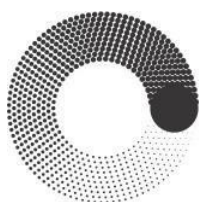


ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
ВЫСШАЯ ШКОЛА ПЕЧАТИ И МЕДИАИНДУСТРИИ

*Институт Принтмедиа и информационных технологий  
Кафедра Информатики и информационных технологий*

направление подготовки

09.03.02 «Информационные системы и технологии»,

## КУРСОВОЙ ПРОЕКТ

Дисциплина: Back-end разработка

Тема: Онлайн магазин электроники

Выполнил: студент группы 221-374

Кузин Артём Александрович

Дата, подпись \_\_\_\_\_  
(Дата) (Подпись)

Проверил: \_\_\_\_\_  
(Фамилия И.О., степень, звание)

Дата, подпись \_\_\_\_\_  
(Дата) (Подпись)

Замечания: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

Москва

2024

## Оглавление

ВВЕДЕНИЕ .....	3
ГЛАВА 1 Проектирование .....	4
1.1 Описание предметной области .....	4
1.2 Выбор инструментов разработки .....	4
ГЛАВА 2. РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ.....	7
2.1 Проектирование и физическая реализация базы данных.....	7
2.2 Разработка API .....	14
2.3 Разработка UI и реализация Fetch API.....	28
ЗАКЛЮЧЕНИЕ .....	37
Библиографический список .....	38

## **ВВЕДЕНИЕ**

Тема курсового проекта – разработка на базе кроссплатформенных WEB технологий информационной системы онлайн магазина электроники, обеспечивающей удобную покупку техники.

В современном мире электронная коммерция становится все более популярной и востребованной среди потребителей. С развитием интернет-технологий и расширением возможностей онлайн платформ, возникла необходимость в создании удобных и функциональных информационных систем для онлайн магазинов. Удобная и качественная информационная система позволит пользователям без особых проблем купить тот товар, который их интересует.

Цель данного проекта заключается в создании высококачественной информационной системы, которая обеспечит пользователям удобный и эффективный процесс покупки техники через онлайн магазин. Информационная система позволит клиентам легко найти необходимые товары, совершить покупку с минимальными усилиями и удобно произвести оплату.

Исходя из поставленной цели и описанной проблемы, были сформированы следующие задачи

1. Изучение предметной области;
2. Изучение технологий для разработки WEB API;
3. Изучение методов работы с запросами HTTP;
4. Выбор оптимальных инструментов разработки для создания системы;
5. Проектирование базы данных, определение структуры и связей между таблицами;
6. Реализация физической модели данных.

# **ГЛАВА 1 Проектирование**

## **1.1 Описание предметной области**

Онлайн магазин – это информационная система, позволяющая пользователям совершать различные покупки, не выходя из дома. При этом пользователи хотят пользоваться удобным, информативным и понятным сервисом.

Для этого была создана информационная система, упрощающая процесс покупки и других нужд пользователя.

Приложение будет использоваться обычными пользователями, а также администраторами информационной системы.

Пользователь системы может посмотреть список товаров в интернет-магазине, выбрать интересующий его товар, количество и оформить заказ. Вся информация о заказе будет автоматически добавлена в базу данных.

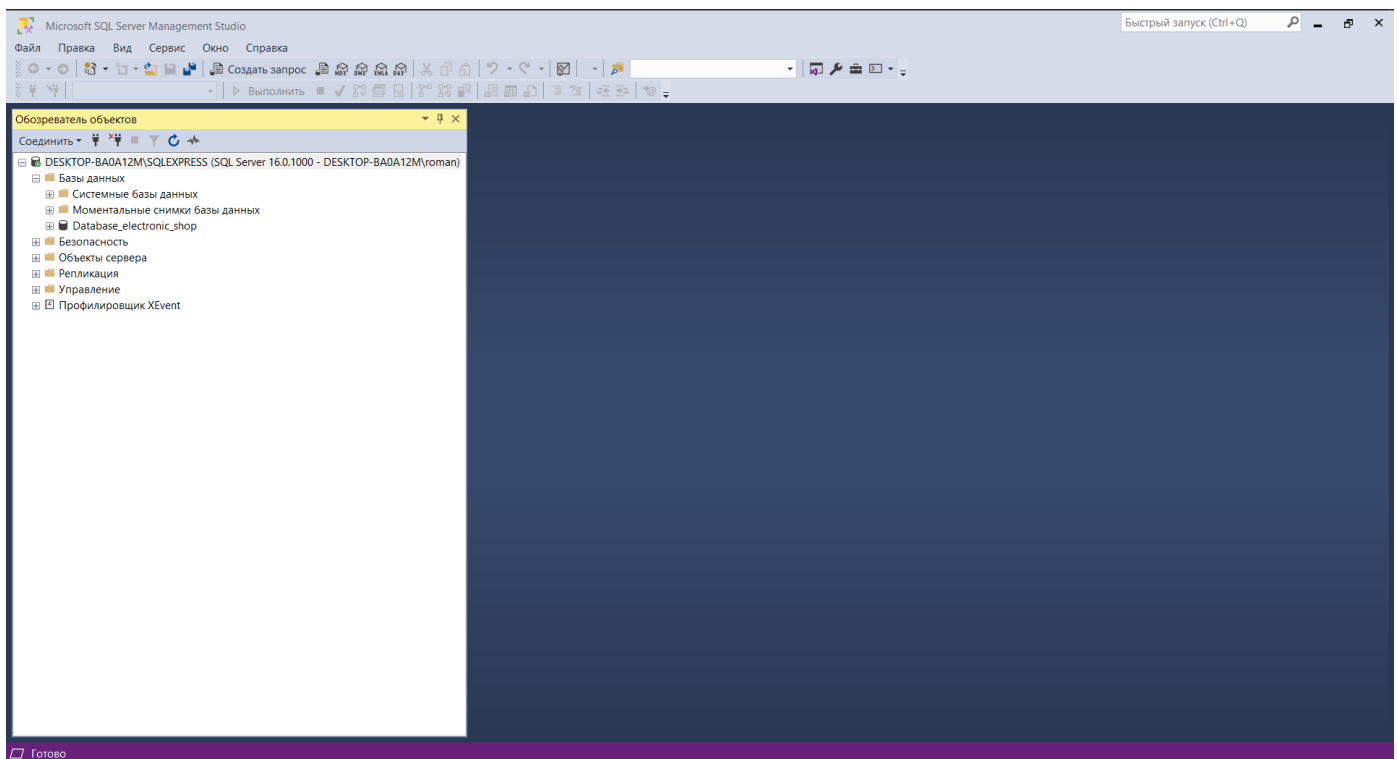
Функционал для администраторов приложения предоставляет большой функционал для работы с информационной системы. Администратор имеет возможность добавлять, изменять или удалять данные о товарах в магазине, пользователях, просматривать заказы пользователей.

## **1.2 Выбор инструментов разработки**

Выбор оптимальных инструментов для разработки информационной системы имеет решающее значение для успешной реализации проекта и обеспечения комфортной работы разработчиков и пользователей системы.

Важным критерием выбора является функциональность и удобство работы

с базой данных. В реализуемом проекте был выбран такой инструмент, как SQL Server 2020 Express - это бесплатная и легкая в использовании версия управляемой реляционной системы управления базами данных Microsoft SQL Server. Она предлагает множество возможностей SQL Server, таких как управление данными, создание запросов, хранение и обработку информации, но в урезанной по сравнению с полными версиями функциональности.



