](#_Toc134778809)

[2.6 Вывод по разделу 17](#_Toc134778810)

[3. Разработка веб-приложения 17](#_Toc134778811)

[3.1 Разработка приложения сервера 17](#_Toc134778812)

[3.2 Разработка приложения клиента 21](#_Toc134778813)

[3.3 Выводы по разделу 31](#_Toc134778814)

[4 Тестирование веб – приложения 31](#_Toc134778815)

[4.1 Тестирование десктопной версии 31](#_Toc134778816)

[4.2 Вывод по разделу 47](#_Toc134778817)

# Введение

Овощебаза будет иметь удобный и интуитивно понятный интерфейс, который позволит пользователям быстро находить нужные продукты. В приложении будут представлены различные овощи и фрукты, а также информация о поставщиках.

Для создания приложения мы будем использовать React JS, одну из самых популярных библиотек для создания пользовательских интерфейсов. Эта библиотека позволяет создавать масштабируемые и высокопроизводительные приложения, которые легко поддерживать и расширять.

Мы также будем использовать различные инструменты для разработки и тестирования приложения, такие как Git, Babel, Jest. Эти инструменты помогут нам создать высококачественное и надежное приложение, которое будет готово к развертыванию на сервере.

В итоге, Овощебаза на React JS будет удобным и функциональным приложением для всех любителей здорового питания. Оно поможет пользователям быстро находить нужные продукты у надежных поставщиков, сравнивать цены и оставлять отзывы о продукции. Мы уверены, что наше приложение станет незаменимым помощником для всех, кто следит за своим здоровьем и питается правильно.

2. Проектирование веб-приложения

Для разработки веб-приложения были выбраны следующие инструменты, а именно Json-Server*, React.js* и *Node.js*.

При разработке дипломного проекта весь процесс был разбит на три следующие составляющие:

база данных;

сервер;

клиент.

2.1 Проектирование структуры базы данных

В данном дипломного проекте была использована не реляционная, то есть *NoSQL*, база данных *Json-Server*. Хранящая в себе документы приближенные к формату *JSON*, позволяя более простым методом разработать схемы данных. Это все возможно за счет того, что данный формат наиболее приближен к объектам языка *JavaScript*. Реляционные БД с их строгими схемами предъявляют требования ко всем полям каждой из сохраняемой строки данных. Документоориентированные *NoSQL* хранилища более гибкие и эффективны в этом плане.

Для реализации базы данных веб-приложения было разработано семь таблиц и схем для них. Они покрывают необходимые требования, для реализации полноты функционала продукта.

Пример схемы таблицы vegitable представлен на листинге 2.1.

|  |
| --- |
| "vegitable": [  {  "name": "Огурец",  "weight": "0",  "image": "https://calorizator.ru/sites/default/files/imagecache/product\_512/product/cucumber-1.jpg",  "price": "0",  "id": 1  },  {  "name": "Бульба",  "weight": "0",  "image": "https://kakvarim.ru/wp-content/uploads/2020/01/kak-varit-kartoshku-post.jpg",  "price": "0",  "id": 2  },  {  "name": "Лук",  "weight": "0",  "image": "https://calorizator.ru/sites/default/files/imagecache/product\_512/product/bulb-onion.jpg",  "price": "0",  "id": 3  },  {  "name": "Помидор",  "weight": "0",  "image": "https://m.dom-eda.com/uploads/images/catalog/item/c6ebcf64ba/e87b941b85\_500.jpg",  "price": "0",  "id": 5  },  {  "name": "Свекла",  "weight": "0",  "image": "https://m-fermer.by/images/svekla.png",  "price": "0",  "id": 6  }  ], |

Листинг 2.1 – схема таблицы *vegitable*

Сама же таблицаvegitable, представляет собой таблицу с овощами, который можно посмотреть на сайте. Таблица состоит из полей:

*name* – Именование Товара

*weight* – Вес товара

*image –* Изображение товара

*price* – Цена товара

*id* – Идентификатор товара

Пример схемы таблицы *postsTovar* представлен ниже на листинге 2.2.

|  |
| --- |
| "postsTovar": [  {  "name": "Андрей",  "phone": "+375297579370",  "id": 1  },  {  "name": "Ванек",  "phone": "+3234253455",  "id": 4  }  ], |

Листинг 2.1 – схема таблицы *postsTovar*

Таблица*postsTovar* представляет собой таблицу с поставщиками, которые предоставляют наш товар. Данная таблица содержит следующие поля:

*name* – Имя поставщика

*phone* – Номер поставщика

*id* – Идентификатор поставщика

Пример схемы таблицы *changedTovar* представлен ниже на листинге 2.3.

|  |
| --- |
| "changedTovar": [  {  "name": "Лук",  "weight": "1000",  "post": "Ванек",  "price": "100000",  "id": 1  },  {  "name": "Лук",  "weight": "1000",  "post": "Ванек",  "price": "20000",  "id": 2  },  {  "name": "Лук",  "weight": "500",  "post": "Андрей",  "price": "1000",  "id": 3  },  {  "name": "Огурец",  "weight": "1000",  "post": "Андрей",  "price": "3210",  "id": 4  },  {  "name": "Огурец",  "weight": "6000",  "post": "Андрей",  "price": "200",  "id": 5  },  {  "name": "Огурец",  "weight": "50",  "post": "Андрей",  "price": "20",  "id": 7  },  {  "name": "Огурец",  "weight": "5",  "post": "Андрей",  "price": "30",  "id": 8  },  {  "name": "Бульба",  "weight": "200",  "post": "Андрей",  "price": "30",  "id": 9  },  {  "name": "Свекла",  "weight": "300",  "post": "Андрей",  "price": "50",  "id": 11  },  {  "name": "Бульба",  "weight": "400",  "post": "Андрей",  "price": "1000",  "id": 12  },  {  "name": "Бульба",  "weight": "500",  "post": "Андрей",  "price": "2000",  "id": 13  },  {  "name": "Помидор",  "weight": "234234",  "post": "Саня",  "price": "23",  "id": 14  },  {  "name": "Свекла",  "weight": "20",  "post": "Владимир",  "price": "35",  "id": 15  },  {  "name": "Сельдерей",  "weight": "200",  "post": "Андрей",  "price": "350",  "id": 16  },  {  "name": "Сельдерей",  "weight": "200",  "post": "Андрей",  "price": "30",  "id": 17  },  {  "name": "Чеснок",  "weight": "400",  "post": "Андрей",  "price": "2000",  "id": 18  }  ], |

Листнинг 2.3 – Схема таблицы *changedTovar*

Данная таблица хранит все изменения товара.

*name* – Наименование товара

*weight* – Вес товара  
*post* – Имя поставщика данного товара  
*price* – Цена товара  
*id* – Идентификатор товара

Пример схемы таблицы *sellTovar* представлен ниже на листинге 2.4.

|  |
| --- |
| "sellTovar": [  {  "name": "Лук",  "weight": "500",  "price": "30000",  "id": 1  },  {  "name": "Лук",  "weight": "1500",  "price": "30000",  "id": 2  },  {  "name": "Лук",  "weight": "500",  "price": "2000",  "id": 3  },  {  "name": "Огурец",  "weight": "500",  "price": "1000",  "id": 4  },  {  "name": "Огурец",  "weight": "500",  "price": "3000",  "id": 5  },  {  {  "name": "Огурец",  "weight": "-20",  "price": "-1",  "id": 8  },  {  "name": "Огурец",  "weight": "-3",  "price": "-6",  "id": 9  },  {  "name": "Бульба",  "weight": "150",  "price": "250",  "id": 11  },  {  "name": "Свекла",  "weight": "30",  "price": "20",  "id": 12  },  {  "name": "Помидор",  "weight": "234234",  "price": "0",  "id": 13  },  {  "name": "Помидор",  "weight": "234234",  "price": "0",  "id": 14  },  {  "name": "Огурец",  "weight": "250",  "price": "3000",  "id": 15  },  {  "name": "Сельдерей",  "weight": "500",  "price": "50",  "id": 16  },  {  "name": "Сельдерей",  "weight": "30",  "price": "10",  "id": 17  },  {  "name": "Сельдерей",  "weight": "40",  "price": "20",  "id": 18  },  {  "name": "Чеснок",  "weight": "150",  "price": "250",  "id": 19  }  ], |

Листинг 2.4 – Схема таблицы *sellTovar*

Данная таблица содержит историю продажи товара.

*name* – Именование Товара

*weight* – Вес товара

*price* – Цена товара

*id* – Идентификатор товара

Пример схемы таблицы *password* представлен ниже на листинге 2.5.

|  |
| --- |
| "password": [  {  "pass": "564",  "id": 1  }  ] |

Листинг 2.5 – Схема таблицы *password*

Таблица содержит данные о пароле для входа в панель администратора.

*pass* – Пароль для входа

*id* – Идентификатор

**2.2 Проектирование приложения сервера**

В данном разделе будет рассмотрено проектирование приложения сервера с использованием json-server. Json-server является простым в использовании и настройке сервером, который позволяет легко создавать и запускать REST API для быстрого прототипирования веб-приложений.

Анализ требований к серверу

Перед началом проектирования сервера необходимо провести анализ требований к серверу. В данном случае, основными требованиями являются:

Поддержка CRUD-операций (Create, Read, Update, Delete) для объектов приложения;

Реализация аутентификации пользователей;

Обработка запросов с использованием формата данных JSON.

Проектирование схемы базы данных

Для реализации поддержки CRUD-операций необходима база данных, которая будет хранить объекты приложения. В данном случае, будет использоваться база данных SQLite, которая является компактной и легко поддерживаемой.

Схема базы данных включает в себя таблицы для каждого объекта приложения, а также таблицу для хранения пользователей и их учетных записей. Каждая таблица объектов будет содержать следующие поля:

Уникальный идентификатор объекта;

Название объекта;

Описание объекта;

Другие свойства, которые зависят от конкретного объекта приложения.

Разработка API-интерфейса

Для работы с объектами приложения необходимо разработать API-интерфейс, который будет реализовывать CRUD-операции. API-интерфейс должен быть легко понятным и использоваться с помощью запросов HTTP.

Для каждого объекта приложения должны быть реализованы следующие эндпоинты:

GET /object - получение всех объектов;

GET /object/:id - получение объекта по идентификатору;

POST /object - создание нового объекта;

PUT /object/:id - обновление объекта по идентификатору;

DELETE /object/:id - удаление объекта по идентификатору.

Реализация аутентификации пользователей

Для реализации аутентификации пользователей будет использоваться механизм токенов доступа. Пользователи должны иметь возможность создавать учетные записи, входить в систему с помощью своих учетных данных и получать доступ к объектам приложения, которые им разрешены.

Тестирование и отладка

После разработки сервера необходимо провести тестирование и отладку приложения. Тестирование должно включать в себя проверку всех эндпоинтов API.

**2.3 Проектирование приложения клиента**

При проектировании клиентской части стоит учесть все особенности *React*.*js*, в частности разделение страниц на компоненты, позволяющие обеспечивать повторное использование элементов страницы в других частях продукта. Повторное использование очень удобно при разработке веб-приложения, ведь оно позволяет не повторяться, а писать один и тот же код и использовать его множество раз. Особенное внимание стоит уделить архитектуре клиента. Пример архитектуры файлов клиента приведен на рисунке 2.3.

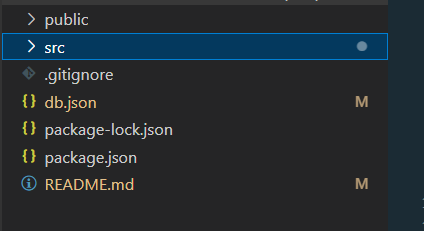


Рисунок 2.1 – Пример архитектуры файлов клиента

Рассмотрим подробнее папки, представленные на рисунке ранее, и их назначение:

* *public* – статические файлы клиента;
* *src* – папка с основными файлами клиента;

При разработке проекта было применено разделение на компоненты и страницы, на которых эти компоненты и будут использованы. Страницы, по своей сути, являются основополагающими компонентами, их основное отличие состоит в том, что на них ведет маршрутизация со стороны клиента. Каждая страница отвечает за отдельные функции, предназначенные именно этой странице.

Пример архитектуры файлов страниц клиента приведен на рисунке 2.4.

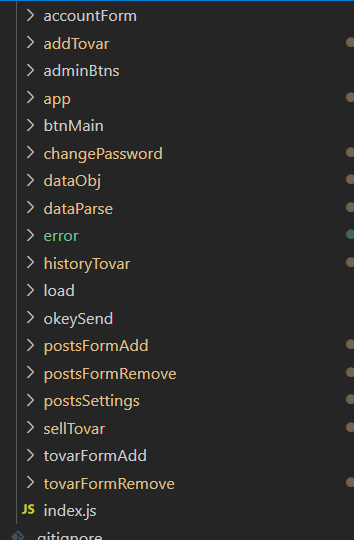


Рисунок 2.2 – Пример архитектуры файлов страниц клиента

Страницы были разделены на следующие логические сущности:

* *accountForm* – форма авторизации;
* *addTovar* – форма добавления еденицы товара;
* *adminBtns* – панель администратора;
* *app* – основной компонент приложения;
* *btnMain* – основные кнопки панели приложения;
* *changePassword* – форма изменения пароля;
* *dataObj* – компонент отображения товаров;
* *error* – компонент с ошибкой;
* *historyTovar* – отображение истории;
* *load* – компонент загрузки;
* *okeySend* – компонент успешного выполнения действия;
* *postsFormAdd* – форма добавления поставщика;
* *postsFromRemove* – форма удаления поставщика;
* *postsSettings* – компонент добавления/удаления поставщиков и товаров;
* *sellTovar* – форма продажи товара;
* *tovarFormAdd* – форма добавления товара;
* *tovarFormRemove* – форма удаления товара.

Такое большое количество страниц было нужно для обеспечения максимального комфорта и интуитивного взаимодействия пользователя с сайтом. Всегда удобно, когда на определенную функцию сайта есть своя страница.

**2.4 Диаграмма вариантов использования**



Рисунок 2.3 – Диаграмма вариантов использования

Веб-приложение для организации хранения товаров на складе предполагает две роли: роль администратора приложения, роль работника склада. Одним из возможных типов отношений между актерами, т.е. ролями в приложении, является отношение обобщения, это значит, что если есть варианты использования, которые присущи нескольким ролям, то можно их обобщить и выделить общую абстрактную роль. В этом дипломном проекте данная связь имеется между ролями «Администратор», «Работника склада». Это означает, что «Администратор» может выполнять такие же действия что и «Работник скалда», но также имеет свои дополнительные, Рассмотрим преценденты «Администратора»:

Добавление и удаление поставщиков;

Добавление и удаление вида товара на склад;

Изменения пароля;

Рассмотрим преценденты «Работника склада»:

Просмотр товара на складе;

Приём товара;

Продажа товара со склада;

**2.5 Структурная схема веб-приложения**

Существует целый ряд архитектурных шаблонов, применяемых при проектировании приложений. К ним относятся: клиент-сервер, многослойная архитектура, компонентная архитектура, архитектура, основанная на шине сообщений и сервисно-ориентированная архитектура (*SOA*).

Для проектирования веб-приложения была выбрана клиент-серверная архитектура. Пример клиент-серверной архитектуры представлен на рисунке 2.5.

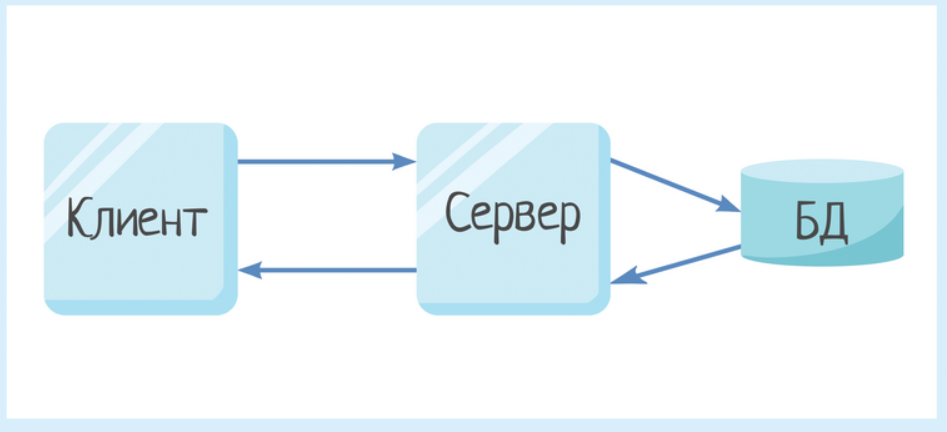


Рисунок 2.4 – Схема клиент-серверной архитектуры

У нас есть клиент, который обращается к серверу и база данных, к которой обращается сервер.

В разрабатываемом веб-приложении должен быть клиент, который будет обращаться к серверу и получать от него ответы. В свою очередь, сервер должен обращаться к базе данных для получения, сохранения, изменения или удаления данных. База данных должна вернуть ответ серверу, а сервер уже вернуть ответ клиенту. Структурная схема приложения представлена в приложении Д, а также на рисунке 2.5.



Рисунок 2.5 – Структурная схема веб-приложения

**2.6 Вывод по разделу**

В данном разделе были спроектированы все необходимые компоненты для начала разработки веб-приложения. Была спроектирована база данных с таблицами и связями между ними, была спроектирована структура файлов на клиенте и сервере. Были продуманы контроллеры для обработки клиентских запросов. Также были продуманы страницы на клиентской стороне.

3. Разработка веб-приложения

В данном разделе происходит разработка веб-приложения, которое должно соответствовать описанию из раздела проектирования**.**

3.1 Разработка приложения сервера

Реализация базы данных с использованием json-server - это процесс создания и настройки базы данных с использованием JSON-файлов в качестве источника данных и json-server в качестве инструмента для предоставления HTTP-сервера для этой базы данных. В данном разделе дипломной работы будет рассмотрена реализация базы данных с помощью json-server, включая создание JSON-файлов, настройку json-server и особенности его использования.

