

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

*Факультет Информационных технологий
Кафедра Информатики и информационных технологий*

направление подготовки

09.03.02 «Информационные системы и технологии»

КУРСОВОЙ ПРОЕКТ

Дисциплина: Технологии кроссплатформенного программирования

Тема: Разработка сайта и информационной системы, обеспечивающей
связь сайта с базой данных благодаря REST API

Выполнил(а): студент(ка) группы **201-724**

Зупник Егор Вячеславович

(Фамилия И.О.)

Дата, подпись _____

(Дата)

(Подпись)

Проверил: _____

(Фамилия И.О., степень, звание)

Дата, подпись _____

(Дата)

(Подпись)

Замечания: _____

Москва

2022

Оглавление

ВВЕДЕНИЕ	3
ГЛАВА 1. ПРОЕКТИРОВАНИЕ	4
Описание предметной области	4
Выбор инструментов	4
ГЛАВА 2. РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ.	7
Проектирования и физическая реализация базы данных на СУБД.	7
Разработка программного продукта	15
ЗАКЛЮЧЕНИЕ	53
Библиографический список	54

ВВЕДЕНИЕ

Тема проекта – сайт для продажи популярных компьютерных частей.

В современном мире практически у всех есть ПК – он стал немалой частью нашей жизни. Одним из удобств ПК является его модулярность: практически любой элемент можно вытащить и заменить. Главной проблемой является ограниченная доступность подобных деталей по отдельности: очень сложно найти магазин, который согласится продать деталь отдельно, а также доставить её. Именно поэтому появилась идея создать сайт, обеспечивающий заказ и доставку отдельных деталей для ПК.

Цель – создание сайта для продажи деталей с подключением к БД.

Исходя из поставленной цели, были сформулированы следующие задачи:

1. Изучение предметной области;
2. Выбор и обоснование инструментов разработки;
3. Проектирование базы данных;
4. Реализация физической модели данных.

ГЛАВА 1. ПРОЕКТИРОВАНИЕ

Описание предметной области

Из-за большого влияния ПК на мир людей его практически постоянно используют. С развитием технологий и износом деталей разрабатываются отдельные элементы нужные для работы ПК, а также развита модулярная архитектура устройства самого компьютера, позволяющая свободно подключать, отключать, и, соответственно, заменять отдельные детали.

Рынок компьютерных деталей развивается очень быстро, до такой степени что детали, которые в начале года имели очень высокую цену, к концу года могут быть намного дороже из-за появления новых ещё более мощных компонентом. Благодаря этому бизнес компьютерных деталей имеет гарантию постоянного массового присутствия на рынке.

Таким образом, было принято решение разработать сайт для продажи компьютерных деталей.

Выбор инструментов

Проект будет представлять из себя сайт, так как в современном мире веб-сайт является самым удобным и универсальным способом ведения бизнеса.

Формат веб-приложения, или сайта, подразумевает кроссплатформенный функционал: сайт должен быть открываем и с мобильного устройства, и с компьютера. Для этого на этапе разработке было решено, что этот сайт будет использовать платформу visual studio, он будет написан на фреймворке с# ASP.Net core.

ASP.NET core —свободно-распространяемый кроссплатформенный фреймворк для создания веб-приложений на платформе .NET с открытым исходным кодом. Данная платформа разрабатывается компанией Майкрософт совместно с сообществом и имеет большую производительность по сравнению с ASP.NET. Имеет модульную структуру и совместима с такими операционными системами как Windows, Linux и macOS.

C# — объектно-ориентированный язык программирования общего назначения. Разработан в 1998—2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота как язык разработки приложений для платформы Microsoft .NET Framework и .NET Core. C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, переменные, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Для создания баз данных в роли СУБД в данном проекте выступает Microsoft SQL, который используется по умолчанию Visual Studio и проектированием которого будет заниматься EntityFramework Code First.

Microsoft SQL Server — система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase.

ADO.NET Entity Framework (EF) — объектно-ориентированная технология доступа к данным, является object-relational mapping (ORM) решением для .NET Framework от Microsoft. Предоставляет возможность взаимодействия с объектами как посредством LINQ в виде LINQ to Entities, так и с использованием Entity SQL. Для облегчения построения web-решений используется как ADO.NET Data Services (Astoria), так и связка из Windows Communication Foundation и Windows Presentation Foundation, позволяющая строить многоуровневые приложения, реализуя один из шаблонов проектирования MVC, MVP или MVVM.

Сайт будет собран при помощи смеси структуры MVC и методов специально разработанного API.

Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и

контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.

Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.

Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

API (Application Programming Interface) — описание способов взаимодействия одной компьютерной программы с другими. Обычно входит в описание какого-либо интернет-протокола (например, SCIM), программного каркаса (фреймворка) или стандарта вызовов функций операционной системы. Часто реализуется отдельной программной библиотекой или сервисом операционной системы. Используется программистами при написании всевозможных приложений.

IIS (Internet Information Services) – это проприетарный набор серверов для нескольких служб интернета от компании Microsoft. Основным компонентом IIS является веб-сервер, который позволяет размещать в Интернете сайты. IIS поддерживает протоколы HTTP, HTTPS, FTP, POP3, SMTP, NNTP. По данным компании Netcraft на июнь 2015 года, почти 22 млн сайтов обслуживаются веб-сервером IIS, что составляет 12,32 % от общего числа веб-сайтов. Он был выбран в качестве среды разработки баз данных, созданных EF в Microsoft SQL Server.

Для проверки запросов перед подключением к приложению будет использован Swagger. Swagger-это язык описания интерфейса для описания RESTful API, выраженных с помощью JSON. Swagger используется вместе с набором программных средств с открытым исходным кодом для проектирования, создания, документирования и использования веб-служб RESTful. Swagger включает в себя автоматизированную документацию, генерацию кода (на многие языки программирования) и генерацию тестовых примеров.

ГЛАВА 2. РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ.

Проектирования и физическая реализация базы данных на СУБД.

Для реализации данного проекта была сгенерирована база данных при помощи метода Entity Network Code First. Хотя во время разработки эта база данных постоянно менялась, финальная версия БД состоит из, состоит из двенадцати таблиц. Таблица реляционной базы данных — это совокупность связанных данных, хранящихся в структурированном виде в базе данных. Таблицы базы данных на рис.1.

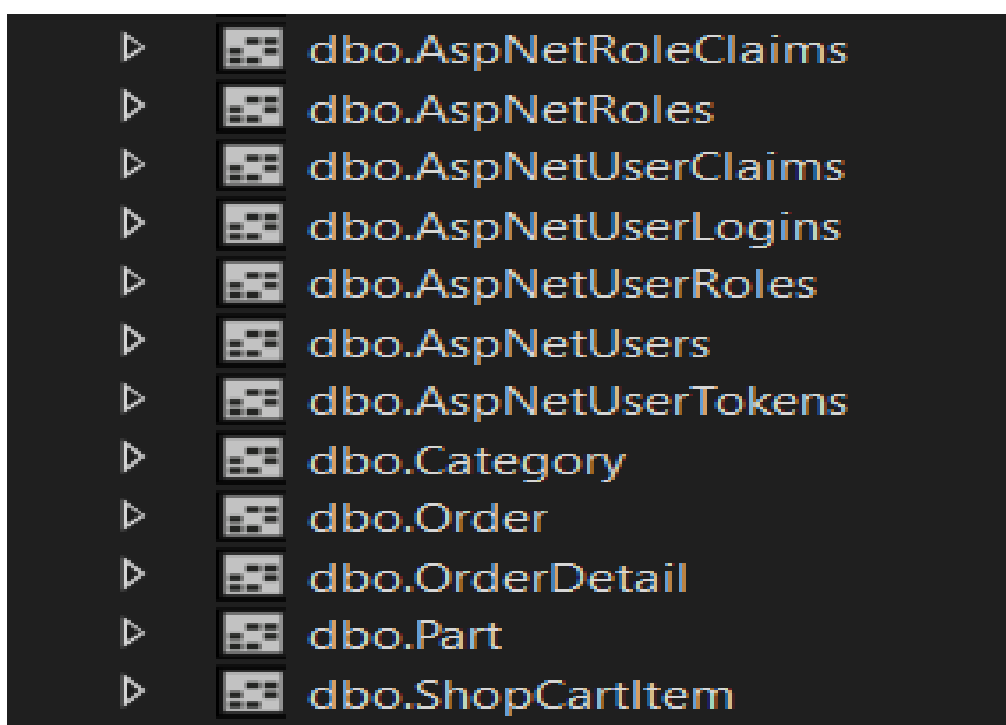


Рисунок 1. Таблицы БД.

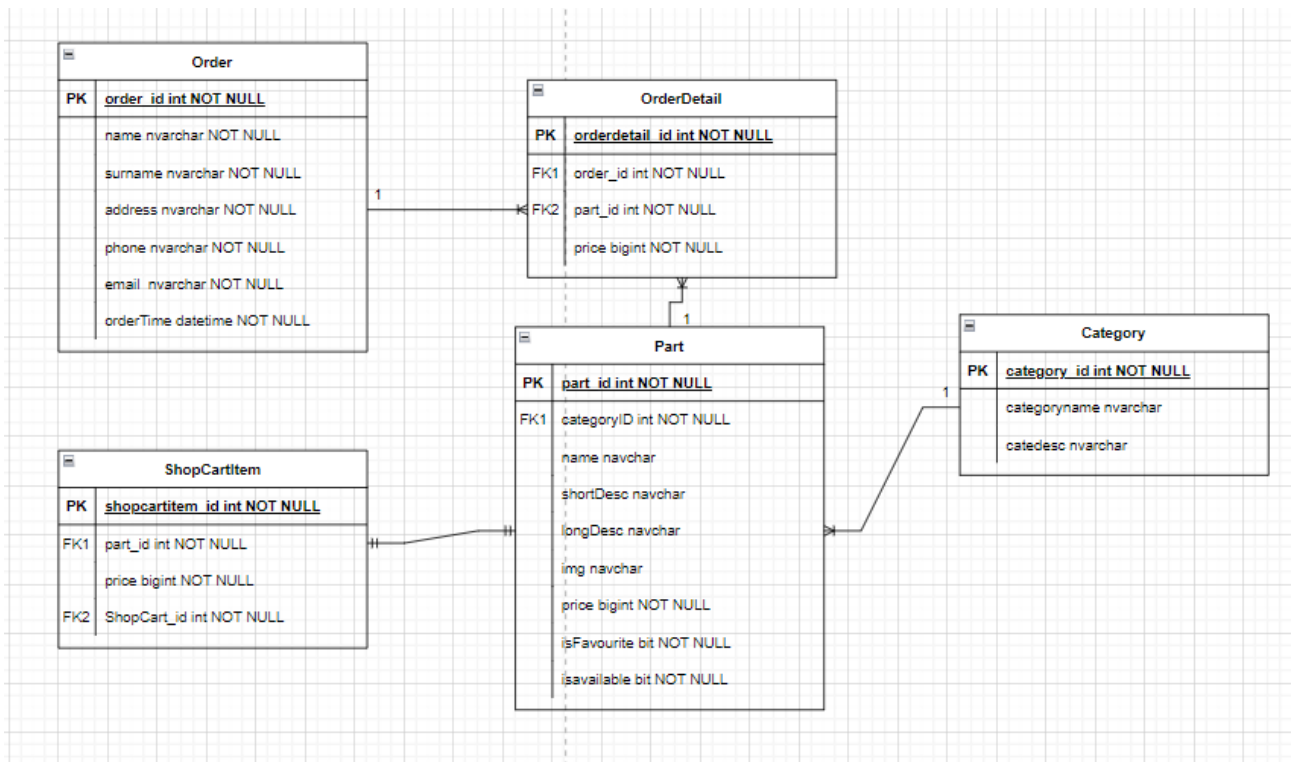


Рисунок 2. ER-диаграмма БД.

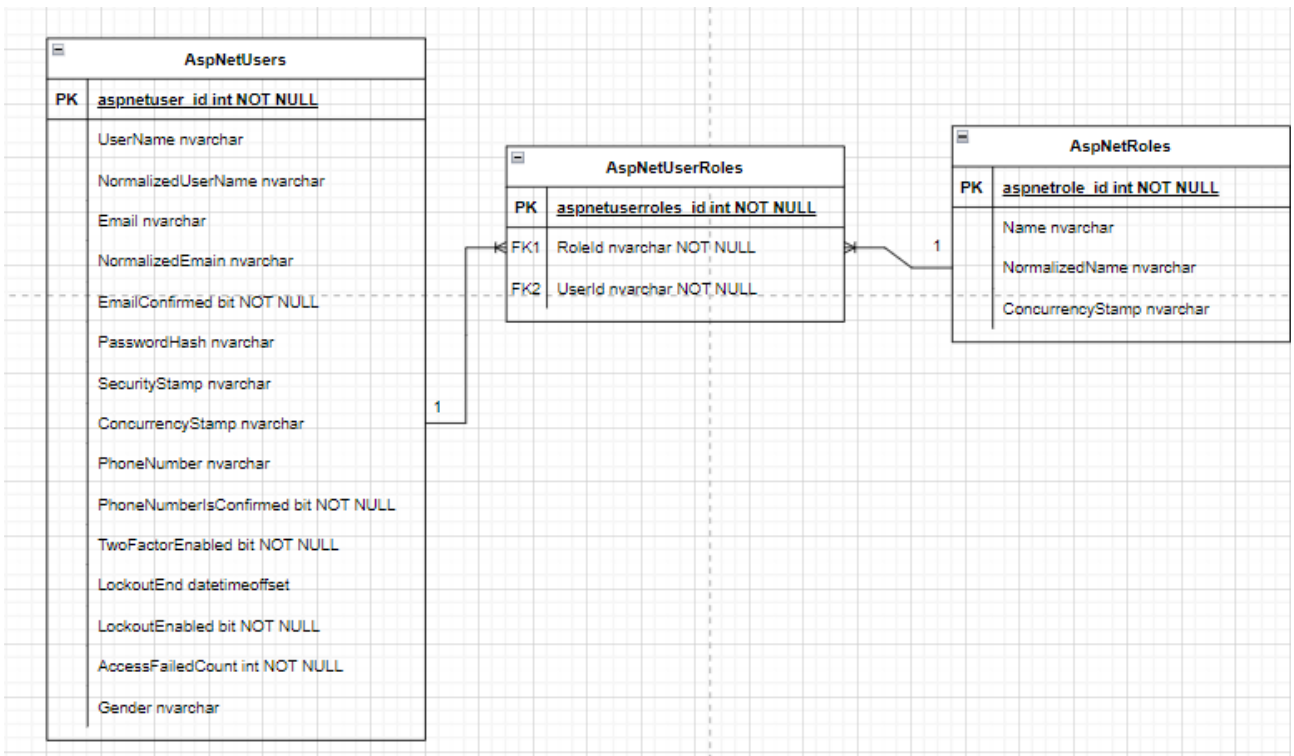


Рисунок 3 ER-диаграмма части БД, предоставленной IdentityCore, с которым проводится прямое взаимодействие.