*Рисунок 1 – Оболочка SQL Server 2020 Express*

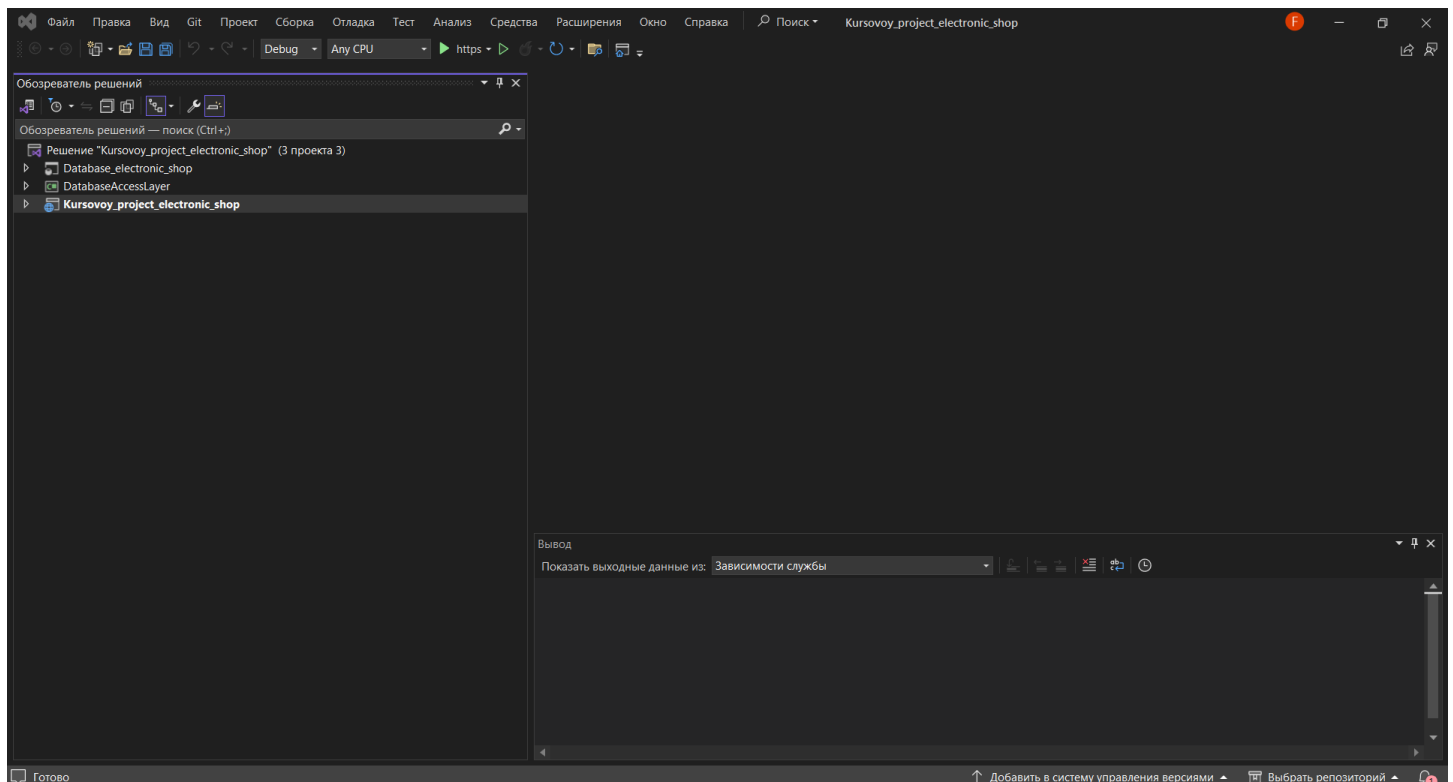
Для реализации проекта было выбрано WEB-приложение. Данный формат проекта был выбран потому, что люди пользуются различными операционными системами, а доступ к оформлению заказа должен быть с разных платформ (IOS, Android). Для создания информационной системы использовался язык программирования C#. Язык является мощным и гибким языком программирования, разработанным компанией Microsoft. Он широко используется для создания разнообразных приложений, включая веб-приложения. C# это объектно-ориентированный язык программирования,

который также поддерживает компонентно-ориентированное программирование.

В качестве фреймворка для создания API был выбран ASP.NET Core.

Для разработки web-приложения «онлайн магазин электроники» была выбрана такая среда программирования, как Microsoft Visual Studio, так как с помощью неё есть возможность разрабатывать множество типов приложения под разные платформы и обладает хорошей интеграцией с другими компонентами Microsoft.

Также, среда разработки Microsoft Visual Studio предоставляет возможность работы с фреймворком ASP.NET Core, который обеспечивает множество дополнительных инструментов и библиотек для создания эффективного API.

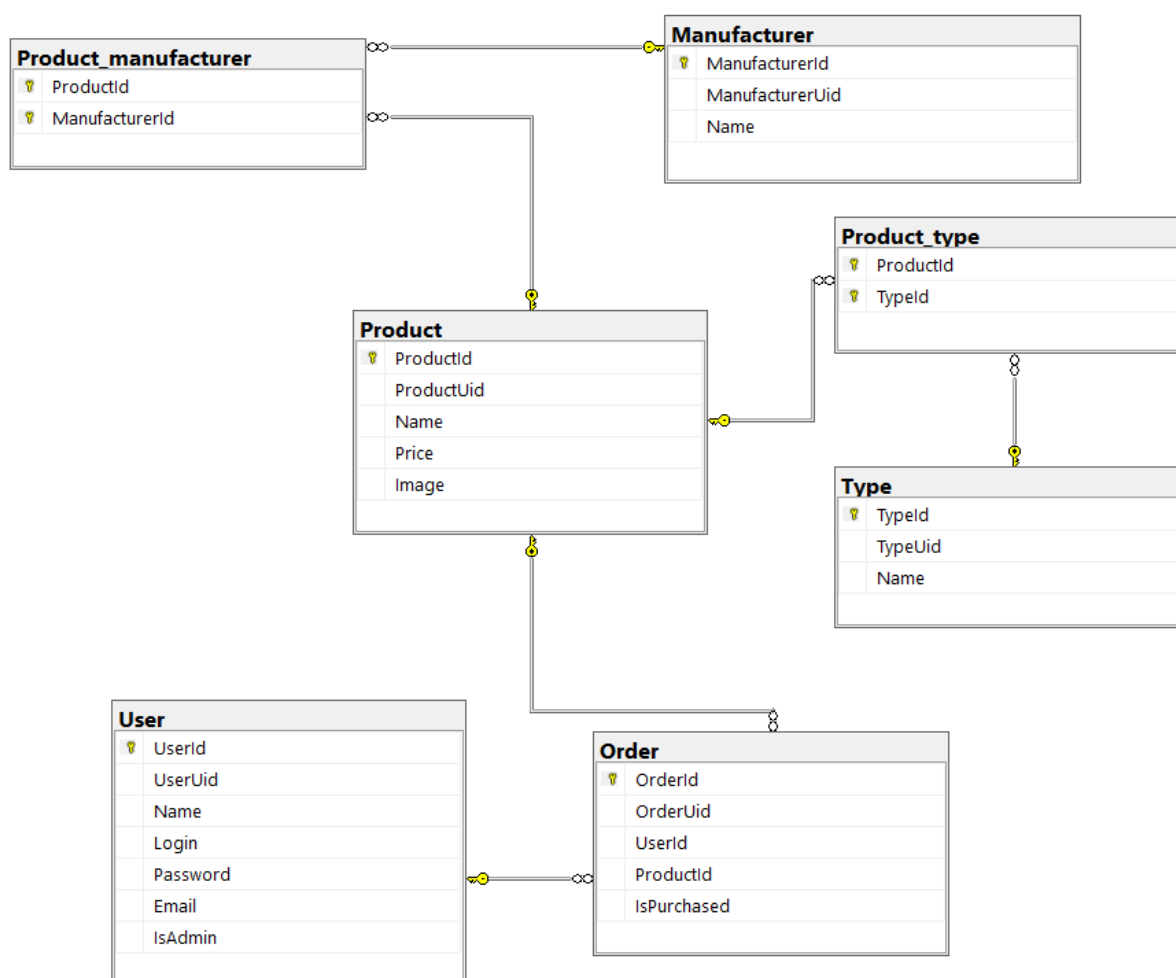


*Рисунок 2 – Оболочка Microsoft Visual Studio*

## ГЛАВА 2. РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ «Онлайн магазин электроники»


### 2.1 Проектирование и физическая реализация базы данных

В рамках данного проекта была спроектирована и создана база данных, содержащая семь связанных таблиц. Таблица реляционной базы данных – это совокупность связанных данных, хранящихся в структурированном виде в базе данных. Данная таблица показана на рисунке ниже.




