

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Факультет Информационных технологий

Кафедра Информатики и информационных технологий

направление подготовки

09.03.02 «Информационные системы и технологии»

КУРСОВОЙ ПРОЕКТ

Дисциплина: Базы Данных

Тема: Разработка интернет-магазина с использованием базы данных

Выполнил: студент группы 221-3711

__Лапин Сергей Витальевич

(Фамилия И.О.)

Дата, подпись 26.09.2024

(Дата) (Подпись)

Проверил: _____

(Фамилия И.О., степень, звание)

Дата, подпись _____

(Дата) (Подпись)

Замечания: _____

Москва

2024

Содержание

| | |
|---------------------------------------|----------|
| ВВЕДЕНИЕ..... | 3 |
| 1. ГЛАВА 1 Проектирование..... | 6 |
| 1.1.Описание предметной области..... | 6 |
| 1.2.Выбор стека технологий..... | 6 |
| 2. ГЛАВА 2 Разработка..... | 9 |
| 2.1.База данных..... | 9 |
| 2.2.Серверная часть (Back-End)..... | 12 |
| 2.3.Клиентская часть (Front-End)..... | 19 |
| ЗАКЛЮЧЕНИЕ..... | 36 |
| СПИСОК ЛИТЕРАТУРЫ..... | 37 |
| ПРИЛОЖЕНИЯ..... | 38 |

Введение

В рамках дисциплины "Базы данных" была выполнена курсовая работа на тему "Разработка интернет-магазина с использованием базы данных". Основной целью данной работы являлось создание прототипа интернет-магазина, который включает в себя основные компоненты и функции современного онлайн-магазина, такие как каталог товаров, корзина покупок, система регистрации и авторизации пользователей, а также управление товарами для администратора.

В современном мире электронная коммерция играет ключевую роль в развитии бизнеса, а интернет-магазины стали одной из самых популярных форм онлайн-торговли. Они предоставляют удобные платформы для взаимодействия между продавцами и покупателями, предлагая широкий ассортимент товаров и услуг. Основой эффективной работы любого интернет-магазина является правильно спроектированная база данных, которая обеспечивает хранение, обработку и управление информацией о товарах, заказах и клиентах.

Разработка интернет-магазина с использованием базы данных требует глубокого понимания принципов организации данных, их структурирования и оптимизации процессов взаимодействия с пользователем. В данной курсовой работе рассмотрен процесс проектирования и реализации интернет-магазина с использованием современных технологий баз данных.

Целью данной работы является разработка прототипа интернет-магазина, который включает в себя основные компоненты и функции современного онлайн-магазина, а также получение практических навыков работы с базами данных, веб-разработки и управления данными.

Разработка подобного прототипа даст возможность приобрести практические навыки работы с базами данных, а также познакомиться с

основами веб-разработки и управления данными.

Задачи:

1. Создание базы данных:

- Спроектировать структуру базы данных для хранения информации о пользователях, товарах, категориях товаров и корзинах покупок.
- Создать ER-диаграмму и определить связи между таблицами.

2. Создание серверной части приложения:

- Реализовать серверную платформу.
- Обеспечить обработку запросов от клиента и взаимодействие с базой данных.
- Реализовать функции регистрации и авторизации пользователей.
- Обеспечить возможность добавления, редактирования и удаления товаров для администратора.

3. Разработка клиентской части приложения:

- Разработать удобный интерфейс для взаимодействия пользователя с магазином
- Реализовать отображение каталога товаров, страницу товара, страницу пользователя, корзину покупок и оформление заказа.
- Обеспечить возможность загрузки и отображения изображений товаров.

4. Тестирование и интеграция

- Провести интеграцию клиентской и серверной частей.
- Провести тестирование функциональности интернет-магазина.
- Исправить выявленные ошибки и улучшить

производительность.

Глава 1 Проектирование

1.1 Описание предметной области

Создание интернет-магазина является одним из наиболее популярных и востребованных направлений в сфере веб-разработки. Интернет-магазины предоставляют пользователям возможность совершать покупки, не выходя из дома, и предлагают широкий ассортимент товаров и услуг.

Интернет-магазины представляют собой сложные информационные системы, включающие в себя множество компонентов, таких как каталог товаров, система управления пользователями, корзина покупок, система оплаты и многое другое. Эти компоненты взаимосвязаны и взаимодействуют друг с другом для обеспечения удобства покупок.

Одной из ключевых задач при разработке интернет-магазина является обеспечение правильного и надежного хранения данных. В данном проекте используются реляционные базы данных, которые позволяют эффективно организовать и управлять данными о пользователях, товарах и заказах.

1.2 Основные аспекты предметной области

Основными компонентами интернет-магазина являются:

- Каталог товаров - предоставляет пользователям информацию о доступных товарах, их характеристиках и ценах. В данном проекте используется база данных для хранения информации о товарах, включая их название, описание, категорию, цену и количество на складе.
- Корзина покупок - временное хранилище, в котором покупатели могут сохранять выбранные товары перед оформлением заказа. Корзина позволяет пользователям добавлять, удалять и изменять количество товаров.
- Система управления пользователями - включает функции

регистрации, авторизации и управления профилем пользователя. Важно обеспечить безопасность хранения данных пользователей, особенно паролей, используя методы хеширования.

- Административная панель - интерфейс для администраторов, позволяющий управлять каталогом товаров и обрабатывать заказы. Административная панель предоставляет доступ к статистике и отчетам, что помогает в принятии управленческих решений.

В рамках проекта выделяются две основные категории пользователей:

- Покупатели - пользователи, которые посещают интернет-магазин с целью выбора и покупки товаров. Для них предусмотрены функции регистрации, авторизации, просмотра каталога товаров, добавления товаров в корзину и оформления заказа.
- Администраторы - пользователи с расширенными правами доступа, отвечающие за управление ассортиментом товаров, обработку заказов и общую поддержку работы интернет-магазина. Администраторы могут добавлять, редактировать и удалять товары, а также просматривать статистику продаж.

1.2 Выбор стека технологий

Для разработки серверной части мной был выбран веб-фреймворк ASP.NET Core с версией .NET 8.0. Данный фреймворк был изучен мной в ходе освоения курса Back-end разработки и представляет собой удобный инструмент для создания кросс платформенных веб-приложений, работой с API и HTTP запросами. Так же для совмещения базы данных с работой серверной части бы установлен фреймворк Entity Framework Core — это ORM (Object-Relational Mapping) от Microsoft, который упрощает взаимодействие с базой данных путем автоматического сопоставления объектов в коде с таблицами в базе данных, что упрощает и систематизирует написания запросов к БД.

Для управления базой данных мной была выбрана СУБД MySQL т. к. она легко интегрируется с ASP NET Core.

Для разработки клиентской же части мой выбор был сделан в пользу библиотеки React, React — это JavaScript-библиотека для построения пользовательских интерфейсов, а в качестве языка программирования TypeScript — это язык программирования, расширяющий возможности JavaScript за счет введения статической типизации. Так же в проект были интегрированы 3 вспомогательные библиотеки:

1. Material-UI: Библиотека компонентов для React, обеспечивающая простое создание интерфейсов для более удобного написания и читаемости кода.
2. Axios: Библиотека для выполнения HTTP-запросов из клиентской части к серверу.
3. React Router: Библиотека для управления маршрутизацией в React-приложениях.

В качестве среды разработки была выбрана Microsoft Visual Studio.

Глава 2 Разработка

2.1 База данных

В данном разделе разберем созданную базу данных, таблицы, их атрибуты и связи между сущностями. Как уже было описано в качестве СУБД была выбрана MySQL, в MySQL Workbench была создана EER диаграмма для упрощения проектированной базы данных и настройки связей.

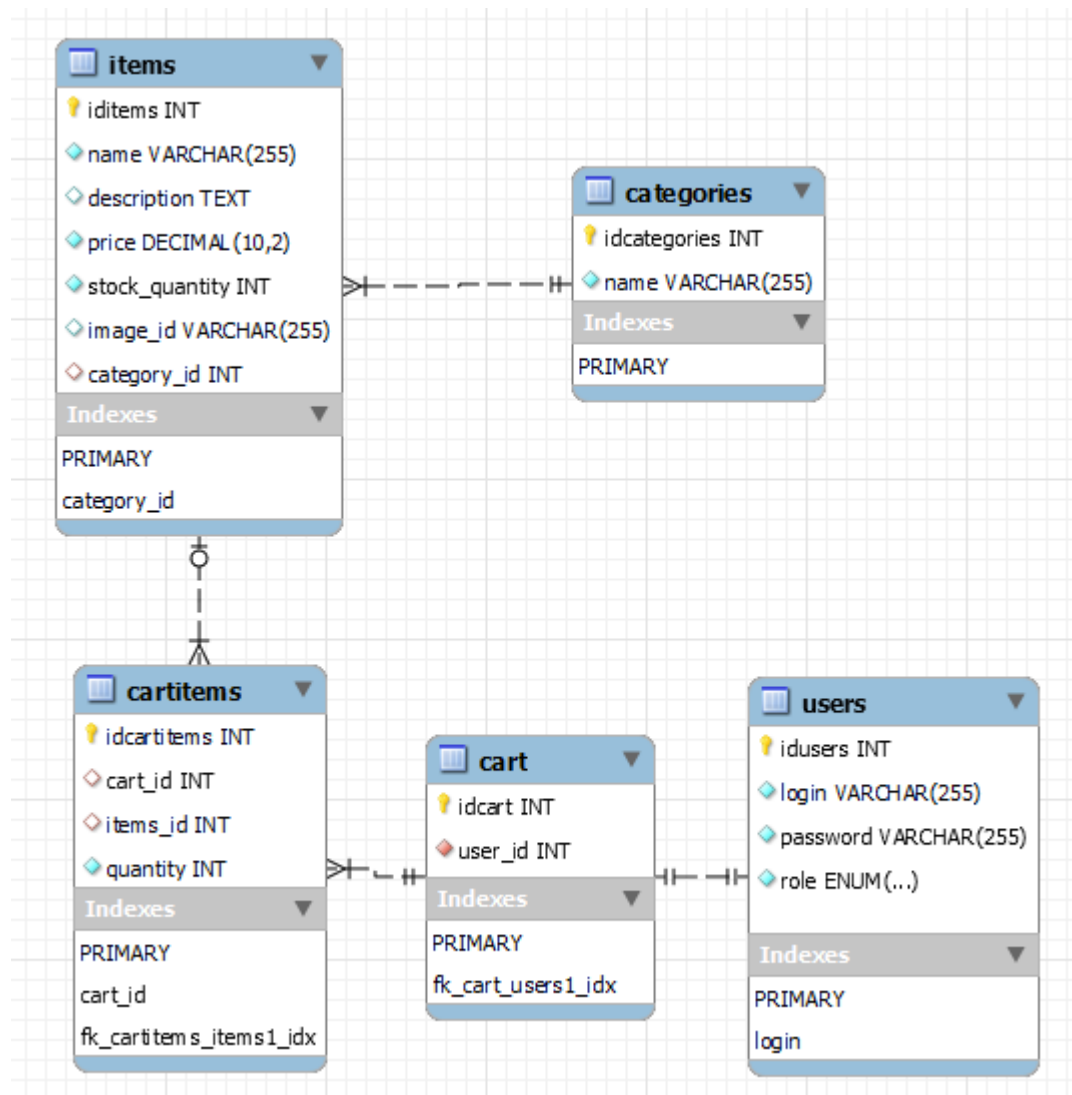


Рисунок 2.1 EER диаграмма базы данных

Данная диаграмма наглядно иллюстрирует структуру базы данных для хранения всех необходимых в разработке интернет-магазина данных. Разберем подробнее каждый элемент.