Создание JSON-файлов:

Первым шагом в реализации базы данных с использованием json-server является создание JSON-файлов, которые будут использоваться в качестве источника данных. JSON (JavaScript Object Notation) - это формат данных, основанный на тексте, который представляет данные в виде пар "имя: значение", разделенных запятыми, и заключенных в фигурные скобки. JSON-файлы могут быть созданы в текстовом редакторе или генерироваться автоматически с использованием других инструментов или языков программирования. JSON-файлы могут содержать различные структуры данных, такие как массивы, объекты и примитивные типы данных. Данный файл у нас создан db.json и находится в корневой папке.

Настройка json-server:

После создания JSON-файлов необходимо настроить json-server, чтобы он мог предоставлять HTTP-сервер для этих файлов. Json-server - это инструмент командной строки, который позволяет легко настроить RESTful API на основе JSON-файлов. Он поддерживает стандартные HTTP-методы, такие как GET, POST, PUT и DELETE, для взаимодействия с данными в JSON-файлах. Json-server также предоставляет дополнительные функции, такие как пагинация, сортировка и фильтрация данных.

Для настройки сервера нам достаточно записать небольшую конструкцию в наш недавно созданный файл db.json:

|  |
| --- |
| {  postsTovar": [  {  "name": "Андрей",  "phone": "+375297579370",  "id": 1  },  {  "name": "Артем",  "phone": "321443534",  "id": 5  },  {  "name": "Евгений",  "phone": "375297579343",  "id": 7  }  ]  } |

Листинг 3.1

Представленный JSON-файл содержит массив объектов, представляющих данные о поставщиках, и имеет название "postsTovar". Каждый объект в массиве содержит следующие поля:

"name": поле, содержащее имя пользователя, которое может быть строковым значением, например "Андрей", "Артем", "Евгений".

"phone": поле, содержащее номер телефона пользователя, который может быть строковым значением, например "+375297579370", "321443534", "375297579343".

"id": поле, содержащее уникальный идентификатор пользователя, который может быть числовым значением, например 1, 5, 7.

Таким же образом создадим другие массивы для хранения видов овощей, данных о покупке/продаже, а также для хранения пароля от панели администратора.

Теперь наш файл настроен и частично заполнен первичными данными. Для запуска сервера необходимо открыть командную строку в файле с нашим json-файлом и прописать команду npx json-server db.json

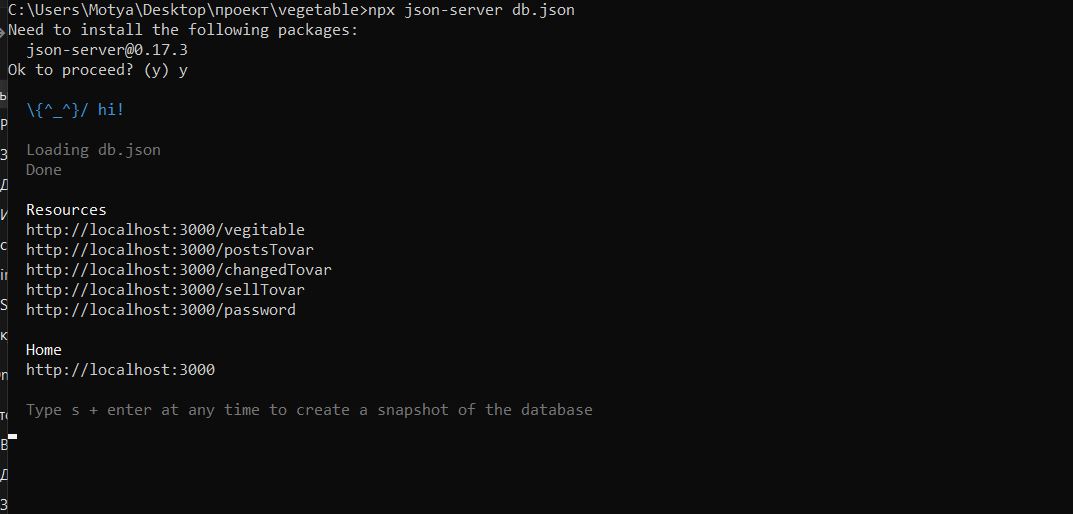


Рисунок 3.1

Готово. Наш сервер работает. Командная строка выдала нам адреса, которые мы будем использовать для получения и отправки данных.

Для получения данных в приложении React мы будет использовать метод fetch, который выполняет любой http запрос и возвращает Promise.

|  |
| --- |
| const getData = async (url) => {  const res = await fetch(url);  return await res;  } |

Листинг 3.2

В данном примере реализована асинхронная функция, которая выполняет GET по заданному url запрос и возвращает Pomise.

В заключение, реализация базы данных с использованием json-server предоставляет простой и удобный способ создания HTTP-сервера для работы с JSON-файлами в качестве источника данных. Однако, перед использованием json-server в production-среде, необходимо учитывать его ограничения и применять соответствующие меры безопасности для защиты данных.

3.2 Разработка приложения клиента

Разберемся с основными компонентами нашего приложения.

Самым главным компонентом является App.js. Данный компонент React представляет собой основной контейнер приложения с несколькими вложенными компонентами. Он содержит следующие импорты:

* BtnsMain: компонент кнопок для основного меню
* Vegitable: компонент отображения овощей
* AddTovar: компонент добавления товара
* SellTovar: компонент продажи товара
* AccontForm: компонент формы авторизации
* PostsSettings: компонент настроек поставщиков
* ChangePassword: компонент изменения пароля
* ParseData: компонент для обработки данных
* LoadState: компонент загрузки состояния
* Error: компонент ошибки

Компонент App является классовым компонентом и имеет следующее состояние:

* vegitable: флаг, указывающий, открыт ли компонент отображения овощей
* addTovar: флаг, указывающий, открыт ли компонент добавления товара
* sellTovar: флаг, указывающий, открыт ли компонент продажи товара
* posts: флаг, указывающий, открыты ли настройки поставщиков
* btnsOpen: флаг, указывающий, открыто ли основное меню
* backBtn: флаг, указывающий, открыта ли кнопка "Назад"
* accountForm: флаг, указывающий, открыта ли форма авторизации
* singed: флаг, указывающий, авторизован ли пользователь
* tovars: флаг, указывающий, открыты ли настройки товаров
* changePass: флаг, указывающий, открыт ли компонент изменения пароля
* errorPassword: флаг, указывающий, есть ли ошибка авторизации
* error: флаг, указывающий, есть ли ошибка при получении данных
* loading: флаг, указывающий, идет ли загрузка данных

Компонент содержит несколько методов для управления состоянием, таких как openBlock, openBack, openForm, closeForm, singIn, singOut, getPassword, closeErrorPassword, которые изменяют состояние компонента при определенных действиях пользователя.

|  |
| --- |
| state = {  vegitable: false,  addTovar: false,  sellTovar: false,  posts: false,  btnsOpen: true,  changePass: false,  backBtn: false,  accountForm: false,  singed: false,  password: '',  error: false,  loading: false  };  openBlock = (btnData) => {  this.setState({  [btnData]: !this.state[btnData],  btnsOpen: false,  backBtn: true,  })  }    openBack = () => {  this.setState({  vegitable: false,  addTovar: false,  sellTovar: false,  posts: false,  tovars: false,  changePass: false,  errorPassword: false,  btnsOpen: true,  backBtn: false  })  }  openForm = () => {  this.setState({  accountForm: true  })  }  closeForm = () => {  this.setState({  accountForm: false  })  }  singIn = (pass, e) => {  e.preventDefault();  if (this.state.password == pass) {  this.setState({  singed: true,  accountForm: false  })  }else{  this.setState({  errorPassword: true  })  }  }  singOut = () => {  this.setState({  singed: false,  })  }  getPassword = () => {  this.setState({  loading: true  })  this.dataObj.getData('http://localhost:3000/password')  .then(res => res.json())  .then((res) => {  this.setState({  password: res[0].pass,  loading: false  })  }).catch((e) => {  this.setState({  loading: false,  error: true  })  })  }  closeErrorPassword = () => {  this.setState({  errorPassword: false  })  } |

Листинг 3.3

Компонент также содержит рендеринг различных дочерних компонентов в зависимости от текущего состояния, таких как BtnsMain, Vegitable, AddTovar, SellTovar, AccontForm, PostsSettings, ChangePassword, PasswordError, Error и LoadState, которые отображаются или скрываются в зависимости от значения

Метод жизненного цикла componentDidMount() используется для вызова метода getPassword(), который извлекает данные из URL-адреса "http://localhost:3000/password" и обновляет состояние с полученным значением пароля.

Метод render возвращает компонент Container из библиотеки styled-components, который служит основным контейнером для всего компонента. Внутри Container есть условный рендеринг нескольких дочерних компонентов на основе значений логических флагов в состоянии.

|  |
| --- |
| render() {  const {vegitable, addTovar, sellTovar, posts, btnsOpen, backBtn, accountForm, singed, tovars, changePass, errorPassword, error, loading} = this.state;  const load = loading ? <LoadState /> : null;  const buttons = btnsOpen ? <BtnsMain onSingOut={this.singOut} signedFlag={singed} openForm={this.openForm} openBlock={this.openBlock}/> : null;  const vegitableOpen = vegitable ? <Vegitable/> : null;  const addNewTovar = addTovar ? <AddTovar /> : null;  const sell = sellTovar ? <SellTovar /> : null;  const account = accountForm ? <AccontForm onSingIn={this.singIn} onCloseForm={this.closeForm}/> : null;  const postsSettigs = posts ? <PostsSettings dataOpen="posts" /> : null;  const tovarSettings = tovars ? <PostsSettings dataOpen="tovars" /> : null;  const passwordChange = changePass ? <ChangePassword getPassword={this.getPassword} dataOpen="changePass" /> : null;  const errorPs = errorPassword ? <PasswordError>Неверный пароль. Повторите попытку  <button className="btn btn-danger" onClick={() => this.closeErrorPassword()}>Закрыть</button> </PasswordError> : null;  const back = backBtn ? <button className='btn btn-primary mt-3' onClick={() => this.openBack()}>Назад</button> : null;  if (!loading && error) {  return <Error />  }else{  return(  <Container>  {buttons}  {vegitableOpen}  {addNewTovar}  {sell}  {postsSettigs}  {tovarSettings}  {account}  {passwordChange}  {errorPs}    {back}  {load}  </Container>  );  }  } |

Листинг 3.4

В целом, компонент App представляет собой контейнерный компонент, который управляет состоянием и поведением нескольких дочерних компонентов в более крупном приложении React. Он включает условный рендеринг, обработку событий и взаимодействие с внешними API для извлечения данных.

Довольно сложный компонент Vegetable. Данный компонент представляет собой контейнер для отображения списка овощей. Он использует стороннюю библиотеку Styled-components. Vegitable является классовым компонентом, наследующим от Component из библиотеки React.

Описание структуры компонента Vegitable:

1. В начале файла импортируются необходимые модули, такие как styled-components, ParseData, LoadState, HistoryTovar и Error.
2. Затем создается стилизованный компонент VegitableContainer с помощью библиотеки styled-components. Этот контейнер содержит список овощей, отображаемых в виде карточек с изображением, названием и весом.

|  |
| --- |
| const VegitableContainer = styled.div`  display: flex;  flex-wrap: wrap;  margin-left: -10px;  .item{  width: 30%;  max-width: 100%;  box-sizing: border-box;  padding: 15px;  margin: 1.5%;  box-shadow: 0px 0px 20px rgba(1, 1, 1, .2);  border-radius: 5px;  .image{  width: 100%;  max-width: 100%;  height: 150px;  position: relative;  overflow: hidden;  img{  min-height: 150px;  max-height: 150px;  display: block;  position: absolute;  left: 50%;  top: 50%;  transform: translate(-50%, -50%);  }  }  .tittle{  font-size: 17px;  font-weight: 500;  line-height: 1;  margin-top: 15px;  }  .weight, .post{  font-size: 16px;  line-height: 1;  margin-top: 15px;  }  }  `; |

Листинг 3.5

1. Затем определяется начальное состояние компонента, включающее массивы items, changed, sell для хранения данных о овощах, измененных овощах и проданных овощах, а также флаги history, loading и error для управления состоянием загрузки и ошибок.
2. Создается экземпляр класса ParseData для работы с данными.

|  |
| --- |
| state = {  items: [],  changed: [],  sell: [],  history: false,    loading: false,  error: false  };  dataObj = new ParseData(); |

1. В методе componentDidMount() компонента Vegitable вызываются методы getItems(), getChanged() и getSell() для загрузки данных о товарах, измененных товарах и проданных товарах при монтировании компонента.
2. Далее определяются методы openHistory(), closeHistory(), getItems(), getChanged() и getSell() для работы с данными, получения их из внешнего источника и обновления состояния компонента.

|  |
| --- |
| componentDidMount(){  this.getItems();  this.getChanged();  this.getSell();  }  openHistory = (name) => {  this.setState({  history: name  });  }  closeHistory = () => {  this.setState({  history: false  })  }  getItems = () => {  this.setState({loading: true})  this.dataObj.getData('http://localhost:3000/vegitable')  .then(res => res.json())  .then(res => {  this.setState({  items: res,  loading: false  })  }).catch(e => this.setState({  error: true,  loading: false  }))  }    getChanged = () => {  this.dataObj.getData('http://localhost:3000/changedTovar')  .then(res => res.json())  .then(res => {  this.setState({  changed: res  })  });  }  getSell = () => {  this.dataObj.getData('http://localhost:3000/sellTovar')  .then(res => res.json())  .then(res => {  this.setState({  sell: res  })  });  } |

Листинг 3.6

1. Внутри метода render() определяется переменная dataHistory, которая содержит компонент HistoryTovar, если флаг history имеет значение true, иначе она равна null.
2. Далее определяется переменная vegitables, содержащая массив карточек овощей, полученных из массива items и обработанных с помощью методов changed и sell.

|  |
| --- |
| const {items, changed, sell, history, loading, error} = this.state;  const dataHistory = history ? <HistoryTovar closeHistory={this.closeHistory} name={history} sell={sell} add={changed} /> : null;  const vegitables = items.map(({name, weight, image}, i) => {  changed.forEach(chang => {  if (name == chang.name){  weight = +weight + +chang.weight;  }  })  sell.forEach(selled => {  if (name == selled.name) {  if (weight >= selled.weight) {  weight = +weight - +selled.weight  }  }  })  return (  <div className="item" key={i} onClick={() => this.openHistory(name)}>  <div className="image"><img src={image}/></div>  <div className="tittle">{name}</div>  <div className="weight">{weight}</div>  </div>  );  }) |

Листинг 3.7

1. Затем возвращается разметка компонента, включающая заголовок "Список товара", контейнер VegitableContainer, который содержит компонент HistoryTovar, карточки овощей (vegitables) и компонент Error, который отображается в случае ошибки, если флаг error имеет значение true, иначе он равен null.

|  |
| --- |
| return (  <>  <h1 className="mb-4">Список товара</h1>  <VegitableContainer>  {dataHistory}  {!loading ? vegitables : <LoadState />}  {error && !loading ? <Error /> : null}  </VegitableContainer>  </>  ); |

Листинг 3.8

Одним из самый важных элементов нашего приложения является класс ParseData:

|  |
| --- |
| class ParseData{  getData = async (url) => {  const res = await fetch(url);  return await res;  }  newData = async (formData, url) => {  const res = await fetch(url, {  method: 'POST',  headers: {  'Accept': 'application/json',  'Content-Type': 'application/json'  },  body: JSON.stringify(formData)  });  return await res;  }  sellTovar = async (sellData) => {  const res = await fetch('http://localhost:3000/sellTovar', {  method: 'POST',  headers: {  'Accept': 'application/json',  'Content-Type': 'application/json'  },  body: JSON.stringify(sellData)  });  return await res;  }  deleteItem = async (url, id) => {  const res = await fetch(url + id, {  method: 'DELETE',  headers: {  'Accept': 'application/json',  'Content-Type': 'application/json'  }  });  return await res;  }  changePassword = async (password) => {  const res = await fetch('http://localhost:3000/password/1', {  method: 'PUT',  headers: {  'Accept': 'application/json',  'Content-Type': 'application/json'  },  body: JSON.stringify(password)  });  return await res;  }  }  export default ParseData; |

Листинг 3.9

Данный класс представляет собой класс для работы с асинхронными запросами на сервер для получения, создания, обновления и удаления данных. Он содержит следующие методы:

getData(url): Метод для отправки GET-запроса на указанный URL и получения данных. Возвращает результат запроса (res) после его выполнения.

newData(formData, url): Метод для отправки POST-запроса на указанный URL с переданными данными в формате JSON (formData). Возвращает результат запроса (res) после его выполнения.

sellTovar(sellData): Метод для отправки POST-запроса на URL http://localhost:3000/sellTovar с переданными данными о продаже (sellData) в формате JSON. Возвращает результат запроса (res) после его выполнения.

deleteItem(url, id): Метод для отправки DELETE-запроса на указанный URL с переданным идентификатором (id) для удаления соответствующего элемента. Возвращает результат запроса (res) после его выполнения.

changePassword(password): Метод для отправки PUT-запроса на URL http://localhost:3000/password/1 с переданными данными о новом пароле (password) в формате JSON. Возвращает результат запроса (res) после его выполнения.

Класс ParseData экспортируется по умолчанию, что позволяет использовать его в других модулях вашего проекта для взаимодействия с сервером и выполнения указанных операций с данными.

3.3 Выводы по разделу

По итогу разработки было сделано веб-приложение, основанное на разделе посвященному проектированию, покрывая описанные в нем требования. Был разработан клиент и сервер, для обработки клиентских запросов. Также в разделе приведены листинги наиболее важных участков кода.

# 4 Тестирование веб – приложения

В данном разделе будет протестировано данное веб – приложение

## **4.1 Тестирование десктопной версии**

При запуске веб – приложения, вы попадаете на главную страницу, причём сразу уже как работник склада. Начальная страница для работника склада состоит из 2 уровней.