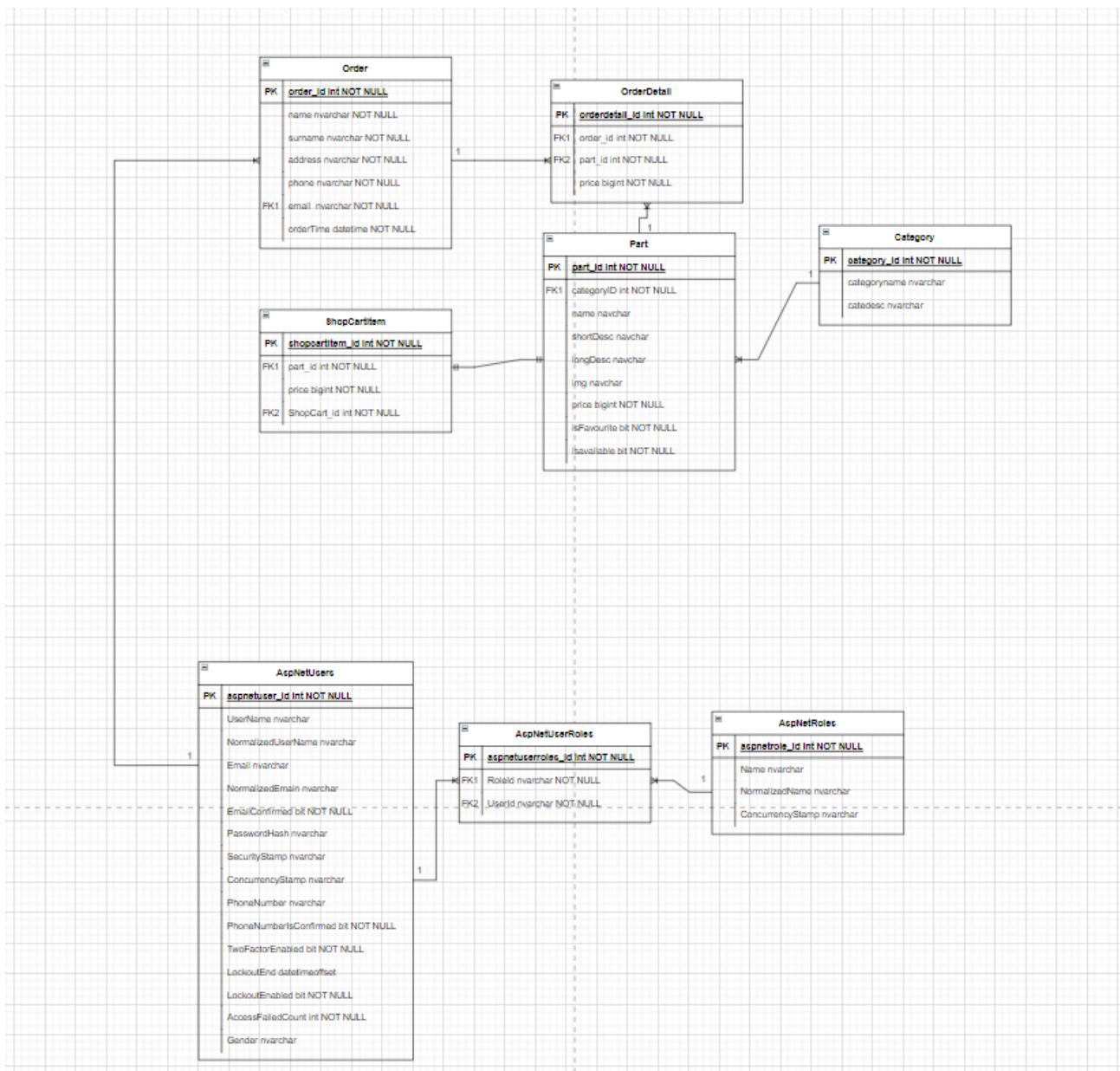



Рисунок 4 Общий вид ER диаграммы БД.

Далее представлены структуры всех таблиц с пояснением к каждой.

Таблица Part – части – таблица, в которой хранится вся информация о товарах в каталоге магазина. Данная таблица содержит:

- Name – Название товара;
- ShortDesc – краткое описание товара;
- LongDesc – полное описание товара;
- img – ссылка на изображение товара;
- Price – цена товара;
- isFavourite – является ли товар популярным (да/нет)
- isAvailable – является ли товар доступным к покупке (да/нет)

- categoryID – внешний ключ, требуемый для определения категории товара (Оперативная память, графическая карта или процессор)


	Имя	Тип данных	Допустимы значения NULL
	id	int	<input type="checkbox"/>
	name	nvarchar(MAX)	<input checked="" type="checkbox"/>
	shortDesc	nvarchar(MAX)	<input checked="" type="checkbox"/>
	longDesc	nvarchar(MAX)	<input checked="" type="checkbox"/>
	img	nvarchar(MAX)	<input checked="" type="checkbox"/>
	price	bigint	<input type="checkbox"/>
	isFavourite	bit	<input type="checkbox"/>
	isavailable	bit	<input type="checkbox"/>
	categoryID	int	<input type="checkbox"/>

	id	name	shortDesc	longDesc	img	price	isFavourite	isavailable	categoryID
▶	1	NVIDIA Geforce...	Видеокарта се...	Видеокарта се...	/img/Nvidia-Ge...	30190	True	True	1
	2	NVIDIA Geforce...	Видеокарта се...	Видеокарта се...	/img/GeForce_...	21000	False	False	1
	3	Intel Core i5-10...	Процессор Int...	Процессор Int...	/img/intel-core...	11990	False	True	2
	4	Оперативная п...	Четвёртое пок...	Четвёртое пок...	/img/1_small.jpg	2190	False	True	3
	5	Оперативная п...	Четвёртое пок...	Четвёртое пок...	/img/14800.970...	4835	False	False	3
	6	Radeon RX 580X	Radeon	Radeon RX	763e267bf28a1...	12000	True	True	1
	7	AwesomeProdu...	dasdasdas	dadsa	4Съемка иност...	699669	True	True	1
	8	EtoBaza	what	thehell	4Съемка иност...	545345	True	True	2
	9	Test	test	test	Nvidia-GeForce...	12331321	True	True	1
	10	Test	test	TEST	string	321313	True	True	1
	11	FinalTest	testfinal	adasdasd	intel-core-i5f-1...	66666	True	True	2
	12	FinalFINALtest	testtt	ttttt	intel-core-i5f-1...	666777	False	False	3
⊞	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Рисунок 5. Таблица Part.

Таблица Category – категории – таблица, в которой хранится информация о всех категориях товаров, которые доступны в каталоге сайта магазина. Данная таблица содержит:

- CategoryName – название категории;
- CateDesc – описание категории;

	Имя	Тип данных	Допустимы значения NULL	По умолчанию
	id	int	<input type="checkbox"/>	
	categoryname	nvarchar(MAX)	<input checked="" type="checkbox"/>	
	catedesc	nvarchar(MAX)	<input checked="" type="checkbox"/>	
			<input type="checkbox"/>	



	id	categoryname	catedesc
	1	Графическая к...	Позволяет ото...
	2	Процессор	Главный мозг ...
	3	Оперативная п...	Обеспечивает ...
	NULL	NULL	NULL

Рисунок 6. Таблица Category.

Таблица Order – заказы – таблица, в которой хранятся данные об оформленных заказах на сайте.

Данная таблица содержит:

- Name – имя заказчика;
- Surname – фамилия заказчика;
- Address – адрес заказчика.
- Phone – телефон заказчика.
- Email – электронная почта заказчика
- OrderTime – (автоматически заполняется) дата и время заказа

	Имя	Тип данных	Допустимы значения NULL	По умолчанию
	id	int	<input type="checkbox"/>	
	name	nvarchar(25)	<input type="checkbox"/>	
	surname	nvarchar(25)	<input type="checkbox"/>	
	address	nvarchar(35)	<input type="checkbox"/>	
	phone	nvarchar(20)	<input type="checkbox"/>	
	email	nvarchar(25)	<input type="checkbox"/>	
	orderTime	datetime2(7)	<input type="checkbox"/>	
			<input type="checkbox"/>	

	id	name	surname	address	phone	email	orderTime
▶	8	aaaaaaaa	asd	das	sad	dsa@dasd	15.12.2022 12:2...
	9	Egor	Zupnik	daadasdasdasd...	79165236172	zupnikegor@g...	17.12.2022 12:3...
	10	Egorr	goRE	leningradskaya ...	1231242342341	zupnikegg@gm...	22.12.2022 21:1...
	11	Egoriyok	Egoriyok	12th downer str...	21312312312	zupnikegg@gm...	22.12.2022 23:0...
	12	ImCool	IWork	69420 skateboa...	4206942069	zupnikegg@gm...	22.12.2022 23:0...
	13	dsa	das	dsad	dadsa	zupnikegg@gm...	01.01.2023 12:4...
⊗	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Рисунок 7. Таблица Order

Таблицы OrderDetail и ShopCartItem являются дополнительными таблицами.

ShopCartItem это таблица одного из элементов и имеет 3 столбца:

- partId – содержит ID отдельной части в корзине.
- Price – цена отдельной части в корзине.
- ShopCartId – ID корзины, к которой этот отдельный элемент принадлежит

	id	int	<input type="checkbox"/>	
	partid	int	<input checked="" type="checkbox"/>	
	price	bigint	<input type="checkbox"/>	
	ShopCartId	nvarchar(MAX)	<input checked="" type="checkbox"/>	
			<input type="checkbox"/>	

Рисунок 8. Таблица ShopCartItem

OrderDetail – таблица изображающая отдельный элемент заказа, содержит 3 столбца:

- partId – содержит ID отдельной части в корзине
- orderId – содержит ID заказа, к которому относится этот элемент
- Price – цена отдельного элемента в заказе

id	int	<input type="checkbox"/>
orderid	int	<input checked="" type="checkbox"/>
partid	int	<input checked="" type="checkbox"/>
price	bigint	<input type="checkbox"/>

Рисунок 9. Таблица OrderDetail

Таблицы AspNetRoleClaims, AspNetRoles, AspNetUserClaims, AspNetUserLogins, AspNetUserRoles, AspNetUsers, AspNetUserTokens – автоматически сгенерированные таблицы, требуемые для работы плагина AspNet Identity Core.

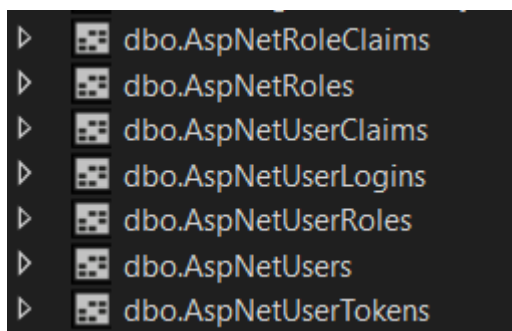


Рисунок 10. Сгенерированные таблицы Identity Core.

В данном случае работа будет вестись напрямую с таблицами AspNetUsers и AspNetRoles.

Таблица AspNetUsers содержит в себе список всех зарегистрированных на сайт пользователей. У неё есть всего 15 таблиц, но прямое взаимодействие будет проводиться всего с 4:

- UserName – Имя/«Никнейм» пользователя. В данном случае он будет принимать значения столбца Email.
- Email – адрес электронной почты пользователя
- PasswordHash – поле, получившая захешированную версию пароля, используется при попытке входа пользователя в аккаунт.

Id	nvarchar(450)	<input type="checkbox"/>	
UserName	nvarchar(256)	<input checked="" type="checkbox"/>	
NormalizedUserName	nvarchar(256)	<input checked="" type="checkbox"/>	
Email	nvarchar(256)	<input checked="" type="checkbox"/>	
NormalizedEmail	nvarchar(256)	<input checked="" type="checkbox"/>	
EmailConfirmed	bit	<input type="checkbox"/>	
PasswordHash	nvarchar(MAX)	<input checked="" type="checkbox"/>	
SecurityStamp	nvarchar(MAX)	<input checked="" type="checkbox"/>	
ConcurrencyStamp	nvarchar(MAX)	<input checked="" type="checkbox"/>	
PhoneNumber	nvarchar(MAX)	<input checked="" type="checkbox"/>	
PhoneNumberConfirmed	bit	<input type="checkbox"/>	
TwoFactorEnabled	bit	<input type="checkbox"/>	
LockoutEnd	datetimeoffset(7)	<input checked="" type="checkbox"/>	
LockoutEnabled	bit	<input type="checkbox"/>	
AccessFailedCount	int	<input type="checkbox"/>	
Gender	nvarchar(MAX)	<input checked="" type="checkbox"/>	
		<input type="checkbox"/>	

Рисунок 11. Таблица *AspNetUsers*.

Таблица *AspNetRoles* содержит в себе список всех существующих на сайте ролей. У неё есть всего 3 таблицы, но напрямую в данном случае взаимодействие будет с одной: таблицей *Name*, которая содержит в себе название роли.

Id	nvarchar(450)	<input type="checkbox"/>	
Name	nvarchar(256)	<input checked="" type="checkbox"/>	
NormalizedName	nvarchar(256)	<input checked="" type="checkbox"/>	
ConcurrencyStamp	nvarchar(MAX)	<input checked="" type="checkbox"/>	
		<input type="checkbox"/>	

Рисунок 12. Таблица *AspNetRoles*.

Разработка программного продукта

Рассмотрим, как данный проект был реализован, для это необходимо обратиться к основным частям программного кода (язык программирования – с#). Далее представлены главные функции приложения в программном виде. Изначально сайт представлял из себя полностью монолитный MVC проект, который позже был конвертирован под использование API.

Поскольку это сайт, который будет использовать API, проект в Visual Studio будет состоять из нескольких частей:

- Web-приложение, служащее фронтом, ответственное за функционал корзины магазина а также имеющая контроллеры, модели и виды для фронтенда сайта.
- Web-API, ответственная за подавляющее большинство обращений сайта к прикреплённому к нему БД. Он содержит модели и контроллеры, позволяющие передавать информацию из БД сайту. Данные передаются в виде сериализованных файлов JSON.
- Contract – библиотека классов, содержащая контракты – модели которые использует API.
- DataAccess – библиотека классов, содержащая БД и все связанные с ней классы а также модели, которые позже конвертируются в фронтенд-приложении.
- Client – библиотека классов, содержащее класс Web-клиента, который делает запросы из фронтенда бэкэнду.