Таблица users хранит информацию о пользователях интернет-магазина. Она содержит следующие атрибуты:

1. idusers (INT): Первичный ключ, уникальный идентификатор пользователя.
2. login (VARCHAR(255)): Логин пользователя, используется для аутентификации.
3. password (VARCHAR(255)): Пароль пользователя, используется для аутентификации.
4. role (ENUM): Роль пользователя, может принимать значения 'admin' или 'user'.

Таблица categories хранит информацию о категориях товаров. Она содержит следующие атрибуты:

1. idcategories (INT): Первичный ключ, уникальный идентификатор категории.
2. name (VARCHAR(255)): Название категории.

Таблица items хранит информацию о товарах. Она содержит следующие атрибуты:

1. iditems (INT): Первичный ключ, уникальный идентификатор товара.
2. name (VARCHAR(255)): Название товара.
3. description (TEXT): Описание товара.
4. price (DECIMAL(10,2)): Цена товара.
5. stock_quantity (INT): Количество товара на складе.
6. image_id (VARCHAR(255)): Путь к изображению товара.
7. category_id (INT): Внешний ключ, указывающий на категорию товара.

Таблица cart хранит информацию о корзинах пользователей. Она содержит следующие атрибуты:

1. idcart (INT): Первичный ключ, уникальный идентификатор корзины.
2. user_id (INT): Внешний ключ, указывающий на пользователя, которому принадлежит корзина.

Таблица cartitems хранит информацию о товарах в корзинах. Она содержит следующие атрибуты:

1. idcartitems (INT): Первичный ключ, уникальный идентификатор записи в корзине.
2. cart_id (INT): Внешний ключ, указывающий на корзину.
3. item_id (INT): Внешний ключ, указывающий на товар.
4. quantity (INT): Количество товара в корзине.

Связи между таблицами определены следующий образом:

- Users и Carts: Один пользователь может иметь одну корзину. Связь осуществляется через атрибут user_id в таблице cart, который является внешним ключом, ссылающимся на атрибут idusers в таблице users.
- Categories и Items: Одна категория может включать множество товаров. Связь осуществляется через атрибут category_id в таблице items, который является внешним ключом, ссылающимся на атрибут idcategories в таблице categories.
- Carts и CartItems: Одна корзина может содержать множество записей о товарах. Связь осуществляется через атрибут cart_id в таблице cartitems, который является внешним ключом, ссылающимся на атрибут idcart в таблице cart.
- Items и CartItems: Один товар может быть включен во множество записей корзины. Связь осуществляется через атрибут item_id в таблице cartitems, который является внешним ключом, ссылающимся на атрибут iditems в таблице items.

Данная структура предоставляет собой хорошую основу для разработки прототипа интернет-магазина, имеющего все типичные для такой системы функции.

2.2 Серверная часть (Back-End)

После создания базы данных была проведена работа по разработке серверной части приложения.

2.2.1 Контекст базы данных и его составляющие

Первым шагом после настройки среды разработки и установки необходимых пакетов через NuGet, был создан контекст базы данных. Контекст базы данных в приложении ASP.NET Core с использованием Entity Framework Core является центральным компонентом, который управляет подключением к базе данных и предоставляет API для выполнения операций с данными. Он представляет собой производный класс от DbContext, содержащий наборы сущностей, соответствующие таблицам в базе данных. Для этого создаем модели данных, повторяющие структуру данных для каждой таблицы в базе данных.

Листинг 2.2.1.1 Модель данных users

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Back_end.Models
{
    [Table("Users")]
    public class User
    {
        [Key]
        [Column("idusers")]
        public int Id { get; set; }

        [Required]
        [Column("login")]
        public string Login { get; set; }

        [Required]
        [Column("password")]
        public string Password { get; set; }

        [Required]
        [Column("role")]
        public string Role { get; set; }
    }
}
```

Данный фрагмент демонстрирует общую структуру реализации

моделей данных для создания контекста БД. Поля класса, их типы данных и название класса должны в точности совпадать с названиями и типами в таблице для автоматической связи, но в нашем случае если использовать названия отличные от тех, что находятся в БД, как в нашем случае для удобства написания некоторых из них, нужно использовать атрибуты `Table` и `Column` для точного указания к какой таблице и атрибуту относится то или иное поле, при этом типы данных должны все так же совпадать.

Общая структура каталога `Models` выглядит следующим образом:

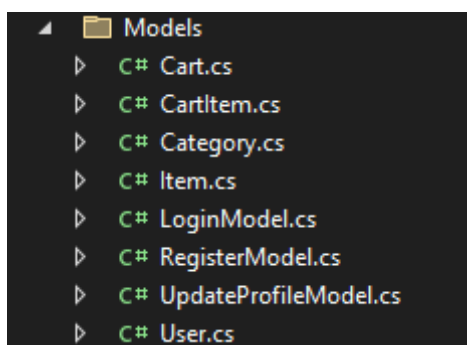


Рисунок 2.2.1 Общая структура каталога `Models`

В нем представлены модели для каждой сущности, а также 2 вспомогательные, используемые в контроллерах, для регистрации и обновления информации в профиле, чтобы ограничить прямое взаимодействие с моделью, содержащей полную информацию о пользователе.

Теперь создаем сам контекст базы данных, контекст создается в отдельном скрипте, с классом `ShopContext`, который наследуется от класса `DbContext`.

Листинг 2.2.1.2 Класс `ShopContext`

```
using Back_end.Models;
using Microsoft.EntityFrameworkCore;

public class ShopContext : DbContext
{
    public DbSet<User> Users { get; set; }
    public DbSet<Category> Categories { get; set; }
    public DbSet<Item> Items { get; set; }
    public DbSet<Cart> Carts { get; set; }
    public DbSet<CartItem> CartItems { get; set; }

    public ShopContext(DbContextOptions<ShopContext> options) : base(options) { }
```

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);
}
}
```

В контексте указываются все модели, относящиеся к базе данных для добавления в него, а в конструкторе указываются параметры, так как мы настроили их в файле Program (а именно подключение и версию MySQL), то здесь указываем их как options.

2.2.2 Подключение к базе данных

Подключение к базе данных можно настроить разными способами, но в нашем случае добавим его в классе Program, для этого указывается сервер, порт, имя пользователя, имя БД и пароль подключения.

Листинг 2.2.2 Подключение к базе данных

```
builder.Services.AddDbContext<ShopContext>(options =>
    options.UseMySQL("Server=127.0.0.1;Port=3306;Database=shop;Uid=root;Pwd=SQL_Root;",
        new MySqlServerVersion(new Version(8, 0, 36))));
```

2.2.3 Контроллеры

После создания основы для работы с базой данных, были созданы контроллеры. Созданные API контроллеры будут служить для получения и обработки запросов с клиентской стороны приложения, всего их 5, каждый имеет свою четкую функцию: Работа с данными пользователя, данными товаров, корзиной, изображениями и категориями товаров. Рассмотрим их функционал.

ItemsController содержит большое количество запросов, полный код содержится в приложении А. Рассмотрим все типы запросов, которые способен обработать данный контроллер и их принцип работы.

Получение списка всех товаров (GET: api/Items): Возвращает список всех товаров из таблицы Items базы данных. Метод асинхронно выполняет запрос к базе данных и возвращает все записи в виде списка.

Получение товара по идентификатору (GET: api/Items/{id}): Возвращает данные о конкретном товаре, используя его идентификатор. Если

товар не найден, возвращается статус 404 (Not Found). Метод асинхронно ищет товар в базе данных по идентификатору.

Добавление нового товара (POST: `api/Items`): Принимает данные нового товара и добавляет его в таблицу `Items` базы данных. Проверяет валидность модели, сохраняет изменения в базе данных и возвращает созданный товар с HTTP статусом 201 (Created) и ссылкой на созданный товар.

Обновление существующего товара (PUT: `api/Items/{id}`): Принимает идентификатор товара и обновленные данные товара. Проверяет, что идентификатор в запросе совпадает с идентификатором в данных товара. Обновляет данные в базе данных, сохраняет изменения. Если товар не найден, возвращает статус 404 (Not Found). В случае успешного обновления возвращает статус 204 (No Content).

Удаление товара (DELETE: `api/Items/{id}`): Принимает идентификатор товара и удаляет его из таблицы `Items` базы данных. Если товар не найден, возвращает статус 404 (Not Found). Удаляет товар из базы данных, сохраняет изменения и возвращает статус 204 (No Content).

Проверка существования товара (приватный метод): Проверяет наличие товара с заданным идентификатором в таблице `Items`. Возвращает `true`, если товар существует, и `false` в противном случае. Используется в методе `UpdateItem` для проверки существования товара перед обновлением данных.

Таким образом данный контроллер позволяет осуществлять CRUD (create, read, upload, delete) операции с товарами из базы данных.

Самый мало функциональный из всех, это `CategoriesController`, отвечающий за передачу списка всех категорий товаров, чтобы использовать его для на стороне клиента для отображения правильной категории товара или присвоения во время создания нового. Данный Get запрос выведен в отдельный контроллер, чтобы разгрузить довольно объемный `ItemsController`.