- Верхняя часть, панель для входа в аккаунт администратора

- Нижняя часть, панель управления работника склада

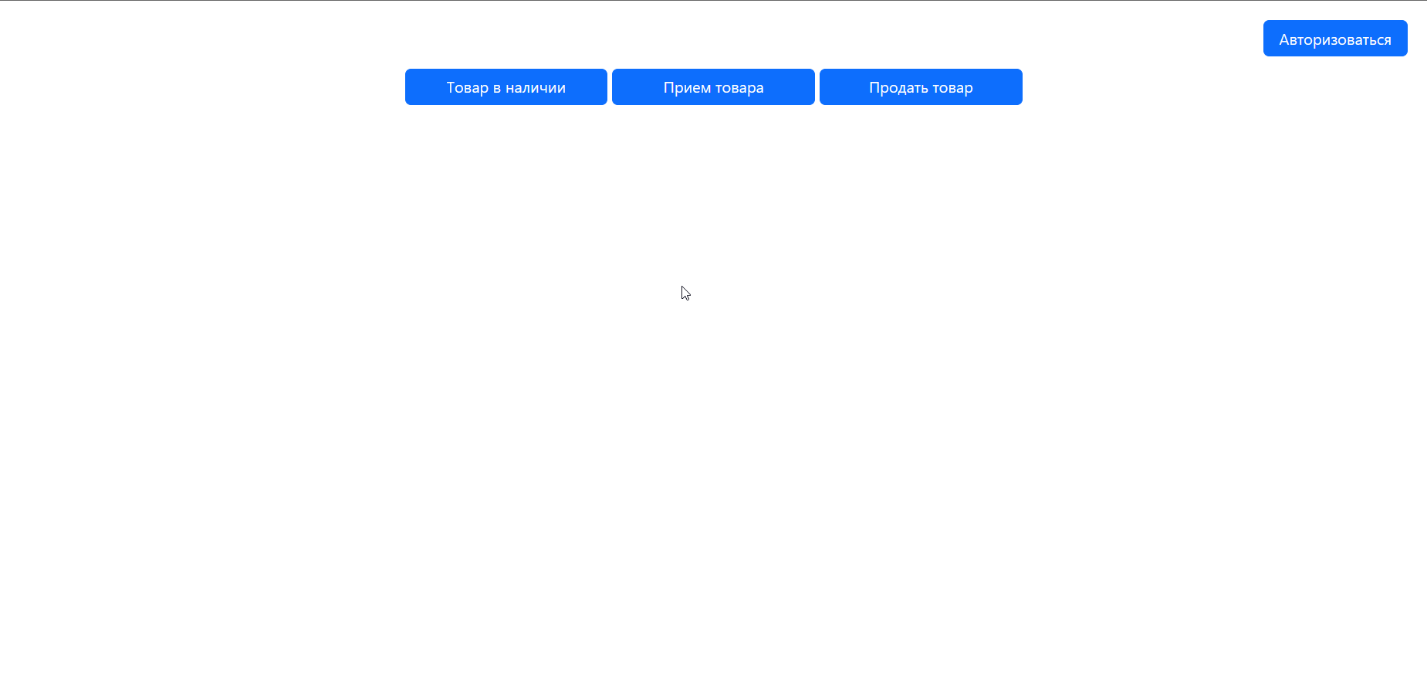


Рисунок 4.1 – Главная страница для работника склада

Давайте сразу же войдём в аккаунт администратора, т.к это не убавит функционал, а только его увеличит.

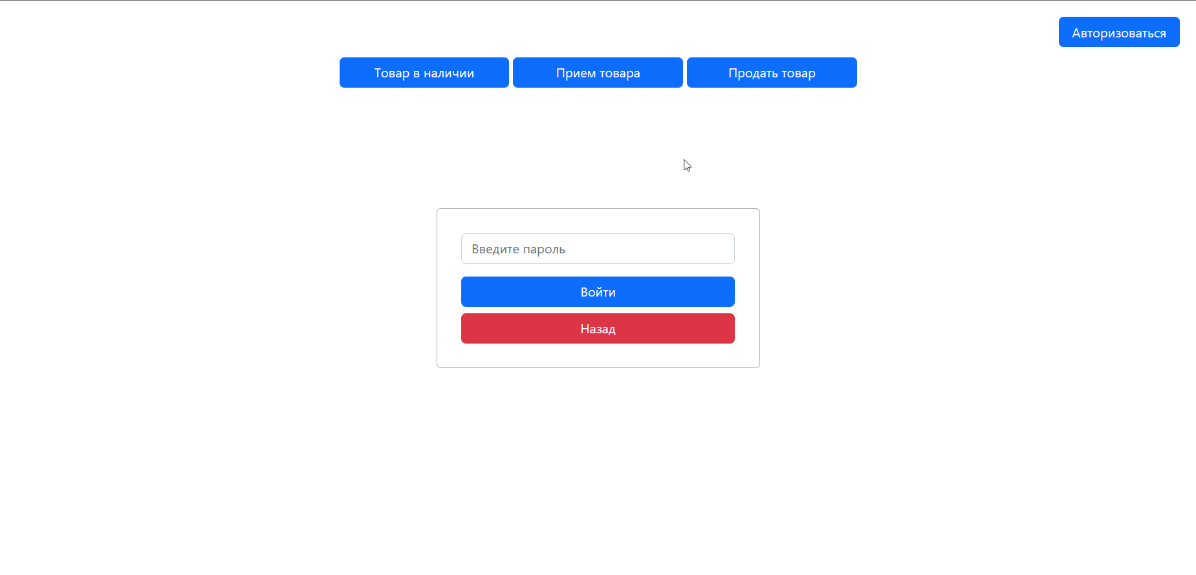


Рисунок 4.2 – Форма для входа

После нажатия кнопки “Авторизоваться” нас встречает форма для входа в панель администратора, с единственным окном для ввода – пароля.

При попытке ввода неверного пароля, нам сразу же будет об этом сообщено и будет дана повторная попытка.

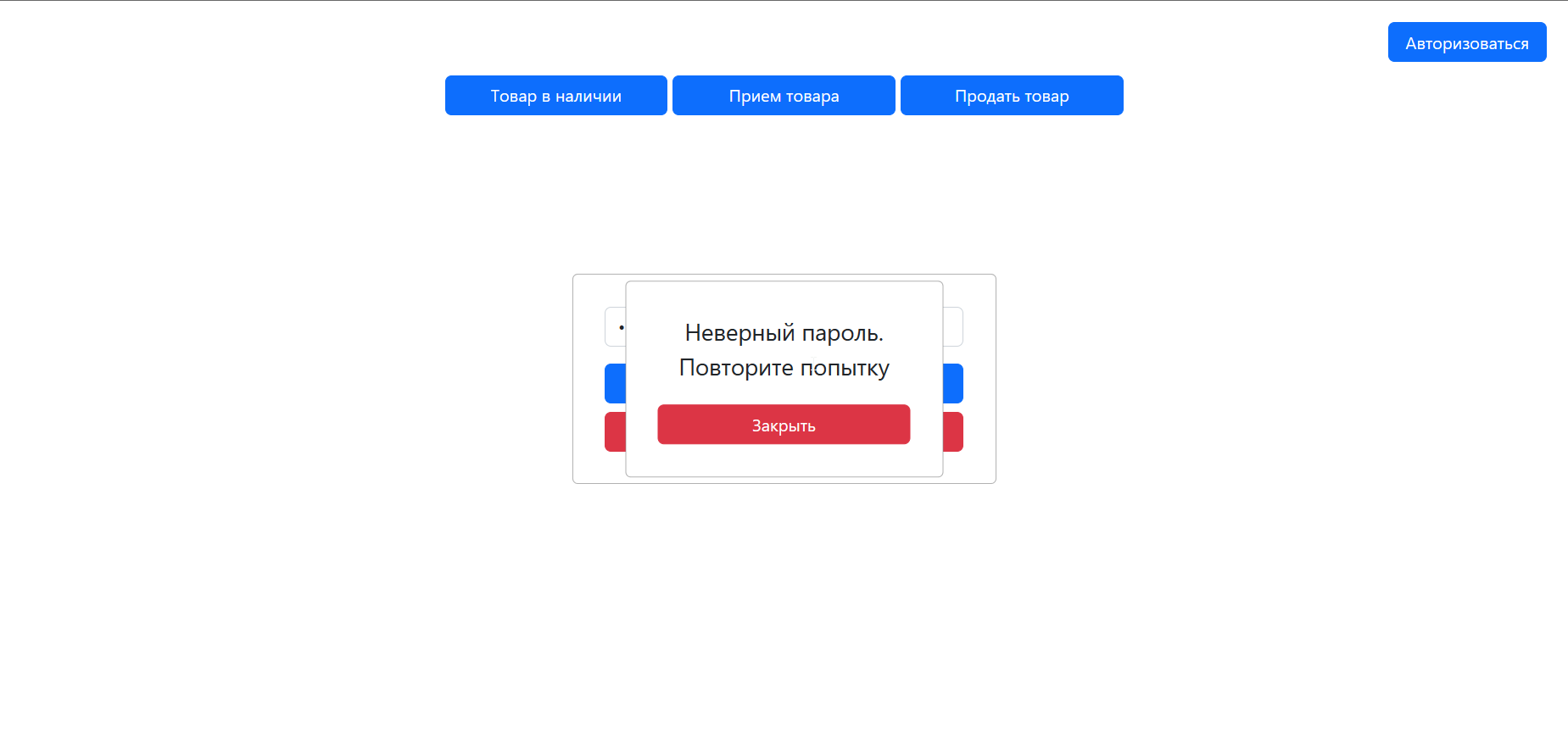


Рисунок 4.3 – Неверный пароль

А при вводе верного мы сразу же войдём в аккаунт администратора, и вот как он выглядит

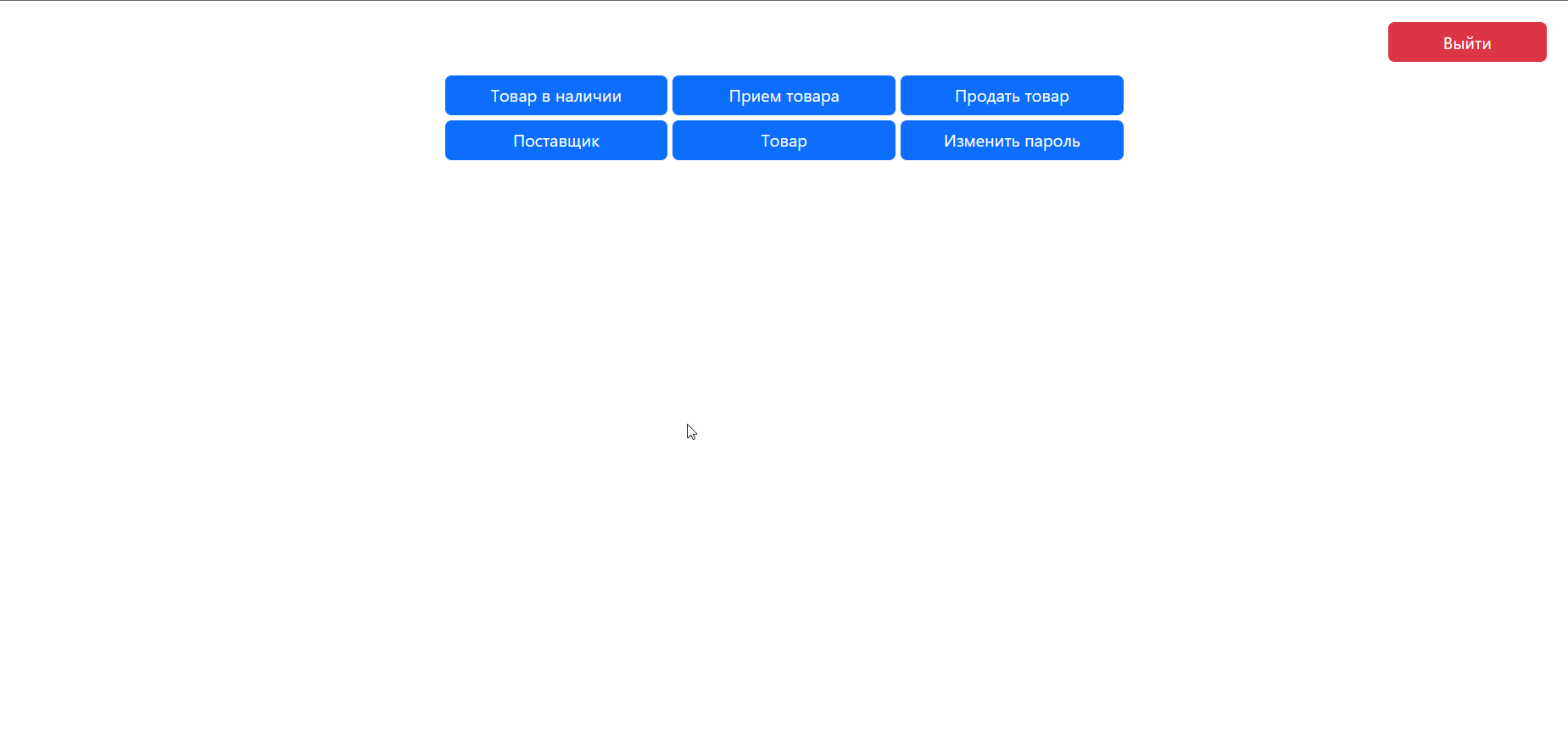


Рисунок 4.4 – Главная страница с аккаунта администратора

Как мы видим, перед кнопка “Авторизоваться” сменилась на красную кнопку “Выйти”, а так же были добавлены 3 новые панели.

Давайте далее просмотрим какой товар у нас есть в наличии

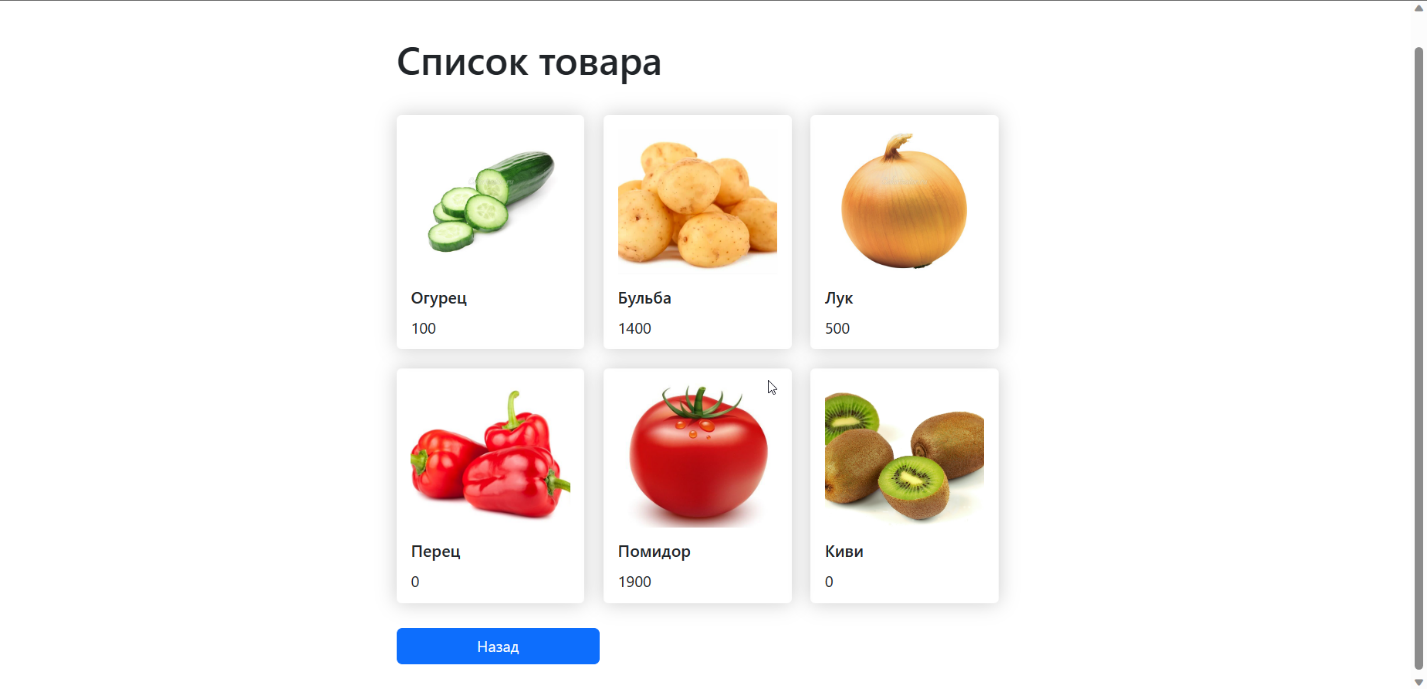


Рисунок 4.5 – “Просмотр товаров”

Тут отображается товар находящийся у нас в наличии и его число в килограммах.