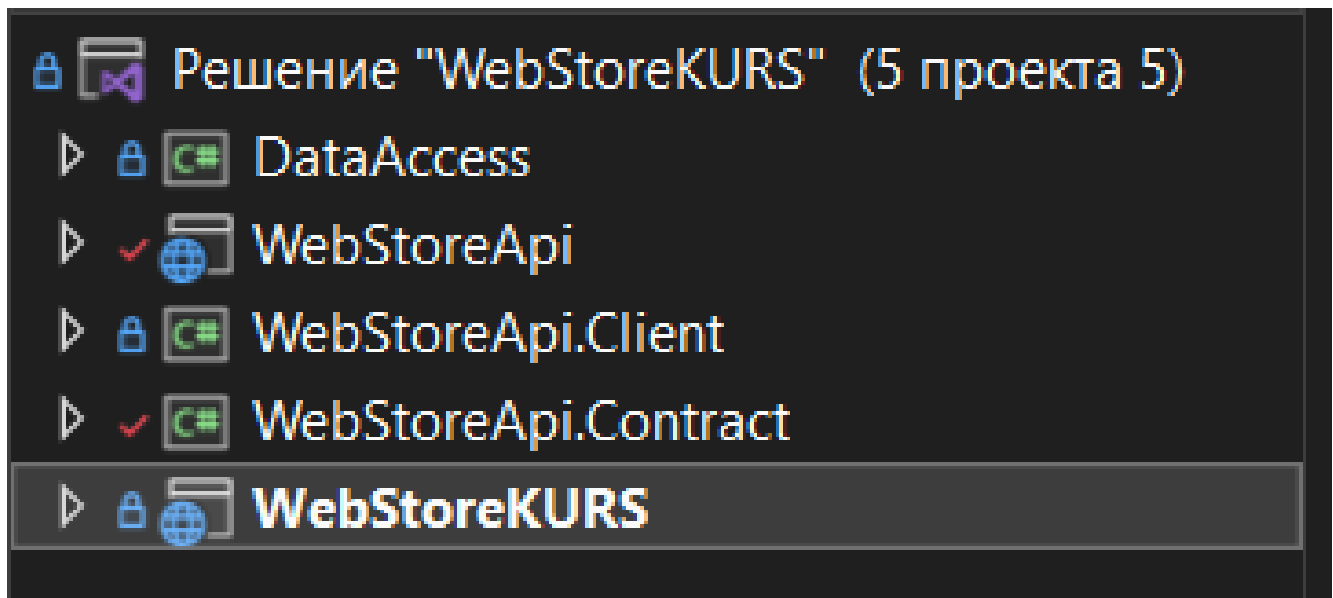


Рисунок 13. Содержимое проекта Visual Studio.

Рассмотрим основной код фронтенд-приложения.

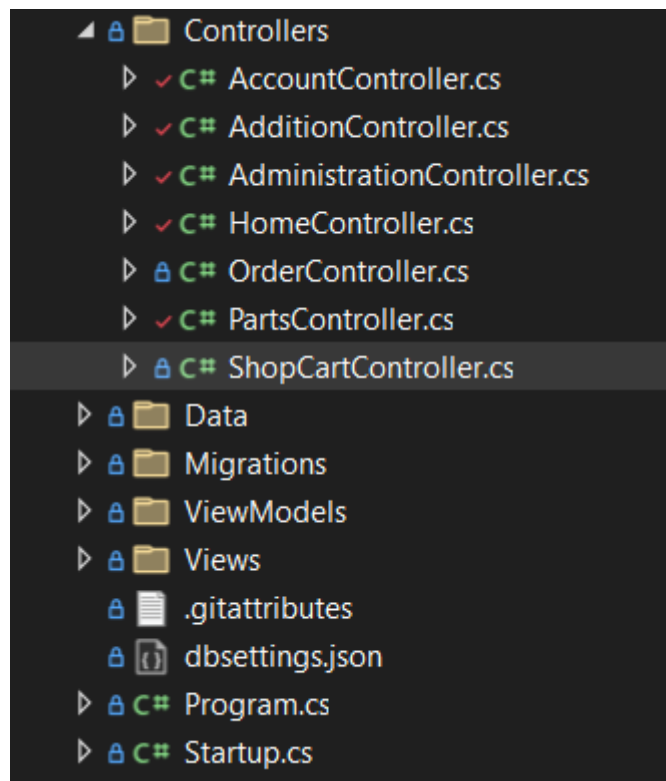


Рисунок 14. Содержимое приложения-фронтенда.

Приложение содержит в себе все нужные элементы для архитектуры MVC: Контроллеры (папка Controllers), Модели (содержимое папки Data) и модели (папка ViewModels).

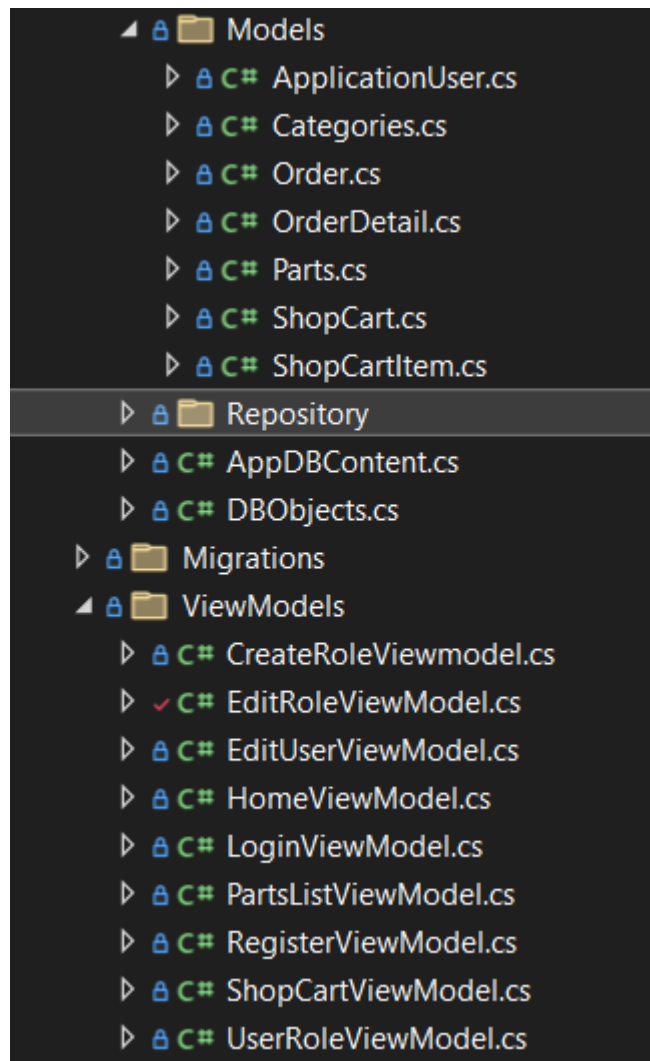


Рисунок 15. Содержимое папок.

Рассмотрим контроллеры.

AccountController – контроллер, отвечающий за работу с пользователями. В нём есть функции входа и выхода из аккаунта, а также регистрация пользователя, которая требует взаимодействия с БД, поэтому в этой функции выполняется вызов соответствующего POST запроса в API веб-клиентом.

```
[HttpPost]
Ссылка: 0
public async Task<IActionResult> Logout()
{
    await signInManager.SignOutAsync();
    return RedirectToAction("TopSellers", "Home");
}
```

Рисунок 16. Функция выхода из аккаунта.

```

[HttpPost]
[AllowAnonymous]
Ссылка 0
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl)
{
    if (ModelState.IsValid)
    {
        var result = await signInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, false);

        if (result.Succeeded)
        {
            if (!string.IsNullOrEmpty(returnUrl) && Url.IsLocalUrl(returnUrl))
            {
                return Redirect(returnUrl);
            }
            else
            {
                return RedirectToAction("TopSellers", "Home");
            }
        }

        ModelState.AddModelError(string.Empty, "Invalid login attempt");
    }
    return View(model);
}

```

Рисунок 17. Функция входа в существующий аккаунт (POST-действие).

```

[HttpPost]
[AllowAnonymous]
Ссылка 0
public async Task<IActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser { UserName = model.Email, Email = model.Email, Gender = model.Gender };
        var reguser = new CreateUserContract { Email = model.Email, Gender = model.Gender, Password = model.Password, ConfirmPassword = model.ConfirmPassword };
        var result = await _client.CreateUser(reguser).ConfigureAwait(false);

        if (result.IsSuccessStatusCode)
        {
            if (signInManager.IsSignedIn(user) && user.IsInRole("Admin"))
            {
                return RedirectToAction("ListUsers", "Administration");
            }
            await signInManager.SignInAsync(user, isPersistent: false);
            return RedirectToAction("TopSellers", "Home");
        }

        /*foreach (var error in result.Errors)
        {
            ModelState.AddModelError("", error.Description);
        }*/
    }
    return View(model);
}

```

Рисунок 18. Функция регистрации аккаунта (POST-запрос к API).

Рассмотрим AdditionController. Этот контроллер ответственен за POST запросы к API для того, чтобы добавить новый товар в БД, используя введенные данные на специальной странице. Для использования функций этого контроллера пользователь должен иметь роль администратора.

```

[HttpPost]
[Authorize]
Ссылка: 0
public async Task<IActionResult> addPart(Parts part)
{
    Parts partToAdd = new Parts();
    partToAdd.name = part.name;
    partToAdd.isavailable = part.isavailable;
    partToAdd.isFavourite = part.isFavourite;
    partToAdd.shortDesc = part.shortDesc;
    partToAdd.longDesc = part.longDesc;
    partToAdd.price = part.price;
    partToAdd.img = part.img;
    partToAdd.categoryID = part.categoryID;
    PartsContract PartToAdd = new PartsContract
    {
        name = part.name,
        isavailable = part.isavailable,
        isFavourite = part.isFavourite,
        shortDesc = part.shortDesc,
        longDesc = part.longDesc,
        price = part.price,
        img = part.img,
        categoryID = part.categoryID
    };
    /*IEnumerable<Categories> categories = null;
    categories = allCategories.AllCategories.OrderBy(i => i.id);
    switch (partToAdd.categoryID)
    {
        case 1:
            partToAdd.Category = categories.ElementAt(0);
            break;
        case 2:
            partToAdd.Category = categories.ElementAt(1);
            break;
        case 3:
            partToAdd.Category = categories.ElementAt(2);
            break;
    }*/
    var result = await _client.AddPart(PartToAdd).ConfigureAwait(false);
    if (result.IsSuccessStatusCode)
    {
        return RedirectToAction("Completion");
    }
    else
    {
        return RedirectToAction("Failure");
    }
    return View(partToAdd);
}

```

Рисунок 19. Функция создания товара, делающая POST-запрос с данными со страницы.

Этот контроллер, как и все остальные, использует клиент для общения с API, а затем использует результат который возвращается после завершения запроса чтобы решить что делать дальше. В данном случае возвращается код операции. Если код успешный (код 200), то контроллер загружает страницу,

указывающую успех операции, иначе контроллер загрузит страницу, указывающую на то, что во время операции что-то пошло не так.

Рассмотрим `AdministrationController`. Для использования функций этого контроллера пользователю обязательно надо иметь статус администратора. В функционал `AdministrationController` входит просмотр ролей, создание новых ролей или изменение, а также удаление существующих ролей. Также администраторы имеют право добавлять или убирать роли у отдельных пользователей. Кроме этого, у администраторов также есть возможность также самостоятельно зарегистрировать/создать нового пользователя. Для всего этого используются запросы к разработанному API, разработанному и соответственно прикреплённому к сайту.

```

}
[HttpPost]
[Authorize]
Ссылка: 0
public async Task<IActionResult> CreateRole(CreateRoleViewModel model)
{
    if (ModelState.IsValid)
    {
        CreateRoleContract contract = new CreateRoleContract
        {
            RoleName = model.RoleName
        };

        var result = await _client.AddRole(contract).ConfigureAwait(false);

        if (result.IsSuccessStatusCode)
        {
            return RedirectToAction("ListRoles");
        }
    }

    return View(model);
}

```

Рисунок 20. `CreateRole` – вызов к методу API, создающий новую роль

```

[HttpGet]
Ссылка: 0
public async Task<IActionResult> EditRole(string id)
{
    var role = await _client.GetEditRole(id).ConfigureAwait(false);
    if (role.Id==null)
    {
        ViewBag.ErrorMessage = $"Роль с Id = {id} не найдена.";
        return View("NotFound");
    }

    /*if (role==null)
    {
        ViewBag.ErrorMessage = $"Роль с Id = {id} не найдена.";
        return View("NotFound");
    }*/

    var model = new EditRoleViewModel
    {
        Id = role.Id,
        RoleName = role.RoleName,
        Users = role.Users,
        UserIds = role.UserIds
    };

    return View(model);
}

```

Рисунок 21. GetEditRole – вызов к методу API, получающий нужную модель нужной роли

```

[HttpPost]
Ссылка: 0
public async Task<IActionResult> EditRole(EditRoleViewModel model)
{
    var role = await _client.GetEditRole(model.Id).ConfigureAwait(false);

    if (role.Id == null)
    {
        ViewBag.ErrorMessage = $"Роль с Id = {model.Id} не найдена.";
        return View("NotFound");
    }
    else
    {
        role.RoleName = model.RoleName;
        role.Users = model.Users;
        role.UserIds = model.UserIds;
        if (ModelState.IsValid)
        {
            var result = await _client.UpdateRoles(role);

            if (result.IsSuccessStatusCode)
            {
                return RedirectToAction("ListRoles");
            }
        }
    }
    return View(model);
}

```

Рисунок 22. EditRole – вызов к методу API, изменяющий данные роли на те которые были введены

```

[HttpGet]
Ссылка: 0
public async Task<IActionResult> EditUser(string id)
{
    //var user = await userManager.FindByIdAsync(id);
    var user = await _client.GetEditUser(id).ConfigureAwait(false);
    if (user==null)
    {
        ViewBag.ErrorMessage = $"Пользователь с Id = {id} не найден";
        return View("NotFound");
    }
    //var userRoles = await userManager.GetRolesAsync(user);
    var model = new EditUserViewModel
    {
        Id = user.userId,
        Email = user.Email,
        UserName = user.userName,
        Gender = user.Gender,
        Roles = user.userRoles
    };

    return View(model);
}

[HttpPost]
Ссылка: 0
public async Task<IActionResult> EditUser(EditUserViewModel model)
{
    //var user = await userManager.FindByIdAsync(model.Id);
    var user = await _client.GetEditUser(model.Id);
    if (user == null)
    {
        ViewBag.ErrorMessage = $"Пользователь с Id = {model.Id} не найден";
        return View("NotFound");
    }
    else
    {
        var updateuser = new EditUserContract
        {
            userId = model.Id,
            Email = model.Email,
            userName = model.UserName,
            Gender = model.Gender,
            userRoles = model.Roles
        };
        var result = await _client.UpdateUser(updateuser);
        if (result.IsSuccessStatusCode)
        {
            return RedirectToAction("ListUsers");
        }
        return View(model);
    }
}
}

```

Рисунок 23. GET и POST методы EditUser – вызовы к методам API, которые соответственно получают нужного и изменяют поля и значения нужного пользователя

```

Ссылка: 0
public async Task<IActionResult> DeleteUser(string id)
{
    //var user = await userManager.FindByIdAsync(id);
    var user = await _client.GetUser(id);
    if (user == null)
    {
        ViewBag.ErrorMessage = $"Пользователь с Id = {id} не найден";
        return View("NotFound");
    }
    else
    {
        DeleteUserContract contract = new DeleteUserContract
        {
            userId = user.userId
        };
        var result = await _client.DeleteUser(contract);
        if (result.IsSuccessStatusCode)
        {
            return RedirectToAction("ListUsers");
        }
        return View("ListUsers");
    }
}

[HttpPost]
Ссылка: 0
public async Task<IActionResult> DeleteRole(string id)
{
    DeleteRoleContract delet = new DeleteRoleContract
    {
        RoleId = id
    };
    var result = await _client.DeleteRole(delet).ConfigureAwait(false);
    if (result.IsSuccessStatusCode)
    {
        return RedirectToAction("ListRoles");
    }

    return View("ListRoles");
}

```

Рисунок 24. POST методы DeleteUser и DeleteRole – вызовы к методам API, которые удаляют пользователей и ролей из БД соответственно

Рассмотрим HomeController. Единственная функция этого контроллера – отображать главную страницу, в которой отображаются все элементы которые обозначены как «популярные». Это выполняется запросом к GET – методу, возвращающему список/массив элементов с соответствующим фильтром.

```

Ссылка: 0
public async Task<ViewResult> TopSellers ()
{
    string category = "fav";
    PartsContract[] contract = await _client.PartsList(category);
    var parts = contract.Select(x => new Parts { id = x.id, name = x.name, isavailable = x.isavailable, isFavourite = x.isFavourite, categoryID = x.categoryID, img = x.img, price = x.price });
    var homeParts = new HomeViewModel
    {
        favParts = parts
    };
    return View(homeParts);
}

```

Рисунок 25. GET метод TopSellers, делающий вызов к методу API, который возвращает список товаров с фильтром «Fav», а затем отображает этот список пользователю.