Листинг 2.2.3.1 `CategoriesController`

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Threading.Tasks;
using Back_end.Models;

namespace Back_end.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class CategoriesController : ControllerBase
    {
        private readonly ShopContext _context;

        public CategoriesController(ShopContext context)
        {
            _context = context;
        }

        // GET: api/Categories
        [HttpGet]
        public async Task<ActionResult<IEnumerable<Category>>> GetCategories()
        {
            return await _context.Categories.ToListAsync();
        }
    }
}

```

Следующим идет AccountController, который осуществляет контроль над учетной записью пользователя, авторизацию, редактирование данных, вход и т.д., полный код представлен в приложении Б, а ниже рассмотрим все возможные действия:

Регистрация нового пользователя (POST: api/Account/Register): Принимает данные для регистрации нового пользователя. Проверяет валидность модели и наличие пользователя с таким же логином. Если пользователь уже существует, возвращает статус 409 (Conflict). Если данные валидны, создает нового пользователя и сохраняет его в базе данных и возвращает сообщение об успешной регистрации и идентификатор пользователя.

Вход пользователя (POST: api/Account/Login): Принимает данные для входа пользователя. Проверяет валидность модели и наличие пользователя с указанным логином и паролем. Если пользователь не найден или данные неверны, возвращает статус 401 (Unauthorized) и логирует предупреждение. Если данные валидны, возвращает роль и идентификатор пользователя.

Выход пользователя (POST: api/Account/Logout): Обработывает запрос

на выход пользователя. Логирует успешный выход и возвращает статус 200 (OK) с сообщением об успешном выходе.

Обновление профиля пользователя (PUT: api/Account/{id}): Принимает идентификатор пользователя и данные для обновления профиля. Проверяет валидность модели и наличие пользователя с указанным идентификатором. Если пользователь не найден, возвращает статус 404 (Not Found). Если данные валидны, обновляет логин и пароль пользователя в базе данных, сохраняет изменения и возвращает сообщение об успешном обновлении профиля.

Далее рассмотрим контроллер для работы с изображениями, изображения для товаров, хранятся в папке wwwroot, откуда легко можно получить изображения для фронтэнд части, при обновлении или добавлении товара, данный контроллер сохраняет путь к изображению для дальнейшего внесения его в базу данных, откуда его сможет использовать React приложение.

Листинг 2.2.3.2 ImageUploadController

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System.IO;
using System.Threading.Tasks;

namespace Back_end.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ImageUploadController : ControllerBase
    {
        [HttpPost]
        public async Task<IActionResult> Upload(IFormFile file)
        {
            if (file == null || file.Length == 0)
                return BadRequest("No file uploaded");

            var filePath = Path.Combine("wwwroot/images", file.FileName);

            using (var stream = new FileStream(filePath, FileMode.Create))
            {
                await file.CopyToAsync(stream);
            }

            var relativePath = $"/images/{file.FileName}";
            return Ok(new { filePath = relativePath });
        }
    }
}
```

Последним контроллером, завершающим функциональность бэкенда, является контроллер управления корзиной пользователя, его код представлен в приложении В, а функциональность каждого запроса описана ниже:

Получение всех товаров в корзине пользователя (GET:api/Cart/{userId}): Возвращает список всех товаров, добавленных в корзину указанного пользователя. Асинхронно выполняет запрос к базе данных для получения корзины пользователя и всех товаров в этой корзине. Если корзина не найдена, возвращает пустой список.

Добавление товара в корзину (POST: api/Cart/{itemId}): Добавляет указанный товар в корзину пользователя. Идентификатор пользователя извлекается из заголовка авторизации. Если корзины у пользователя нет, она создается. Если товар уже есть в корзине, его количество увеличивается на 1, иначе товар добавляется в корзину с количеством 1.

Удаление товара из корзины (DELETE: api/Cart/{itemId}): Удаляет указанный товар из корзины пользователя. Идентификатор пользователя извлекается из заголовка авторизации. Если корзина или товар в корзине не найдены, возвращается соответствующий статус ошибки.

Очистка корзины (DELETE: api/Cart/clear/{userId}): Удаляет все товары из корзины указанного пользователя. Если корзина не найдена, возвращает

статус 404 (Not Found). Если корзина найдена, все товары из неё удаляются, и корзина очищается.

Оформление заказа (POST: `api/Cart/checkout`): Обработывает оформление заказа для пользователя. Идентификатор пользователя извлекается из заголовка авторизации. Проверяет наличие корзины и достаточность каждого товара в корзине. Если товара недостаточно на складе, возвращает сообщение об ошибке. Если все проверки пройдены, уменьшает количество каждого товара на складе на количество, указанное в корзине, очищает корзину и возвращает сообщение об успешном оформлении заказа.

Таким образом были выполнены все задачи, относящиеся к бэкэнд части, приложение теперь имеет действия для обработки всех необходимых для функционирования запросов.

2.3 Клиентская часть (Front-End)

Завершающей частью разработки, стало создание react-приложения для клиентской части интернет-магазина, в ней были реализованы все необходимые страницы, запросы к серверу, а также пользовательский интерфейс.

Клиентская часть взаимодействует с серверной частью через API-запросы, выполненные с использованием библиотеки `axios`. Это позволяет эффективно управлять данными и обеспечивать обновление интерфейса в реальном времени. Приложение разделено на модули, включающие в себя страницы и отдельные компоненты, такие как формы для ввода, хэдеры и компоненты уведомлений. Они разделены по каталогам `pages` и `components`.

Компонент `App` является основной точкой входа в React-приложение. Полный код данного компонента содержится в приложении Е. `App` использует библиотеку `react-router-dom` для управления маршрутизацией и отображения различных страниц в зависимости от роли пользователя. В него импортируются необходимые компоненты и страницы, которые будут

использоваться для отображения содержимого и навигации. Далее используется хук `useState` для хранения роли пользователя и хук `useEffect` для установки роли пользователя при загрузке приложения. Роль пользователя так же сохранена в локальном хранилище. После чего идет настройка маршрутизации, которая в зависимости от роли пользователя позволяет заходить на определенные страницы и отображает соответствующий хэдер для навигации. Хэдеров всего 3 типа, для неавторизованного пользователя (только название сайта), для пользователя (каталог, корзина, профиль, выход) и для администратора (редактирование, добавление, выход).

Перейдем к авторизации, за данный процесс отвечают несколько компонентов, первый из них это `auth.ts` он содержит 2 функции `POST` для регистрации и входа с использованием `axios`.

Листинг 2.3.1 Компонент авторизации `auth.tsx`

```
import axios from 'axios';

const API_URL = 'https://localhost:7009/api/Account';

interface AuthResponse {
  message: string;
  role?: string;
  userId?: string;
}

export const register_ = async (login: string, password: string, role: string): Promise<AuthResponse> => {
  const response = await axios.post<AuthResponse>(`${API_URL}/Register`, { login, password, role });
  return response.data;
};

export const login_ = async (login: string, password: string): Promise<AuthResponse> => {
  const response = await axios.post<AuthResponse>(`${API_URL}/Login`, { login, password });
  return response.data;
};
```

Далее рассмотрим компонент `LoginPage` представляющий страницу для входа, этот компонент отображает формы для ввода данных и вызывает функцию входа из `auth` для аутентификации при отправке формы (`handleLogin`). В случае положительного ответа от сервера, айди пользователя сохраняется в локальном хранилище, чтобы затем корректно синхронизировать пользователя и его корзину. Так же в зависимости от роли, используется перенаправление на страницы администратора, где дается инструментарий администратора (управление товарами) или пользователя (интерфейс

покупок и просмотра).

Листинг 2.3.2 Функции страницы входа

```
import React, { useState } from 'react';
import { useNavigate, Link } from 'react-router-dom';
import { Button, TextField, Box, Typography, Container } from '@mui/material';
import { login_ } from '../api/auth';

interface LoginPageProps {
  onLogin: (role: string) => void;
}

const LoginPage: React.FC<LoginPageProps> = ({ onLogin }) => {
  const [login, setLogin] = useState("");
  const [password, setPassword] = useState("");
  const navigate = useNavigate();

  const handleLogin = async () => {
    try {
      const response = await login_(login, password);
      const role = response.role;
      const userId = response.userId;
      if (role && userId) {
        localStorage.setItem('userRole', role);
        localStorage.setItem('userId', userId);
        onLogin(role);
        if (role === 'admin') {
          navigate('/items');
        } else if (role === 'user') {
          navigate('/user-items');
        }
      } else {
        console.error('No role or userId received from the server');
      }
    } catch (error) {
      console.error('Login failed', error);
    }
  };
};
```

Визуальное отображение данной страницы в свою очередь выполнено с использованием Material-UI, похожая структура используется на многих страницах, таких как регистрация, добавление или редактирование товара, но со своими специфическими полями ввода и другими элементами интерфейса.

Листинг 2.3.3 Визуальная составляющая страницы входа

```
return (
  <Container component="main" maxWidth="xs">
    <Box
      sx={{
        marginTop: 8,
        display: 'flex',
        flexDirection: 'column',
        alignItems: 'center',
      }}
    >
      <Typography component="h1" variant="h5">
        Вход
      </Typography>
      <Box component="form" noValidate sx={{ mt: 1 }}>
```

```

<TextField
  margin="normal"
  required
  fullWidth
  label="Логин"
  name="login"
  autoComplete="login"
  autoFocus
  value={login}
  onChange={(e) => setLogin(e.target.value)}
/>
<TextField
  margin="normal"
  required
  fullWidth
  name="password"
  label="Пароль"
  type="password"
  autoComplete="current-password"
  value={password}
  onChange={(e) => setPassword(e.target.value)}
/>
<Button
  fullWidth
  variant="contained"
  sx={{ mt: 3, mb: 2 }}
  onClick={handleLogin}
>
  Войти
</Button>
<Link to="/register">
  <Button
    fullWidth
    variant="outlined"
    sx={{ mt: 1 }}
  >
    Зарегистрироваться
  </Button>
</Link>
</Box>
</Box>
</Container>
);

```

Для регистрации страница и структура похожа, с несколькими отличиями, данные, введенные пользователем так же отправляются на сервер через auth после чего если процесс регистрации успешен, пользователь перенаправляется обратно на страницу входа. Полный код данного компонента содержится в приложении Ж.

Визуал страниц входа и регистрации у запущенного приложения представлен ниже.

The screenshot shows the login page of a website titled "КиберМагазин". At the top, there is a blue header with the site name. Below the header, the word "Вход" (Login) is centered. There are two input fields: the first is labeled "Логин*" (Login*) and contains the text "admin"; the second is labeled "Пароль*" (Password*) and contains six dots. Below these fields are two buttons: a blue button labeled "ВОЙТИ" (Login) and a white button with a blue border labeled "ЗАРЕГИСТРИРОВАТЬСЯ" (Register).

Рисунок 2.3.1 Страница входа

The screenshot shows the registration page of the same website, "КиберМагазин". The header is identical. Below it, the word "Регистрация" (Registration) is centered. There are three input fields: the first is labeled "Логин*" (Login*) and contains "user"; the second is labeled "Пароль*" (Password*) and contains six dots; the third is labeled "Тип аккаунта*" (Account type*) and is a dropdown menu currently showing "Покупатель" (Buyer). Below these fields is a blue button labeled "ЗАРЕГИСТРИРОВАТЬСЯ" (Register).

Рисунок 2.3.2 Страница регистрации

После авторизации пользователь или администратор попадают на страницу со списком всех товаров, пользователь может перейти на более подробную страницу товара или добавить в корзину, а администратор может удалять или редактировать товары, а также добавить новый.

Отображение списка товаров воспроизводится путем запроса на получение данных о них, на подобных страницах алгоритм схож.

Рассмотрим получение информации о товарах со страницы каталога у

пользователя, полный код данного компонента содержится в приложении Г.

Для начала, используем хук `useState` для создания состояний, необходимых для хранения данных о товарах, категориях и уведомлениях. Состояния `items` и `categories` будут хранить данные о товарах и категориях соответственно, а состояние `notification` будет использоваться для управления уведомлениями, которые отображаются пользователю.