При нажатии на какой – либо из товаров нам будет открыта, более подробная информация

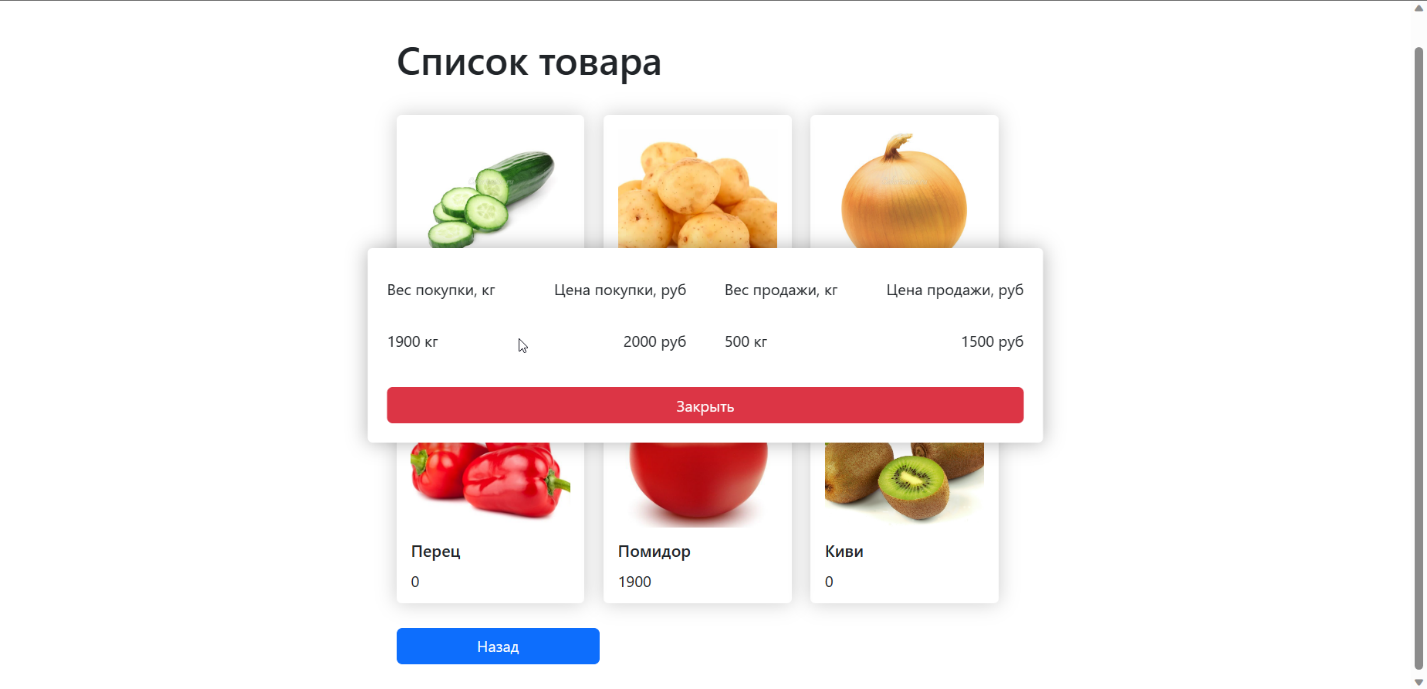


Рисунок 4.6 - Подробная информация про картофель

Далее если мы нажмём кнопку назад, нас тут же вернёт на главную панель

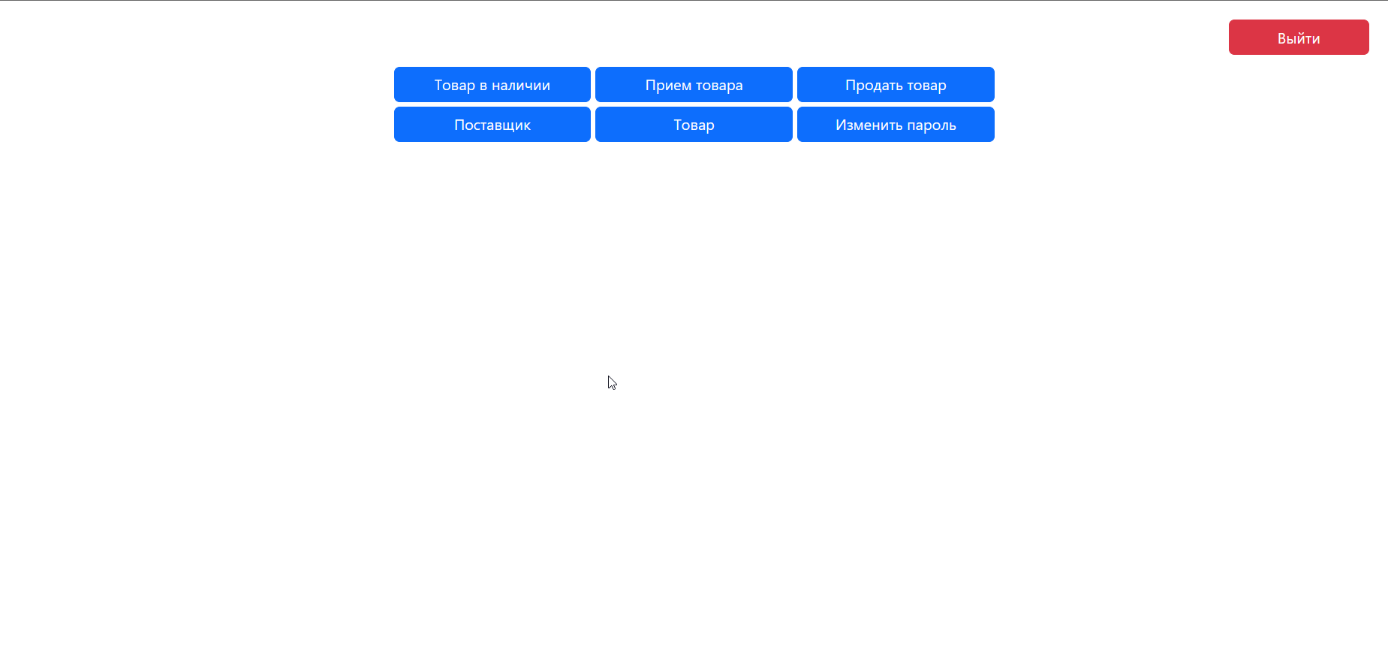


Рисунок 4.7 – После нажатия на “Назад”

Далее попытаемся продать какой либо из элементов на складе, для это перейдём в “Продать товар”.

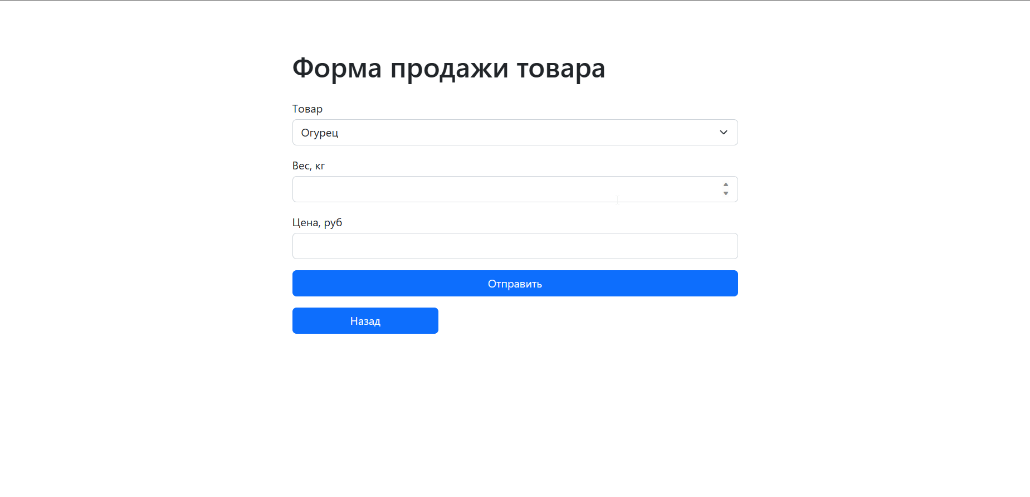


Рисунок 4.8 – Форма продажи товара

Выбираем имеющийся товар, вводим продаваемый вес и цену в рублях после чего нажимаем отправить и видим сообщение о том, что всё прошло успешно

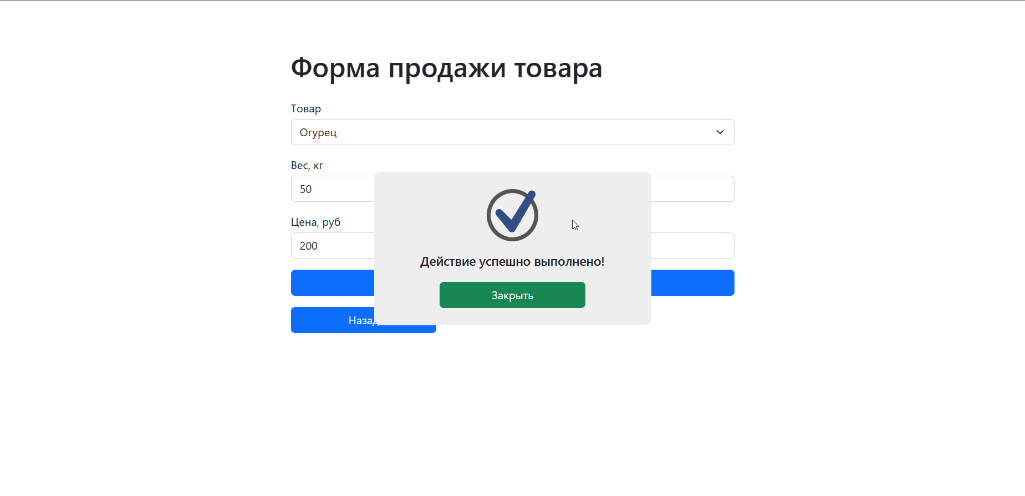


Рисунок 4.9 – Успешная продажа товара

После чего обратно перейдя к просмотру товара, мы сможем увидеть изменения.

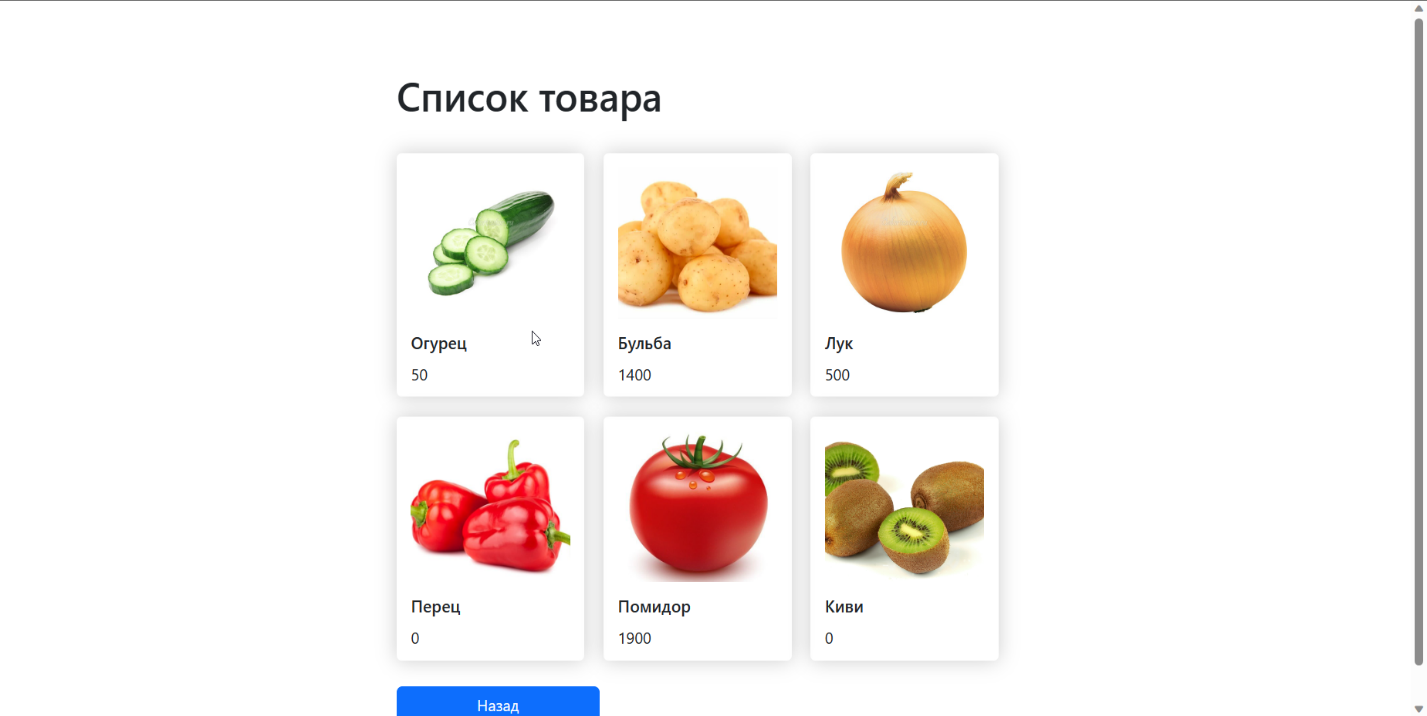


Рисунок 4.10 – Изменения после продажи

И если просмотрим подробную информацию, то и заметим там так же изменения

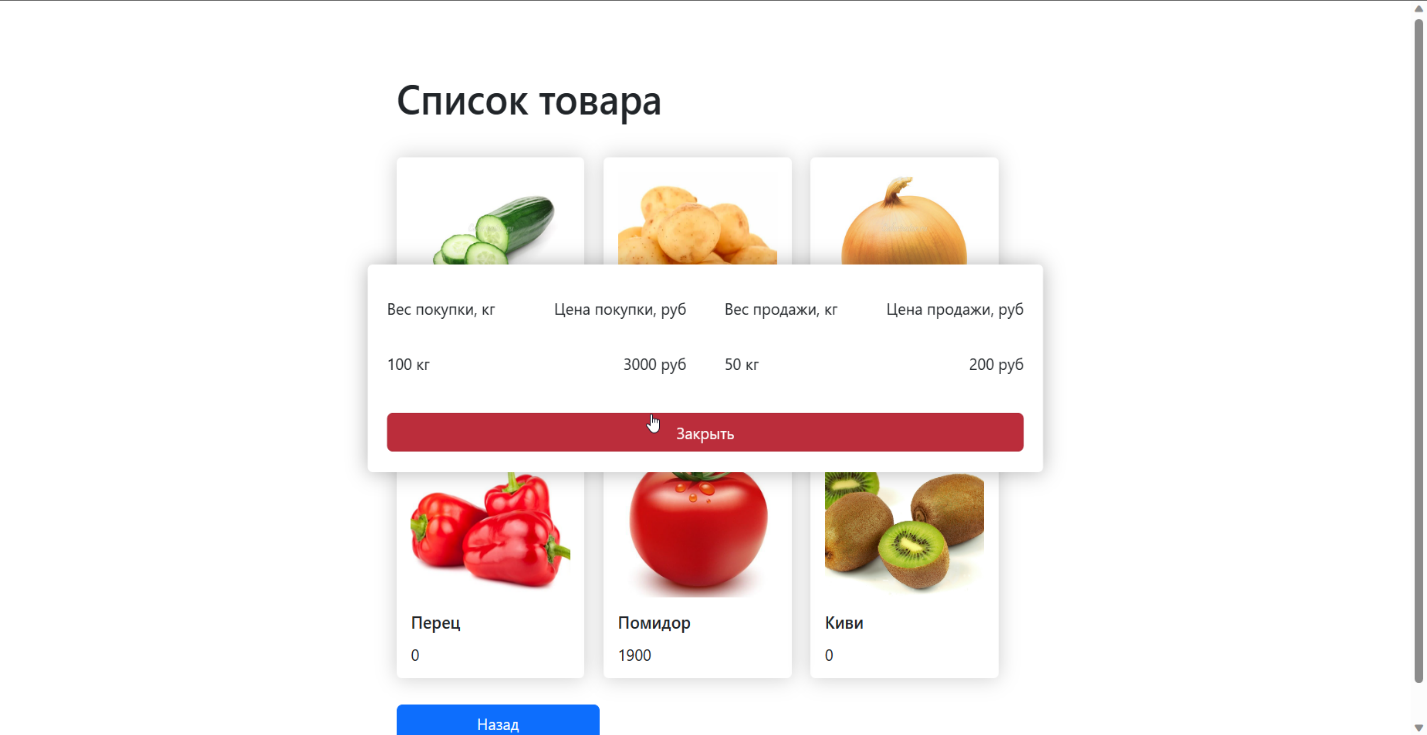


Рисунок 4.11 – Изменения после продажи, подробная информация

Так же мы можем принять товар на склад, для этого перейдём во вкладку “Приём товара”.

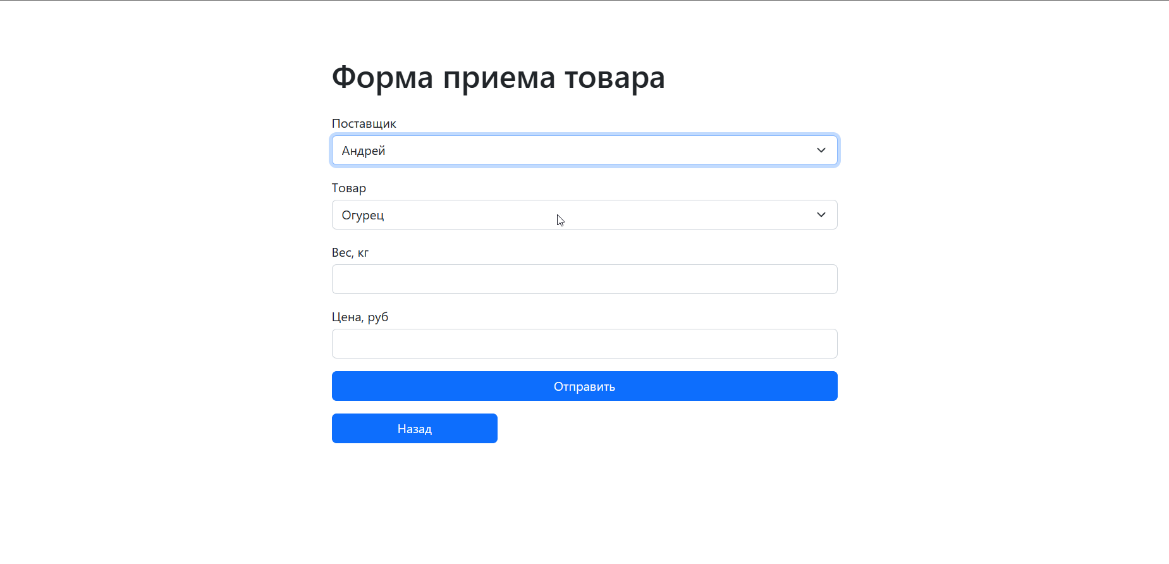


Рисунок 4.12 – форма приёма товара

Тут уже мы видим новое поле, поставщики, о которых мы поговорим чутка позже. Выберем тот же тип что товара что мы продали и введём вес покупки и цену покупки.