*Рисунок 3 – Диаграмма базы данных*

Далее будут представлены структуры всех таблиц с пояснением к каждой скриншотом её представления в системе SQL Server Management Studio 20.

Product	
	ProductId
	ProductUid
	Name
	Price
	Image

*Рисунок 4 – Таблица товаров*

- ProductId – уникальный идентификатор товара в базе данных
- ProductUid – уникальный идентификатор товара на сайте
- Name – название товара
- Price – цена товара
- Image – картинка товара


User	
	UserId
	UserUid
	Name
	Login
	Password
	Email
	IsAdmin

*Рисунок 5 – Таблица пользователей*

- UserId – уникальный идентификатор пользователя в базе данных




- UserId – уникальный идентификатор пользователя на сайте
- Name – Имя пользователя
- Login – Логин пользователя
- Password – пароль пользователя
- Email – электронная почта пользователя
- IsAdmin – флаг, который указывает является ли пользователь администратором

Order	
	OrderId
	OrderUid
	UserId
	ProductId
	IsPurchased


*Рисунок 6 – Таблица заказов*

- OrderId – уникальный идентификатор заказа в базе данных
- OrderUid – уникальный идентификатор заказа на сайте
- UserId – идентификатор пользователя
- ProductId – идентификатор товара
- isPurchased – флаг, который указывает является ли заказ оплаченным

Manufacturer	
	ManufacturerId
	ManufacturerUid
	Name



*Рисунок 7 – Таблица производителей*

- ManufacturerId – уникальный идентификатор производителя в базе данных
- ManufacturerUid – уникальный идентификатор производителя на сайте
- Name – название производителя

Type	
	TypeId
	TypeUid
	Name

*Рисунок 8 – Таблица типов товаров*



- TypeId – уникальный идентификатор типа товара в базе данных
- TypeUid – уникальный идентификатор типа товара на сайте
- Name – название типа товара

Product_type	
	ProductId
	TypeId

*Рисунок 9 – Таблица продуктов и типов товаров*

Таблица Product\_type является связующей таблицей и содержит информацию о товарах и их типах. Она содержит такие поля как:

- ProductId – идентификатор товара
- TypeId – идентификатор типа товара

Product_manufacturer	
	ProductId
	ManufacturerId

*Рисунок 10 – Таблица продуктов и их производителей*

Таблица Product\_manufacturer является связующей таблицей и содержит информацию о товарах и их производителях. Она содержит такие поля как:

- ProductId – идентификатор товара
- ManufacturerId – идентификатор производителя товара

С помощью SQL Server 2020 Express была реализована база данных для информационной системы «онлайн магазин электронной техники»

Следом была разработана диаграмма классов:

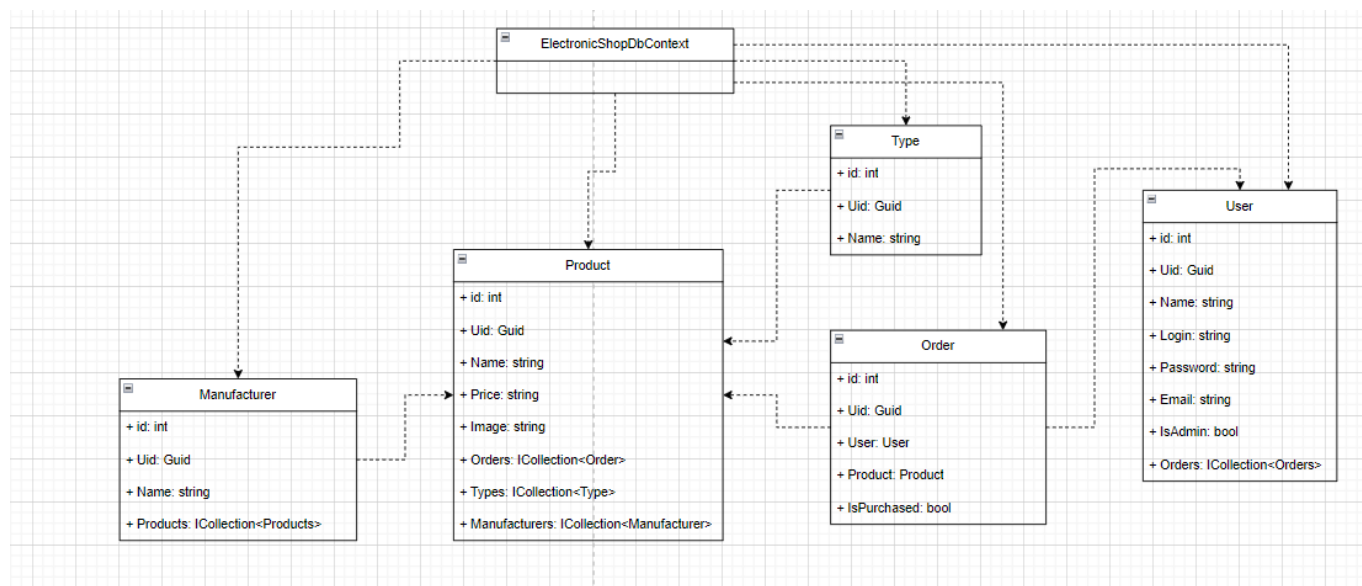


Рисунок 11 – Диаграмма классов

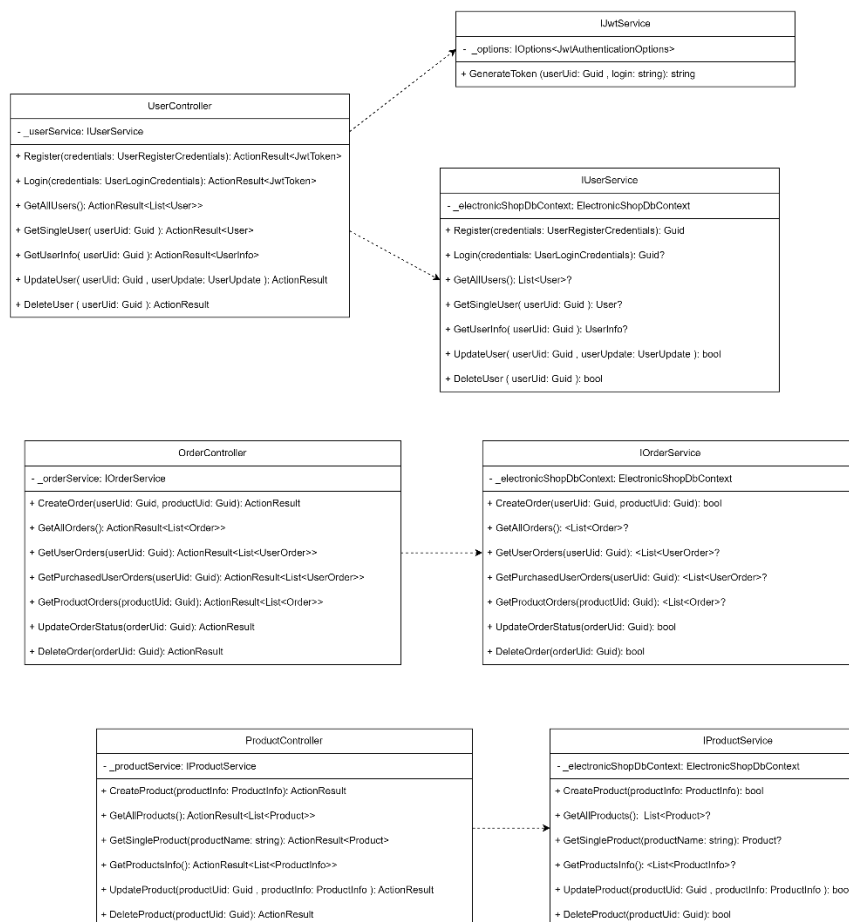


Рисунок 12.1 – API

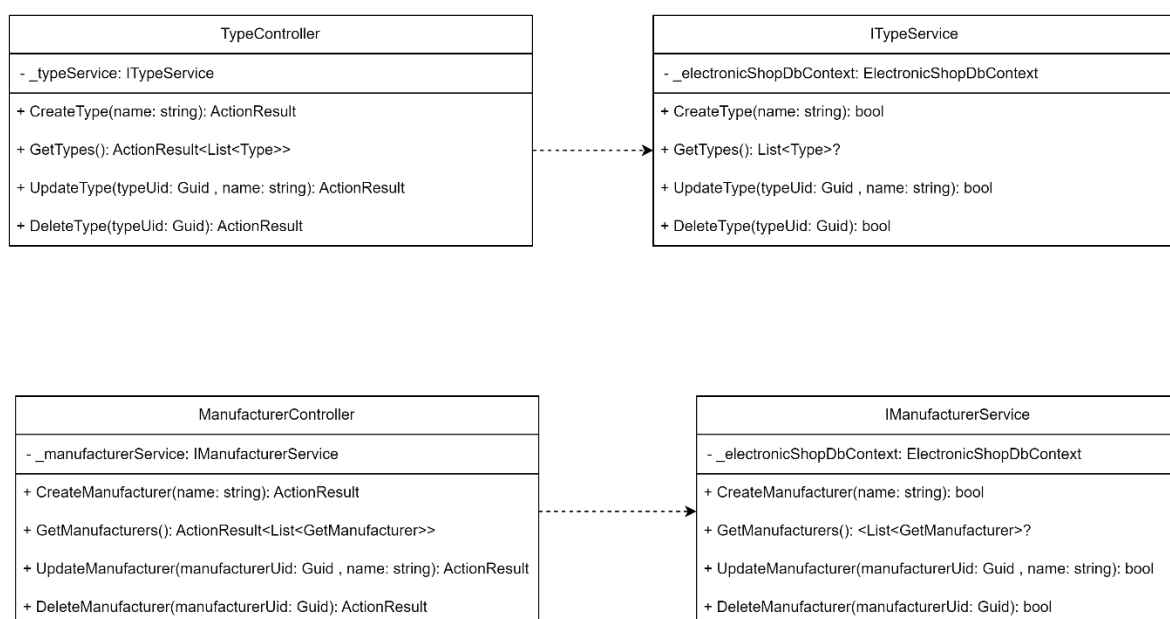
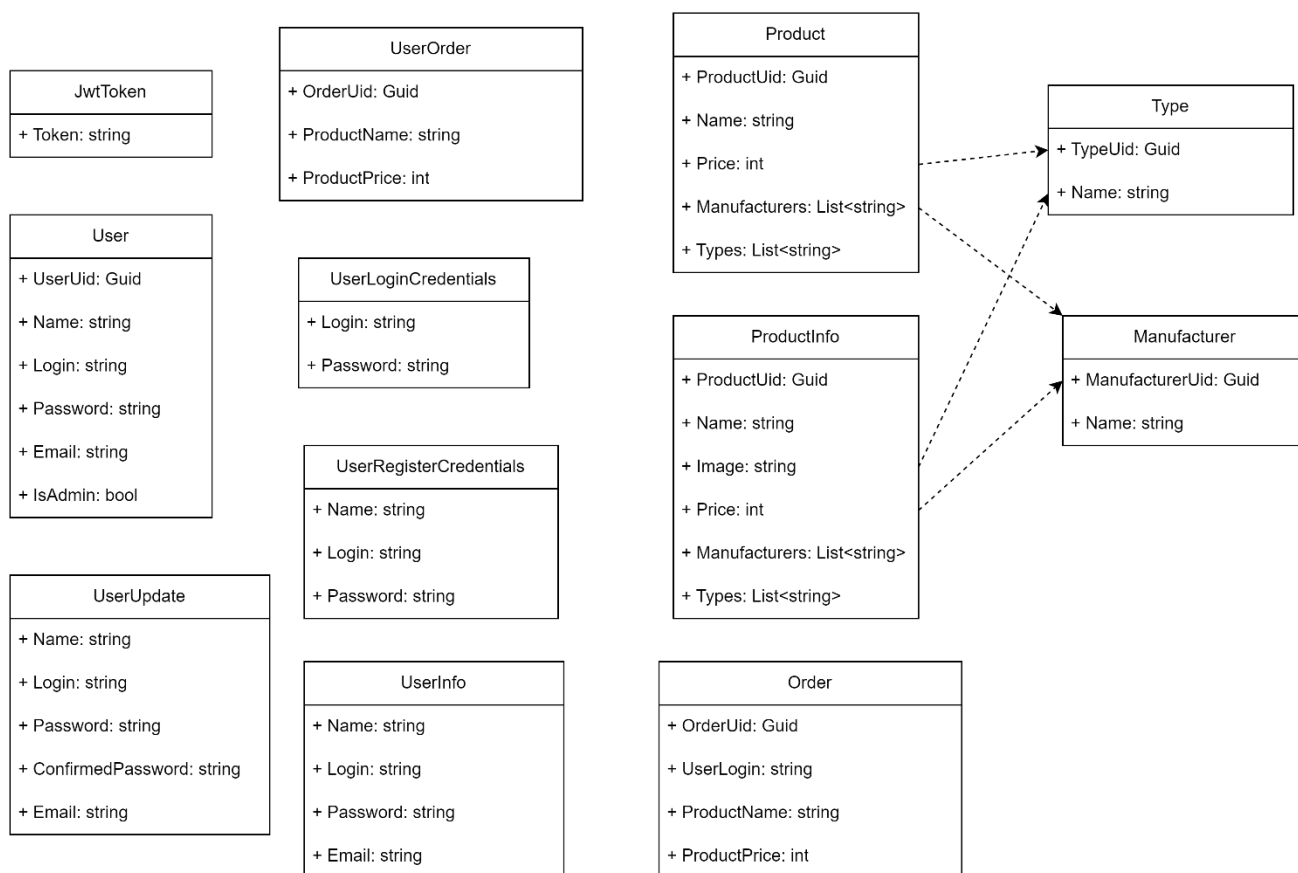


Рисунок 12.2 – API



*Рисунок 13 – Контрактные модели*

## 2.2 Разработка API

Для понимания того, как проходила разработка данного проекта необходимо сначала изучить его ключевые компоненты, которые были реализованы на языке C#.

В архитектуре ASP.NET Core одним из основных компонентов является контроллер, который отвечает за обработку запросов. При поступлении запроса, система маршрутизации определяет соответствующий контроллер для его обработки и передаёт ему данные запроса. Контроллер обрабатывает запрос и возвращает результат выполненной операции.

Другим важным компонентом является сервис, который выполняет

определённые операции и обработку данных. Сервис взаимодействует с базой данных, файловой системой и другими сервисами для получения или обновления данных. Контроллер вызывает методы сервиса для работы с данными.

Контрактные модели (или модели представления) – ещё один компонент, используемый для передачи данных между контроллером и клиентом. Они определяют структуру данных, передаваемую через сеть, и содержат только необходимую информацию. Это упрощает взаимодействие между сервером и клиентом.

И, наконец, класс `DbContext` представляет собой подключение к базе данных. `DbContext` обеспечивает взаимодействие с базой данных, выполнение операции `CRUD` и содержит информацию о структуре данных. Этот класс облегчает работу с данными и позволяет легко управлять таблицами, столбцами и связями в базе данных.