Для получения информации о товарах и категориях используются два хука `useEffect`. Первый хук выполняется при монтировании компонента и вызывает асинхронную функцию `fetchItems`, которая отправляет GET-запрос к API `https://localhost:7009/api/Items`. После успешного выполнения запроса данные о товарах сохраняются в состояние `items`. Второй хук также выполняется при монтировании компонента и вызывает функцию `fetchCategories`, которая отправляет GET-запрос к API `https://localhost:7009/api/Categories`. Полученные данные о категориях сохраняются в состояние `categories`.

Функция `fetchItems` отвечает за выполнение асинхронного запроса к серверу для получения списка товаров. Аналогично, функция `fetchCategories` выполняет запрос для получения списка категорий. Оба этих запроса выполняются один раз при загрузке компонента благодаря хукам `useEffect`.

После получения данных о товарах и категориях, компонент `UserItemsListPage` отображает информацию с использованием компонентов из библиотеки `Material-UI`. Товары отображаются в виде сетки (`Grid`), где каждый товар представлен в карточке (`Card`). Карточка товара включает в себя изображение товара (`CardMedia`), название, описание, категорию и цену (`CardContent`), а также кнопку для добавления товара в корзину. Теперь рассмотрим функцию добавления в корзину, так как она одинакова, что для общей страницы так и конкретного товара. Для добавления товара в корзину используется функция `addToCart`. При нажатии на кнопку "Купить", функция проверяет наличие `userId` в `localStorage`, чтобы убедиться, что пользователь

авторизован. Если пользователь не авторизован, отображается уведомление с просьбой войти в систему. Если пользователь авторизован, отправляется POST-запрос к API `https://localhost:7009/api/Cart/{itemId}`, чтобы добавить товар в корзину. В случае успешного добавления товара в корзину, отображается уведомление о том, что товар был успешно добавлен. Функция `handleNotificationClose` используется для закрытия уведомлений. Она обновляет состояние `notification`, чтобы закрыть компонент `Snackbar`, который отображает уведомления пользователю.

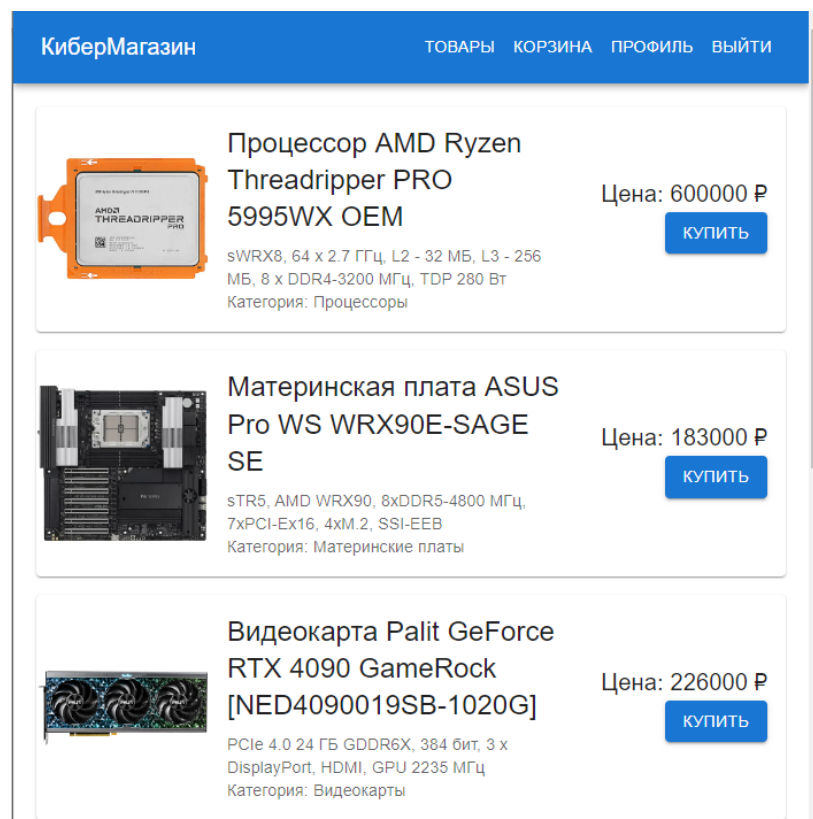


Рисунок 2.3.3 Страница каталога товаров для покупателя

Так же с данной страницы можно перейти на детальную страницу конкретного товара, на которой будет указано еще и количество выбранного товара, переход осуществляется по клику на карточку товара с помощью `Link`: Компонент `Link` оборачивает информацию о товаре, включая название, описание и категорию. Он создает ссылку на страницу детального просмотра товара, используя путь `/item/${item.id}`.

Путь `/item/${item.id}` включает идентификатор товара (`item.id`). Это

позволяет передать уникальный идентификатор товара в URL, что затем используется для получения данных о конкретном товаре на странице детального просмотра.

На странице детального просмотра товара используется компонент `useParams` из `react-router-dom` для получения параметра `id` из URL. Этот идентификатор используется для выполнения запроса к серверу и получения данных о конкретном товаре.

Листинг 2.3.4 Функция получения информации о конкретном товаре

```
const ItemDetailPage: React.FC = () => {
  const { id } = useParams<{ id: string }>();
  const [item, setItem] = useState<Item | null>(null);
  const [categories, setCategories] = useState<Category[]>([]);
  const [notification, setNotification] = useState({ open: false, message: '', severity: 'success' as 'success' | 'error' | 'warning' | 'info' });

  useEffect(() => {
    const fetchItem = async () => {
      const response = await axios.get(`https://localhost:7009/api/Items/${id}`);
      setItem(response.data);
    };

    const fetchCategories = async () => {
      const response = await axios.get('https://localhost:7009/api/Categories');
      setCategories(response.data);
    };

    fetchItem();
    fetchCategories();
  }, [id]);
```

Данный фрагмент работает по следующей схеме:

- `useParams` извлекает `id` из параметров URL.
- `useEffect` выполняет запрос к серверу, используя идентификатор товара для получения данных о конкретном товаре.
- Данные о товаре сохраняются в состояние `item`, которое затем используется для отображения информации на странице.

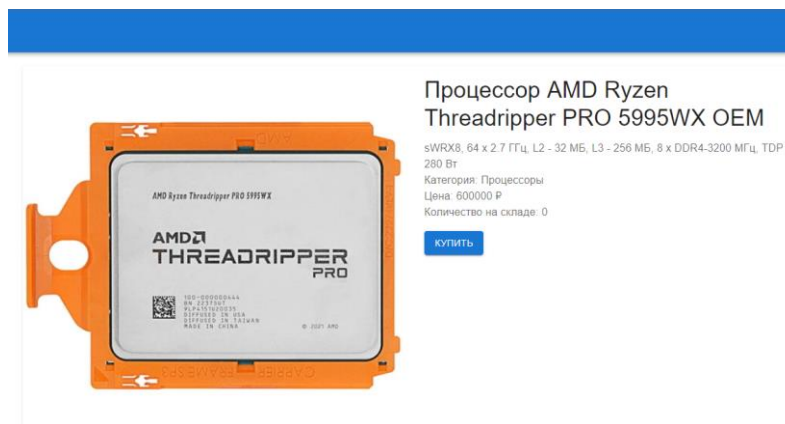


Рисунок 2.3.4 Страница товара

Теперь перейдем к корзине, она является одним из завершающих компонентов пользовательской функциональности интернет-магазина. Полный код страницы корзины содержится в приложении 3.

При загрузке компонента с помощью `useEffect` вызывается функция `fetchCart`, которая выполняет запрос к серверу для получения товаров в корзине текущего пользователя по его `userId`.

Функция `fetchCart`:

- Запрашивает идентификатор пользователя из `localStorage`.
- Выполняет GET-запрос к серверу по адресу `https://localhost:7009/api/Cart/${userId}`.
- Если запрос успешен, обновляет состояние `cartItems`.

Далее идут 2 функции, удаление одного типа товара из корзины или полная очистка товаров, работают они сходным образом отправляя DELETE запрос к контроллеру управления корзиной, передавая сначала айди пользователя, а затем айди предмета, или сразу удаляет объекты по айди пользователя.

Последняя функция корзины — это оформление заказа. `handleOrder` проверяет наличие товаров в корзине и выполняет POST-запрос на сервер для оформления заказа. Если товары успешно заказаны, корзина очищается и выводится сообщение об успешном оформлении. В случае недостаточного количества товаров на складе или других ошибок выводится

соответствующее сообщение.

Сами товары же расположены по сетке в виде своих карточек с кнопками для удаления их из корзины.

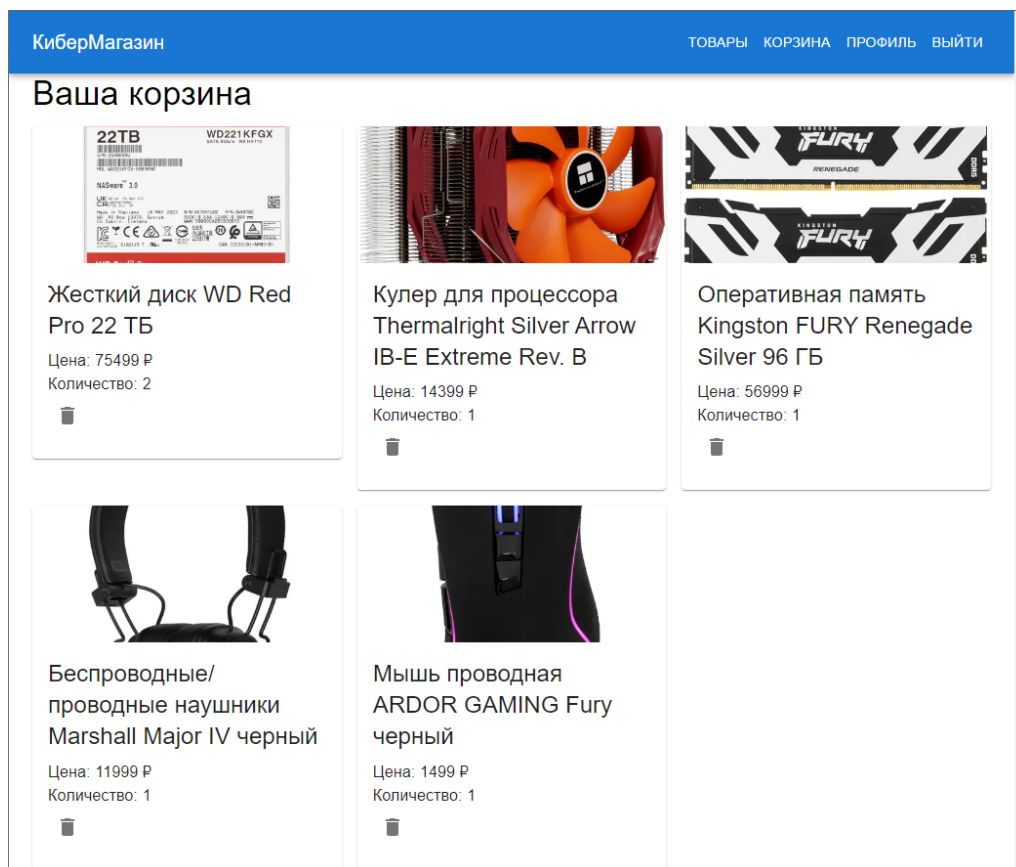



Рисунок 2.3.5 Страница корзины

Последним компонентом является страница профиля, на ней можно обновить данные об аккаунте, при вводе новых данных и нажатии на кнопку обновить происходит PUT запрос к контроллеру аккаунтов, который обновляет данные для юзера, айди которого все так же передается из локального хранилища.