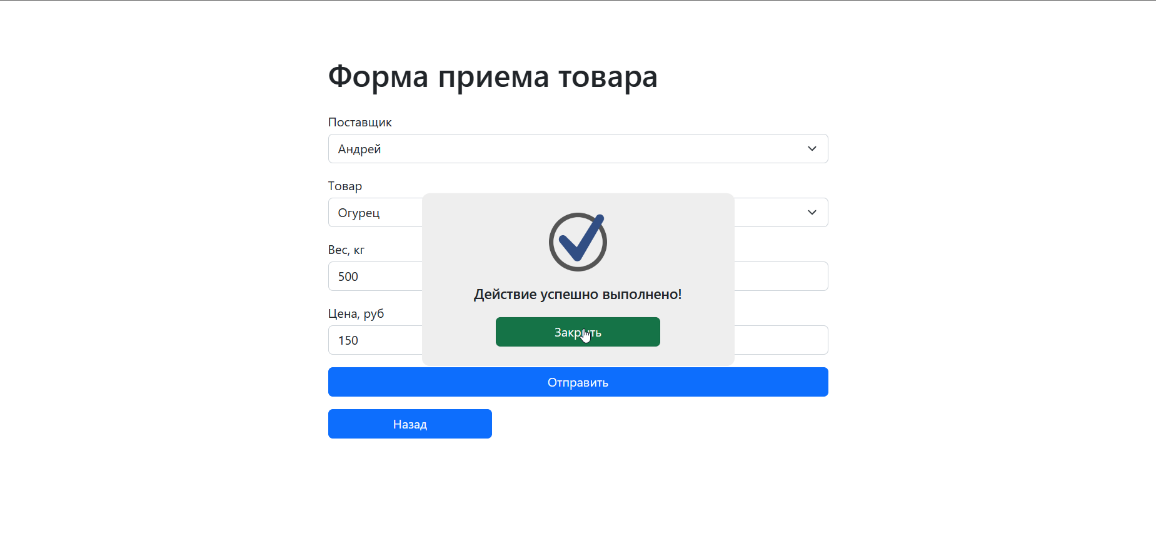


Рисунок 4.13 – Успешная продажа товара

После чего сразу же перейдём к нашему товару в наличии и проверим

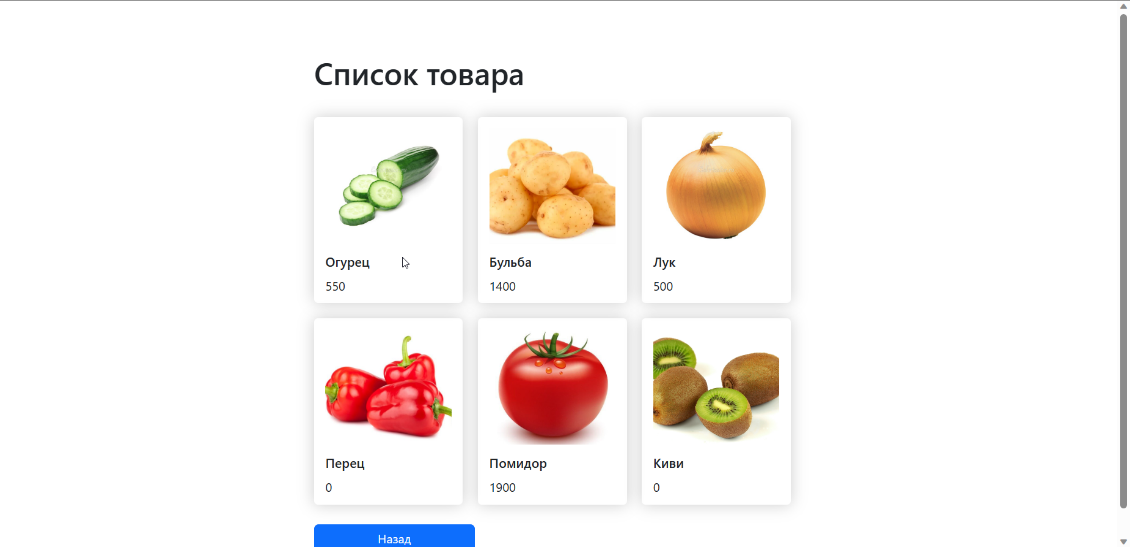


Рисунок 4.14 – Прибавление на складе

А так же просмотрим более подробную информацию о товаре

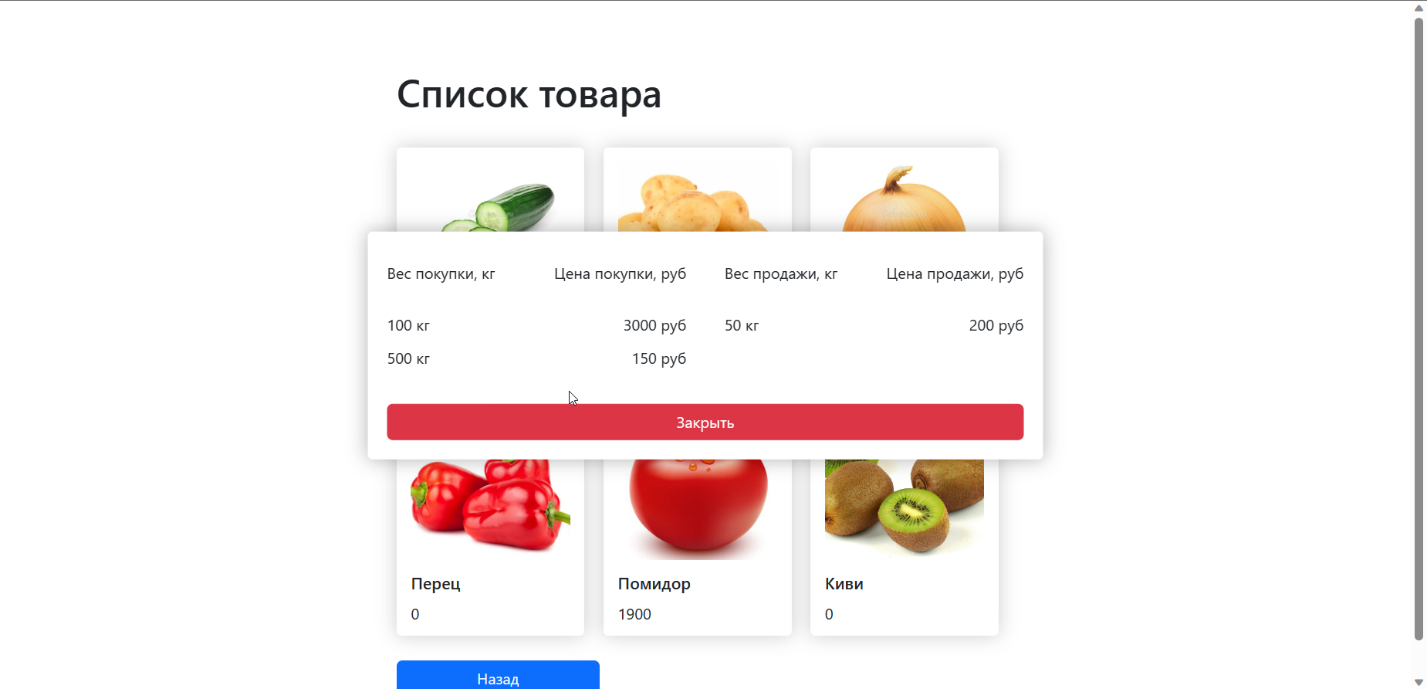


Рисунок 4.15 – Изменение, более подробно

И так же заметит тут изменения

Однако, при попытке ввода отрицательных значений, нам будет сообщено об ошибке

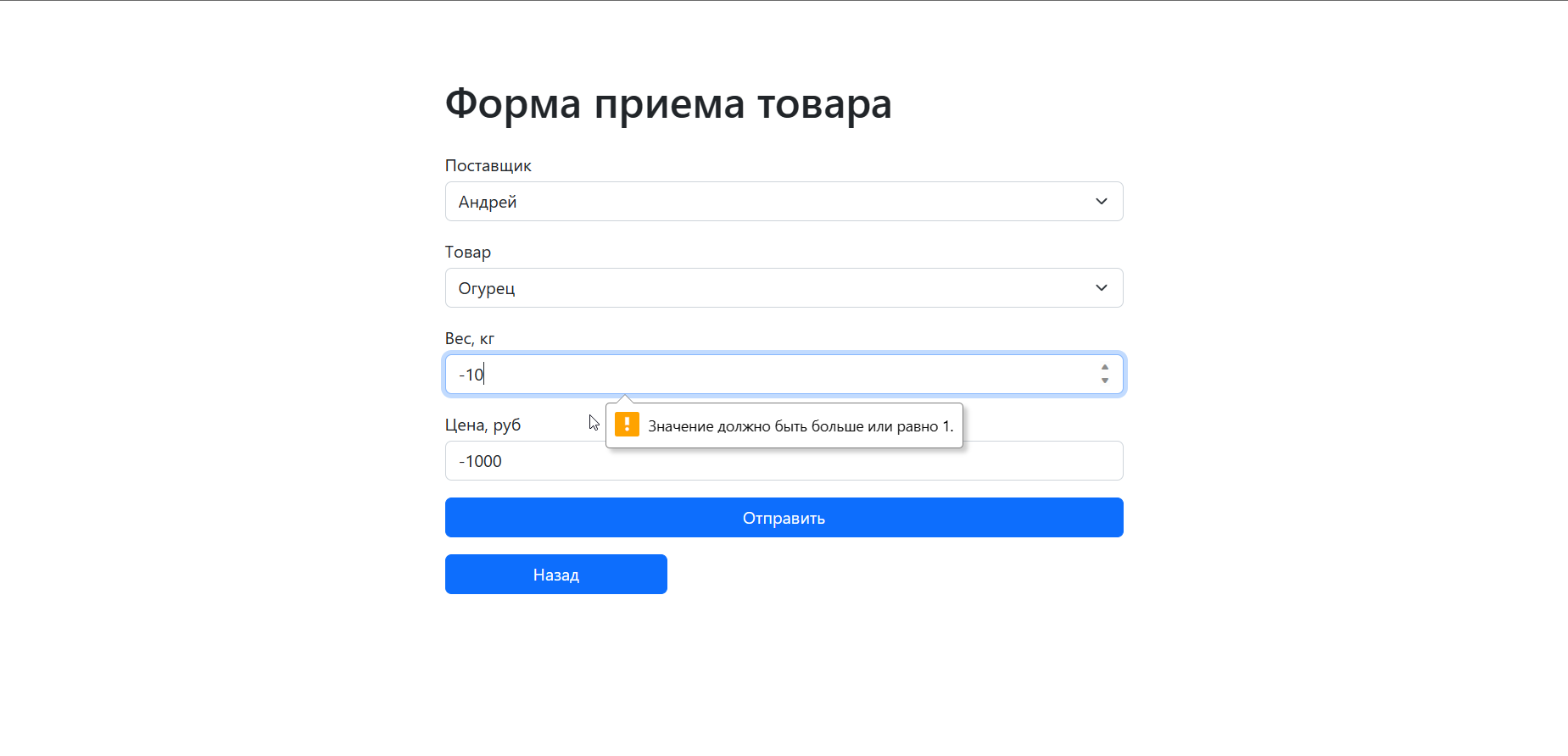


Рисунок 4.16 – Отрицательные значения

Далее перейдём во вкладку товар

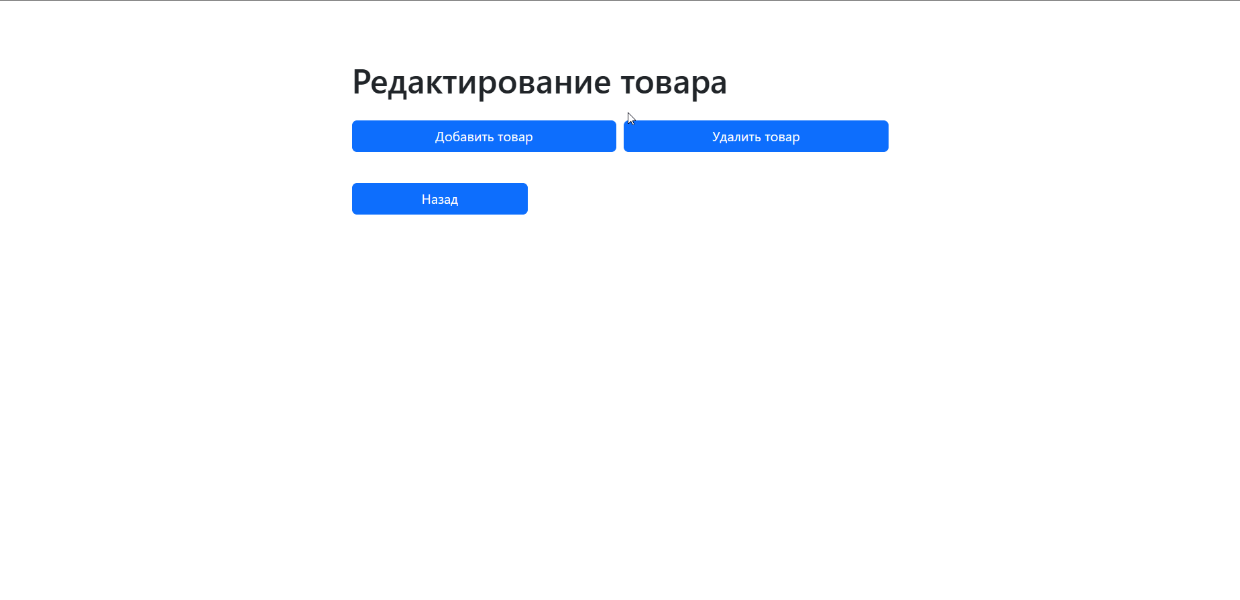


Рисунок 4.17 – Вкладка товар

Мы видим тут 2 других вкладки, добавить и удалить товар. Для начала перейдём в удалить.

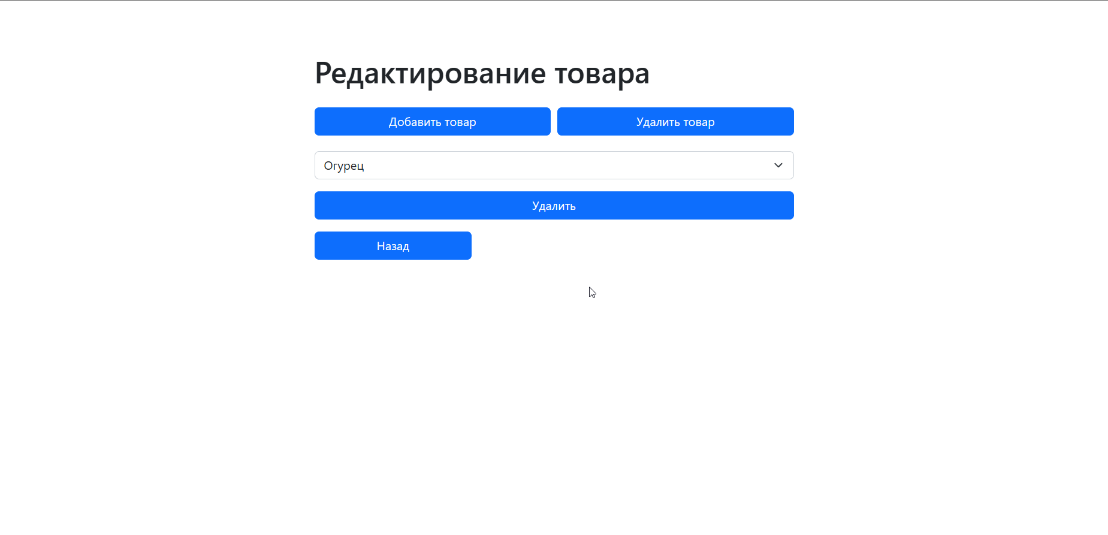


Рисунок 4.18 – Удаление товара

Далее выберем тот же тип товара на складе, что мы и использовали ранее и нажмём на кнопку “Удалить”.

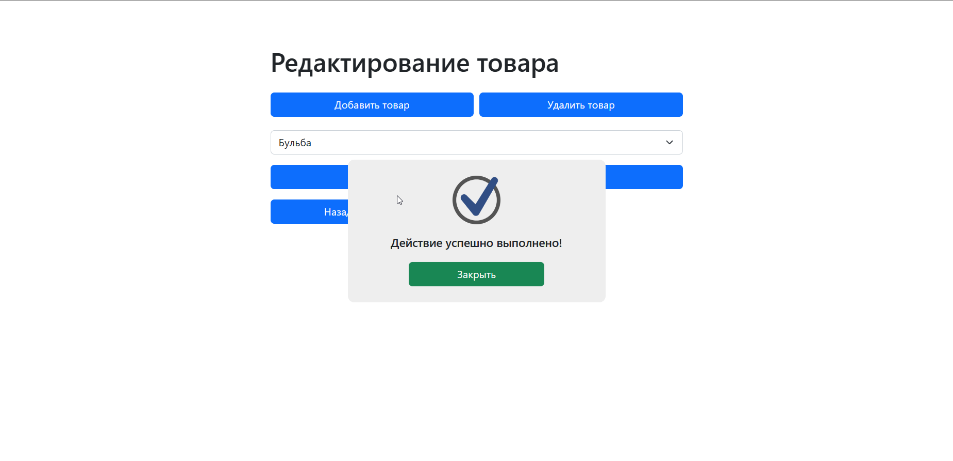


Рисунок 4.19 – Успешное удаление со склада

Как мы можем отметить, товар удалился в том числе и из поля для удаления.

Далее вновь перейдём к просмотру товара в наличии и увидим, что “Огурец” пропал.