Рассмотрим контроллер OrderController. Этот контроллер отвечает за создание формы для заполнения пользователем для оформления заказа. К несчастью, API не работает с элементами типа «Order» (объект – заказ),

несмотря на постоянные попытки это исправить, поэтому это единственный контроллер, который производит манипуляции с БД сервером напрямую.

```
[HttpPost]
[Authorize]
Ссылка: 0
public IActionResult Checkout(Order order)
{
    shopCart.listShopItems = shopCart.GetShopItems();
    string useremail = User.FindFirstValue(ClaimTypes.Email);

    if (shopCart.listShopItems.Count==0)
    {
        ModelState.AddModelError("", "Корзина пуста!");
    }
    if (ModelState.IsValid)
    {
        allOrders.createOrder(order, useremail);
        return RedirectToAction("Complete");
    }
    return View(order);
}
```

Рисунок 26. POST метод Checkout, использующий полученные введённые пользователем данные и добавляющий их в БД в форме заказа.

Рассмотрим контроллер PartsController. Задача этого контроллера заключается в том, чтобы получить списки деталей под разными категориями. В данном случае он делает вызов в соответствующий API метод и передаёт ему строку-критерий. Это могут быть популярные товары или отдельные виды, например графические карты или процессоры.

```
[Route("Parts/List")]
[Route("Parts/List/{category}")]
Ссылка: 0
public async Task<ViewResult> List(string category)
{
    string _category = category;
    IEnumerable<Part> parts = null;
    string currCategory = "";
    //List<Part> contractParts = null;
    PartsContract[] contract = await _client.PartsList(category);
    parts = contract.Select(x => new Part { id = x.id, name = x.name, isavailable = x.isavailable, isFavourite = x.isFavourite, categoryID = x.categoryID, img = x.img, price = x.price, shortDesc = x.shortDesc, longDesc = x.longDesc }).ToList();
    /*if(category.Equals("") )
    {
        parts = allParts.AllParts.OrderBy(i => i.id);
    }
    else
    {
        if(string.Equals("gpu", category, StringComparison.OrdinalIgnoreCase))
        {
            parts = allParts.AllParts.Where(i => i.Category.categoryname.Equals("Графическая карта")).OrderBy(i => i.id);
        }
        else if (string.Equals("processor", category, StringComparison.OrdinalIgnoreCase))
        {
            parts = allParts.AllParts.Where(i => i.Category.categoryname.Equals("Процессор")).OrderBy(i => i.id);
        }
        else if (string.Equals("ram", category, StringComparison.OrdinalIgnoreCase))
        {
            parts = allParts.AllParts.Where(i => i.Category.categoryname.Equals("Оперативная память")).OrderBy(i => i.id);
        }
    }
    currCategory = _category;
    */
    var partObj = new PartsListViewModel
    {
        AllParts = parts,
        currCategory = currCategory
    };
    ViewBag.Title = "Список товаров";
    return View(partObj);
}
```

Рисунок 27. POST метод List, использующий введённый критерий для вызова метода API, возвращающий список с элементами под данным критерием.

Рассмотрим последний контроллер фронтенд – приложения: ShopCartController. Это контроллер корзины, а именно объекта ShopCart. Этот

контроллер добавляет элементы товары в корзину, а также убирает их из неё. Он отвечает за то, какие товары в результате будут включены в заказ.

```
Ссылка: 0
public RedirectToActionResult addToCart(int id)
{
    var item = _partRep.AllParts.FirstOrDefault(i => i.id == id);
    if(item != null)
    {
        _shopCart.AddToCart(item);
    }
    return RedirectToAction("ShopCartPage");
}

Ссылка: 0
public RedirectToActionResult removeFromCart(int id)
{
    var items = _shopCart.GetShopItems();
    _shopCart.listShopItems = items;
    var item = _shopCart.listShopItems.FirstOrDefault(i => i.id == id);
    if(item != null)
    {
        _shopCart.RemoveFromCart(item);
    }
    return RedirectToAction("ShopCartPage");
}
```

Рисунок 28. Методы `addToCart` и `removeFromCart`, которые соответственно добавляют и убирают соответствующие товары из корзины.

Теперь можно рассмотреть вторую половину сайта: API.

Прикреплённое к сайту API имеет похожее к фронтенд-приложению содержание и структуру: API использует архитектуру «Модель-Вид-Контроллер», только в данном случае фигурируют только модели и контроллеры, так как API сам по себе не используется пользователем поэтому сам по себе API-приложение не будет отображать эти данные. Эту роль занимает Swagger, что позволяет программисту проверять и видеть результаты действия.

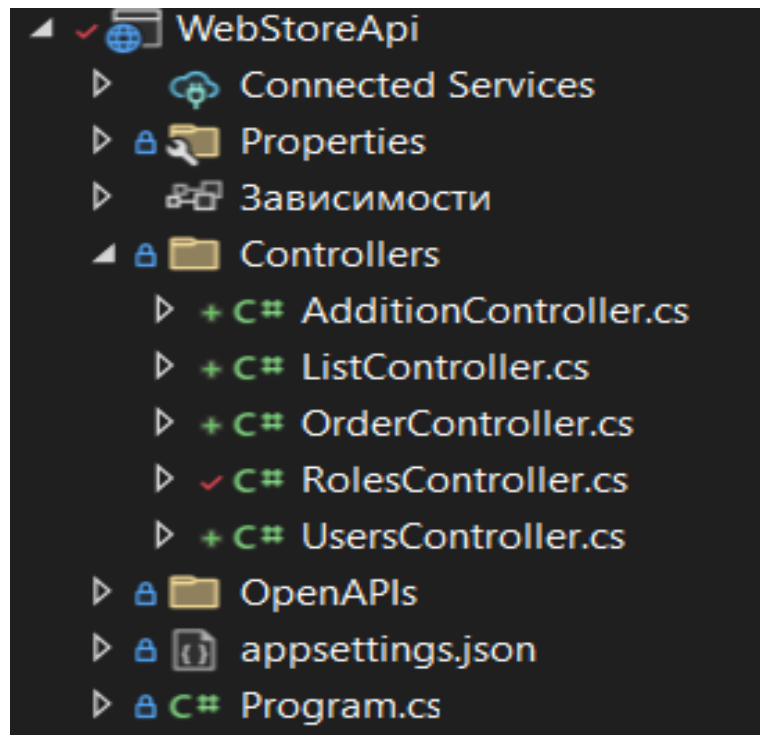


Рисунок 29. Содержимое прикреплённого к приложению API.

API использует контекст БД для прямого подключения и взаимодействия с нужной Базой Данных, 5 контроллеров имеют в себе функции которые нужны сайту: GET и POST функции. GET функции «достаю» из БД требуемые данные будь они отдельный элемент или массивы-списки этих элементов. POST функции имеют разные применения: добавление данных на сервер, изменение данных на сервере или даже удаление данных с сервера, и всё это происходит с использованием моделей которые в данном случае называются «контрактами».

Рассмотрим контроллер AdditionController. Этот контроллер отвечает за добавку на сайт новых деталей, используя полученные данные из соответствующего контракта, полученного от фронтенд-приложения. Этот контроллер возвращает свою нужную версию контракта, который соответствующий контроллер в фронтенде переводит в нужный объект для дальнейшего использования.

```

[HttpPost]
Ссылка: 0
public async Task<IActionResult> AddPart(PartsContract part)
{
    Parts partToAdd = new Parts();
    partToAdd.name = part.name;
    partToAdd.isavailable = part.isavailable;
    partToAdd.isFavourite = part.isFavourite;
    partToAdd.shortDesc = part.shortDesc;
    partToAdd.longDesc = part.longDesc;
    partToAdd.price = part.price;
    partToAdd.img = part.img;
    partToAdd.categoryID = part.categoryID;

    IEnumerable<Categories> categories = null;
    categories = appDBContent.Category.OrderBy(i => i.id);
    switch (partToAdd.categoryID)
    {
        case 1:
            partToAdd.Category = categories.ElementAt(0);
            break;
        case 2:
            partToAdd.Category = categories.ElementAt(1);
            break;
        case 3:
            partToAdd.Category = categories.ElementAt(2);
            break;
    }
    if (partToAdd.categoryID != null && partToAdd.Category != null)
    {
        await appDBContent.Part.AddAsync(partToAdd);
        await appDBContent.SaveChangesAsync();
        return Ok();
    }
    else
    {
        return BadRequest();
    }
}

```

Рисунок 30. Метод AddPart, добавляющий новый товар в общий каталог товаров в БД.

Демонстрация функции AddPart в Swagger:

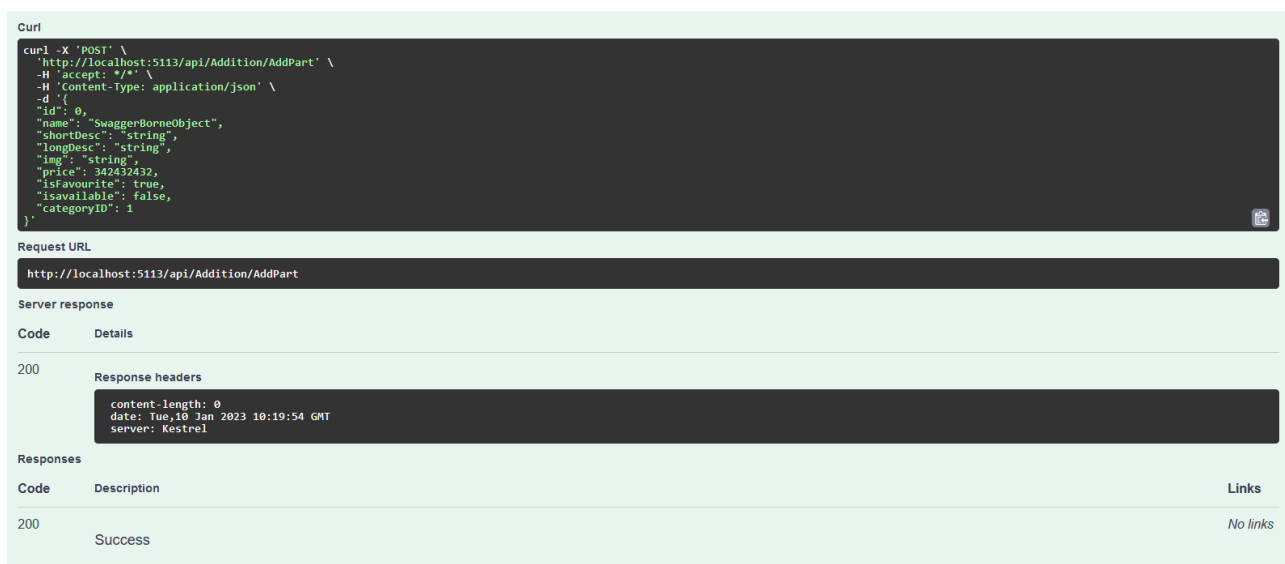
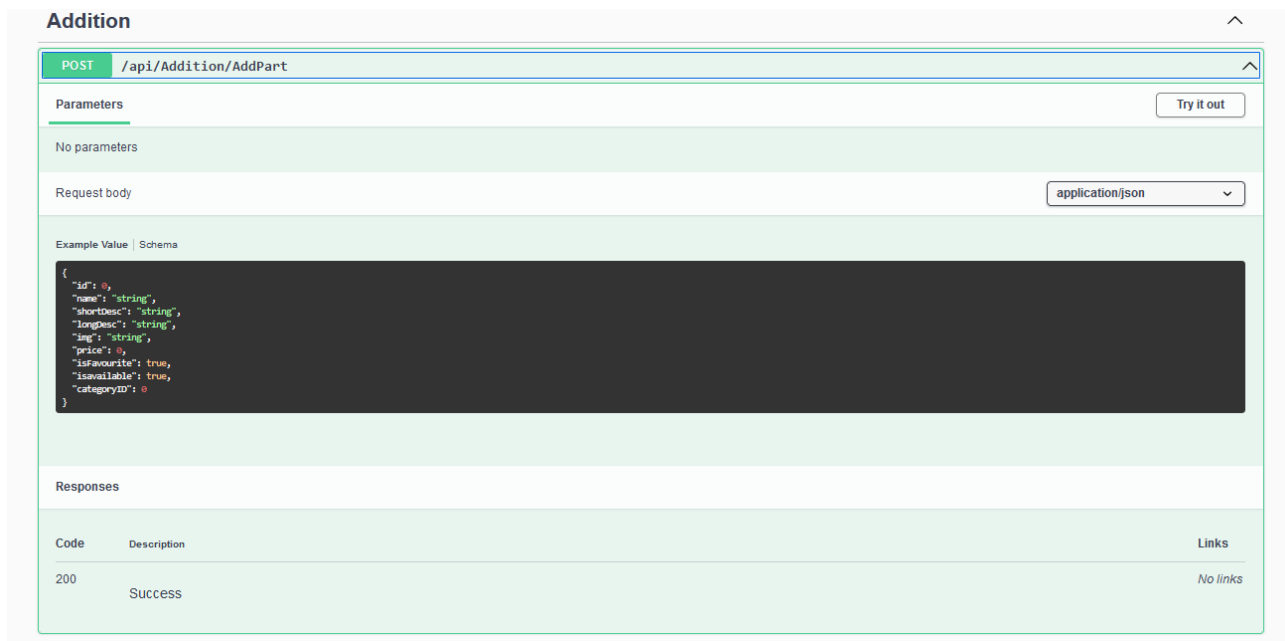


Рисунок 31. Метод `AddPart` в Swagger.