Профиль

Логин *

Новый пароль

ОБНОВИТЬ

Рисунок 2.3.5 Страница профиля пользователя

Помимо пользовательской зоны, приложение содержит и панель администратора с возможностью редактирования, и удаления товаров. При входе в аккаунт администратора пользователь попадает на страницу, где отображаются все товары.

КиберМагазин Панель администратора


ТОВАРЫ ДОБАВИТЬ ТОВАР ВЫЙТИ

Список товаров

ДОБАВИТЬ НОВЫЙ ТОВАР

Процессор AMD Ryzen Threadripper PRO 5995WX OEM


Категория: Процессоры
Цена: 600000 Р
Количество на складе: 0



РЕДАКТИРОВАТЬ УДАЛИТЬ

Материнская плата ASUS Pro WS WRX90E-SAGE SE


Категория: Материнские платы
Цена: 183000 Р
Количество на складе: 5



РЕДАКТИРОВАТЬ УДАЛИТЬ

Видеокарта Palit GeForce RTX 4090 GameRock [NED4090019SB-1020G]

Категория: Видеокарты
Цена: 226000 Р
Количество на складе: 8



РЕДАКТИРОВАТЬ УДАЛИТЬ

Оперативная память Kingston FURY Renegade Silver 96 ГБ

Кулер для процессора Thermalright Silver Arrow IB-E

Жесткий диск WD Red Pro 22 ТБ

Рисунок 2.3.6 Страница каталога товаров для администратора

По своим функциям эта страница практически идентична той, что была разработана для покупателя, но есть отличия, в том, что она содержит кнопку, ведущую на добавление товара, редактирование и удаление выбранной позиции. Полный код данной страницы содержится в приложении И.

Кроме уже знакомых функций по запросу всех товаров списком для их отображения, есть функция удаления товара, которая получает айди нажатого товара для удаления и открывает окно для подтверждения удаления, после подтверждения отправляет DELETE запрос.

Кнопка редактировать формирует ссылку на страницу редактирования товара, после чего хук Params вытягивает переданный в айди параметр и запрашивает данные товара чтобы поместить их в форму для редактирования.

Страницы редактирования и добавления нового товара похожи и имеют одинаковые формы и поля для ввода, единственное отличие в том, что добавление отправляет POST запрос и данные нужно вводить все, страница редактирования же, как было сказано из параметров ссылки получает айди и заполняет все поля для выбранного предмета, после чего их можно изменять и отправлять PUT запросы на обновление. Разберем работу данных систем на примере добавления нового товара, полный код компонента содержится в приложении Д.

Компонент использует хуки useState для управления состояниями таких полей, как название, описание, цена, количество на складе, категория, изображение, предпросмотр изображения, список категорий, а также состояния модального окна и сообщения в нем.

При загрузке компонента с помощью useEffect вызывается функция fetchCategories, которая выполняет запрос к серверу для получения списка категорий. Полученные категории сохраняются в состоянии categories.

Функция handleSubmit обрабатывает отправку формы. При отправке

формы создается объект `FormData`, в который добавляется изображение. Затем выполняется POST-запрос к серверу для загрузки изображения.

После успешной загрузки изображения создается объект нового товара, включающий название, описание, цену, количество, категорию и путь к изображению. Этот объект отправляется на сервер с помощью POST-запроса. В случае успешного добавления товара выводится сообщение об успехе, иначе - сообщение об ошибке.

Функция `handleImageChange` обрабатывает изменение изображения. Когда пользователь выбирает файл изображения, он сохраняется в состоянии `image`, а также создается его предварительный просмотр с помощью `FileReader`.

Функция `handleRemoveImage` позволяет удалить выбранное изображение и сбросить состояние предпросмотра.

Функция `handleModalClose` закрывает модальное окно и перенаправляет пользователя на страницу со списком товаров.

Компонент рендерит форму для ввода данных о товаре. Форма включает поля для ввода названия, описания, цены, количества на складе и выбора категории из выпадающего списка. Также предусмотрена возможность загрузки изображения товара. Форма представлена с помощью компонентов `TextField`, `Button`, `Box`, `Typography` и `Container` из библиотеки `@mui/material`. Кнопка для загрузки изображения открывает диалог выбора файла, а предварительный просмотр изображения отображается под кнопкой. При успешном добавлении товара отображается уведомление с соответствующим сообщением. После закрытия уведомления пользователь перенаправляется на страницу со списком товаров.

Добавить новый товар

Название *

Процессор Intel Core i5-12400 OEM

Описание *

LGA 1700, 6 x 2.5 ГГц, L2 - 7.5 МБ, L3 - 18 МБ, 2

Цена, Р *

13899


Количество на складе *

123

Категория *

Процессоры

ЗАГРУЗИТЬ ИЗОБРАЖЕНИЕ



УДАЛИТЬ ИЗОБРАЖЕНИЕ

ДОБАВИТЬ ТОВАР

Рисунок 2.3.7 Страница добавления товара

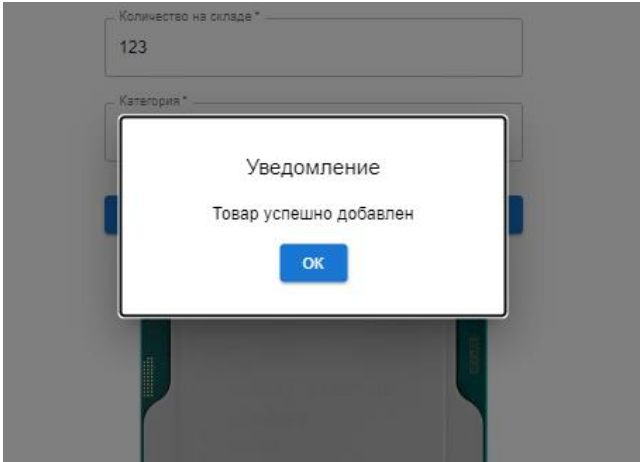


Рисунок 2.3.8 Уведомление о добавления товара

Последними же стоит разобрать служебные компоненты, такие как уведомления и хэдеры, как было упомянуто в начале данного раздела приложения содержит 3 различных хэдера в зависимости от авторизации пользователя, что было так же заметно на демонстрационных изображениях.

Рассмотрим хэдэр для пользователя. По своей сути, данный компонент — это панель с кнопками при нажатии на которые, пользователь перенаправляется на ту или иную страницу, а также при нажатии на кнопку выйти вызывается функции очистки авторизации и перенаправления на страницу входа.

Листинг 2.3.4 Хэдэр для пользователя

```
import React from 'react';
import { useNavigate } from 'react-router-dom';
import { AppBar, Toolbar, Typography, Button } from '@mui/material';

const UserHeader: React.FC = () => {
  const navigate = useNavigate();

  const handleLogout = () => {
    localStorage.removeItem('userRole');
    localStorage.removeItem('userId');
    navigate('/login');
  };

  return (
    <AppBar position="static">
      <Toolbar>
        <Typography variant="h6" component="div" sx={{ flexGrow: 1 }}>
          КиберМагазин
        </Typography>
        <Button color="inherit" onClick={() => navigate('/user-items')}>Товары</Button>
        <Button color="inherit" onClick={() => navigate('/cart')}>Корзина</Button>
        <Button color="inherit" onClick={() => navigate('/profile')}>Профиль</Button>
        <Button color="inherit" onClick={handleLogout}>Выйти</Button>
      </Toolbar>
    </AppBar>
  );
};

export default UserHeader;
```

Хэдеры для администратора и неавторизованного пользователя разработаны аналогичным образом и представлены в приложениях К и Л соответственно.

Уведомления же представляют собой либо всплывающие окна с сообщением или окна с кнопками вызывающие то или иное действие,

например, всплывающее уведомление внизу экрана.

Листинг 2.3.5 Компонент уведомления

```
import React from 'react';
import { Snackbar, Alert } from '@mui/material';

interface NotificationProps {
  open: boolean;
  message: string;
  severity: 'success' | 'error' | 'warning' | 'info';
  onClose: () => void;
}

const Notification: React.FC<NotificationProps> = ({ open, message, severity, onClose }) => {
  return (
    <Snackbar open={open} autoHideDuration={6000} onClose={onClose}>
      <Alert onClose={onClose} severity={severity} sx={{ width: '100%' }}>
        {message}
      </Alert>
    </Snackbar>
  );
};

export default Notification;
```

Данное уведомление имеет только одну функцию для закрытия его и несет в себе только информативный характер, так же данное уведомление автоматически пропадает если его не закрыть. Но есть и более функциональный тип уведомлений — подтверждение удаление товара.

Листинг 2.3.6 Окно подтверждения удаления

```
import React from 'react';
import { Dialog, DialogTitle, DialogContent, DialogActions, Button, Typography } from '@mui/material';

interface ConfirmDeleteDialogProps {
  open: boolean;
  title: string;
  description: string;
  onConfirm: () => void;
  onCancel: () => void;
}

const ConfirmDeleteDialog: React.FC<ConfirmDeleteDialogProps> = ({ open, title, description, onConfirm, onCancel }) => {
  return (
    <Dialog open={open} onClose={onCancel}>
      <DialogTitle>{title}</DialogTitle>
      <DialogContent>
        <Typography>{description}</Typography>
      </DialogContent>
      <DialogActions>
        <Button onClick={onCancel} color="primary">
          Отмена
        </Button>
        <Button onClick={onConfirm} color="secondary">
          Удалить
        </Button>
      </DialogActions>
    </Dialog>
  );
};
```

```
    </Dialog>  
  );  
};  
export default ConfirmDeleteDialog;
```

Функция удаления товара использует вызов `onConfirm`.

Таким образом были рассмотрены все возможности клиентской части приложения из чего так же можно сделать вывод о том, что поставленные цели и задачи были достигнуты.

Заключение

Процесс разработки интернет-магазина с использованием базы данных стал важным этапом в освоении современных технологий веб-разработки. Этот проект охватил все ключевые аспекты создания веб-приложений, начиная с проектирования и реализации серверной части, заканчивая созданием функционального и удобного пользовательского интерфейса.

В ходе работы над проектом была проведена тщательная проработка структуры базы данных. Основное внимание уделялось проектированию таблиц и их взаимосвязям, что обеспечило целостность и согласованность данных. Были разработаны ключевые сущности, такие как пользователи, категории товаров, товары, корзины и элементы корзин.

Реализация серверной части на основе ASP.NET Core обеспечила создание надежного API для управления пользователями, товарами и корзинами покупок.