Далее будут показаны примеры реализации сервисов, контроллеров, контрактных моделей и класса `DbContext`:

На рисунке ниже представлен контроллер для пользователя. Метод `Register()` реализует `HTTP POST` запрос. Он принимает на вход контрактную модель `UserRegisterCredentials`, которая содержит в себе данные для регистрации. После регистрации пользователю присваивается специальный `JWT`-токен. Данный токен используется для безопасной передачи данных между клиентом и сервером с помощью шифрования.

```

[Route("api/[controller]/[action]")]
[ApiController]
Ссылка: 1
public class UserController : ControllerBase
{
    private readonly IUserService _userService;
    private readonly IJwtService _jwtService;

    Ссылка: 0
    public UserController(IUserService userService, IJwtService jwtService)
    {
        _userService = userService;
        _jwtService = jwtService;
    }

    [HttpPost]
    [AllowAnonymous]
    Ссылка: 0
    public ActionResult<JwtToken> Register(UserRegisterCredentials credentials)
    {
        if (!_userService.CheckLoginRegex(credentials.Login))
        {
            ModelState.AddModelError("", "Invalid name format");
            return BadRequest(ModelState);
        }

        if (_userService.CheckLogin(credentials.Login))
        {
            ModelState.AddModelError("", "Login already exists");
            return BadRequest(ModelState);
        }

        var userId = _userService.Register(credentials);

        return new JwtToken
        {
            Token = _jwtService.GenerateToken(userId, credentials.Login, false)
        };
    }
}

```

*Рисунок 14 – метод Register()*

Метод Login() реализует HTTP POST запрос. Он принимает на вход контрактную модель UserLoginCredentials, которая содержит в себе данные авторизации. После авторизации пользователю так же присваивается специальный JWT-токен.



```

[HttpPost]
[AllowAnonymous]
Ссылка: 0
public ActionResult<JwtToken> Login(UserLoginCredentials credentials)
{
    var userId = _userService.Login(credentials);

    if (userId == null)
    {
        ModelState.AddModelError("user", "Invalid login or password");

        return BadRequest(ModelState);
    }

    return new JwtToken
    {
        Token = _jwtService.GenerateToken(userId.Value, credentials.Login, _userService.IsAdmin(userId.Value))
    };
}

```

*Рисунок 15 – метод Login()*

Ниже представлен код реализации метода GenerateToken(), который создаёт Jwt-токен.

```

Ссылка: 2
public class JwtService : IJwtService
{
    private readonly IOption<JwtAuthenticationOptions> _options;

    Ссылка: 0
    public JwtService(IOption<JwtAuthenticationOptions> options)
    {
        _options = options;
    }

    Ссылка: 3
    public string GenerateToken(Guid userId, string login, bool isAdmin)
    {
        var tokenDescriptor = new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(new[]
            {
                new Claim(ClaimTypes.NameIdentifier, userId.ToString()),
                new Claim(ClaimTypes.Name, login),
                new Claim(ClaimTypes.Role, isAdmin ? "Admin" : "User")
            },
            Expires = DateTime.UtcNow.AddDays(1),
            Issuer = _options.Value.Issuer,
            Audience = _options.Value.Audience,
            SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_options.Value.Key)), SecurityAlgorithms.HmacSha256Signature)
        };

        var tokenHandler = new JwtSecurityTokenHandler();
        var token = tokenHandler.CreateToken(tokenDescriptor);

        return tokenHandler.WriteToken(token);
    }
}

```

*Рисунок 17 – метод GenerateToken()*

Такие методы, как GetAllUsers(), GetSingleUser(), GetUserInfo используют HTTP GET запрос. Эти методы используются администратором информационной системы для получения различной информации о пользователях.

```

[HttpGet]
[Authorize (Roles = "Admin")]
Ссылка: 0
public ActionResult<List<User>> GetAllUsers()
{
    var users = _userService.GetAllUsers();

    if (users == null)
    {
        return NotFound("No users found");
    }

    return Ok(users);
}

[HttpGet]
[Authorize(Roles = "Admin")]
Ссылка: 0
public ActionResult<User> GetSingleUser(Guid userId)
{
    var user = _userService.GetSingleUser(userId);

    if (user == null)
    {
        return NotFound("User not found");
    }

    return Ok(user);
}

[HttpGet]
[Authorize(Roles = "Admin, User")]
Ссылка: 0
public ActionResult<UserInfo> GetUserInfo(Guid userId)
{
    var user = _userService.GetUserInfo(userId);

    if (user == null)
    {
        return NotFound("User not found");
    }

    return Ok(user);
}

```

*Рисунок 18 – методы GetAllUsers(), GetSingleUser(), GetUserInfo*

Также администратор имеет возможность изменять информацию о пользователе. Для этого существует метод UpdateUser(). Этот метод использует HTTP PUT запрос. В качестве аргументов метод принимает uid пользователя и контрактную модель UserInfo, в которой содержатся данные для обновления.

```

[HttpPut]
[Authorize(Roles = "Admin, User")]
public ActionResult UpdateUser(Guid userId, UserUpdate userUpdate)
{
    if (userUpdate.Login == null || userUpdate.Password == null || userUpdate.Name == null || userUpdate.ConfirmedPassword == null)
    {
        return BadRequest();
    }

    if (_userService.GetLogin(userId) != userUpdate.Login)
    {
        if (!_userService.CheckLoginRegex(userUpdate.Login))
        {
            ModelState.AddModelError("", "Invalid name format");
            return BadRequest(ModelState);
        }

        if (_userService.CheckLogin(userUpdate.Login))
        {
            ModelState.AddModelError("", "Login already exists");
            return BadRequest(ModelState);
        }
    }

    if (!_userService.CheckEmailRegex(userUpdate.Email))
    {
        ModelState.AddModelError("", "Invalid email format");
        return BadRequest(ModelState);
    }

    if (userUpdate.Password != userUpdate.ConfirmedPassword)
    {
        ModelState.AddModelError("", "Failed to confirm password");
        return BadRequest(ModelState);
    }

    if (!_userService.UpdateUser(userId, userUpdate))
    {
        ModelState.AddModelError("", "Failed to update user");
        return BadRequest(ModelState);
    }

    return Ok("User updated");
}

```

*Рисунок 19 – метод UpdateUser()*

Также администратор информационной системы может удалить пользователя. Для этого существует метод DeleteUser(). Этот метод использует HTTP DELETE запрос. В качестве параметров он принимает uid пользователя, который нужно удалить.

```

public ActionResult DeleteUser(Guid userId)
{
    if (!_userService.DeleteUser(userId))
    {
        ModelState.AddModelError("", "Failed to delete user");
        return BadRequest(ModelState);
    }

    return Ok("User deleted");
}

```

*Рисунок 20 – метод DeleteUser()*

Администратор информационной системы может также работать с другими данными, например со списком товаров на сайте. Так, с помощью метода CreateProduct(), администратор может добавить новый товар на сайт.

Этот метод использует HTTP POST запрос и принимает в качестве аргумента контрактную модель, в которой находятся необходимые данные, такие как название, цена, производитель.

```
[HttpPost]
[Authorize(Roles = "Admin")]
Ссылка: 0
public ActionResult CreateProduct(ProductInfo productInfo)
{
    if (productInfo.Name == null || productInfo.Price <= 0)
    {
        return BadRequest();
    }

    if (!_productService.CheckRegex(productInfo.Name))
    {
        ModelState.AddModelError("", "Invalid product name format");
        return BadRequest(ModelState);
    }

    if (!_productService.CheckRegexList(productInfo.Manufacturers))
    {
        ModelState.AddModelError("", "Invalid manufacturer name format");
        return BadRequest(ModelState);
    }

    if (!_productService.CheckRegexList(productInfo.Types))
    {
        ModelState.AddModelError("", "Invalid type name format");
        return BadRequest(ModelState);
    }

    if (_productService.CheckProductInfo(productInfo))
    {
        ModelState.AddModelError("", "Product already exists");
        return BadRequest(ModelState);
    }

    if (!_productService.CreateProduct(productInfo))
    {
        ModelState.AddModelError("", "Failed to create product");
        return BadRequest(ModelState);
    }

    return Ok("Product created");
}
```

*Рисунок 21 – метод CreateProduct()*

Метод GetAllProducts() позволит администраторы вывести на экран все товары.

```

public ActionResult<List<Product>> GetAllProducts()
{
    var products = _productService.GetAllProducts();

    if (products == null)
    {
        return NotFound("No products found");
    }

    return Ok(products);
}