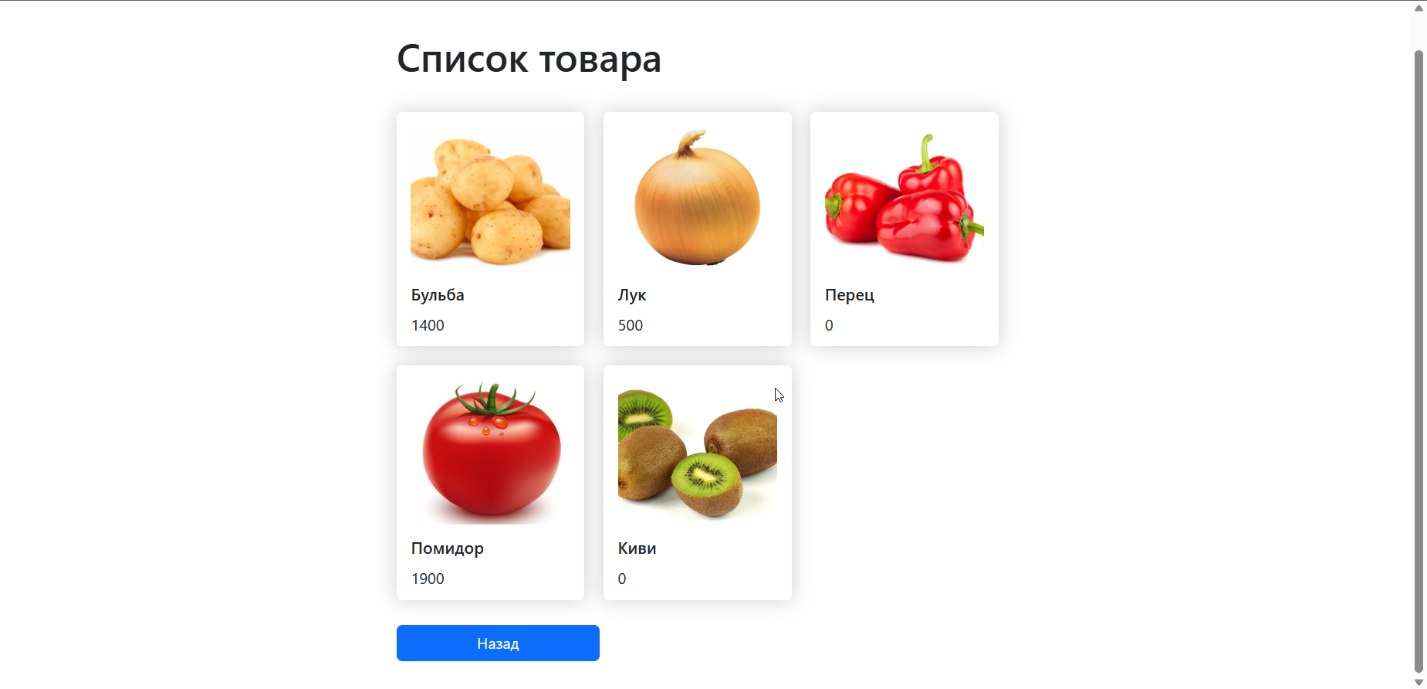
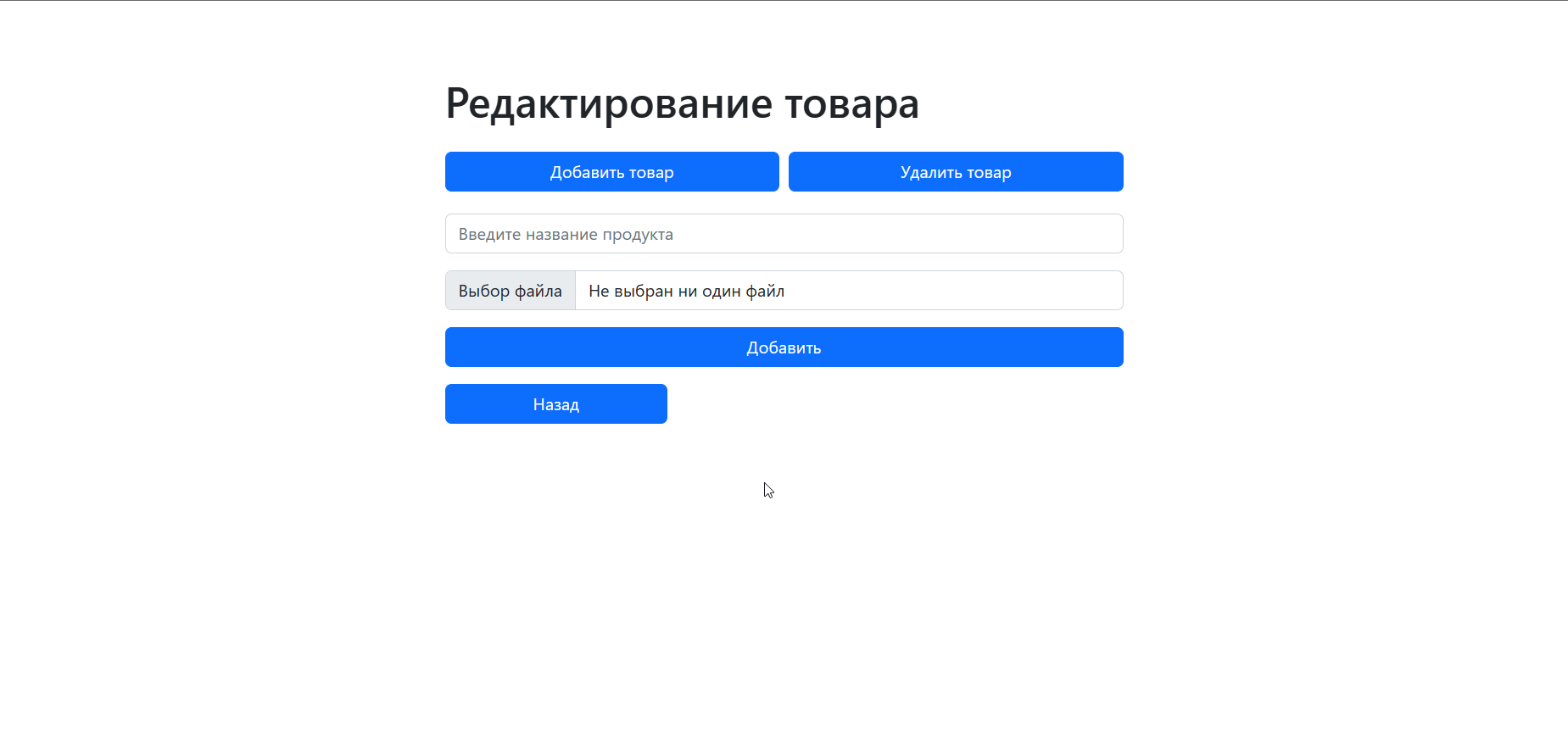


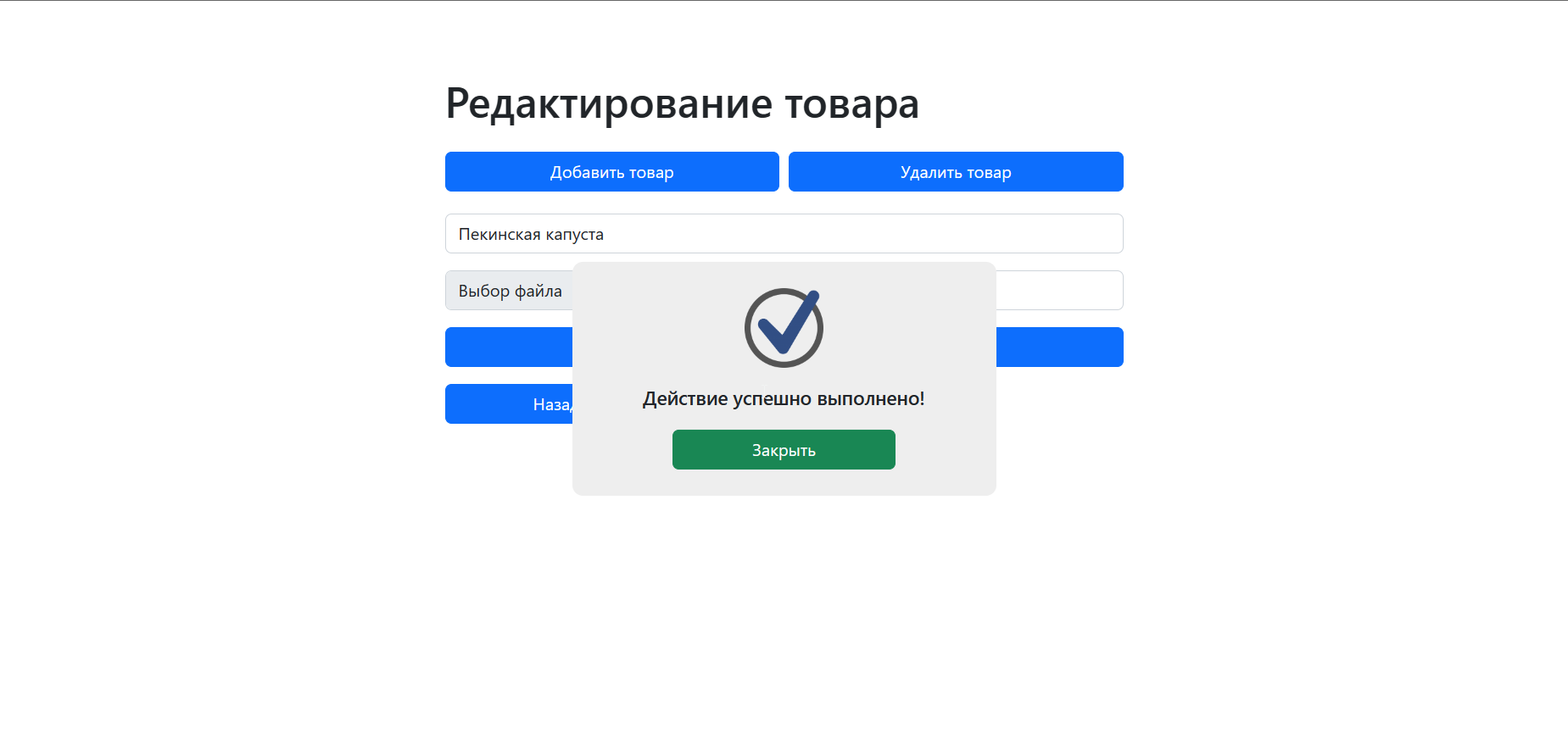
Рисунок 4.20 – Демонстрация изменения на складе

А теперь вновь вернёмся ко вкладке “Товар”, только теперь будем добавлять на склад.

  
Рисунок 4.21 – Добавление товара

Мы можем видеть здесь поле для ввода именования товара и загрузки изображения.

Далее вводим данные и нажимаем на кнопку “Добавить”.

  
Рисунок 4.22 – Успешное добавление товара

Далее перейдём во вкладку с просмотром товара, для отслеживания изменений.

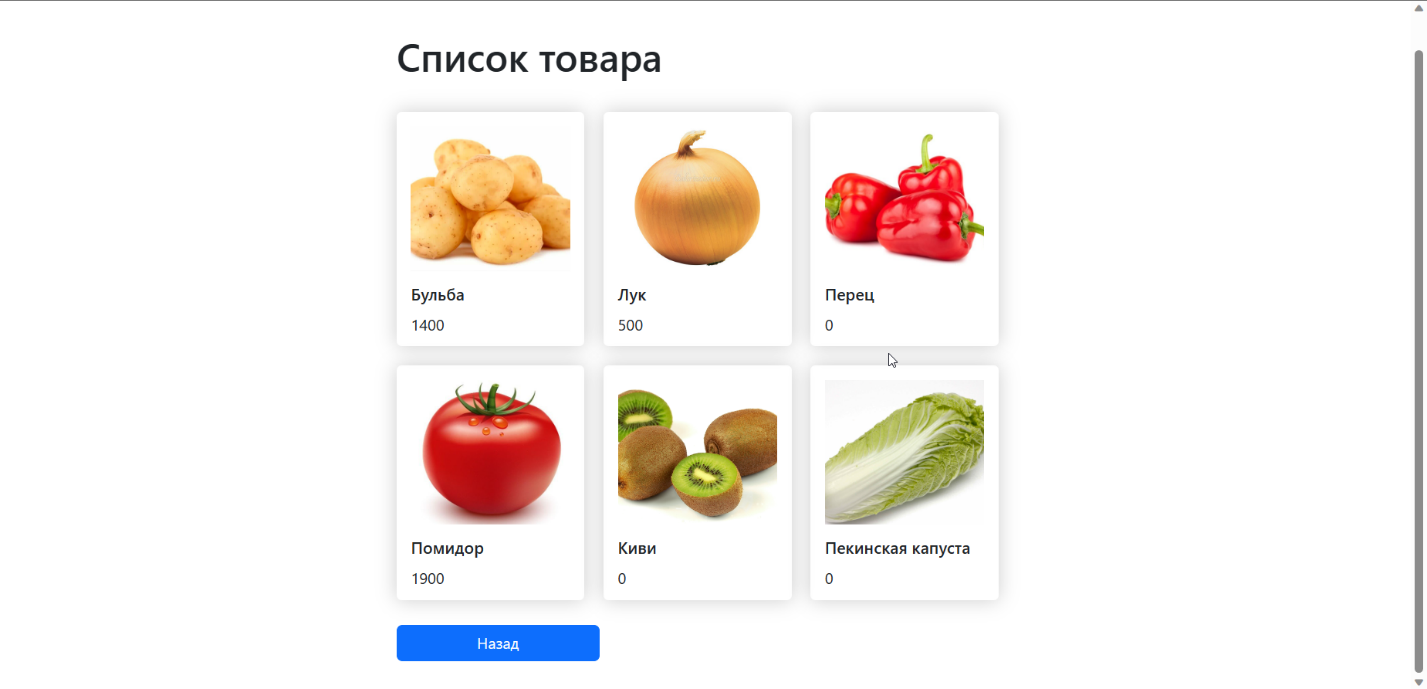
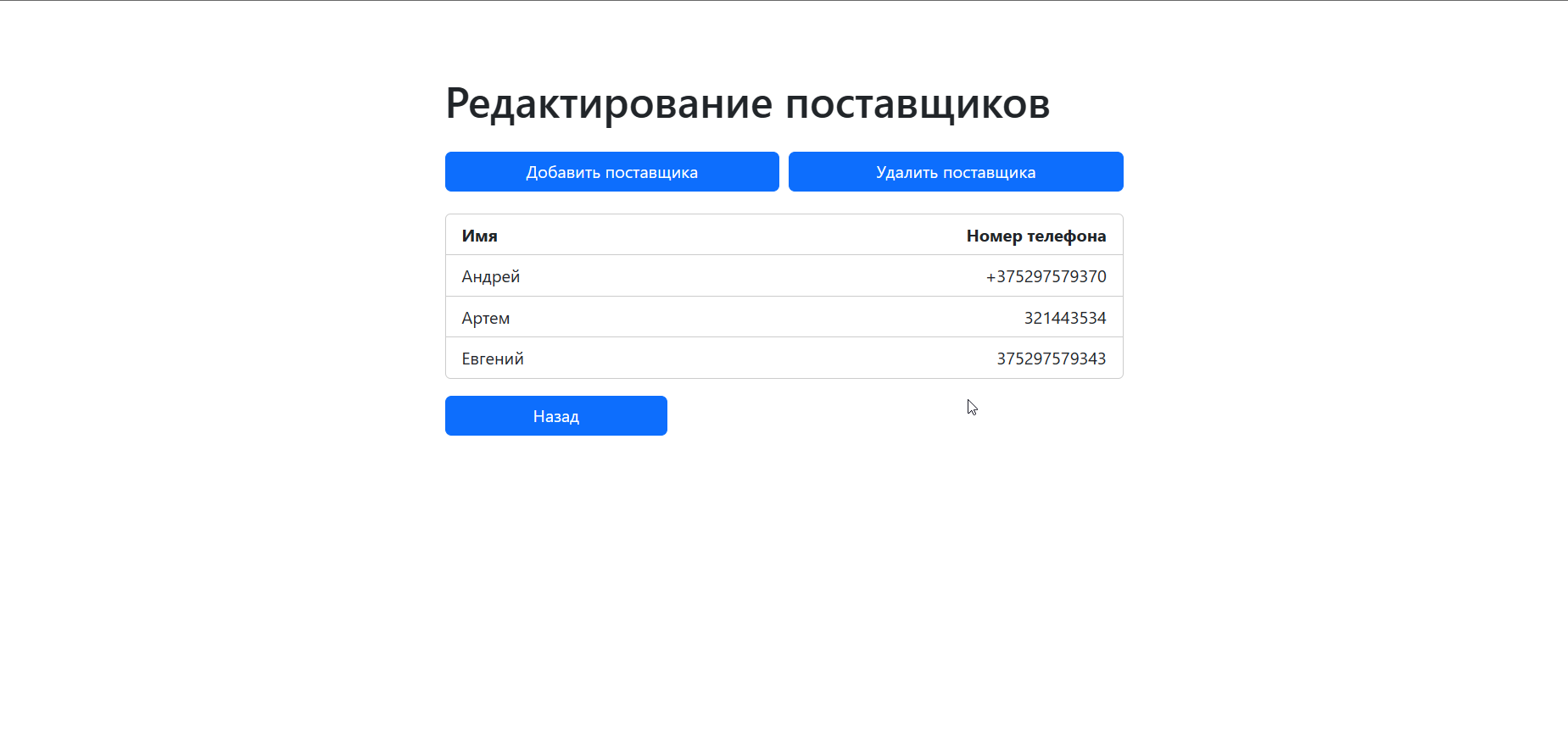


Рисунок 4.23 – Тип товара был добавлен

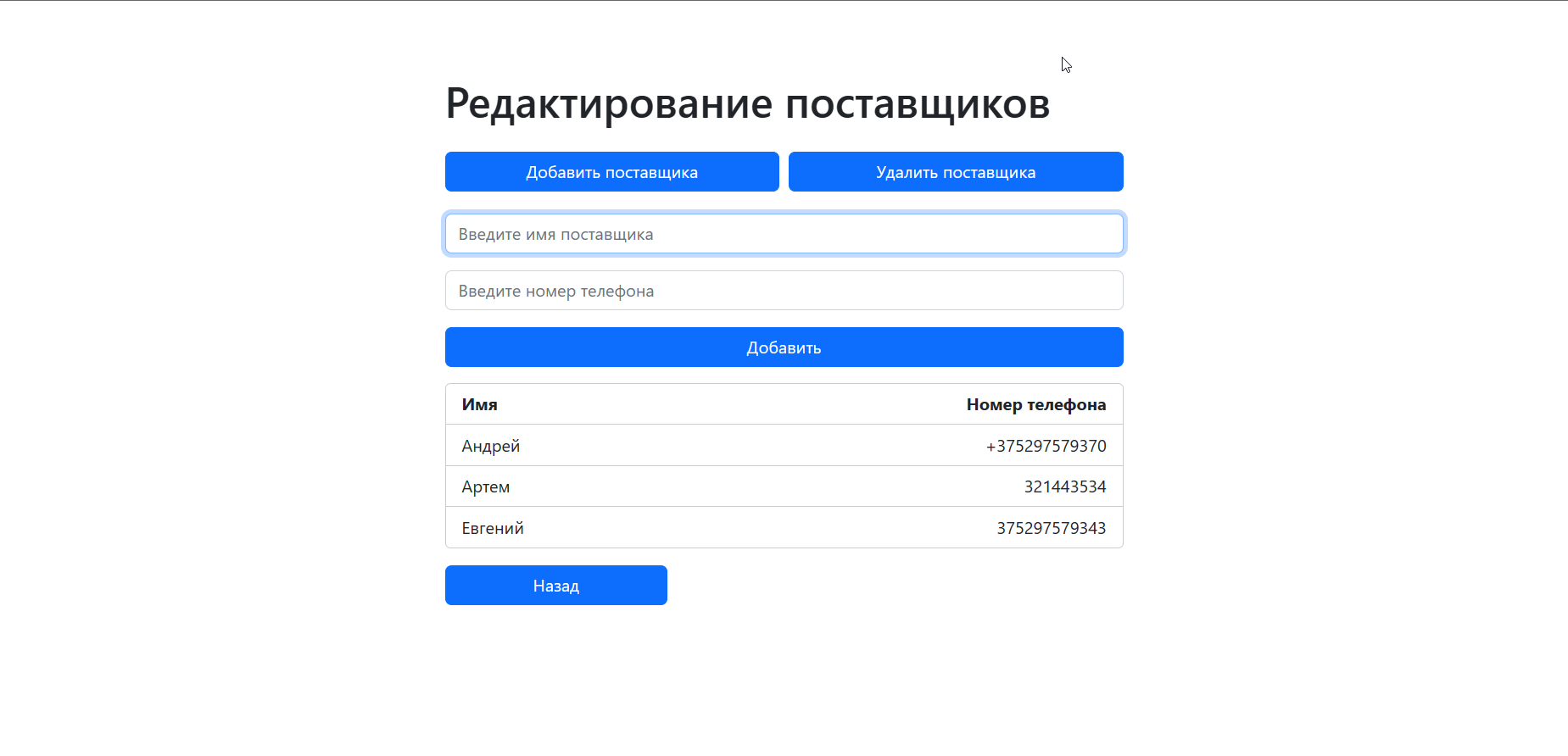
И как мы видим, тип товара был добавлен на склад.