Клиентская часть проекта была разработана с использованием React и библиотеки компонентов Material-UI. Это позволило создать адаптивный пользовательский интерфейс, обеспечивающий удобное взаимодействие пользователей с системой. Управление состоянием приложения и маршрутизация стали ключевыми элементами разработки, обеспечивающими корректное функционирование интерфейса.

В результате проделанной работы был реализован прототип интернет-магазина с необходимой функциональностью, обеспечивающей полноценное функционирование системы.

Список Литературы

- 1) Трофимов В.В. ASP.NET Core: разработка современных веб-приложений. - М.: ДМК Пресс, 2020.
- 2) Зубков С.И. Введение в Entity Framework Core. - СПб.: БХВ-Петербург, 2021.
- 3) Соколов А.В. Архитектура веб-приложений на ASP.NET Core и React. - М.: ДМК Пресс, 2020.
- 4) Документация по библиотеке Material-UI: офиц. сайт. — URL: <https://v4.mui.com/ru/> (дата обращения: 10.09.2024).
- 5) Документация по Entity Framework Core: офиц. сайт. — URL: <https://learn.microsoft.com/ru-ru/ef/> (дата обращения: 10.09.2024).
- 6) Документация по Axios: офиц. сайт. — URL: https://axios-http.com/ru/docs/api_intro (дата обращения: 10.09.2024).
- 7) Документация по React Router: офиц. сайт. — URL: <https://reactrouter.com/en/main> (дата обращения: 10.09.2024).
- 8) Курс базы данных 3-4 семестр [Электронный ресурс]. – Московский Политехнический университет.
- 9) Курс Back-end разработка [Электронный ресурс]. – Московский Политехнический университет.
- 10) Курс Веб программирование и дизайн [Электронный ресурс]. – Московский Политехнический университет.

Приложения

Приложение А

```
// ItemsController.cs
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Back_end.Models;

namespace Back_end.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ItemsController : ControllerBase
    {
        private readonly ShopContext _context;

        public ItemsController(ShopContext context)
        {
            _context = context;
        }

        // GET: api/Items
        [HttpGet]
        public async Task<ActionResult<IEnumerable<Item>>> GetItems()
        {
            return await _context.Items.ToListAsync();
        }

        // GET: api/Items/{id}
        [HttpGet("{id}")]
        public async Task<ActionResult<Item>> GetItem(int id)
        {
            var item = await _context.Items.FindAsync(id);

            if (item == null)
            {
                return NotFound();
            }

            return item;
        }

        // POST: api/Items
        [HttpPost]
        public async Task<ActionResult<Item>> AddItem(Item newItem)
        {
            if (!ModelState.IsValid)
                return BadRequest(ModelState);

            _context.Items.Add(newItem);
            await _context.SaveChangesAsync();

            return CreatedAtAction(nameof(GetItem), new { id = newItem.Id }, newItem);
        }

        // PUT: api/Items/{id}
        [HttpPut("{id}")]
        public async Task<ActionResult> UpdateItem(int id, Item updatedItem)
        {
            if (id != updatedItem.Id)
```

```

    {
        return BadRequest();
    }

    _context.Entry(updatedItem).State = EntityState.Modified;

    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!ItemExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }

    return NoContent();
}

// DELETE: api/Items/{id}
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteItem(int id)
{
    var item = await _context.Items.FindAsync(id);
    if (item == null)
    {
        return NotFound();
    }

    _context.Items.Remove(item);
    await _context.SaveChangesAsync();

    return NoContent();
}

private bool ItemExists(int id)
{
    return _context.Items.Any(e => e.Id == id);
}
}

```

Приложение Б

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Threading.Tasks;
using Back_end.Models;
using Microsoft.Extensions.Logging;

namespace Back_end.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class AccountController : ControllerBase
    {
        private readonly ShopContext _context;
        private readonly ILogger<AccountController> _logger;

        public AccountController(ShopContext context, ILogger<AccountController> logger)
        {
            _context = context;
            _logger = logger;
        }

        // POST: api/Account/Register
        [HttpPost("Register")]
        public async Task<IActionResult> Register(RegisterModel model)
        {
            if (!ModelState.IsValid)
                return BadRequest(ModelState);

            if (await _context.Users.AnyAsync(u => u.Login == model.Login))
                return Conflict("User already exists");

            var user = new User
            {
                Login = model.Login,
                Password = model.Password,
                Role = model.Role
            };

            _context.Users.Add(user);
            await _context.SaveChangesAsync();

            return Ok(new { message = "User registered successfully", userId = user.Id });
        }

        // POST: api/Account/Login
        [HttpPost("Login")]
        public async Task<IActionResult> Login(LoginModel model)
        {
            if (!ModelState.IsValid)
                return BadRequest(ModelState);

            var user = await _context.Users
                .FirstOrDefaultAsync(u => u.Login == model.Login && u.Password == model.Password);

            if (user == null)
            {
                _logger.LogWarning("Invalid login attempt for user: {Login}", model.Login);
                return Unauthorized("Invalid login or password");
            }

            _logger.LogInformation("User {Login} logged in successfully with userId {UserId}", user.Login, user.Id);
            return Ok(new { role = user.Role, userId = user.Id });
        }
    }
}
```



```

    }

    // POST: api/Account/Logout
    [HttpPost("Logout")]
    public IActionResult Logout()
    {
        // успешный ответ
        _logger.LogInformation("User logged out successfully");
        return Ok("Logged out successfully");
    }

    // PUT: api/Account/{id}
    [HttpPut("{id}")]
    public async Task<IActionResult> UpdateProfile(int id, UpdateProfileModel model)
    {
        if (!ModelState.IsValid)
            return BadRequest(ModelState);

        var user = await _context.Users.FindAsync(id);
        if (user == null)
            return NotFound("User not found");

        user.Login = model.Login;
        user.Password = model.Password; // мб надо хешировать

        _context.Users.Update(user);
        await _context.SaveChangesAsync();

        return Ok(new { message = "Profile updated successfully" });
    }
}

```

Приложение В

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using System.Threading.Tasks;
using Back_end.Models;
using System.Collections.Generic;

namespace Back_end.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class CartController : ControllerBase
    {
        private readonly ShopContext _context;

        public CartController(ShopContext context)
        {
            _context = context;
        }

        [HttpGet("{userId}")]
        public async Task<ActionResult<IEnumerable<object>>> GetCartItems(int userId)
        {
            var cart = await _context.Carts.FirstOrDefaultAsync(c => c.UserId == userId);

            if (cart == null)
            {
                return Ok(new List<object>());
            }

            var cartItems = await _context.CartItems
                .Where(ci => ci.CartId == cart.Id)
                .ToListAsync();

            var result = new List<object>();

            foreach (var ci in cartItems)
            {
                var item = await _context.Items.FindAsync(ci.ItemId);
                result.Add(new
                {
                    ci.Id,
                    ci.CartId,
                    ci.ItemId,
                    ci.Quantity,
                    item
                });
            }

            return Ok(result);
        }

        [HttpPost("{itemId}")]
        public async Task<ActionResult> AddToCart(int itemId, [FromHeader] string Authorization)
        {
            if (string.IsNullOrEmpty(Authorization))
                return Unauthorized();

            var userId = int.Parse(Authorization.Replace("Bearer ", ""));
            var cart = await _context.Carts.FirstOrDefaultAsync(c => c.UserId == userId);

            if (cart == null)
```

```

    {
        cart = new Cart { UserId = userId };
        _context.Carts.Add(cart);
        await _context.SaveChangesAsync();
    }

    var cartItem = await _context.CartItems.FirstOrDefaultAsync(ci => ci.CartId == cart.Id && ci.ItemId == itemId);

    if (cartItem != null)
    {
        cartItem.Quantity++;
    }
    else
    {
        cartItem = new CartItem { CartId = cart.Id, ItemId = itemId, Quantity = 1 };
        _context.CartItems.Add(cartItem);
    }

    await _context.SaveChangesAsync();
    return Ok("Товар добавлен в корзину");
}

[HttpDelete("{itemId}")]
public async Task<IActionResult> RemoveFromCart(int itemId, [FromHeader] string Authorization)
{
    if (string.IsNullOrEmpty(Authorization))
        return Unauthorized();

    var userId = int.Parse(Authorization.Replace("Bearer ", ""));
    var cart = await _context.Carts.FirstOrDefaultAsync(c => c.UserId == userId);

    if (cart == null)
        return NotFound("Корзина не найдена");

    var cartItem = await _context.CartItems.FirstOrDefaultAsync(ci => ci.CartId == cart.Id && ci.ItemId == itemId);

    if (cartItem == null)
        return NotFound("Товар не найден в корзине");

    _context.CartItems.Remove(cartItem);
    await _context.SaveChangesAsync();
    return Ok("Товар удален из корзины");
}

[HttpDelete("clear/{userId}")]
public async Task<IActionResult> ClearCart(int userId)
{
    var cart = await _context.Carts.FirstOrDefaultAsync(c => c.UserId == userId);

    if (cart == null)
        return NotFound("Корзина не найдена");

    var cartItems = _context.CartItems.Where(ci => ci.CartId == cart.Id);
    _context.CartItems.RemoveRange(cartItems);
    await _context.SaveChangesAsync();

    return Ok("Корзина очищена");
}

[HttpPost("checkout")]
public async Task<IActionResult> Checkout([FromHeader] string Authorization)
{
    if (string.IsNullOrEmpty(Authorization))
        return Unauthorized();
}

```

```

var userId = int.Parse(Authorization.Replace("Bearer ", ""));
var cart = await _context.Carts.FirstOrDefaultAsync(c => c.UserId == userId);

if (cart == null)
{
    return Ok(new { success = false, message = "Корзина не найдена" });
}

var cartItems = await _context.CartItems
    .Where(ci => ci.CartId == cart.Id)
    .ToListAsync();

foreach (var cartItem in cartItems)
{
    var item = await _context.Items.FindAsync(cartItem.ItemId);
    if (item == null || item.Stock < cartItem.Quantity)
    {
        return Ok(new { success = false, message = "Недостаточно товара на складе" });
    }
}

foreach (var cartItem in cartItems)
{
    var item = await _context.Items.FindAsync(cartItem.ItemId);
    if (item != null)
    {
        item.Stock -= cartItem.Quantity;
    }
}

_context.CartItems.RemoveRange(cartItems);
await _context.SaveChangesAsync();

return Ok(new { success = true, message = "Заказ успешно оформлен" });
}
}
}