```

*Рисунок – 22 метод GetAllProducts()*

Также у администратора есть возможность изменять характеристики товаров. Для этого существует метод UpdateProduct(). Этот метод использует запрос HTTP PUT, принимает в качестве аргумента uid товара и контрактную модель, в которой находится вся необходимая для замены информация.

```

public ActionResult UpdateProduct(Guid productUid, ProductInfo productInfo)
{
    if (productInfo.Name == null || productInfo.Price <= 0 )
    {
        return BadRequest();
    }

    if (!_productService.CheckRegex(productInfo.Name))
    {
        ModelState.AddModelError("", "Invalid product name format");
        return BadRequest(ModelState);
    }

    if (!_productService.CheckRegexList(productInfo.Manufacturers))
    {
        ModelState.AddModelError("", "Invalid manufacturer name format");
        return BadRequest(ModelState);
    }

    if (!_productService.CheckRegexList(productInfo.Types))
    {
        ModelState.AddModelError("", "Invalid type name format");
        return BadRequest(ModelState);
    }

    if (_productService.CheckProductInfo(productUid, productInfo))
    {
        ModelState.AddModelError("", "Product already exists");
        return BadRequest(ModelState);
    }

    if (!_productService.UpdateProduct(productUid, productInfo))
    {
        ModelState.AddModelError("", "Failed to update product");
        return BadRequest(ModelState);
    }

    return Ok("Product updated");
}

```

*Рисунок 23 – метод UpdateProduct()*

Администратор также может удалить товар. Для этого существует метод DeleteProduct(). Метод использует запрос HTTP DELETE и принимает в качестве аргумента uid товара.

```
[HttpDelete]
[Authorize(Roles = "Admin")]
Ссылка: 0
public ActionResult DeleteProduct(Guid productUid)
{
    if (!_productService.DeleteProduct(productUid))
    {
        ModelState.AddModelError("", "Failed to delete product");
        return BadRequest(ModelState);
    }

    return Ok("Product deleted");
}
```

*Рисунок 24 – метод DeleteProduct()*

Далее будут представлены некоторые контрактные модели:

```
namespace Kursovoy_project_electronic_shop.Contracts
{
    Ссылка: 3
    public class UserRegisterCredentials
    {
        Ссылка: 1
        public required string Name { get; init; }

        Ссылка: 4
        public required string Login { get; init; }

        Ссылка: 1
        public required string Password { get; init; }
    }
}
```

*Рисунок 25 – контрактная модель регистрации пользователя*

```

{
  Ссылка: 8
  public class User
  {
    Ссылка: 2
    public Guid UserId { get; init; }

    Ссылка: 2
    public required string Name { get; init; }

    Ссылка: 2
    public required string Login { get; init; }

    Ссылка: 2
    public required string Password { get; init; }

    Ссылка: 2
    public required string? Email { get; init; }

    Ссылка: 2
    public bool IsAdmin { get; init; } = false;
  }
}

```

*Рисунок 26 – контрактная модель пользователя*

```

{
  Ссылка: 3
  public class UserUpdate
  {
    Ссылка: 2
    public required string Name { get; init; }

    Ссылка: 5
    public required string Login { get; init; }

    Ссылка: 3
    public required string Password { get; init; }

    Ссылка: 2
    public required string ConfirmedPassword { get; init; }

    Ссылка: 2
    public required string Email { get; init; }
  }
}

```

*Рисунок 27 – контрактная модель изменения данных о пользователе*

```

Ссылка: 8
public class Order
{
    Ссылка: 2
    public required Guid OrderUid { get; init; }

    Ссылка: 2
    public required string UserLogin { get; init; }

    Ссылка: 2
    public required string ProductName { get; init; }

    Ссылка: 2
    public required int ProductPrice { get; init; }

    Ссылка: 2
    public required List<string> ProductType { get; init; }

    Ссылка: 2
    public required List<string> ProductManufacturer { get; init; }
}

```

*Рисунок 28 – контрактная модель заказа*

```

Ссылка: 14
public class ProductInfo
{
    Ссылка: 1
    public Guid ProductUid { get; init; }

    Ссылка: 9
    public required string Name { get; init; }

    Ссылка: 2
    public string? Image { get; init; }

    Ссылка: 7
    public required int Price { get; init; }

    Ссылка: 5
    public required List<string> Manufacturers { get; init; }

    Ссылка: 5
    public required List<string> Types { get; init; }
}

```

*Рисунок 29 – контрактная модель информации о товаре*

Как уже говорилось ранее, класс `ElectronicShopDbContext` используется для взаимодействия с базой данных.

Fluent API в ASP.NET Core используется для гибкой настройки



отображения сущностей базы данных на объекты модели в Entity Framework Core.

Этот интерфейс позволяет детально настроить связи между сущностями, определяя типы связей (один-к-одному, один-ко-многим, многие-ко-многим), настройку каскадного удаления, управление именами и типами столбцов в базе данных и другие аспекты.

ORM (Object-Relational Mapping) является технологией программирования, которая создает абстракцию между реляционными базами данных и объектно-ориентированными языками программирования. ORM облегчает работу с данными, предоставляя высокоуровневую модель данных и автоматическое отображение объектов на таблицы в базе данных. Это позволяет программистам работать с данными на уровне объектов, без необходимости писать сложные SQL-запросы и работать с низкоуровневыми деталями структуры данных.

```
namespace DatabaseAccessLayer
{
    Ссылка 13
    public class ElectronicShopDbContext : DbContext
    {
        Ссылка 0
        public ElectronicShopDbContext(DbContextOptions<ElectronicShopDbContext> options) : base(options)
        {
        }

        Ссылка 0
        protected override void OnModelCreating(ModelBuilder builder)
        {
            builder.Entity<User>().HasKey(x => x.UserId);
            builder.Entity<User>().HasMany(x => x.Orders).WithOne(x => x.User).HasForeignKey("UserId");

            builder.Entity<Product>().HasKey(x => x.ProductId);
            builder.Entity<Product>().HasMany(x => x.Orders).WithOne(x => x.Product).HasForeignKey("ProductId");

            builder.Entity<Order>().HasKey(x => x.OrderId);
            builder.Entity<Order>().HasOne(x => x.User).WithMany(x => x.Orders).HasForeignKey("UserId");

            builder.Entity<Order>().HasOne(x => x.Product).WithMany(x => x.Orders).HasForeignKey("ProductId");

            builder.Entity<Manufacturer>().HasKey(x => x.ManufacturerId);
            builder.Entity<Manufacturer>().HasMany(x => x.Products).WithMany(x => x.Manufacturers)
                .UsingEntity("Product_manufacturer",
                    l => l.HasOne<typeof(Product)>().WithMany().HasForeignKey("ProductId").HasPrincipalKey(nameof(Product.ProductId)),
                    r => r.HasOne<typeof(Manufacturer)>().WithMany().HasForeignKey("ManufacturerId").HasPrincipalKey(nameof(Manufacturer.ManufacturerId)),
                    j => j.HasKey("ManufacturerId", "ProductId"));

            builder.Entity<Entities.Type>().HasKey(x => x.TypeId);
            builder.Entity<Entities.Type>().HasMany(x => x.Products).WithMany(x => x.Types)
                .UsingEntity("Product_type",
                    l => l.HasOne<typeof(Product)>().WithMany().HasForeignKey("ProductId").HasPrincipalKey(nameof(Product.ProductId)),
                    r => r.HasOne<typeof(Entities.Type)>().WithMany().HasForeignKey("TypeId").HasPrincipalKey(nameof(Entities.Type.TypeId)),
                    j => j.HasKey("TypeId", "ProductId"));
        }
    }
}
```

*Рисунок 30 – класс ElectronicShopDbContext*

Далее будут представлены классы для сущностей в базе данных:

```
public class User
{
    public int UserId { get; init; }

    public required Guid UserUid { get; init; }

    public required string Name { get; set; }

    public required string Login { get; set; }

    Ссылка: 6
    public required string Password { get; set; }

    Ссылка: 4
    public string? Email { get; set; }

    Ссылка: 4
    public required bool IsAdmin { get; set; } = false;

    Ссылка: 2
    public ICollection<Order> Orders { get; set; }
}
```

*Рисунок 31 – класс для сущности User*

```
Ссылка: 22
public class Product
{
    Ссылка: 3
    public int ProductId { get; init; }

    Ссылка: 10
    public required Guid ProductUid { get; init; }

    Ссылка: 13
    public required string Name { get; set; }

    Ссылка: 11
    public required int Price { get; set; }

    Ссылка: 4
    public string? Image { get; set; }

    Ссылка: 2
    public ICollection<Order> Orders { get; set; }

    Ссылка: 18
    public ICollection<Type> Types { get; set; } = new List<Type>();

    Ссылка: 18
    public ICollection<Manufacturer> Manufacturers { get; set; } = new List<Manufacturer>();
}
```