Рассмотрим контроллер `ListController`. Его задача – формировать списки товаров по категориям. Всего в данном случае он может создать 5 видов списков:

Все товары, Популярные товары, Графические карты, Процессоры, Оперативная память. Список собирается по критерию, и отправляется обратно к фронтенду в виде массива, который фронтенд использует чтобы собрать все нужные товары на экране для пользователя.

```

[Route("api/[controller]/[action]")]
[ApiController]
Source 1
public class ListController : ControllerBase
{
    private readonly AppDbContext appDbContext;

    Source 0
    public ListController (AppDbContext appDbContext)
    {
        this.appDbContext = appDbContext;
    }

    [HttpGet]
    Source 0
    public async Task<PartsContract[]> GetAllParts()
    {
        IEnumerable<Parts> AllParts = appDbContext.Part.Include(c => c.Category);
        PartsContract[] arr = AllParts.Select(x => new PartsContract { id = x.id, name = x.name, isavailable = x.isavailable, isFavourite = x.isFavourite, categoryID = x.categoryID, img = x.img, p
        return arr;
    }

    [HttpGet]
    Source 0
    public async Task<PartsContract[]> PartsList(string category)
    {
        string _category = category;
        IEnumerable<Parts> parts = null;
        string currCategory = "";
        if (category.Equals("all"))
        {
            parts = appDbContext.Part.Include(c => c.Category);
            PartsContract[] contrpart = parts.Select(x => new PartsContract { id = x.id, name = x.name, isavailable = x.isavailable, isFavourite = x.isFavourite, categoryID = x.categoryID, img = x
            return contrpart;
        }
        else
        {
            if (string.Equals("gpu", category, StringComparison.OrdinalIgnoreCase))
            {
                parts = appDbContext.Part.Where(i => i.Category.Categoryname.Equals("Графическая карта")).OrderBy(i => i.id);
                PartsContract[] contrpart = parts.Select(x => new PartsContract { id = x.id, name = x.name, isavailable = x.isavailable, isFavourite = x.isFavourite, categoryID = x.categoryID, img
                return contrpart;
            }
            else if (string.Equals("processor", category, StringComparison.OrdinalIgnoreCase))
            {
                parts = appDbContext.Part.Where(i => i.Category.Categoryname.Equals("Процессор")).OrderBy(i => i.id);
                PartsContract[] contrpart = parts.Select(x => new PartsContract { id = x.id, name = x.name, isavailable = x.isavailable, isFavourite = x.isFavourite, categoryID = x.categoryID, img
                return contrpart;
            }
            else if (string.Equals("ram", category, StringComparison.OrdinalIgnoreCase))
            {
                parts = appDbContext.Part.Where(i => i.Category.Categoryname.Equals("Оперативная память")).OrderBy(i => i.id);
                PartsContract[] contrpart = parts.Select(x => new PartsContract { id = x.id, name = x.name, isavailable = x.isavailable, isFavourite = x.isFavourite, categoryID = x.categoryID, img
                return contrpart;
            }
            else if (string.Equals("fav", category, StringComparison.OrdinalIgnoreCase))
            {
                parts = appDbContext.Part.Where(i => i.isFavourite.Equals(true)).OrderBy(i => i.id);
                PartsContract[] contrpart = parts.Select(x => new PartsContract { id = x.id, name = x.name, isavailable = x.isavailable, isFavourite = x.isFavourite, categoryID = x.categoryID, img
                return contrpart;
            }
            PartsContract[] contrpart = parts.Select(x => new PartsContract { id = x.id, name = x.name, isavailable = x.isavailable, isFavourite = x.isFavourite, categoryID = x.categoryID, img =
            return contrpart;
        }
    }
}

```

Рисунок 32. Методы контроллера ListController.

Curl

```
curl -X 'GET' \
  'http://localhost:5113/api/List/GetAllParts' \
  -H 'accept: text/plain'
```

Request URL

```
http://localhost:5113/api/List/GetAllParts
```

Server response

Code	Details		
200	<div> Response body <pre>{ "id": 1, "name": "NVIDIA GeForce GTX 1650", "shortDesc": "Видеокарта серии GeForce GTX 16", "longDesc": "Видеокарта серии GeForce GTX 16. Мощностъ на картата ѝ е много изгодна за покупка, въпреки че е с по-нисък ниво на мощност в сравнение с другите карти NVIDIA.", "img": "/img/Nvidia-geforce-GTX-1650-6.jpg", "price": 30000, "isFavourite": true, "isavailable": true, "categoryID": 1 }, { "id": 2, "name": "NVIDIA GeForce GTX 1050Ti", "shortDesc": "Видеокарта серии GeForce GTX 10", "longDesc": "Видеокарта серии GeForce GTX 10. Огромна мощност за своето време, способна да изобразява картини до 4K и има висока динамична област (HDR).", "img": "/img/GeForce_GTX_1050_Ti_Partners_GALAX_E24FCB040E284EC9893ACED4B668269C.jpg", "price": 21000, "isFavourite": false, "isavailable": false, "categoryID": 1 }, { "id": 3, "name": "Intel Core i5-10400F", </pre> </div> <div> Response headers <pre>content-type: application/json; charset=utf-8 date: Tue, 10 Jan 2023 14:03:54 GMT server: Kestrel transfer-encoding: chunked</pre> </div> <div> Responses </div> <table border="1"> <thead> <tr> <th>Code</th> <th>Description</th> </tr> </thead> <tbody> </tbody> </table> <div>Links</div>	Code	Description
Code	Description		

Рисунок 33. Метод GetAllParts.

Демонстрация функций контроллера в Swagger:

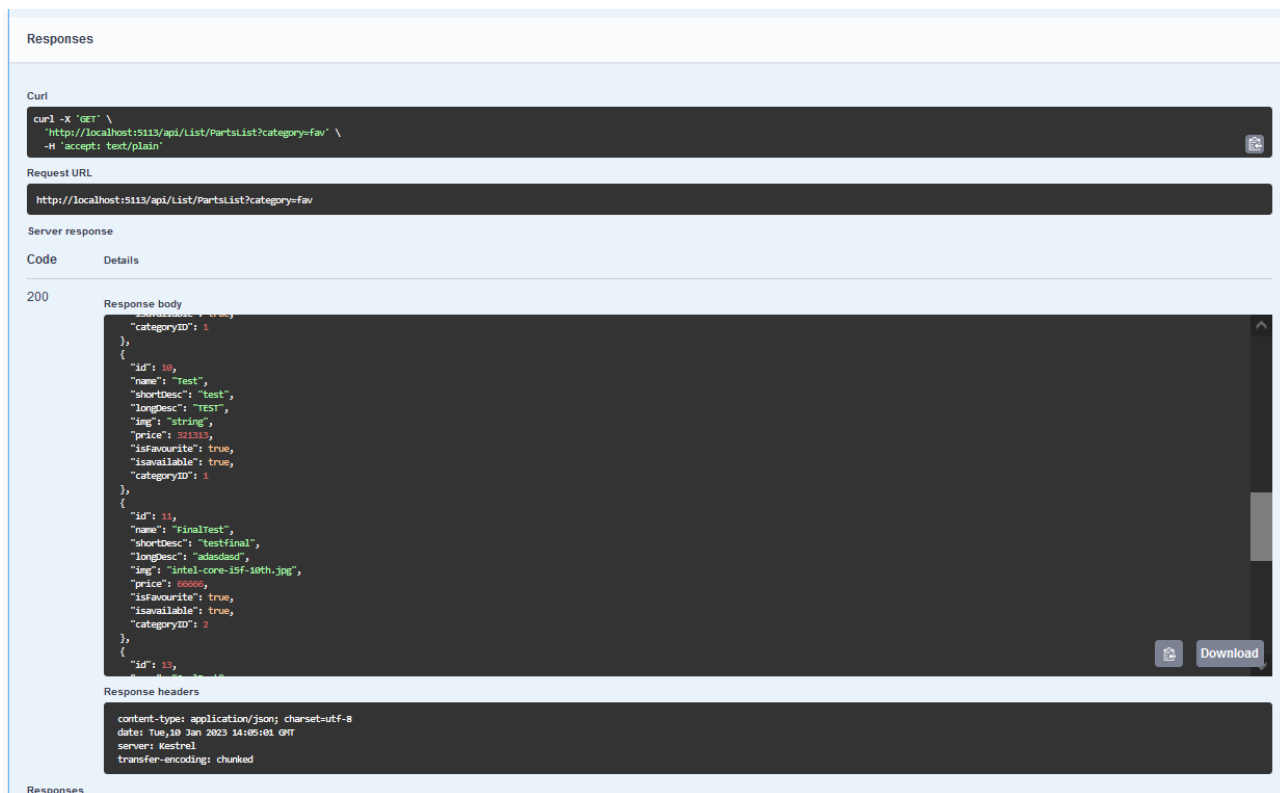


Рисунок 34. Метод *PartsList* с запросом «fav»-все товары, помеченные как популярные.

Рассмотрим контроллер *OrderController*. Его функция – оформлять и добавлять в БД информацию о заказах. Он получает информацию из соответствующего контракта и добавляет её в базу данных. Хотя этот контроллер реализован и работает, попытки подключить его к фронтенд-приложению не увенчались успехом, из-за чего его функционал не связан напрямую с приложением.

Демонстрация функции через Swagger:

The screenshot displays the Swagger UI interface. At the top, the 'Responses' tab is selected. Below it, the 'Curl' section shows a cURL command for a POST request to `http://localhost:5113/api/Order/AddOrder` with a JSON body. The 'Request URL' section shows the same endpoint. The 'Server response' section shows a 200 status code. The 'Response headers' section lists: `content-length: 0`, `date: Tue, 10 Jan 2023 15:16:23 GMT`, and `server: Kestrel`.

```
curl -X 'POST' \
  'http://localhost:5113/api/Order/AddOrder' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "items": [
      {
        "id": 0,
        "part": {
          "id": 0,
          "name": "string",
          "shortDesc": "string",
          "longDesc": "string",
          "img": "string",
          "price": 0,
          "isFavourite": true,
          "isAvailable": true,
          "categoryID": 0
        },
        "price": 0,
        "shopCardId": "string"
      }
    ],
    "order": {
      "id": 0,
      "name": "string",
      "surname": "string",
      "address": "string",
      "phone": "string",
      "email": "string",
      "orderTime": "2023-01-10T15:09:09.032Z"
    }
  }'
```

Request URL

http://localhost:5113/api/Order/AddOrder

Server response

Code Details

200

Response headers

```
content-length: 0
date: Tue, 10 Jan 2023 15:16:23 GMT
server: Kestrel
```

Рисунок 35. Пример работы метода AddOrder.

Рассмотрим контроллер RolesController. Этот контроллер отвечает за все операции, связанные с ролями пользователей. Он может создавать роли, изменять роли, изменять кто находится в роли, а также удалять эти роли, используя специальные модели-контракты.

```
[HttpGet]
[AllowAnonymous]
public ActionResult<RoleContract[]> GetAllRoles()
{
    return _roleManager.Roles.Select(x => new RoleContract { Id = x.Id, Name = x.Name }).ToArray();
}

[HttpGet]
[AllowAnonymous]
public ActionResult<RoleContract> GetRole(string id)
{
    RoleContract[] rolesArray = _roleManager.Roles.Select(x => new RoleContract { Id = x.Id, Name = x.Name }).ToArray();
    //List<RoleContract> roles = rolesArray.ToList<RoleContract>();
    RoleContract roletofind = rolesArray.SingleOrDefault(x => x.Id.Equals(id));
    return roletofind;
}

[HttpGet]
[AllowAnonymous]
public async Task<EditRoleContract> GetEditRole(string id)
{
    var role = await _roleManager.FindByIdAsync(id);
    List<string> userIds = new List<string>();
    List<string> usernames = new List<string>();
    EditRoleContract roletoedit = new EditRoleContract { Id = role.Id, RoleName = role.Name, UserIds=userIds, Users = usernames };
    //if (roletoedit.Id==null && roletoedit.RoleName==null && roletoedit.UserIds==null)
    //{
        //ViewBag.ErrorMessage = $"Роль с Id = {id} не найдена.";
        //return roletoedit;
    //}
    foreach(var user in _userManager.Users)
    {
        if(await _userManager.IsInRoleAsync(user, role.Name))
        {
            roletoedit.UserIds.Add(user.Id);
            roletoedit.Users.Add(user.UserName);
        }
    }

    return roletoedit;
}
```

Рисунок 36. GET методы RolesController.

```
[HttpPost]
Ссылка: 0
public async Task<IActionResult> UpdateRoles(EditRoleContract contract)
{
    var role = _roleManager.Roles.Single(x => x.Id == contract.Id);
    role.Name = contract.RoleName;
    await _roleManager.UpdateAsync(role).ConfigureAwait(false);

    var users = _userManager.Users.Where(x => contract.UserIds.Contains(x.Id));
    foreach(var user in users)
    {
        await _userManager.AddToRoleAsync(user, role.Name).ConfigureAwait(false);
    }

    return Ok();
}

[HttpPost]
Ссылка: 0
public async Task<IActionResult> AddRole(CreateRoleContract contract)
{
    IdentityRole identityRole = new IdentityRole
    {
        Name = contract.RoleName
    };

    IdentityResult result = await _roleManager.CreateAsync(identityRole);

    if (result.Succeeded)
    {
        return Ok();
    }

    return BadRequest();
}

[HttpPost]
Ссылка: 0
public async Task<IActionResult> DeleteRole(string id)
{
    var role = await _roleManager.FindByIdAsync(id);
    if (role == null)
    {
        return BadRequest();
    }
    else
    {
        var result = await _roleManager.DeleteAsync(role);
        if (result.Succeeded)
        {
            return Ok();
        }
    }

    return BadRequest();
}
```

Рисунок 37. POST методы RolesController.

Демонстрация работы методов в Swagger:

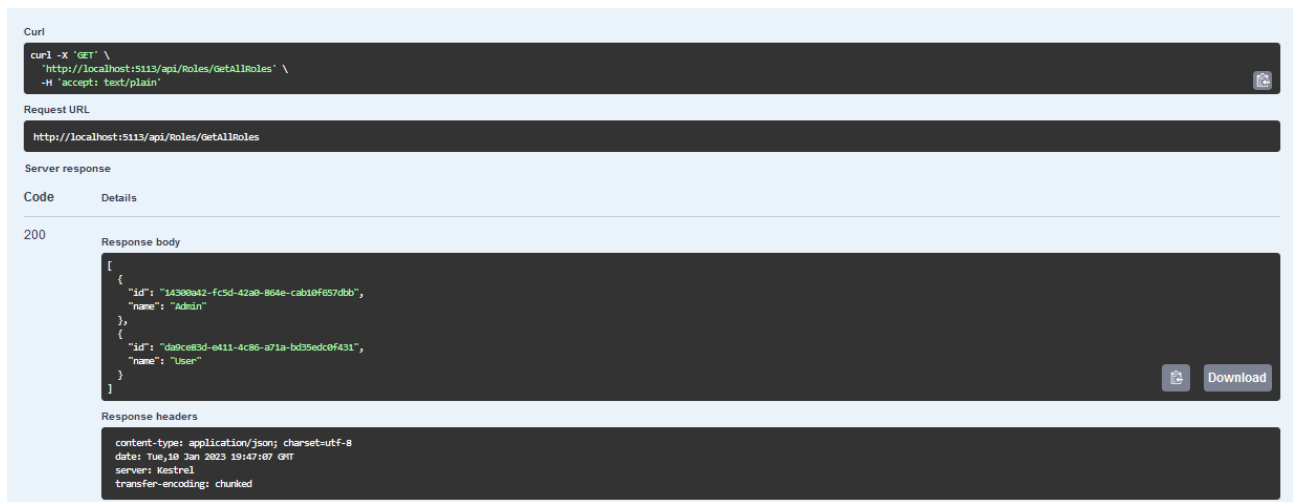


Рисунок 38. Функция *GetAllRoles*

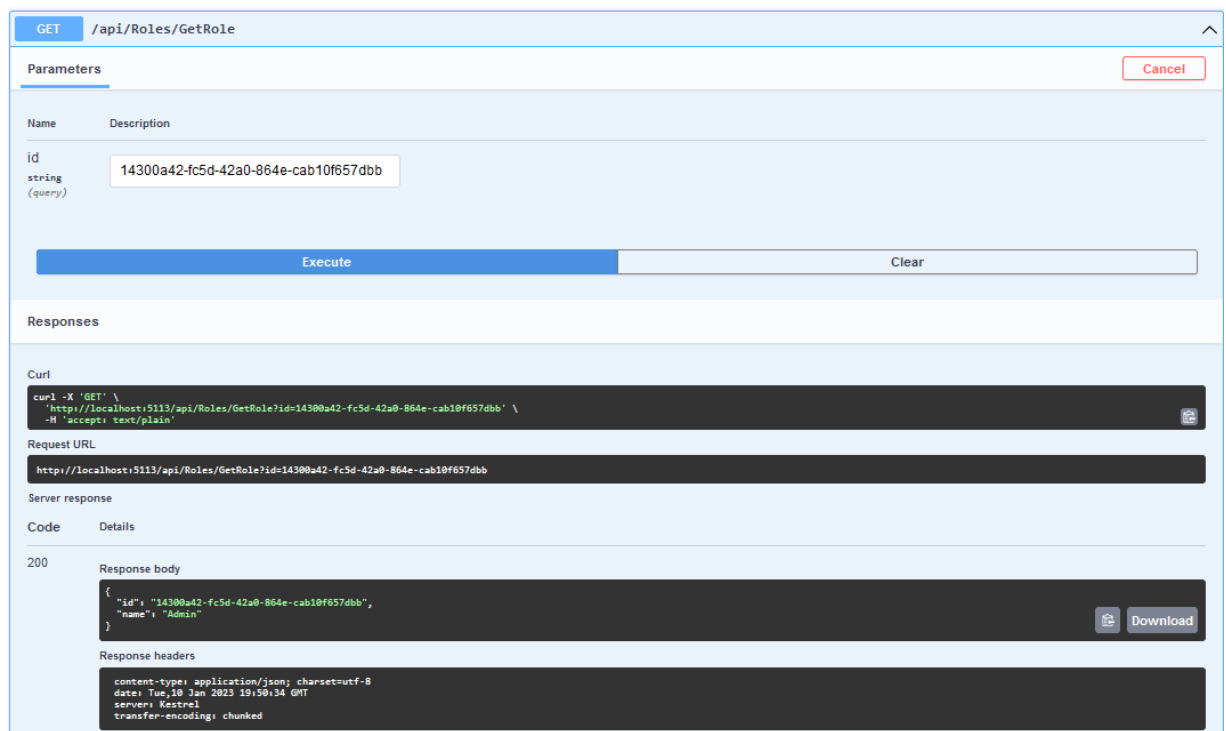


Рисунок 39. Функция *GetRole*

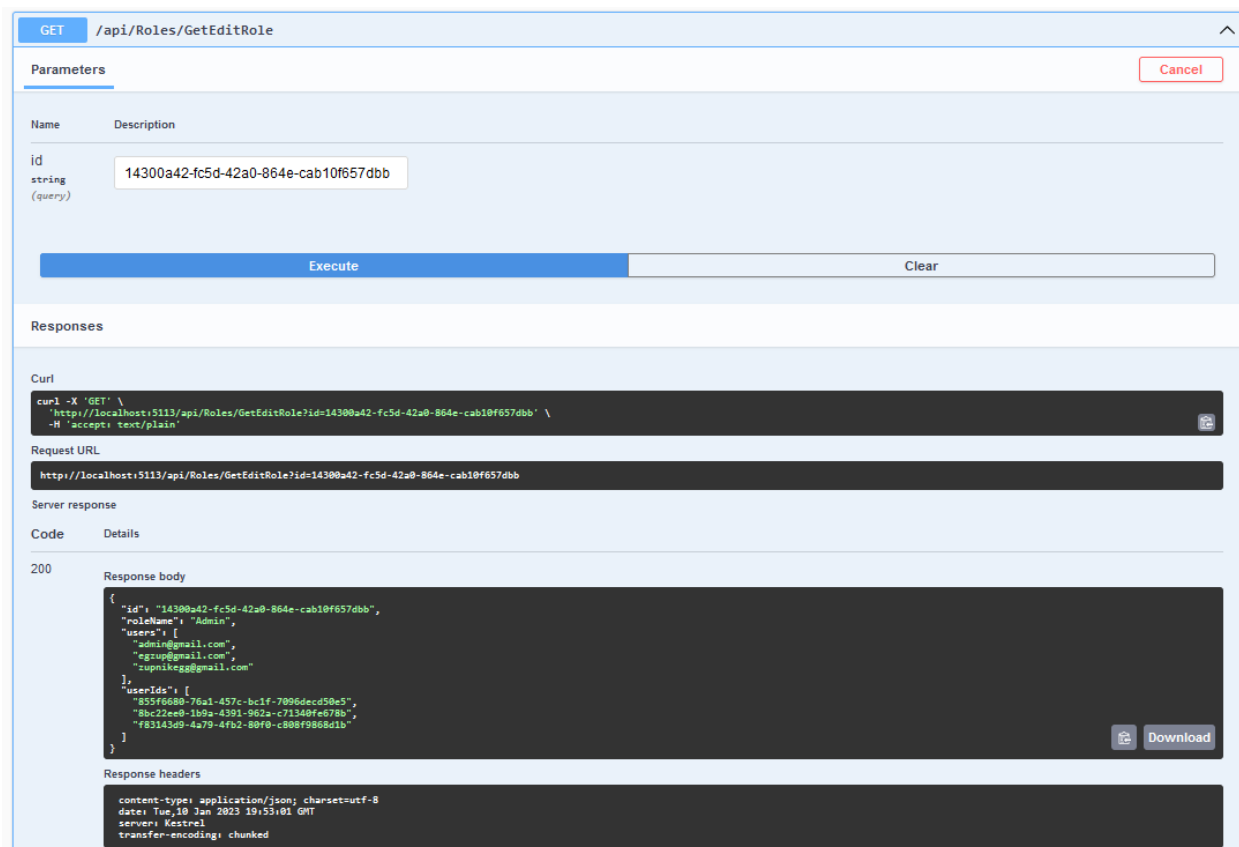


Рисунок 40. Функция *GetEditRole*.

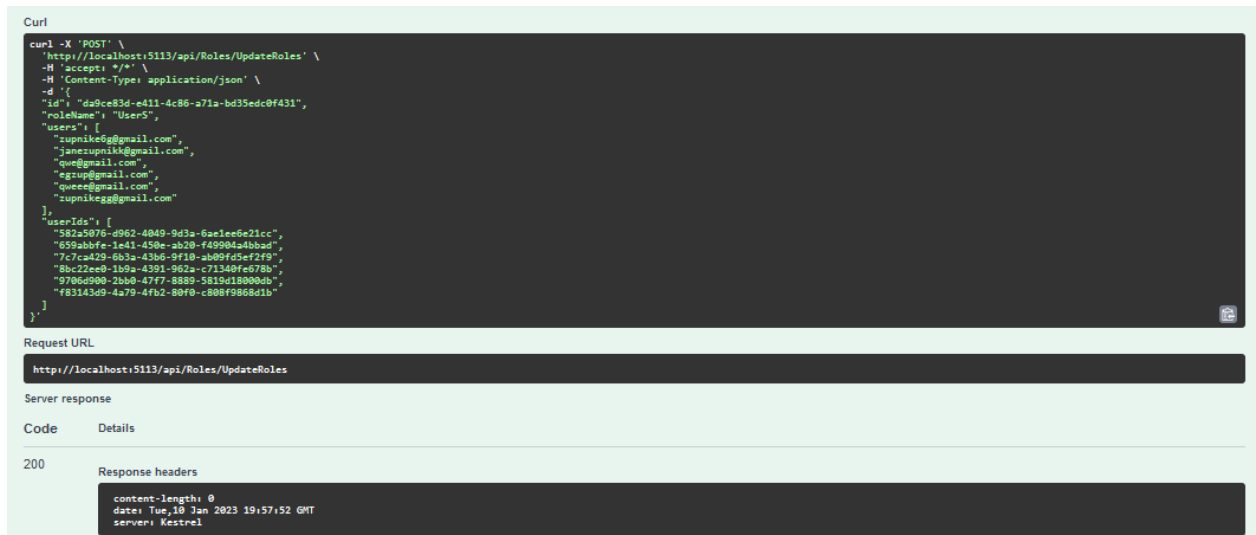


Рисунок 41. Функция *UpdateRoles* (в данном случае было изменено название роли).



Рисунок 42. Функция AddRole

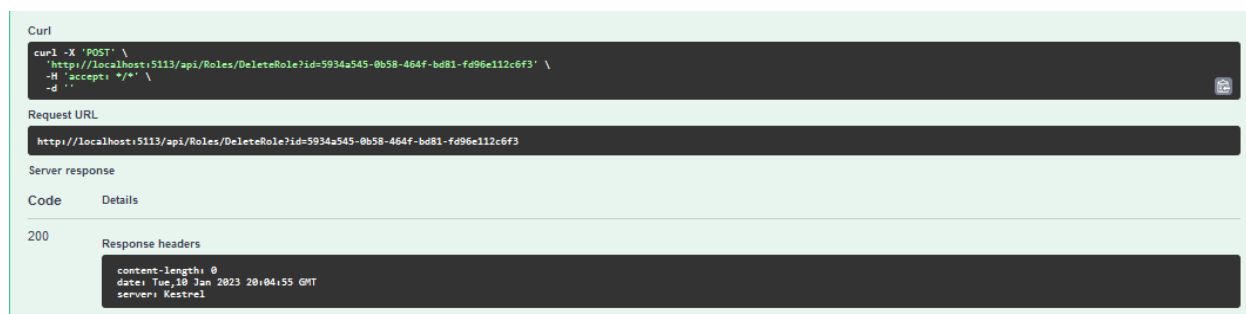


Рисунок 43. Функция DeleteRole (удалена роль, созданная в демонстрации функции AddRole)

Последний контроллер в API-приложении – UsersController. Этот контроллер отвечает за все действия, связанные с пользователями. Благодаря этому контроллеру сайт может регистрировать новых пользователей, удалять ненужных пользователей, или изменять информацию существующих пользователей.

```

}
[HttpGet]
Ссылка: 0
public async Task<UserContract[]> GetAllUsers()
{
    return _userManager.Users.Select(x => new UserContract { userId = x.Id, userName = x.UserName }).ToArray();
}
[HttpGet]
Ссылка: 0
public ActionResult<UserContract> GetUser(string id)
{
    UserContract[] rolesArray = _userManager.Users.Select(x => new UserContract { userId = x.Id, userName = x.UserName }).ToArray();
    //List<RoleContract> roles = rolesArray.ToList<RoleContract>();
    UserContract roletofind = rolesArray.SingleOrDefault(x => x.userId.Equals(id));
    return roletofind;
}
[HttpGet]
Ссылка: 0
public async Task<EditUserContract> GetEditUser(string id)
{
    var user = await _userManager.FindByIdAsync(id);
    List<string> roleids = new List<string>();
    List<string> rolename = new List<string>();
    foreach (var role in _roleManager.Roles)
    {
        if (await _userManager.IsInRoleAsync(user, role.Name))
        {
            roleids.Add(role.Id);
            rolename.Add(role.Name);
        }
    }
    EditUserContract usertoedit = new EditUserContract
    {
        userId = user.Id,
        userName = user.UserName,
        Email = user.Email,
        Gender = user.Gender,
        userRoles = rolename,
        userRoleIds = roleids
    };
    return usertoedit;
}
}

```

```

[HttpGet]
Ссылка: 0
public async Task<List<UserRoleContract>> GetUserRole(string id)
{
    var model = new List<UserRoleContract>();
    var role = await _roleManager.FindByIdAsync(id);
    //var contract = await _client.GetEditUser(roleId).ConfigureAwait(false);
    //
    foreach (var user in _userManager.Users)
    {
        var userRoleContract = new UserRoleContract
        {
            UserId = user.Id,
            UserName = user.UserName
        };
        if (await _userManager.IsInRoleAsync(user, role.Name))
        {
            userRoleContract.IsSelected = true;
        }
        else
        {
            userRoleContract.IsSelected = false;
        }
        model.Add(userRoleContract);
    }
    return model;
}
}

```

Рисунок 44. GET методы контроллера UsersController

```

[HttpPost]
Ссылка: 0
public async Task<IActionResult> UpdateUserRole(UpdateRoleContract contract)
{
    //List<UserRoleContract> model, string roleId
    List<UserRoleContract> model = new List<UserRoleContract>();
    model = contract.Users;
    string roleId = contract.roleId;
    var role = await _roleManager.FindByIdAsync(roleId);
    int k = 0;
    foreach (var el in model)
    {
        var user = await _userManager.FindByIdAsync(el.UserId);
        IdentityResult result = null;
        if (el.IsSelected && !(await _userManager.IsInRoleAsync(user, role.Name)))
        {
            k++;
            result = await _userManager.AddToRoleAsync(user, role.Name);
        }
        else if (!el.IsSelected && (await _userManager.IsInRoleAsync(user, role.Name)))
        {
            k++;
            result = await _userManager.RemoveFromRoleAsync(user, role.Name);
        }
        else
        {
            k++;
            continue;
        }
        if (result != null)
        {
            if (result.Succeeded)
            {
                continue;
            }
            else
            {
                break;
            }
        }
    }
    if (k==model.Count)
    {
        return Ok();
    }
    return BadRequest();
}

```

```

[HttpPost]
Ссылка: 0
public async Task<IActionResult> RegisterUser(CreateUserContract contract)
{
    var user = new ApplicationUser { UserName = contract.Email, Email = contract.Email, Gender = contract.Gender };
    var result = await _userManager.CreateAsync(user, contract.Password);
    if (result.Succeeded)
    {
        return Ok();
    }
    else
    {
        return BadRequest();
    }
}

```

```

[HttpPost]
Ссылка: 0
public async Task<IActionResult> UpdateUser(EditUserContract contract)
{
    var user = await _userManager.FindByIdAsync(contract.userId);
    if (user == null)
    {
        return BadRequest();
    }
    else
    {
        user.Email = contract.Email;
        user.UserName = contract.userName;
        user.Gender = contract.Gender;
        var result = await _userManager.UpdateAsync(user);
        if (result.Succeeded)
        {
            return Ok();
        }
        else
        {
            return BadRequest();
        }
    }
}

[HttpPost]
Ссылка: 0
public async Task<IActionResult> DeleteUser(DeleteUserContract contract)
{
    var user = await _userManager.FindByIdAsync(contract.userId);
    if (user == null)
    {
        return BadRequest();
    }
    else
    {
        var result = await _userManager.DeleteAsync(user);
        if (result.Succeeded)
        {
            return Ok();
        }
        else
        {
            return BadRequest();
        }
    }
}
}

```

Рисунок 45. POST методы контроллера UsersController

Демонстрация работы контроллера в Swagger:

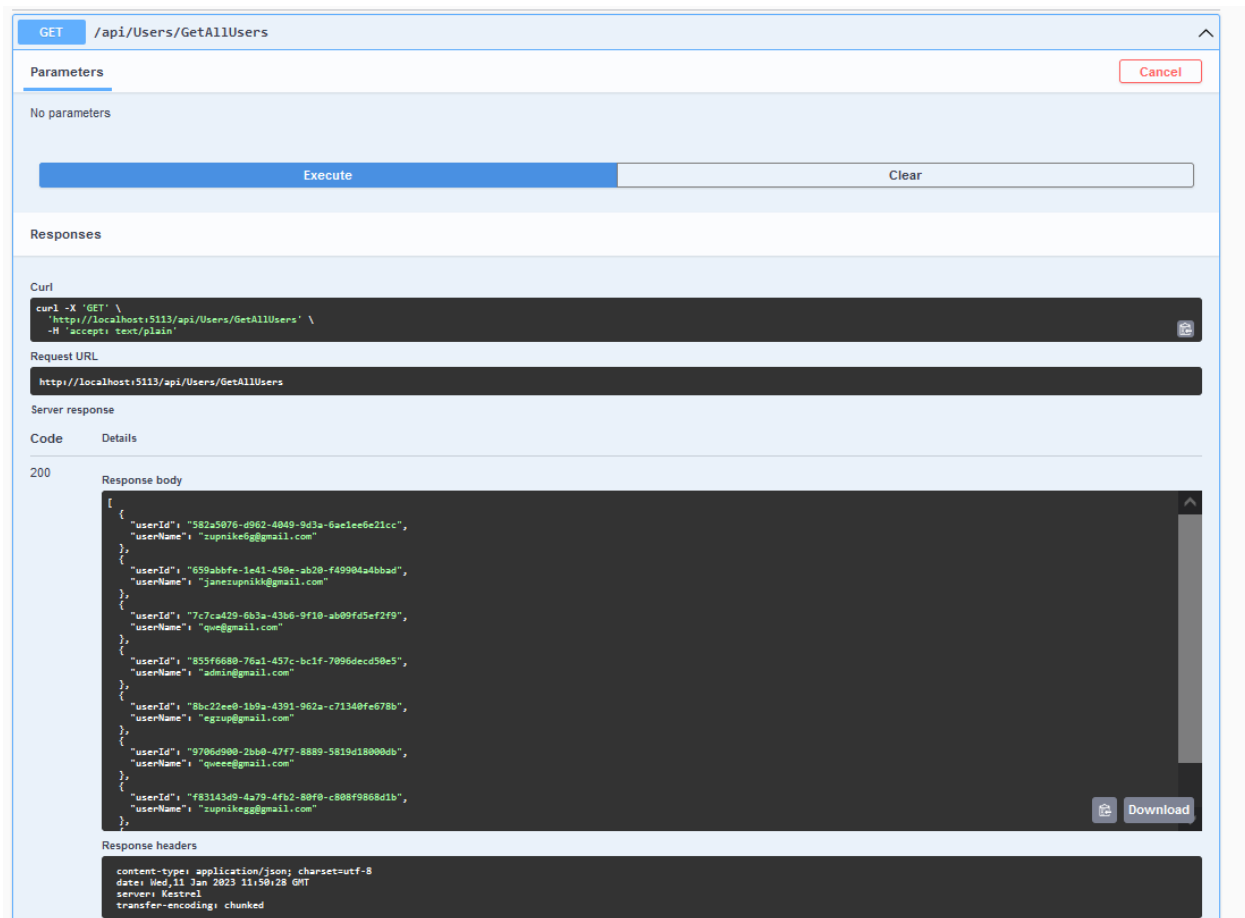


Рисунок 46. Функция GetAllUsers

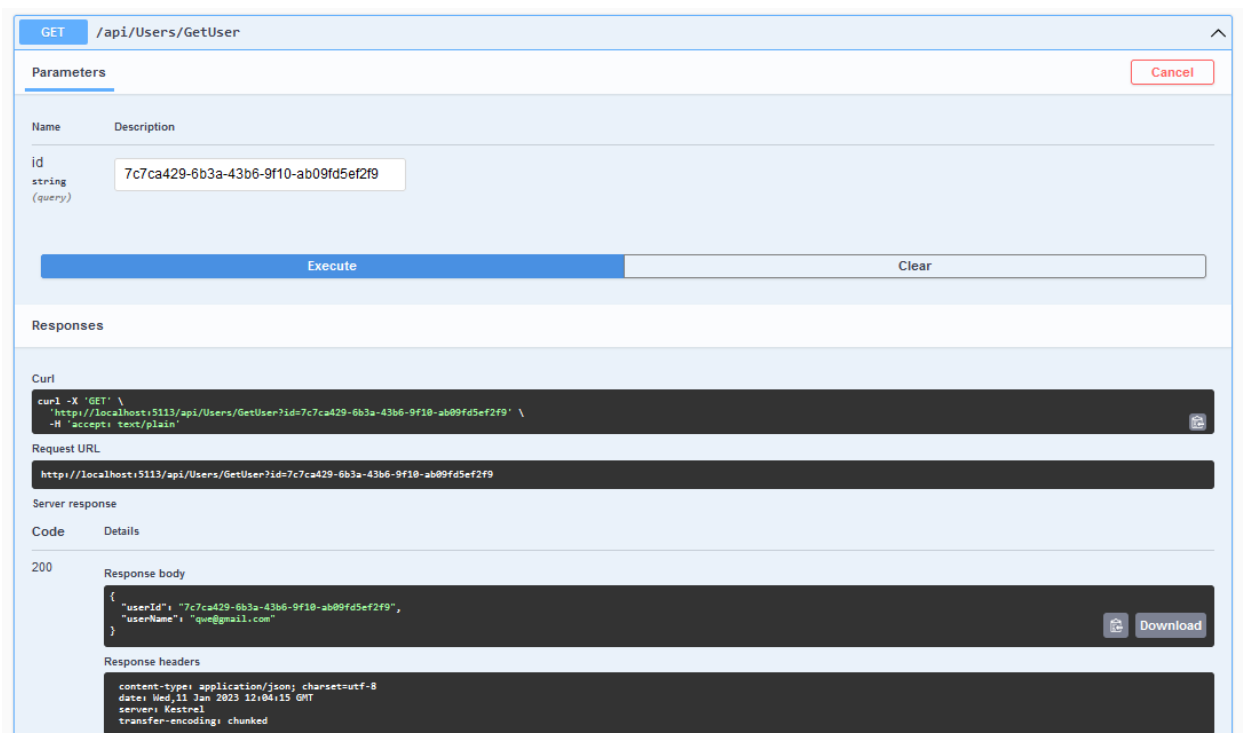


Рисунок 47. Функция *GetUser*

GET /api/Users/GetEditUser

Parameters

Cancel

Name	Description
id	
string	
(query)	

Execute

Clear

Responses

Curl

curl -X 'GET' \

'http://localhost:5113/api/Users/GetEditUser?id=7c7ca429-6b3a-43b6-9f10-ab09fd5ef2f9' \

-H 'accept: text/plain'

Request URL

http://localhost:5113/api/Users/GetEditUser?id=7c7ca429-6b3a-43b6-9f10-ab09fd5ef2f9

Server response

Code

Details

200

Response body

```
{
  "userId": "7c7ca429-6b3a-43b6-9f10-ab09fd5ef2f9",
  "userName": "qwe@gmail.com",
  "email": "qwe@gmail.com",
  "gender": "Male",
  "userRoles": [
    "User"
  ],
  "userRoleId": [
    "da9ce83d-e411-4c86-a71a-bd35edc0f431"
  ]
}
```

Download

Response headers

```
content-type: application/json; charset=utf-8
date: Wed, 11 Jan 2023 12:06:05 GMT
server: Kestrel
transfer-encoding: chunked
```

Рисунок 48. Функция *GetEditUser*

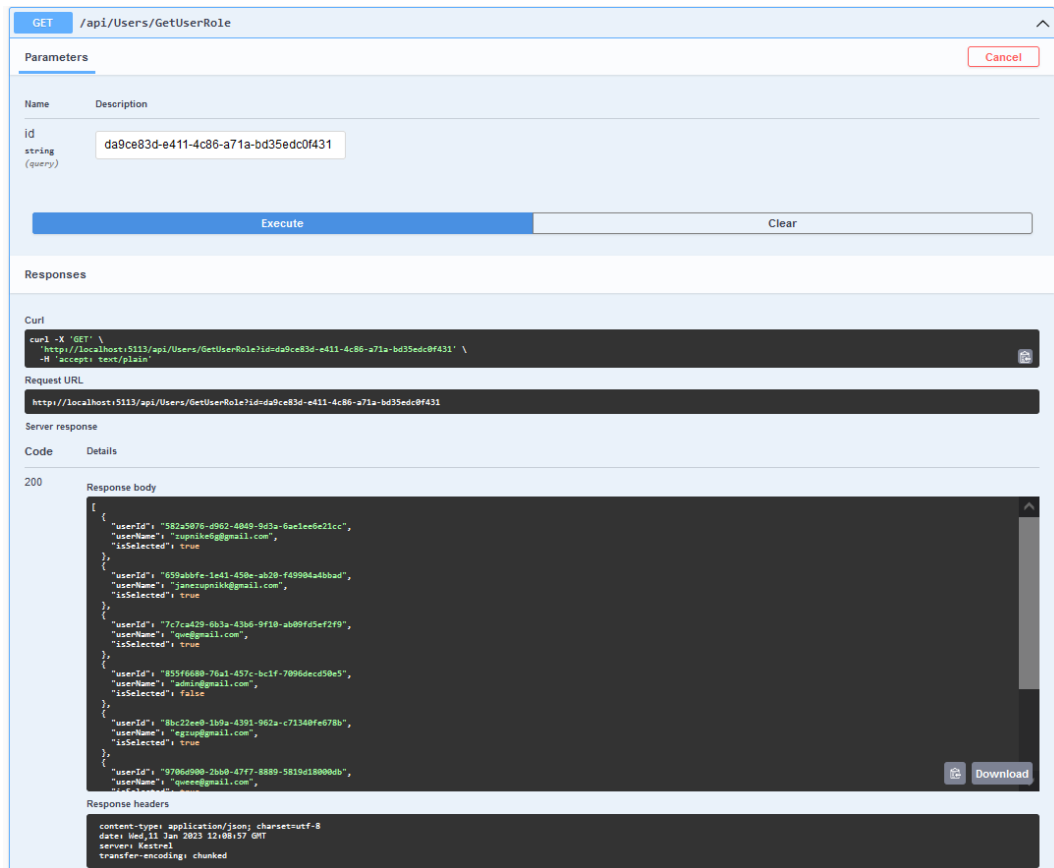
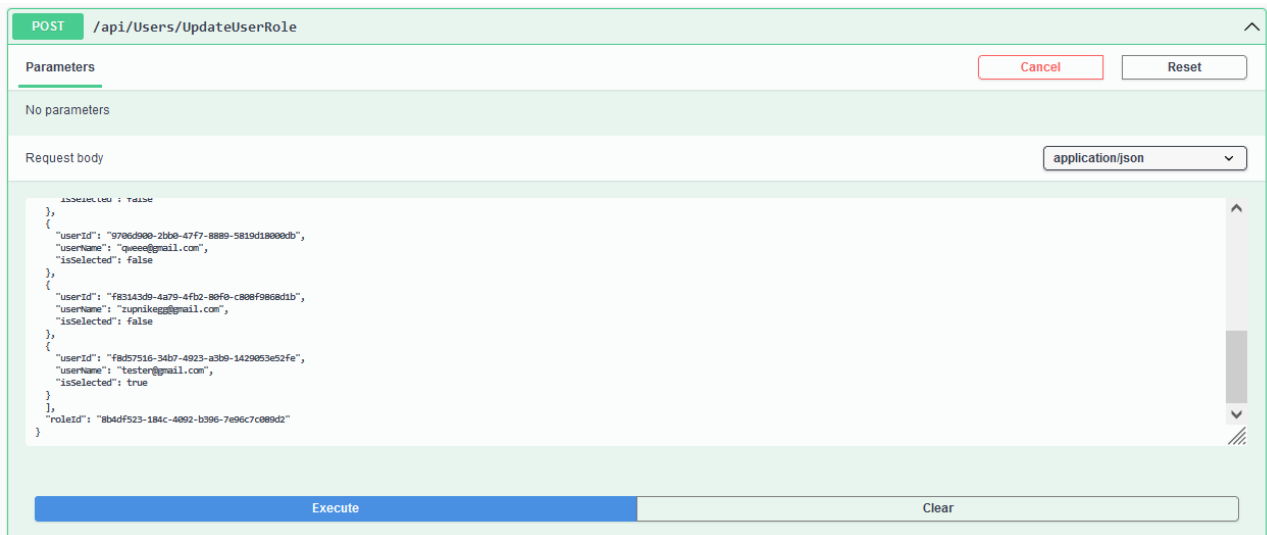


Рисунок 49. Функция GetUserRole



Curl

```
curl -X 'POST' \
  'http://localhost:5113/api/Users/UpdateUserRole' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "users": [
      {
        "userId": "582a5076-d362-4049-9d3a-6ae1ee6e21cc",
        "userName": "zupnik@gmail.com",
        "isSelected": false
      },
      {
        "userId": "659abbfe-1e41-450e-ab20-f49904a4bbad",
        "userName": "jareczupnik@gmail.com",
        "isSelected": false
      },
      {
        "userId": "7c7ca429-8b3a-43b6-9f10-ab09fd5ef2f9",
        "userName": "q@gmail.com",
        "isSelected": false
      },
      {
        "userId": "855f6680-76a1-457c-bc1f-7096decdb9e5",
        "userName": "admin@gmail.com",
        "isSelected": false
      },
      {
        "userId": "8bc22ee0-1b3a-4391-962a-c71340fe678b",
        "userName": "mgrup@gmail.com",
        "isSelected": false
      },
      {
        "userId": "97065900-2b20-47f7-8889-5819d18000db",
        "userName": "qwee@gmail.com",
        "isSelected": false
      },
      {
        "userId": "fa3143db-4a79-4fb2-89f0-c80f96c8d1b",
        "userName": "zupnik@gmail.com",
        "isSelected": false
      },
      {
        "userId": "fad57516-34b7-4923-a3b9-1429053e52fe",
        "userName": "tester@gmail.com",
        "isSelected": true
      }
    ],
    "roleId": "8b4df523-184c-4092-b396-7e96c7c889d2"
  }'
```

Request URL

http://localhost:5113/api/Users/UpdateUserRole

Server response

Code	Details
200	<p>Response headers</p> <p>content-length: 0 date: Wed, 11 Jan 2023 13:16:20 GMT server: Kestrel</p>

Рисунок 50. Функция UpdateUserRole

POST /api/Users/UpdateUser

Parameters

No parameters

Request body

application/json

```
{
  "userId": "7c7ca429-8b3a-43b6-9f10-ab09fd5ef2f9",
  "username": "qweeeeeeeeeee@gmail.com",
  "email": "q@gmail.com",
  "password": "190618",
  "roles": [
    "user"
  ],
  "permissions": [
    "viewedit-0111-4c86-a71a-bd35edc9f431"
  ]
}
```