```

Приложение Г

```
import React, { useState, useEffect } from 'react';
import { Link } from 'react-router-dom';
import axios from 'axios';
import { Card, CardContent, CardMedia, Typography, Button, Grid, Snackbar } from '@mui/material';
import Alert from '@mui/material/Alert';

interface Item {
  id: number;
  name: string;
  description: string;
  price: number;
  stock: number;
  categoryId: number;
  imagePath: string;
}

interface Category {
  id: number;
  name: string;
}

const UserItemsListPage: React.FC = () => {
  const [items, setItems] = useState<Item[]>([]);
  const [categories, setCategories] = useState<Category[]>([]);
  const [notification, setNotification] = useState({ open: false, message: '', severity: 'success' as 'success' | 'error' | 'warning' | 'info' });

  useEffect(() => {
    const fetchItems = async () => {
      const response = await axios.get('https://localhost:7009/api/Items');
      setItems(response.data);
    };

    const fetchCategories = async () => {
      const response = await axios.get('https://localhost:7009/api/Categories');
      setCategories(response.data);
    };

    fetchItems();
    fetchCategories();
  }, []);

  const addToCart = async (itemId: number) => {
    try {
      const userId = localStorage.getItem('userId');
      if (!userId) {
        setNotification({ open: true, message: 'Пожалуйста, войдите в систему, чтобы добавить товар в корзину.', severity: 'warning' });
        return;
      }
      await axios.post('https://localhost:7009/api/Cart/${itemId}', {}, {
        headers: {
          'Authorization': `Bearer ${userId}`
        }
      });
      setNotification({ open: true, message: 'Товар добавлен в корзину', severity: 'success' });
    } catch (error) {
      console.error('Error adding item to cart:', error);
      if ((error as any).response && (error as any).response.status === 401) {
        setNotification({ open: true, message: 'Ваша сессия истекла. Пожалуйста, войдите в систему снова.', severity: 'error' });
      } else {

```

```

        setNotification({ open: true, message: 'Произошла ошибка при добавлении товара в корзину.', severity: 'error'
    });
    }
  }
};

const handleNotificationClose = () => {
  setNotification({ ...notification, open: false });
};

return (
  <Grid container spacing={2} style={{ padding: '20px' }}>
    {items.map(item => (
      <Grid item xs={12} key={item.id}>
        <Card style={{ display: 'flex', flexDirection: 'row', alignItems: 'center' }}>
          <CardMedia
            component="img"
            style={{ width: 150, height: 150, objectFit: 'contain' }}
            image={`https://localhost:7009/${item.imagePath}`}
            alt={item.name}
          />
          <CardContent style={{ flex: 1 }}>
            <Link to={`/item/${item.id}`} style={{ textDecoration: 'none', color: 'inherit' }}>
              <Typography gutterBottom variant="h5" component="div">
                {item.name}
              </Typography>
              <Typography variant="body2" color="text.secondary">
                {item.description}
              </Typography>
              <Typography variant="body2" color="text.secondary">
                Категория: {categories.find(category => category.id === item.categoryId)?.name}
              </Typography>
            </Link>
          </CardContent>
          <CardContent style={{ display: 'flex', flexDirection: 'column', alignItems: 'flex-end' }}>
            <Typography variant="h6" component="div">
              Цена: {item.price} ₺
            </Typography>
            <Button variant="contained" color="primary" style={{ marginTop: 'auto' }} onClick={() =>
addToCart(item.id)}>
              Купить
            </Button>
          </CardContent>
        </Card>
      </Grid>
    ))}
    <Snackbar open={notification.open} autoHideDuration={6000} onClose={handleNotificationClose}>
      <Alert onClose={handleNotificationClose} severity={notification.severity} sx={{ width: '100%' }}>
        {notification.message}
      </Alert>
    </Snackbar>
  </Grid>
);
};

export default UserItemsListPage;

```

Приложение Д

```
// AddItemPage.tsx
import React, { useState, useEffect } from 'react';
import { TextField, Button, Box, Typography, Container, MenuItem } from '@mui/material';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import ModalNotification from '../components/ModalNotification';

const AddItemPage: React.FC = () => {
  const [name, setName] = useState("");
  const [description, setDescription] = useState("");
  const [price, setPrice] = useState("");
  const [stock, setStock] = useState("");
  const [categoryId, setCategoryId] = useState("");
  const [image, setImage] = useState<File | null>(null);
  const [preview, setPreview] = useState<string | null>(null);
  const [categories, setCategories] = useState<{ id: number, name: string }[]>([]);
  const navigate = useNavigate();
  const [modalOpen, setModalOpen] = useState(false);
  const [modalMessage, setModalMessage] = useState("");

  useEffect(() => {
    const fetchCategories = async () => {
      try {
        const response = await axios.get('https://localhost:7009/api/Categories');
        setCategories(response.data);
      } catch (error) {
        console.error('Error fetching categories:', error);
      }
    };
    fetchCategories();
  }, []);

  const handleSubmit = async (event: React.FormEvent) => {
    event.preventDefault();
    if (!image) return;

    const formData = new FormData();
    formData.append('file', image);

    try {
      const imageResponse = await axios.post('https://localhost:7009/api/ImageUpload', formData, {
        headers: {
          'Content-Type': 'multipart/form-data',
        },
      });

      const newItem = {
        name,
        description,
        price: parseFloat(price),
        stock: parseInt(stock, 10),
        categoryId: parseInt(categoryId, 10),
        imagePath: imageResponse.data.filePath,
      };

      await axios.post('https://localhost:7009/api/Items', newItem);
      setModalMessage("Товар успешно добавлен");
      setModalOpen(true);
    } catch (error) {
      console.error('Error adding item:', error);
      setModalMessage("Ошибка при добавлении товара");
      setModalOpen(true);
    }
  };
};
```

```

    }
  };

  const handleImageChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    if (e.target.files && e.target.files[0]) {
      const file = e.target.files[0];
      setImage(file);
      const reader = new FileReader();
      reader.onloadend = () => {
        setPreview(reader.result as string);
      };
      reader.readAsDataURL(file);
    }
  };

  const handleRemoveImage = () => {
    setImage(null);
    setPreview(null);
  };

  const handleModalClose = () => {
    setModalOpen(false);
    navigate('/items'); // Redirect to items list page
  };

  return (
    <Container component="main" maxWidth="xs">
      <Box
        sx={{
          marginTop: 8,
          display: 'flex',
          flexDirection: 'column',
          alignItems: 'center',
        }}
      >
        <Typography component="h1" variant="h5">
          Добавить новый товар
        </Typography>
        <Box component="form" onSubmit={handleSubmit} sx={{ mt: 1 }}>
          <TextField
            margin="normal"
            required
            fullWidth
            label="Название"
            name="name"
            value={name}
            onChange={(e) => setName(e.target.value)}
            autoFocus
          />
          <TextField
            margin="normal"
            required
            fullWidth
            label="Описание"
            name="description"
            value={description}
            onChange={(e) => setDescription(e.target.value)}
          />
          <TextField
            margin="normal"
            required
            fullWidth
            label="Цена, ₽"
            name="price"

```



```

        type="number"
        value={price}
        onChange={(e) => setPrice(e.target.value)}
      />
      <TextField
        margin="normal"
        required
        fullWidth
        label="Количество на складе"
        name="stock"
        type="number"
        value={stock}
        onChange={(e) => setStock(e.target.value)}
      />
      <TextField
        margin="normal"
        required
        fullWidth
        label="Категория"
        name="category"
        select
        value={categoryId}
        onChange={(e) => setCategoryId(e.target.value)}
      >
        {categories.map((category) => (
          <MenuItem key={category.id} value={category.id}>
            {category.name}
          </MenuItem>
        ))}
      </TextField>
      <Button
        variant="contained"
        component="label"
        fullWidth
        sx={{ mt: 3, mb: 2 }}
      >
        Загрузить изображение
        <input
          type="file"
          hidden
          onChange={handleImageChange}
        />
      </Button>
      {preview && (
        <Box sx={{ mt: 2, mb: 2 }}>
          <img src={preview} alt="Превью" style={{ width: '100%' }} />
          <Button variant="contained" color="secondary" fullWidth onClick={handleRemoveImage}>
            Удалить изображение
          </Button>
        </Box>
      )}
      <Button type="submit" fullWidth variant="contained" sx={{ mt: 3, mb: 2 }}>
        Добавить товар
      </Button>
    </Box>
    <ModalNotification
      open={modalOpen}
      message={modalMessage}
      onClose={handleModalClose}
    />
  </Box>
</Container>
);
};

```

```
export default AddItemPage;
```

Приложение Е

```
import React, { useEffect, useState } from 'react';
import { BrowserRouter as Router, Route, Routes, Navigate } from 'react-router-dom';
import LoginPage from './pages/LoginPage';
import RegisterPage from './pages/RegisterPage';
import ItemsListPage from './pages/ItemsListPage';
import AddItemPage from './pages/AddItemPage';
import EditItemPage from './pages/EditItemPage';
import UserItemsListPage from './pages/UserItemsListPage';
import ItemDetailPage from './pages/ItemDetailPage';
import CartPage from './pages/CartPage';
import ProfilePage from './pages/ProfilePage';
import AdminHeader from './components/AdminHeader';
import UserHeader from './components/UserHeader';
import LoginHeader from './components/LoginHeader';

const App: React.FC = () => {
  const [userRole, setUserRole] = useState<string | null>(localStorage.getItem('userRole'));

  useEffect(() => {
    const role = localStorage.getItem('userRole');
    if (role) {
      setUserRole(role);
    }
  }, []);

  return (
    <Router>
      <Routes>
        <Route path="/login" element={<>
          <LoginHeader />
          <LoginPage onLogin={(role) => setUserRole(role)} />
        </> } />
        <Route path="/register" element={<>
          <LoginHeader />
          <RegisterPage />
        </> } />
        {userRole === 'admin' ? (
          <>
            <Route path="/items" element={<>
              <AdminHeader />
              <ItemsListPage />
            </> } />
            <Route path="/add-item" element={<>
              <AdminHeader />
              <AddItemPage />
            </> } />
            <Route path="/edit-item/:id" element={<>
              <AdminHeader />
              <EditItemPage />
            </> } />
            <Route path="/" element={<Navigate to="/items" /> } />
          </>
        ) : userRole === 'user' ? (
          <>
            <Route path="/user-items" element={<>
              <UserHeader />
              <UserItemsListPage />
            </> } />
            <Route path="/item/:id" element={<>
              <UserHeader />
              <ItemDetailPage />
            </> } />
          </>
        ) : null
      </Routes>
    </Router>
  );
};
```

```

    <Route path="/cart" element={<>
      <UserHeader />
      <CartPage />
    </> } />
    <Route path="/profile" element={<>
      <UserHeader />
      <ProfilePage />
    </> } />
    <Route path="/" element={<Navigate to="/user-items" /> } />
  </>
) : (
  <Route path="/" element={<Navigate to="/login" /> } />
)
</Routes>
</Router>
);
};

export default App;