Так же, с добавленным типом товара можно проводить все те же действия, что и товарами что были изначально.

Далее перейдём во вкладку поставщики

  
Рисунок 4.24 – Список поставщиков

Тут мы видим список существующих поставщиков с их номерами телефонов, а так же две кнопки, “Добавить поставщика” и “Удалить поставщика”. Сначала давайте добавим его.

  
Рисунок 4.25 – Форма для добавления

Введём данные и нажмём на кнопку добавить.

Если попытаться ввести в поле для номера телефона символы, то ничего вводиться не будет.

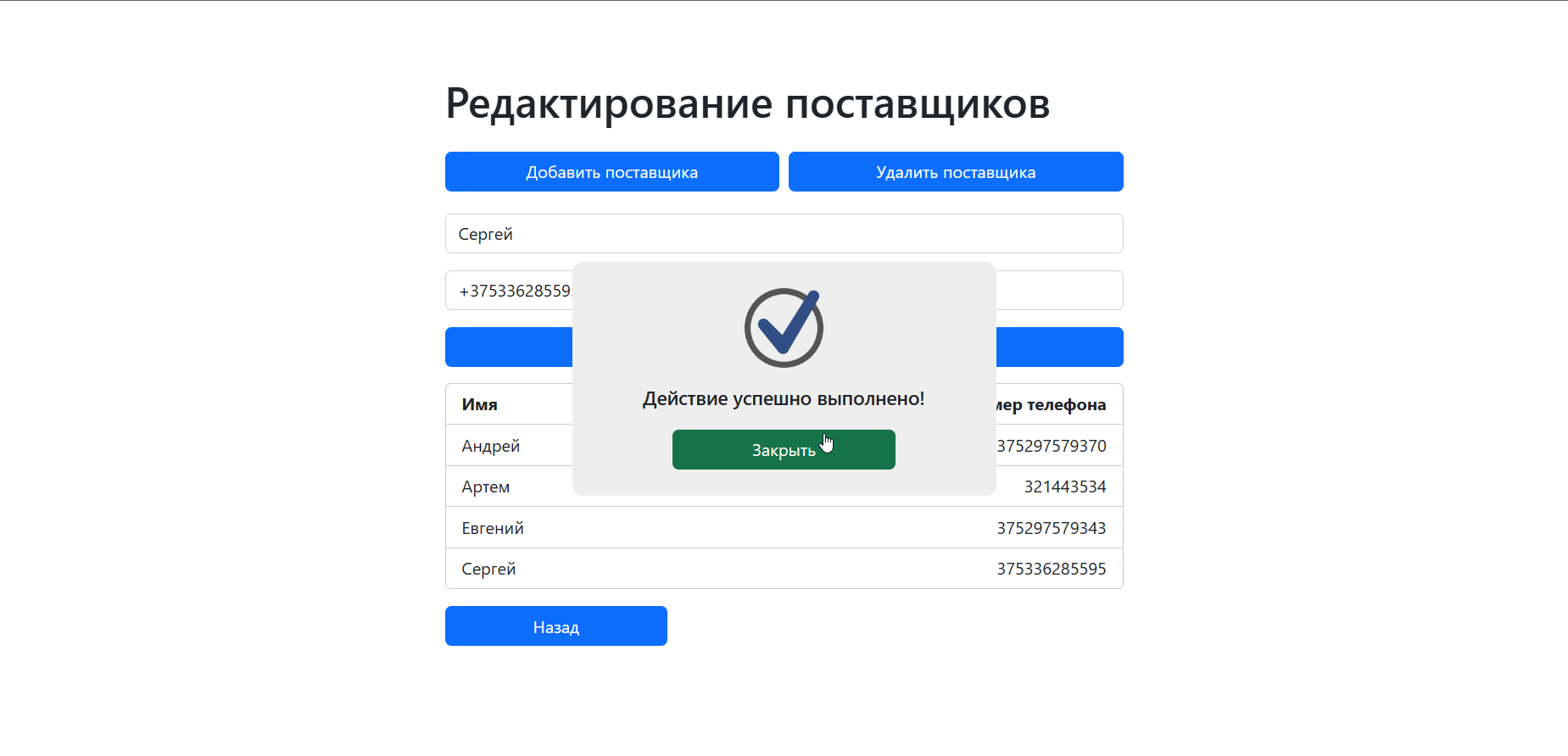


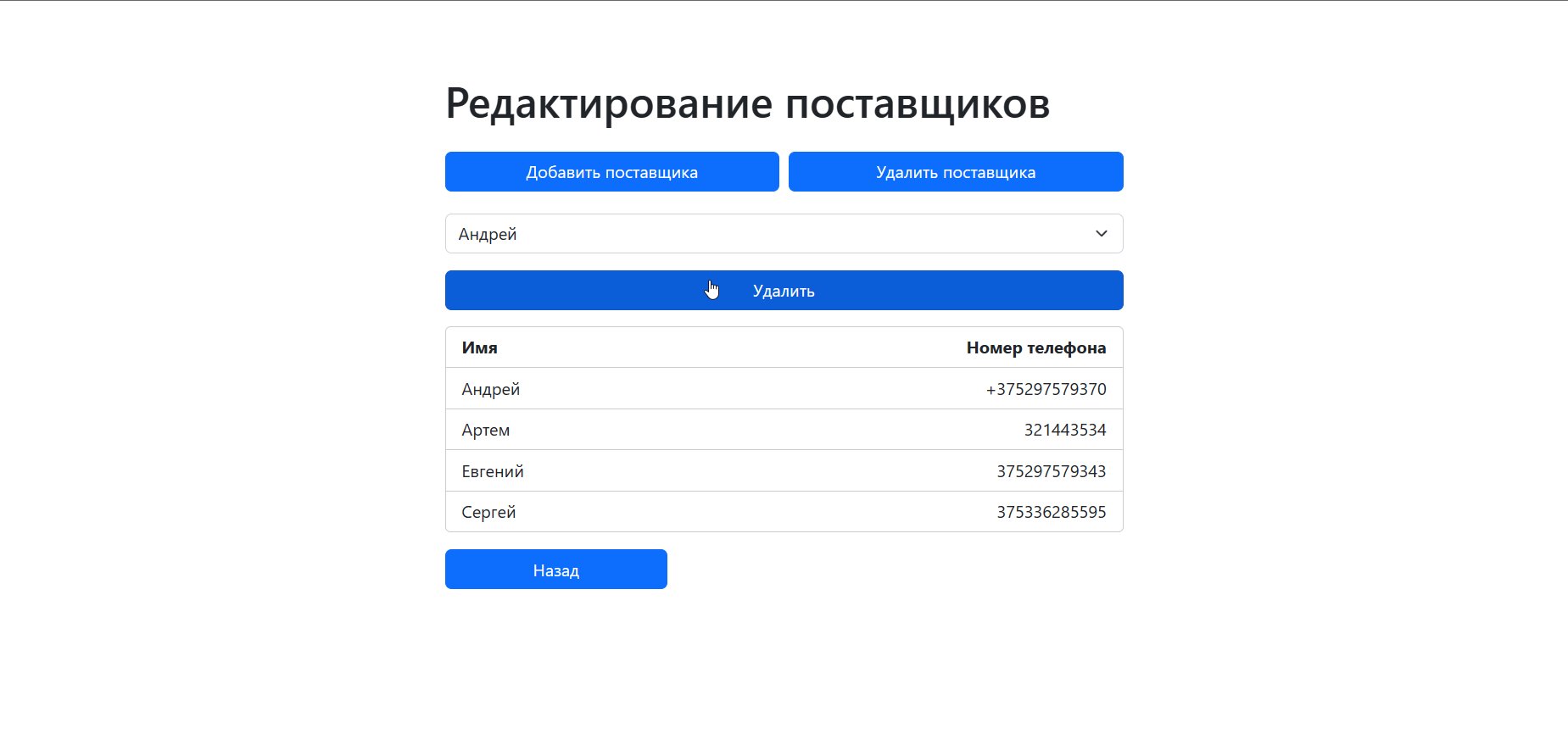
Рисунок 4.26 – Успешное добавление

Как видим, поставщик был добавлен и сразу же отображён снизу.

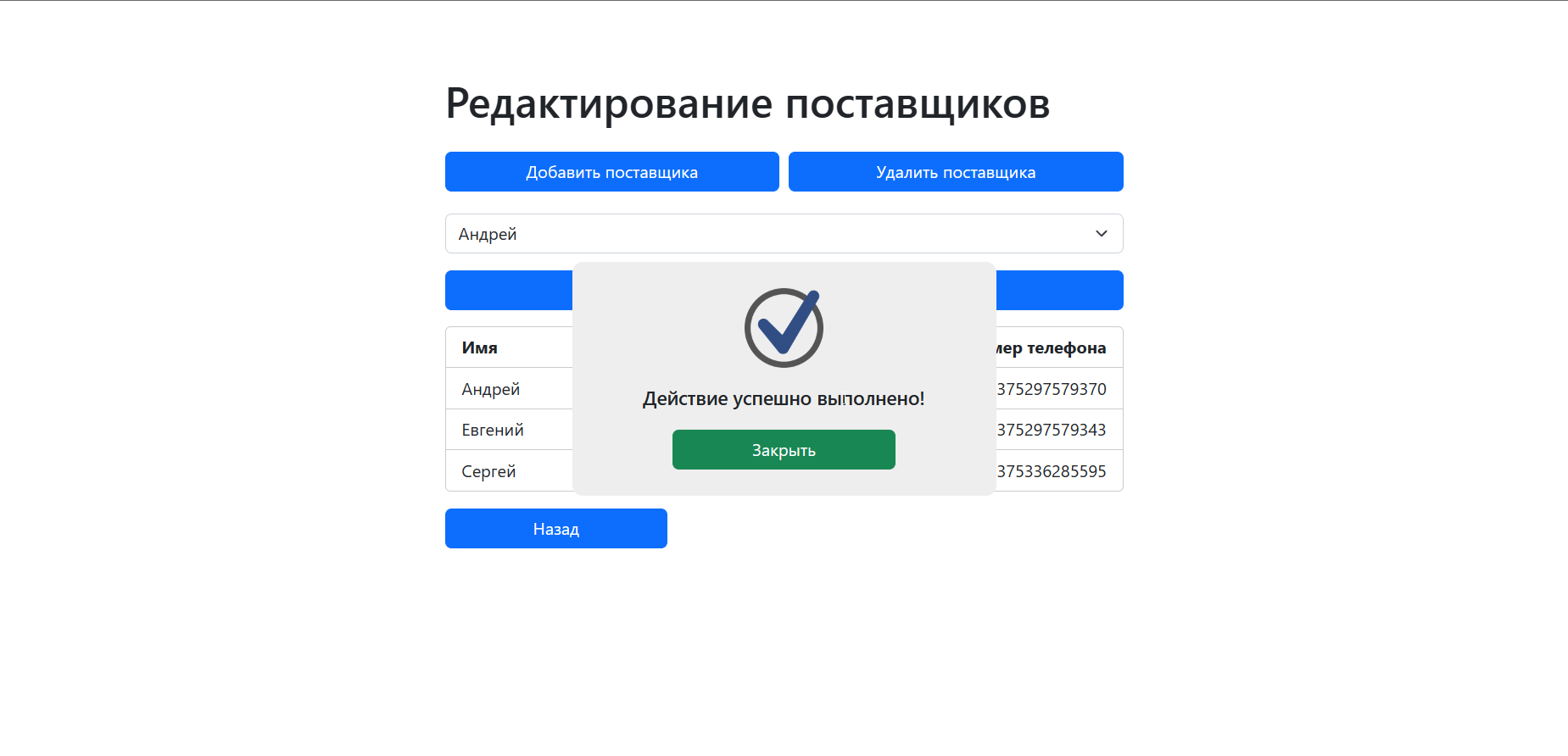
Добавленных поставщиком так же можно использовать в операциях по добавлению товара, как и изначальных.

Далее протестируем удаления поставщиков.

Нажимаем на кнопку “Удалить поставщика”

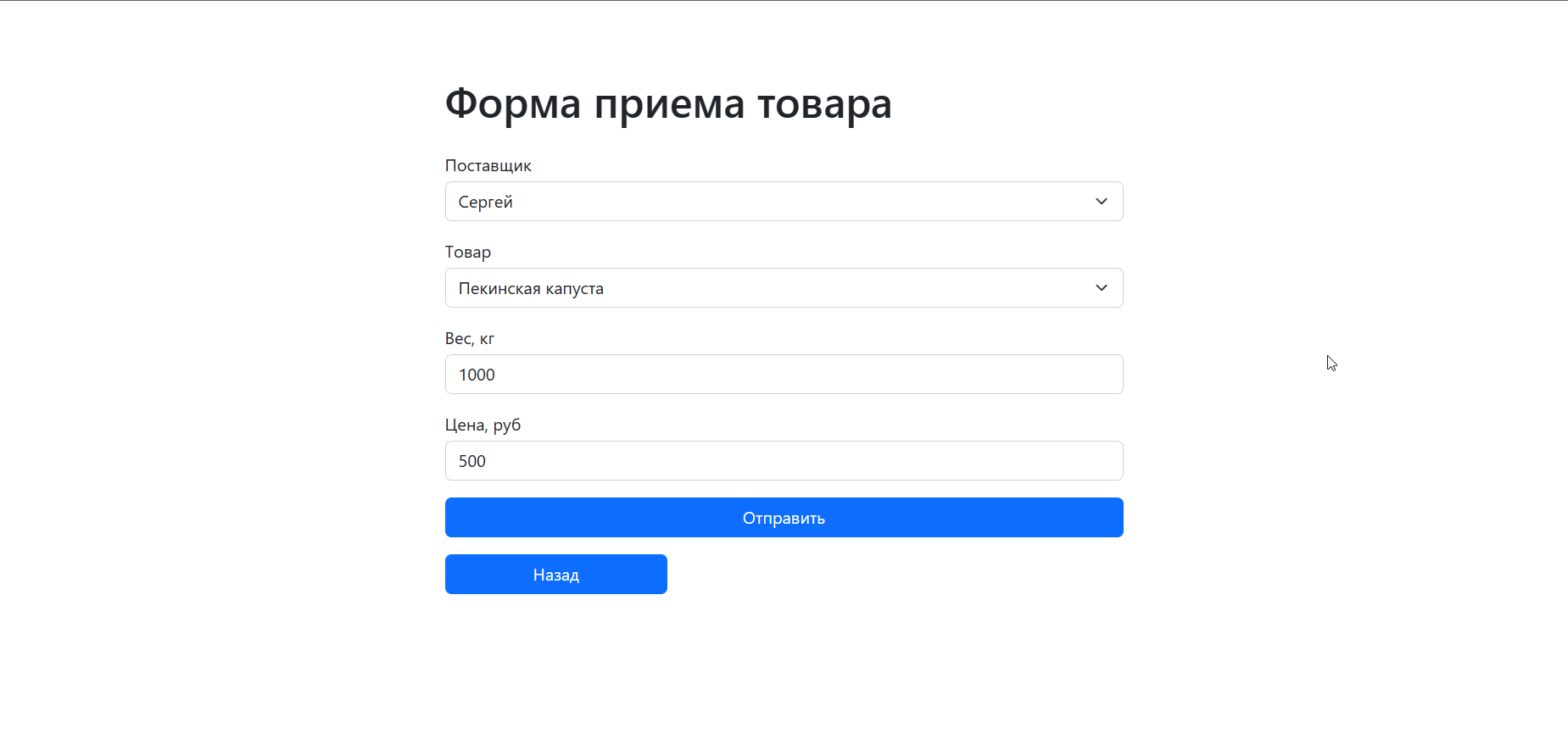
  
Рисунок 4.27 – Удаление поставщика

Мы видим тут список с поставщиками, выберем “Артём” и нажмём на кнопку удалить.

  
Рисунок 4.28 – Удаление поставщика

И как мы можем видеть, поставщик пропал.