Execute Clear

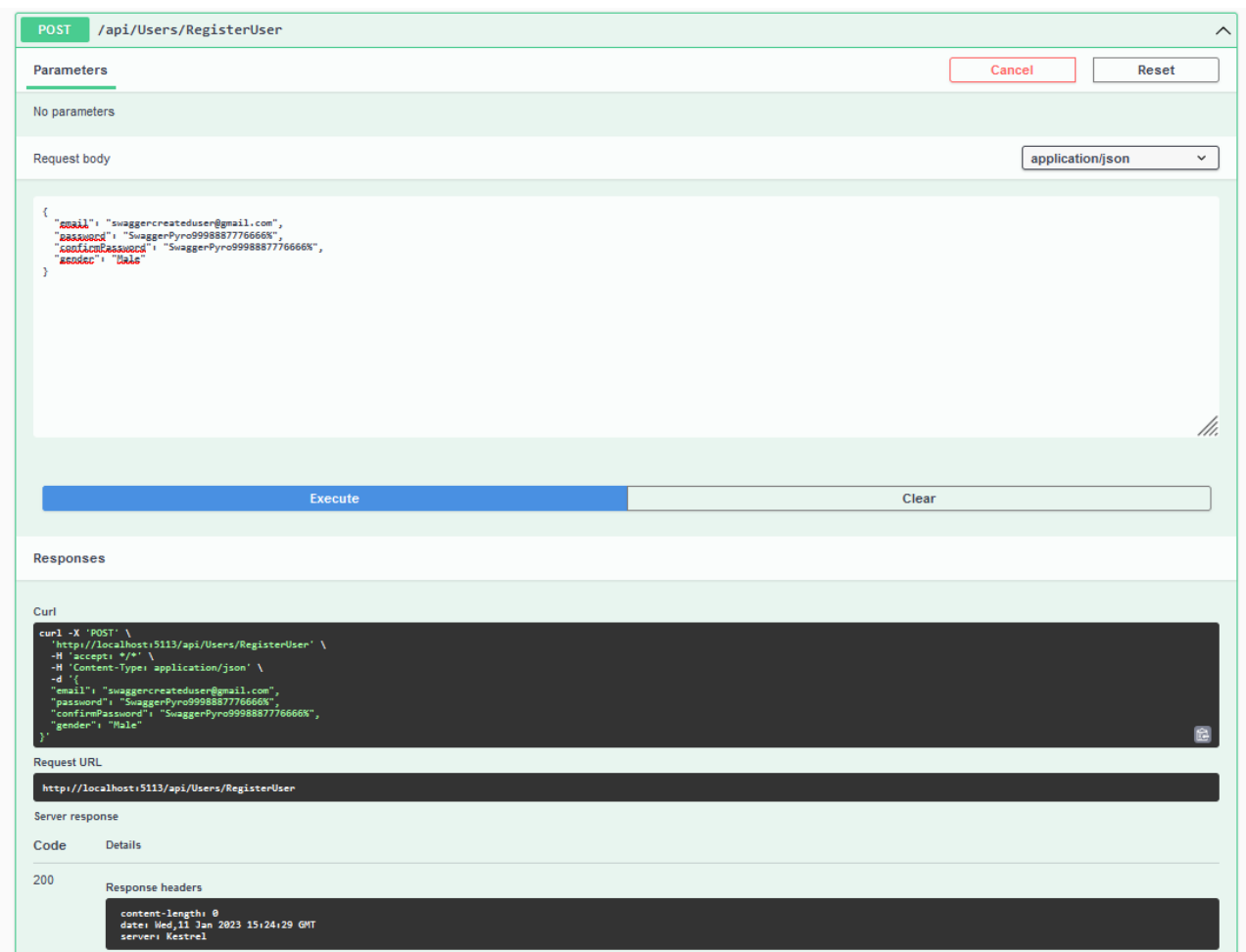


Рисунок 53. Функция RegisterUser

При переходе на сайт пользователя встречает главная страница, содержащая все товары, которые считаются популярными. Вверху справа есть кнопки входа или регистрации. Это обязательно, так как без регистрации пользователь не сможет оформлять заказы.

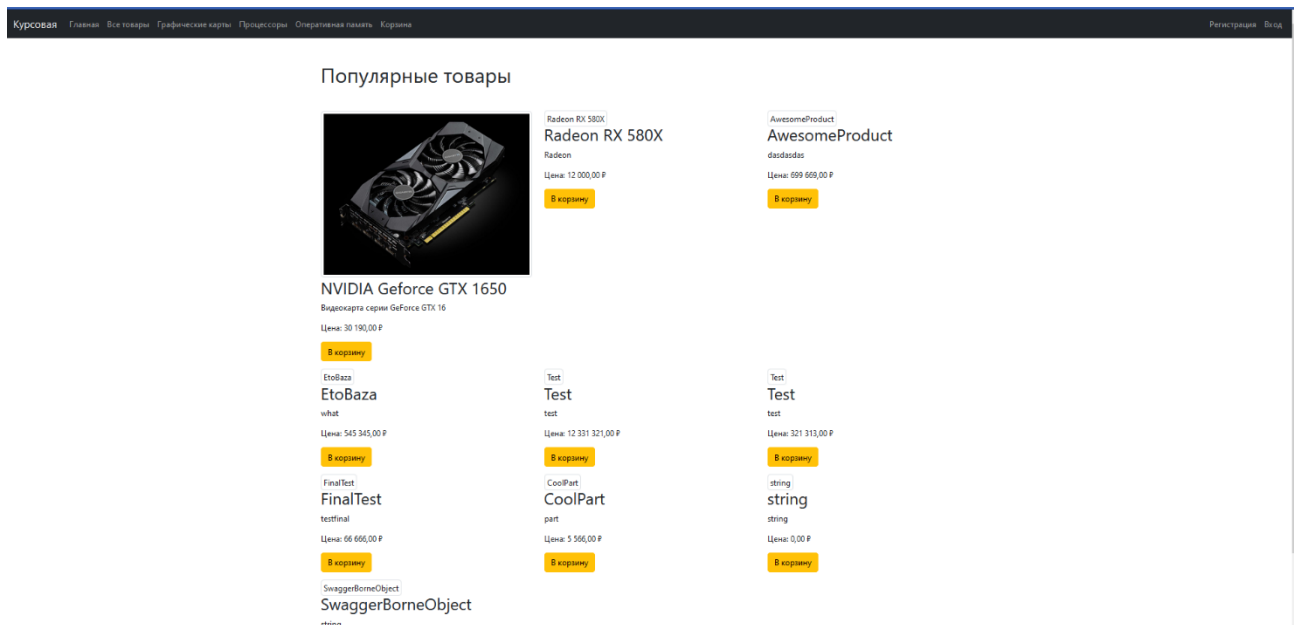


Рисунок 54. Главный экран.

Регистрация пользователя

Email
admin@gmail.com

Password
••••••••••

Confirm password

Gender

[Зарегистрироваться](#)

©2022 Company, Inc. · [Соглашение](#) [Вернуться наверх](#)

Рисунок 55. Окно входа в аккаунт

Вход в аккаунт

Email
admin@gmail.com

Password
••••••••••

☐ Запомнить логин

[Вход](#)

©2022 Company, Inc. · [Соглашение](#) [Вернуться наверх](#)

Рисунок 56. Окно регистрации

Пользователи, вошедшие в свой аккаунт, могут оформлять свои заказы, рассмотреть другие товары в каталоге товаров.

Деталь: NVIDIA Geforce GTX 1650
Цена: 30 190,00 Р
Удалить из корзины

Деталь: Intel Core i5-10400F
Цена: 11 990,00 Р
Удалить из корзины

Деталь: NVIDIA Geforce GTX 1050Ti
Цена: 21 000,00 Р
Удалить из корзины

Оплатить

©2022 Company, Inc. · [Соглашение](#)

Вернуться наверх

Добавить элемент

Оформление заказа

Введите имя

Egor

Введите имя

Zurrik

Введите адрес

leningradsкая 28 street Dubna

Введите номер телефона

79165236172

Завершить заказ

©2022 Company, Inc. · [Соглашение](#)

Вернуться наверх

Добавить элемент

Заказ успешно обработан!

©2022 Company, Inc. · [Соглашение](#)

Вернуться наверх

Добавить элемент

Рисунок 57. Корзина и оформление заказа.

Все товары



NVIDIA Geforce GTX 1650

Видеокарта серии GeForce GTX 16

Цена: 30 190,00 Р

В корзину



NVIDIA Geforce GTX 1050Ti

Видеокарта серии GeForce GTX 10

Цена: 21 000,00 Р

В корзину



Intel Core i5-10400F

Процессор Intel Core серии i5

Цена: 11 990,00 Р

В корзину



Оперативная память DDR4 8Gb

Четвёртое поколение DDR 8Гб памяти

Цена: 2 190,00 Р

В корзину



Оперативная память DDR4 16Gb

Четвёртое поколение DDR 16Гб памяти

Цена: 4 835,00 Р

В корзину

Radeon RX 580X

Radeon RX 580X

Radeon

Цена: 12 000,00 Р

В корзину



NVIDIA Geforce GTX 1650

Видеокарта серии GeForce GTX 16

Цена: 30 190,00 Р

В корзину

AwesomeProduct

AwesomeProduct

dasdasdas

Цена: 699 669,00 Р

В корзину

string

string

string

Цена: 0,00 Р

В корзину



NVIDIA Geforce GTX 1050Ti

Видеокарта серии GeForce GTX 10

Цена: 21 000,00 Р

В корзину

Test

Test

test

Цена: 12 331 321,00 Р

В корзину

SwaggerBorneObject

SwaggerBorneObject

string

Цена: 342 432 432,00 Р

В корзину

Radeon RX 580X

Radeon RX 580X

Radeon

Цена: 12 000,00 Р

В корзину

Test

Test

test

Цена: 321 313,00 Р

В корзину



Intel Core i5-10400F

Процессор Intel Core серии i5

Цена: 11 990,00 Р

В корзину

CoolPart

CoolPart

part

Цена: 5 566,00 Р

В корзину

© 2022 Company, Inc. · [Соглашение](#)

[Добавить элемент](#)

EtoBaza

EtoBaza

what

Цена: 545 345,00 Р

В корзину

FinalTest

FinalTest

testfinal

Цена: 66 666,00 Р

В корзину

TestProcessor

TestProcessor

processes

Цена: 124 124 124,00 Р

В корзину

[Вернуться наверх](#)



Оперативная память DDR4 8Gb

Четвёртое поколение DDR 8Гб памяти

Цена: 2 190,00 Р

В корзину

© 2022 Company, Inc. · [Соглашение](#)

[Добавить элемент](#)



Оперативная память DDR4 16Gb

Четвёртое поколение DDR 16Гб памяти

Цена: 4 835,00 Р

В корзину

FinaFINALtest

FinaFINALtest

testtt

Цена: 666 777,00 Р

В корзину

[Вернуться наверх](#)

Рисунок 58. Список товаров.

Если пользователь зарегистрирован и имеет роль администратора, то у него есть дополнительные опции. Он может добавить новые товары в список товаров, он может изменить список ролей, или состав пользователей этих ролей. Администраторы также способны изменять информацию об аккаунтах пользователей, а также если это нужно, регистрировать новые аккаунты на сайт.

[Список пользователей](#) [Список ролей](#)

[Добавить элемент](#)

Рисунок 57. Элементы, доступные только администраторам.

Добавление детали в каталог

name	<input type="text"/>
shortDesc	<input type="text"/>
longDesc	<input type="text"/>
img	<div><div>Обзор...</div><div>Файл не выбран.</div></div>
price	<input type="text"/>
isavailable	<input type="checkbox"/>
isFavourite	<input type="checkbox"/>
categoryID	<input type="text"/>
<div>Создать элемент</div>	

Все пользователи сайта

[Зарегистрировать пользователя](#)

User Id : 582a5076-d962-4049-9d3a-6ae1ee6e21cc
zupnike6g@gmail.com
Изменить Удалить

User Id : 659abbfe-1e41-450e-ab20-f49904a4bbad
janezupnikk@gmail.com
Изменить Удалить

User Id : 7eea3a39-8a4c-4991-b167-6e10f130fd02
swaggercreateduser@gmail.com
Изменить Удалить

User Id : 855f6680-76a1-457c-bc1f-7096decd50e5
admin@gmail.com
Изменить Удалить

User Id : 8bc22ee0-1b9a-4391-962a-c71340fe678b
egzup@gmail.com
Изменить Удалить

Рисунок 59. Страница создания новой детали для добавления в каталог.

Редактирование данных пользователя

Id	582a5076-d962-4049-9d3a-6ae1ee6e21cc
Email	zupnike44g@gmail.com
UserName	zupnike6g@gmail.com
Gender	Male

[Обновить](#) [Отменить](#)

Роли пользователя
UserS
Управлять ролями

Рисунок 60. Страница пользователей, которую может открыть администратор.

Все роли

[Добавить новую роль](#)

Id Роли : 14300a42-fc5d-42a0-864e-cab10f657dbb
Admin
Изменить Удалить

Id Роли : 8b4df523-184c-4092-b396-7e96c7c089d2
TestRoleRole
Изменить Удалить

Id Роли : da9ce83d-e411-4c86-a71a-bd35edc0f431
UserS
Изменить Удалить

Изменить Роль

Id	14300a42-fc5d-42a0-864e-cab10f657dbb
RoleName	Admin

[Обновить](#) [Отменить](#)

Users in this role
admin@gmail.com egzup@gmail.com zupnikegg@gmail.com
Добавить или убрать пользователей

Добавить или убрать пользователей из этой роли

☐ zupnike6g@gmail.com

☐ janezupnikk@gmail.com

☐ swaggercreateduser@gmail.com

☒ admin@gmail.com

☒ egzup@gmail.com

☐ qweeee@gmail.com

☒ zupnikegg@gmail.com

☐ tester@gmail.com

Update

Отменить

©2022 Company, Inc. · [Соглашение](#)

[Вернуться наверх](#)

Рисунок 61. Страница пользователей, которую может открыть администратор.

Итак, было рассмотрено всё содержимое проекта: от его кода до самих интерфейсов, которыми этот проект пользуется для взаимодействия с пользователями.

ЗАКЛЮЧЕНИЕ

По итогу, цель данного проекта была достигнута.

Информационная система была реализована при помощи Entity Framework, ASP NET Core и Visual Studio, благодаря чему был получен опыт работы с этими инструментами. Для создания визуальной части также использовалась оболочка Visual Studio, а также инструмент под названием Razor для визуальной и функциональной составляющей страниц.

В ходе проекта были получены навыки:

- проектирования баз данных;
- работы с Entity Framework;
- создания сайта при помощи ASP.NET;
- интеграции и взаимодействия с созданной базой данных с помощью Entity Framework;
- практические навыки работы с Swagger.

В результате осуществления данного проекта была создана мобильная игра про дрифт, с подключенной базой данных пользователей и машин.

Библиографический список

1. Ник Рендольф, Дэвид Гарднер, Майкл Минутилло, Крис Андерсон. Visual Studio 2010 для профессионалов = Professional Visual Studio 2010. — М.: «Диалектика», 2011. — С. 1184. — ISBN 978-5-8459-1683-9.
2. Алекс Макки. Введение в .NET 4.0 и Visual Studio 2010 для профессионалов = Introducing .NET 4.0: with Visual Studio 2010. — М.: «Вильямс», 2010. — С. 416. — ISBN 978-5-8459-1639-6.
3. Джесс Чедвик и др. ASP.NET MVC 4: разработка реальных веб-приложений с помощью ASP.NET MVC = Programming ASP.NET MVC 4: Developing Real-World Web Applications with ASP.NET MVC. — М.: «Вильямс», 2013. — 432 с. — ISBN 978-5-8459-1841-3.
4. Джон Скит. C# для профессионалов: тонкости программирования, 3-е издание, новый перевод = C# in Depth, 3rd ed.. — М.: «Вильямс», 2014. — 608 с. — ISBN 978-5-8459-1909-0.
5. Адам Фримен. ASP.NET MVC 5 с примерами на C# 5.0 для профессионалов, 5-е издание = Pro ASP.NET MVC 5. — М.: «Вильямс», 2014. — 736 с. — ISBN 978-5-8459-1911-3.