```

Приложение Ж

```
import React, { useState } from 'react';
import { TextField, Button, Box, Typography, Container, FormControl, InputLabel, Select, MenuItem } from
 '@mui/material';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import Notification from '../components/Notification';

const RegisterPage: React.FC = () => {
  const [login, setLogin] = useState("");
  const [password, setPassword] = useState("");
  const [role, setRole] = useState("");
  const [notification, setNotification] = useState({ open: false, message: "", severity: 'success' as 'success' | 'error' | 'warning' |
'info' });
  const navigate = useNavigate();

  const handleSubmit = async (event: React.FormEvent) => {
    event.preventDefault();

    try {
      await axios.post('https://localhost:7009/api/Account/Register', { login, password, role });
      setNotification({ open: true, message: 'User registered successfully', severity: 'success' });
      navigate('/login');
    } catch (error) {
      console.error('Error registering user:', error);
      setNotification({ open: true, message: 'Error registering user', severity: 'error' });
    }
  };

  const handleNotificationClose = () => {
    setNotification({ ...notification, open: false });
  };

  return (
    <Container component="main" maxWidth="xs">
      <Box component="form" onSubmit={handleSubmit} sx={{ mt: 1 }}>
        <Typography component="h1" variant="h5">
          Регистрация
        </Typography>
        <TextField
          margin="normal"
          required
          fullWidth
          label="Логин"
          name="login"
          value={login}
          onChange={(e) => setLogin(e.target.value)}
          autoFocus
        />
        <TextField
          margin="normal"
          required
          fullWidth
          name="password"
          label="Пароль"
          type="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
        />
        <TextField
          margin="normal"
          required
          fullWidth

```

```

        select
        label="Тип аккаунта"
        name="role"
        value={role}
        defaultValue={ "user" }
        onChange={ (e) => setRole(e.target.value)}
      <
      <MenuItem value={ "user" }>
        Покупатель
      </MenuItem>
      <MenuItem value={ "admin" }>
        Администратор
      </MenuItem>
    </TextField>
    <Button type="submit" fullWidth variant="contained" sx={{ mt: 3, mb: 2 }}>
      Зарегистрироваться
    </Button>
    <Notification
      open={ notification.open }
      message={ notification.message }
      severity={ notification.severity }
      onClose={ handleNotificationClose }
    />
  </Box>
</Container>
);
};

export default RegisterPage;

```

Приложение 3

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { Container, Typography, Button, Grid, Card, CardContent, CardMedia, IconButton } from '@mui/material';
import DeleteIcon from '@mui/icons-material/Delete';
import ConfirmDialog from '../components/ConfirmDialog';

const CartPage: React.FC = () => {
  const [cartItems, setCartItems] = useState<any[]>([]);
  const [isDialogOpen, setIsDialogOpen] = useState(false);
  const [dialogTitle, setDialogTitle] = useState("");
  const [dialogDescription, setDialogDescription] = useState("");

  const fetchCart = async () => {
    try {
      const userId = localStorage.getItem('userId');
      if (!userId) {
        throw new Error('User ID not found');
      }

      const response = await axios.get(`https://localhost:7009/api/Cart/${userId}`);
      setCartItems(response.data);
    } catch (error) {
      console.error('Error fetching cart items:', error);
    }
  };

  const removeItem = async (itemId: number) => {
    try {
      const userId = localStorage.getItem('userId');
      if (!userId) {
        throw new Error('User ID not found');
      }

      await axios.delete(`https://localhost:7009/api/Cart/${itemId}`, {
        headers: {
          'Authorization': `Bearer ${userId}`
        }
      });
      fetchCart();
    } catch (error) {
      console.error('Error removing item from cart:', error);
    }
  };

  const clearCart = async () => {
    try {
      const userId = localStorage.getItem('userId');
      if (!userId) {
        throw new Error('User ID not found');
      }

      await axios.delete(`https://localhost:7009/api/Cart/clear/${userId}`);
      fetchCart();
    } catch (error) {
      console.error('Error clearing cart:', error);
    }
  };

  const handleOrder = async () => {
    try {
      const userId = localStorage.getItem('userId');
      if (!userId) {
```

```

    throw new Error('User ID not found');
  }

  if (cartItems.length === 0) {
    setDialogTitle('Корзина пуста');
    setDescription('Нет товаров для оформления заказа.');
    setIsDialogOpen(true);
    return;
  }

  const response = await axios.post(`https://localhost:7009/api/Cart/checkout`, {}, {
    headers: {
      'Authorization': `Bearer ${userId}`
    }
  });

  if (response.data.success) {
    setDialogTitle('Заказ оформлен');
    setDescription('Ваш заказ был успешно оформлен!');
    clearCart();
  } else {
    setDialogTitle('Недостаточно товара');
    setDescription('Один или несколько товаров недостаточно на складе.');
```



```

        </Grid>
      )))
    </Grid>
    {cartItems.length > 0 && (
      <Button variant="contained" color="secondary" onClick={clearCart} style={{ marginTop: '20px' }}>
        Очистить корзину
      </Button>
    )}
    <Button variant="contained" color="primary" style={{ marginTop: '20px' }} onClick={handleOrder}>
      ОФОРМИТЬ ЗАКАЗ
    </Button>
    <ConfirmDialog
      open={isDialogOpen}
      title={dialogTitle}
      description={dialogDescription}
      onConfirm={() => setIsDialogOpen(false)}
      onCancel={() => setIsDialogOpen(false)}
    />
  </Container>
);
};

export default CartPage;

```

Приложение И

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import { Box, Button, Typography, Card, CardContent, CardActions, Grid, Container } from '@mui/material';
import { useNavigate } from 'react-router-dom';
import ModalNotification from '../components/ModalNotification';
import ConfirmDeleteDialog from '../components/ConfirmDeleteDialog';

interface Item {
  id: number;
  name: string;
  description: string;
  price: number;
  stock: number;
  categoryId: number;
  imagePath: string;
}

interface Category {
  id: number;
  name: string;
}

const ItemsListPage: React.FC = () => {
  const [items, setItems] = useState<Item[]>([]);
  const [categories, setCategories] = useState<Category[]>([]);
  const [modalOpen, setModalOpen] = useState(false);
  const [modalMessage, setModalMessage] = useState('');
  const [confirmDialogOpen, setConfirmDialogOpen] = useState(false);
  const [itemToDelete, setItemToDelete] = useState<number | null>(null);
  const navigate = useNavigate();

  useEffect(() => {
    const fetchItems = async () => {
      try {
        const response = await axios.get('https://localhost:7009/api/Items');
        setItems(response.data);
      } catch (error) {
        console.error('Error fetching items:', error);
      }
    };
    const fetchCategories = async () => {
      try {
        const response = await axios.get('https://localhost:7009/api/Categories');
        setCategories(response.data);
      } catch (error) {
        console.error('Error fetching categories:', error);
      }
    };
    fetchItems();
    fetchCategories();
  }, []);

  const handleDelete = async () => {
    if (itemToDelete !== null) {
      try {
        await axios.delete(`https://localhost:7009/api/Items/${itemToDelete}`);
        setModalMessage('Товар успешно удален');
        setModalOpen(true);
        setItems(items.filter(item => item.id !== itemToDelete));
      } catch (error) {
        console.error('Error deleting item:', error);
        setModalMessage('Ошибка при удалении товара');
      }
    }
  };
};
```

```

        setModalOpen(true);
    } finally {
        setItemToDelete(null);
        setConfirmDialogOpen(false);
    }
}
};

const openConfirmDialog = (id: number) => {
    setItemToDelete(id);
    setConfirmDialogOpen(true);
};

const handleModalClose = () => {
    setModalOpen(false);
};

const handleConfirmDialogClose = () => {
    setConfirmDialogOpen(false);
};

return (
    <Container>
        <Box sx={{ marginTop: 8, display: 'flex', flexDirection: 'column', alignItems: 'center' }}>
            <Typography component="h1" variant="h5">
                Список товаров
            </Typography>
            <Button variant="contained" color="primary" onClick={() => navigate('/add-item')} sx={{ marginBottom: 2 }}>
                Добавить новый товар
            </Button>
            <Grid container spacing={3}>
                {items.map(item => (
                    <Grid item key={item.id} xs={12} sm={6} md={4}>
                        <Card sx={{ maxWidth: 600, height: 500, display: 'flex', flexDirection: 'column', justifyContent: 'space-
between' }}>
                            <CardContent sx={{ flexGrow: 1 }}>
                                <Typography gutterBottom variant="h5" component="div">
                                    {item.name}
                                </Typography>
                                <Typography variant="body2" color="text.secondary">
                                    Категория: {categories.find(category => category.id === item.categoryId)?.name}
                                </Typography>
                                <Typography variant="body2" color="text.secondary">
                                    Цена: {item.price} Р
                                </Typography>
                                <Typography variant="body2" color="text.secondary">
                                    Количество на складе: {item.stock}
                                </Typography>
                                {item.imagePath && (
                                    <Box sx={{ height: 250, display: 'flex', justifyContent: 'center', alignItems: 'center' }}>
                                        <img src={`https://localhost:7009${item.imagePath}`} alt={item.name} style={{ maxHeight:
'100%', maxWidth: '100%' }} />
                                    </Box>
                                )}
                            </CardContent>
                            <CardActions>
                                <Button
                                    variant="outlined"
                                    color="primary"
                                    onClick={() => navigate(`/edit-item/${item.id}`)}
                                    fullWidth
                                >
                                    Редактировать
                                </Button>

```

```

        <Button
          variant="outlined"
          color="secondary"
          onClick={() => openConfirmDialog(item.id)}
          fullWidth
        >
          Удалить
        </Button>
      </CardActions>
    </Card>
  </Grid>
))}
</Grid>
</Box>
<ModalNotification
  open={modalOpen}
  message={modalMessage}
  onClose={handleModalClose}
/>
<ConfirmDeleteDialog
  open={confirmDialogOpen}
  title="Подтверждение удаления"
  description="Вы уверены, что хотите удалить этот товар? Это действие нельзя отменить."
  onConfirm={handleDelete}
  onCancel={handleConfirmDialogClose}
/>
</Container>
);
};

export default ItemsListPage;

```

Приложение К

```
import React from 'react';
import { useNavigate } from 'react-router-dom';
import { AppBar, Toolbar, Typography, Button } from '@mui/material';

const AdminHeader: React.FC = () => {
  const navigate = useNavigate();

  const handleLogout = () => {
    localStorage.removeItem('userRole');
    localStorage.removeItem('userId');
    navigate('/login');
  };

  return (
    <AppBar position="static">
      <Toolbar>
        <Typography variant="h6" component="div" sx={{ flexGrow: 1 }}>
          КиберМагазин Панель администратора
        </Typography>
        <Button color="inherit" onClick={() => navigate('/items')}>Товары</Button>
        <Button color="inherit" onClick={() => navigate('/add-item')}>Добавить товар</Button>
        <Button color="inherit" onClick={handleLogout}>Выйти</Button>
      </Toolbar>
    </AppBar>
  );
};

export default AdminHeader;
```

Приложение Л

```
import React from 'react';
import { AppBar, Toolbar, Typography } from '@mui/material';

const LoginHeader: React.FC = () => {
  return (
    <AppBar position="static">
      <Toolbar>
        <Typography variant="h6" component="div" sx={{ flexGrow: 1 }}>
          КиберМагазин
        </Typography>
      </Toolbar>
    </AppBar>
  );
};

export default LoginHeader;
```