*Рисунок 32 – класс для сущности Product*

```
Ссылка: 14
public class Manufacturer
{
    Ссылка: 2
    public int ManufacturerId { get; init; }

    Ссылка: 6
    public required Guid ManufacturerUid { get; init; }

    Ссылка: 13
    public required string Name { get; set; }

    Ссылка: 1
    public ICollection<Product> Products { get; set; }
}
```

*Рисунок 33 – класс для сущности Manufacturer*

```
public class Order
{
    Ссылка: 1
    public int OrderId { get; init; }

    Ссылка: 8
    public required Guid OrderUid { get; init; }

    Ссылка: 11
    public required User User { get; set; }

    Ссылка: 34
    public required Product Product { get; set; }

    Ссылка: 5
    public required bool IsPurchased { get; set; } = false;
}
```

*Рисунок 34 – класс для сущности Order*

```

public class Type
{
    public int TypeId { get; init; }

    public required Guid TypeUid { get; init; }

    public required string Name { get; set; }

    public ICollection<Product> Products { get; set; }
}

```

*Рисунок 35 – класс для сущности Type*

## 2.3 Разработка UI и реализация Fetch API

Для разработки пользовательского интерфейса был выбран HTML, UI фреймворк Bootstrap и необходимые скрипты JavaScript для обработки событий.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</head>
<body>
  <br>
  <div class="container" style="margin-top: 10%">
    <div class="row">
      <div class="col-md-6 offset-md-3">
        <h1 class="text-center">Регистрация</h1>
        <br>
        <div class="form-group">
          <label for="name">Полное имя:</label>
          <input type="text" class="form-control" id="name" name="name" required>
        </div>
        <div class="form-group">
          <label for="login">Логин:</label>
          <input type="text" class="form-control" id="login" name="login" required>
        </div>
        <div class="form-group">
          <label for="password">Пароль:</label>
          <input type="password" class="form-control" id="password" name="password" required>
        </div>
        <button type="button" class="btn btn-primary" onclick="register()">Зарегистрироваться</button>
        <button type="button" class="btn btn-secondary" onclick="window.location.href='authorization.html'">Войти</button>
      </div>
    </div>
  </div>
  <script src="../js/user.js"></script>
</body>
</html>

```

*Рисунок 36 – страница регистрации пользователя*

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet"
        href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" />
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
  <title>Login</title>
</head>
<body>
  <br />
  <div class="container" style="margin-top: 10px">
    <div class="row">
      <div class="col-md-6 offset-md-3">
        <h1 class="text-center">Авторизация</h1>
        <br />
        <div class="form-group">
          <label for="login">Логин:</label>
          <input type="text" class="form-control" id="login" name="login" required />
        </div>
        <div class="form-group">
          <label for="password">Пароль:</label>
          <input type="password" class="form-control" id="password" name="password" required />
        </div>
        <button type="button" class="btn btn-primary" onclick="login()">
          Войти
        </button>
        <button type="button" class="btn btn-secondary" onclick="window.location.href='registration.html'">
          Регистрация
        </button>
      </div>
    </div>
  </div>
  <script src="../js/user.js"></script>
  <script>
    localStorage.clear();
  </script>
</body>
</html>

```

*Рисунок 37 – страница авторизации пользователя*

Далее будут представлены фрагменты кода, отвечающие за работу с запросами и ответами HTTP:

```

async function login() {
  const login = document.getElementById('login').value;
  const password = document.getElementById('password').value;

  const credentials = {
    login: login,
    password: password
  };

  try {
    const response = await fetch('https://localhost:7210/api/User/Login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(credentials)
    });

    if (response.ok) {
      const data = await response.json();
      localStorage.setItem('userToken', data.token);

      var decodedToken = JSON.parse(atob(data.token.split('.')[1]));
      const userRole = decodedToken.role;

      if (userRole === 'Admin') {
        window.location.href = 'admin/user.html';
      } else if (userRole === 'User') {
        window.location.href = 'user/product.html';
      } else {
        console.log(await response.text());
        alert('Произошла ошибка при авторизации');
      }
    } else {
      console.log(await response.text());
      throw new Error('Неправильный логин или пароль');
    }
  } catch (error) {
    console.error(error);
    alert('Неправильный логин или пароль');
  }
}

```

*Рисунок 38 – функция авторизации пользователя*

```

async function register() {
  const name = document.getElementById('name').value;
  const login = document.getElementById('login').value;
  const password = document.getElementById('password').value;

  if (!name || !login || !password) {
    alert('Заполните необходимые поля');
    return;
  }

  const credentials = {
    name: name,
    login: login,
    password: password
  };

  try {
    const response = await fetch('https://localhost:7210/api/User/Register', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(credentials)
    });

    if (response.ok) {
      const data = await response.json();
      localStorage.setItem('userToken', data.token);

      alert('Успешная регистрация');
      window.location.href = 'user/product.html';
    } else {
      console.log(await response.text());
      throw new Error('Ошибка регистрации');
    }
  } catch (error) {
    console.error(error);
    alert('Ошибка регистрации');
  }
}

```

Рисунок 39 – функция регистрации пользователя

```

async function getUserInfo() {
  document.getElementById('password').value = '';
  document.getElementById('confirmedPassword').value = '';

  var decodedToken = JSON.parse(atob(localStorage.getItem('userToken').split('.')[1]));
  const uid = decodedToken.nameid;

  try {
    const response = await fetch('https://localhost:7210/api/User/GetUserInfo?userId=${uid}', {
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${localStorage.getItem('userToken')}`
      },
    });

    if (response.ok) {
      const data = await response.json();

      document.getElementById('name').value = data.name;
      document.getElementById('login').value = data.login;
      document.getElementById('email').value = data.email;
    } else {
      console.log(await response.text());
      throw new Error('Что-то пошло не так');
    }
  } catch (error) {
    console.error(error);
    alert('Ошибка');
  }
}

```

Рисунок 40 – функция получения информации о пользователе

```

async function getAllUsers() {
  try {
    const response = await fetch('https://localhost:7210/api/User/GetAllUsers', {
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${localStorage.getItem('userToken')}`,
      },
    });

    if (response.ok) {
      const data = await response.json();

      const userTable = document.getElementById('userTable');
      userTable.innerHTML = '';

      data.forEach(user => {
        let email;

        if (!user.email) {
          email = '';
        } else {
          email = user.email;
        }

        const row = document.createElement('tr');
        row.innerHTML = `
          <td>${user.userId}</td>
          <td>${user.name}</td>
          <td>${user.login}</td>
          <td>${email}</td>
          <td>${user.isAdmin}</td>
          <td style="text-align: center;"><input type="checkbox" value="${user.userId}"></td>`;
        userTable.appendChild(row);
      });
    } else {
      console.log(await response.text());
      throw new Error('Что-то пошло не так');
    }
  } catch (error) {
    console.error(error);
  }
}

```

Рисунок 41 – функция получения списка всех пользователей

```

async function deleteUser() {
  const selectedCheckboxes = document.querySelectorAll('#userTable input[type="checkbox"]:checked');

  if (selectedCheckboxes.length === 0) {
    alert('Выберите хотя бы одного пользователя');
    return;
  }

  let uids = [];
  for (let i = 0; i < selectedCheckboxes.length; i++) {
    if (selectedCheckboxes[i].checked) {
      uids.push(selectedCheckboxes[i].value);
    }
  }

  uids.forEach(async (uid) => {
    try {
      const response = await fetch('https://localhost:7210/api/User/DeleteUser?userId=${uid}', {
        method: 'DELETE',
        headers: {
          'Content-Type': 'application/json',
          Authorization: `Bearer ${localStorage.getItem('userToken')}`,
        },
      });

      const data = await response.text();

      if (response.ok) {
        console.log(data);
        getAllUsers();
      } else {
        console.log(data);
        throw new Error('Что-то пошло не так');
      }
    } catch (error) {
      console.error(error);
      alert('Ошибка');
    }
  });
}