У нас имеется новый добавленный вид товара и новый поставщик, давайте попытаемся закупить этот вид, от этого поставщика

  
Рисунок 4.29 – Демонстрация с новыми данными

Как мы видим, мы можем с ними взаимодействовать, далее нажмём кнопку “Отправить”

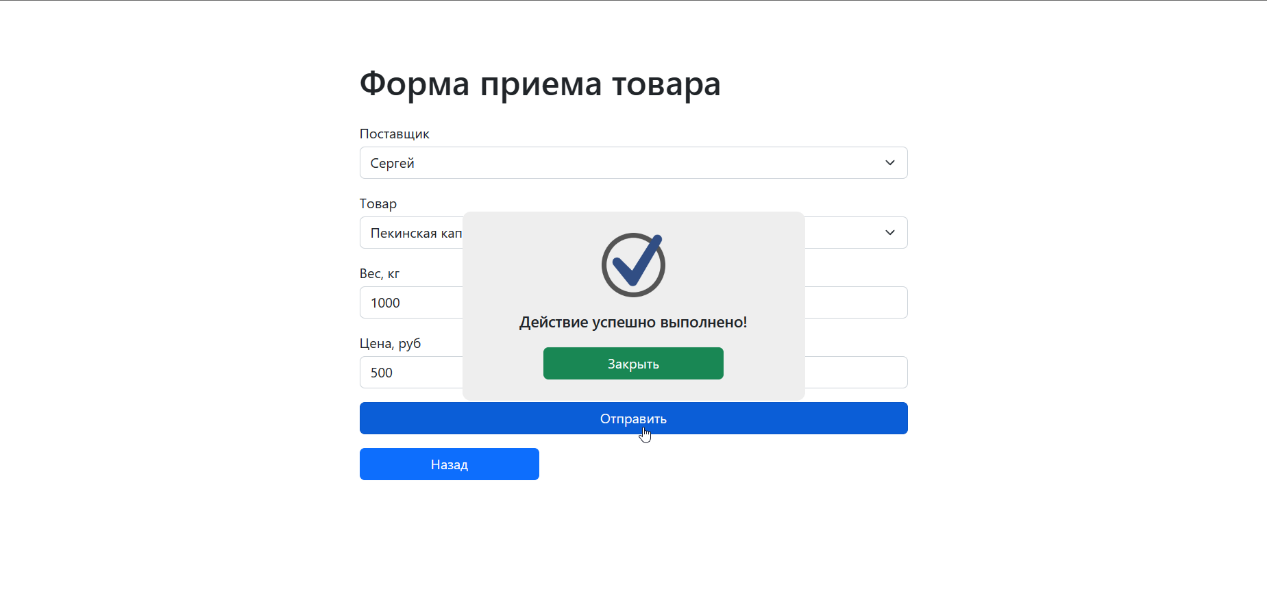


Рисунок 4.30 – Успешное добавление с новыми данными

И теперь зайдём во вкладку со всем нашим товаром.

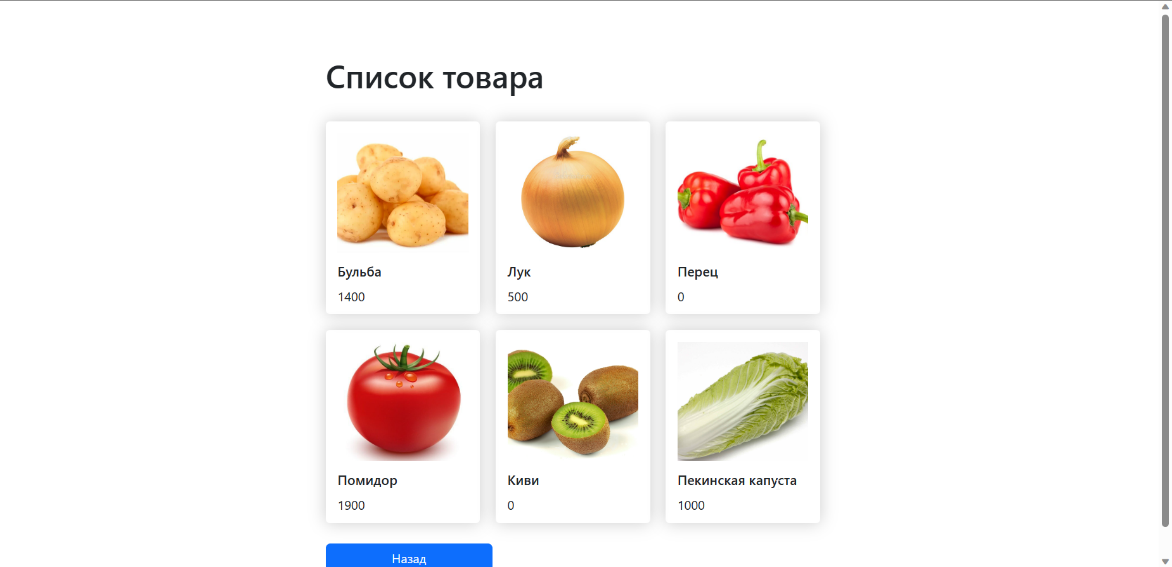


Рисунок 4.31 – Изменения на складе

Число товара увеличилось на складе, что и следовало ожидать.

Далее попытаемся поменять пароль для входа в аккаунт администратора.

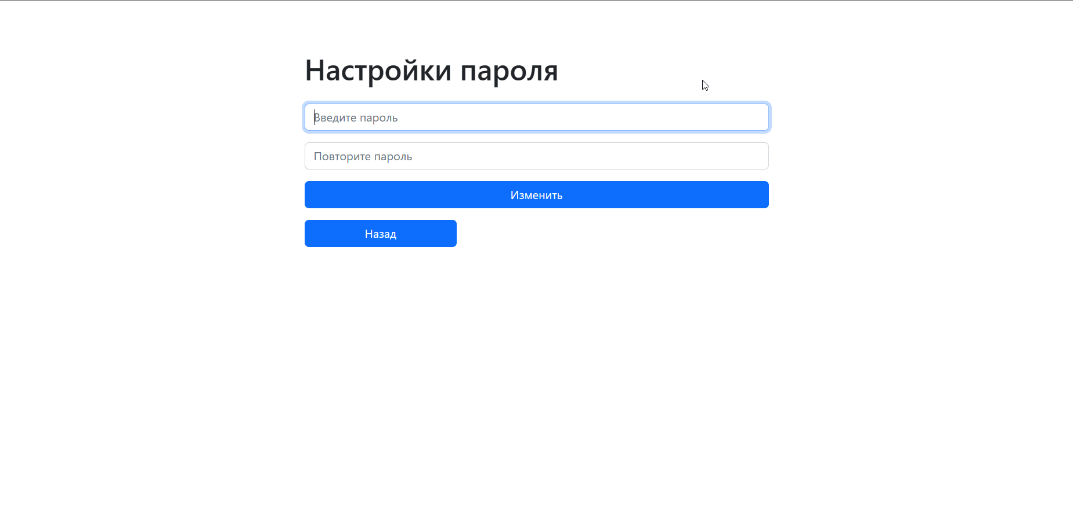
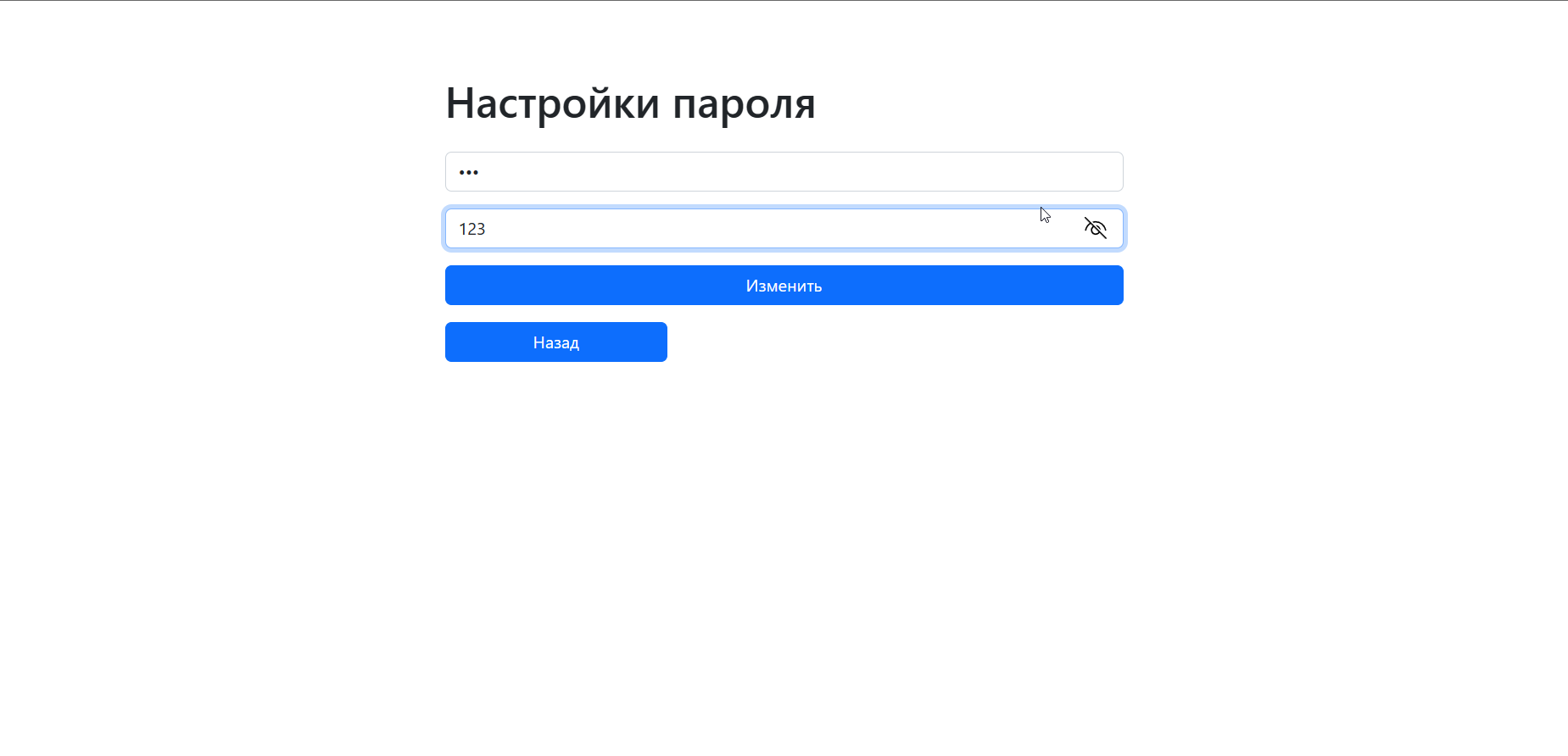


Рисунок 4.32 – Изменения пароля

Мы видим стандартную форму, для таких случаев. Давай попробуем ввести два разных пароля в поля формы. К примеру “321” и “123”.

  
Рисунок 4.33 – Разные пароли

И после чего нажмём кнопку “Изменить”. И ничего не будет происходить, так как вы ввели разные пароли. А теперь, давайте сменим пароль с “123” на “444”.

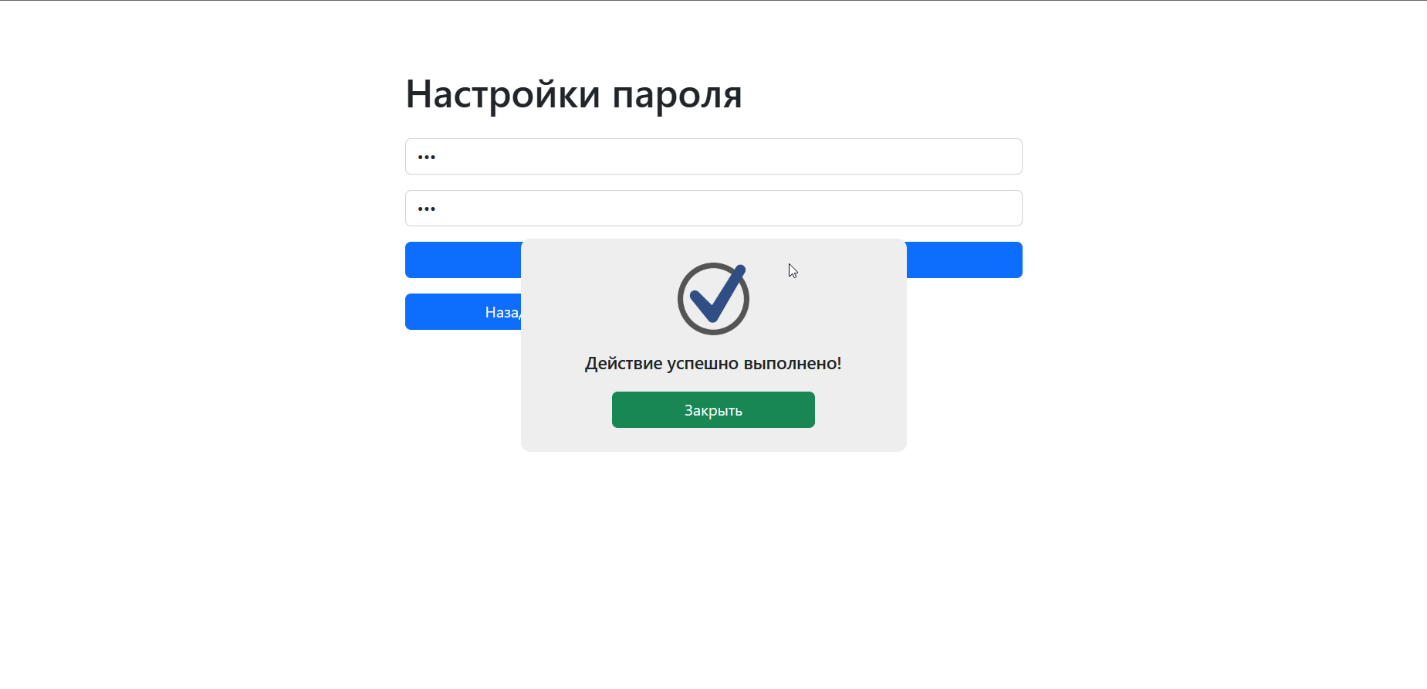


Рисунок 4.34 – Успешная смена пароля

После чего мы получаем сообщение, что всё прошло успешно. Выйдем из аккаунта администратора и попытаемся войти под старым паролем.

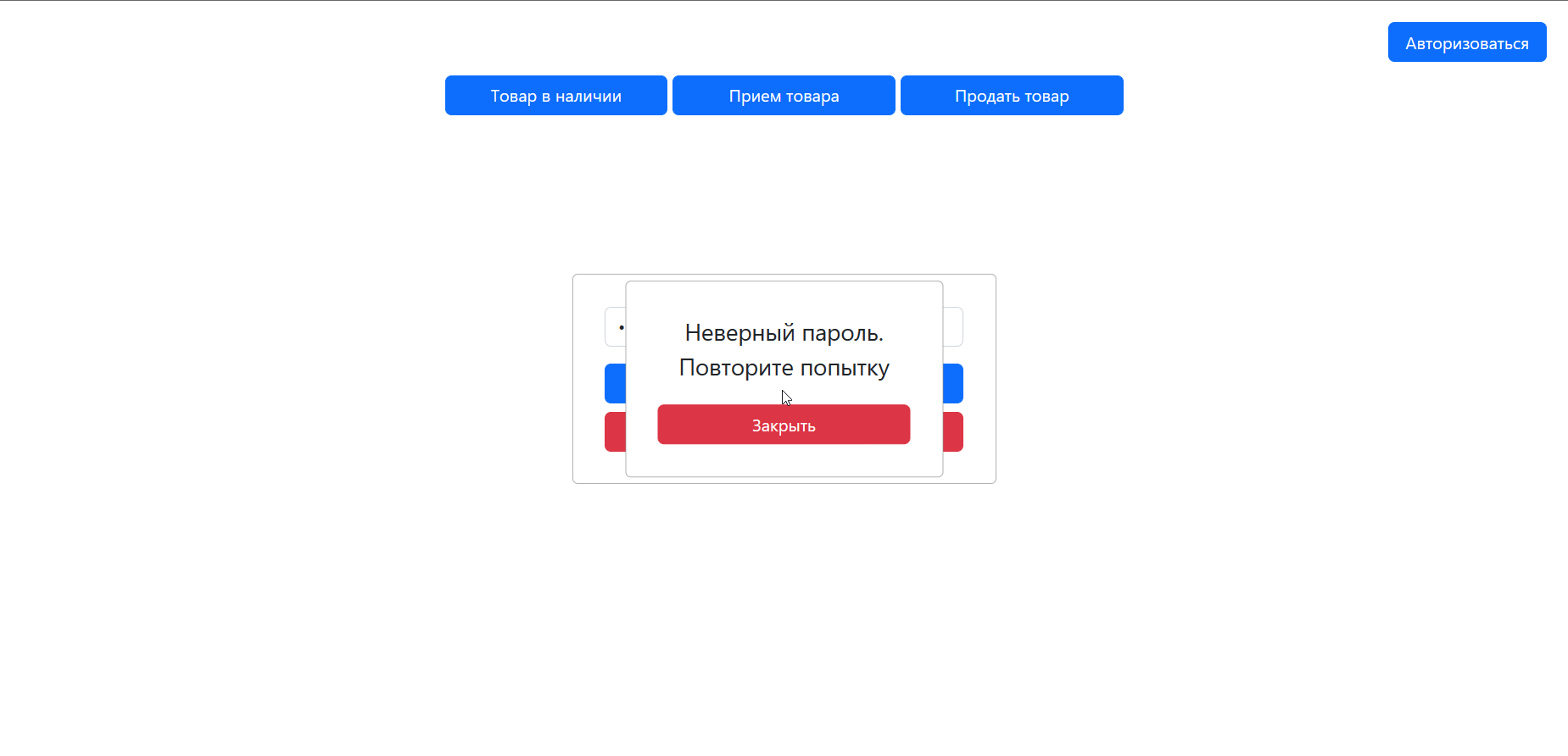


Рисунок 4.35 – Старый пароль

И получаем ошибку, а при вводе нового, мы уже сможем зайти в аккаунт

## **4.2 Вывод по разделу**

В данном разделе было проведено функциональное тестирование разработанного веб-приложения в десктопной версиях. Были протестированы все компоненты программы и каждый компонент отвечает своим требованиям. По итогам тестирования было выяснено, что проект соответствует ранее заданным требованиям и работает корректно.