```

Рисунок 42 – функция удаления пользователя



```

async function getAllProducts() {
  try {
    const response = await fetch('https://localhost:7210/api/Product/GetAllProducts', {
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${localStorage.getItem('userToken')}`,
      },
    });

    if (response.ok) {
      const data = await response.json();

      const productTable = document.getElementById('productTable');
      productTable.innerHTML = '';

      data.forEach(product => {
        const row = document.createElement('tr');
        row.innerHTML = `
          <td>${product.productId}</td>
          <td>${product.name}</td>
          <td>${product.manufacturers}</td>
          <td>${product.types}</td>
          <td>${product.price}</td>
          <td style="text-align: center;"><input type="checkbox" value="${product.productId}"></td>`;
        productTable.appendChild(row);
      });
    } else {
      console.log(await response.text());
      location.reload(true);
      throw new Error('Что-то пошло не так!');
    }
  } catch (error) {
    console.error(error);
  }
}

```

Рисунок 43 – функция получения списка всех товаров

```

async function createProduct() {
  const productName = document.getElementById('productName').value;
  const productPrice = document.getElementById('productPrice').value;
  const productManufacturers = document.getElementById("productManufacturers").value;
  const productTypes = document.getElementById("productTypes").value;
  const productImage = document.getElementById("productImage").value;

  if (!productName || !productPrice) {
    alert('Заполните необходимые поля!');
    return;
  }

  const productInfo = {
    name: productName,
    price: productPrice,
    manufacturers: productManufacturers.split(", "),
    types: productTypes.split(", "),
    image: productImage
  };

  try {
    const response = await fetch('https://localhost:7210/api/Product/CreateProduct', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${localStorage.getItem('userToken')}`
      },
      body: JSON.stringify(productInfo)
    });

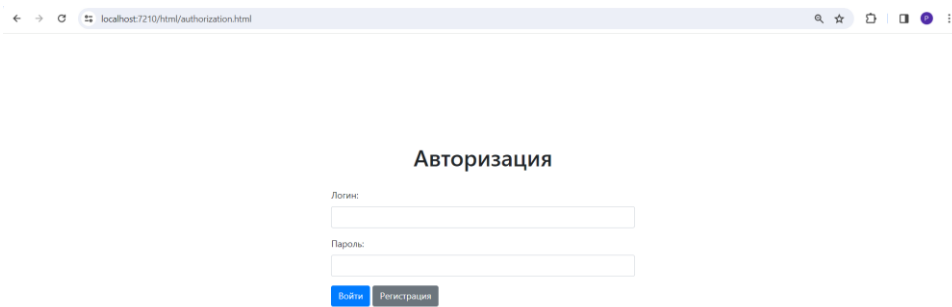
    const data = await response.text();

    if (response.ok) {
      console.log(data);
      getAllProducts();
    } else {
      console.log(data);
      throw new Error('Что-то пошло не так!');
    }
  } catch (error) {
    console.error(error);
    alert('Ошибка!');
  }
}

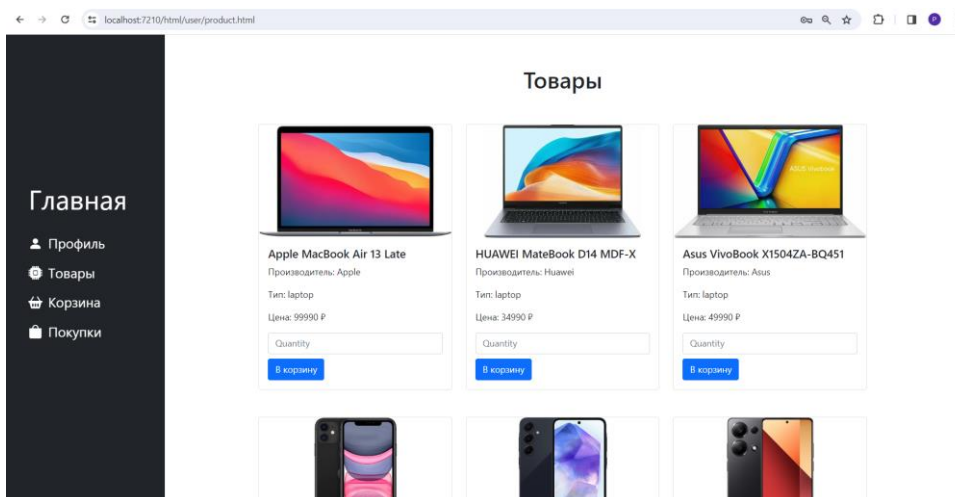
```

Рисунок 44 – функция создания товара

Далее будут предоставлены скриншоты работающей информационной системы:



*Рисунок 45 – страница авторизации*



*Рисунок 46 – страница товаров*

localhost:7210/html/user/userInfo.html

### Данные пользователя

Полное имя:

Email:

Логин:

Пароль:

Подтвердите пароль:

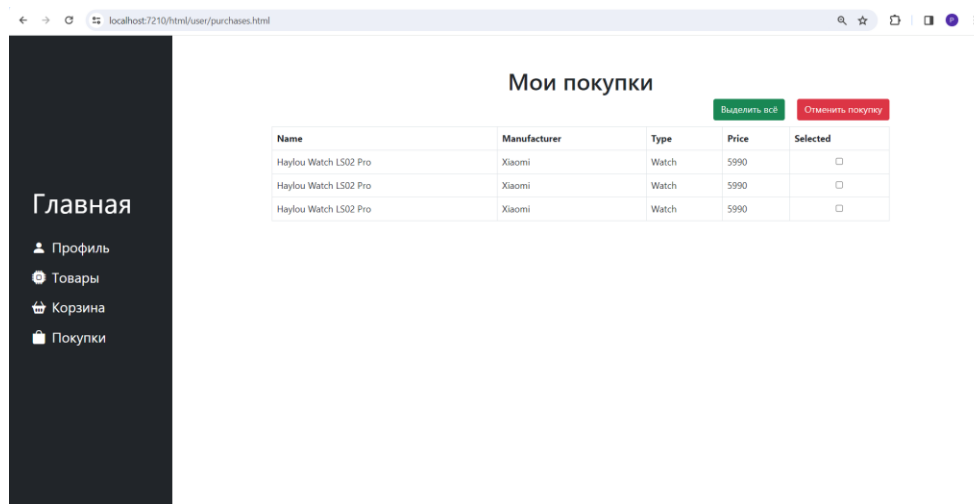
Рисунок 47 – страница профиля

localhost:7210/html/user/cart.html

### Моя Корзина

Name	Manufacturer	Type	Price	Selected
Apple MacBook Air 13 Late	Apple	laptop	99990	<input type="checkbox"/>
Samsung Galaxy A55	Samsung	phone	37990	<input type="checkbox"/>

Рисунок 48 – страница корзины



*Рисунок 49 – страница покупок*

## **ЗАКЛЮЧЕНИЕ**

Итак, была создана информационная система “онлайн магазин электронной техники”, основанная на кроссплатформенных WEB технологиях, которая предоставляет функциональность для покупки электронной техники онлайн.

При реализации проекта использовались средства разработки от Microsoft Visual Studio для создания API на платформе ASP.NET Core. Кроме того, в проекте применялась система управления базами данных SQL Server 2019 Express. В рамках проекта были приобретены навыки в проектировании WEB API, работе с фреймворком ASP.NET Core и Fetch API.

В результате, поставленная цель на проект была достигнута.

## Библиографический список

1. Официальная документация Entity Framework [Электронный ресурс].  
URL: <https://learn.microsoft.com/ru-ru/ef/>
2. Официальная документация ASP.NET Core [Электронный ресурс].  
URL: <https://learn.microsoft.com/ru-ru/ef/>
3. Официальная документация API Fluent [Электронный ресурс].  
URL: <https://learn.microsoft.com/ru-ru/ef/ef6/modeling/code-first/fluent/relationships>
4. Официальная документация Fetch API [Электронный ресурс].  
URL: [https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)
5. JWT Аутентификация в ASP.NET [Электронный ресурс].  
URL: <https://dev.to/fabriziobagala/jwt-authentication-in-aspnet-13ma>