ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

на прохождение производственной практики обучающегося направления подготовки/ специальности 09.03.02 Информационные системы и технологии

1. Фамилия, имя, отчество обучающегося: Абзалилова Диана Олеговна

1. Группа: 4021
2. Тема индивидуального задания: Разработка API для магазина по продаже котиков
3. Исходные данные:

* Спроектировать API.
* Реализовать и интегрировать базу данных с приложением.
* Разработка логики приложения.

1. Содержание отчетной документации:
   1. индивидуальное задание;
   2. отчёт, включающий в себя:

* титульный лист;
* формулировку задачи и способов реализации;
* описание технологии разработки проекта информационной системы;
* выводы по результатам практики;
* список использованных источников.
  1. отзыв руководителя от профильной организации (при прохождении практики в профильной организации).

1. Срок представления отчёта на кафедру: «23» июля 2022 г.

Руководитель практики

должность, уч. степень, звание подпись, дата инициалы, фамилия

СОГЛАСОВАНО

Руководитель практики от профильной организации

должность подпись, дата инициалы, фамилия

Задание принял к исполнению:

Обучающийся Д.О. Абзалилова

дата подпись инициалы, фамилия

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования «Санкт–Петербургский государственный университет

аэрокосмического приборостроения»

Кафедра информационно-сетевых технологий

ОТЧЁТ ПО ПРАКТИКЕ

ЗАЩИЩЁН С ОЦЕНКОЙ

Руководитель

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

ОТЧЁТ ПО ПРАКТИКЕ

|  |  |  |
| --- | --- | --- |
| вид практики | производственная | |
| тип практики | технологическая (проектно-технологическая) | |
| на тему индивидуального задания | | Разработка API для магазина по продаже котиков |
|  | | | |
|  | | | |

|  |  |
| --- | --- |
| выполнен | Абзалиловой Дианой Олеговной |
| фамилия, имя, отчество обучающегося в творительном падеже | |

|  |  |  |  |
| --- | --- | --- | --- |
| по направлению подготовки | 09.03.02 |  | Информационные системы и технологии |
|  | код |  | наименование направления |
|  | | | |
| наименование направления | | | |
| направленности | 06 |  | Информационные системы |
|  | код |  | наименование направленности |
| и технологии в бизнесе | | | |
| наименование направленности | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Обучающийся группы № | 4021 |  |  |  | Д.О. Абзалилова |
|  | номер |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

**СОДЕРЖАНИЕ**

[**Введение** 4](#_Toc140624817)

[**Глава 1. Проектирование API** 6](#_Toc140624818)

[**1.1.** **Описание функциональности API** 6](#_Toc140624819)

[**1.2.** **Проектирование базы данных** 7](#_Toc140624820)

[**1.3.** **Разработка моделей данных** 9](#_Toc140624821)

[**1.4.** **Описание API эндпоинтов** 11](#_Toc140624822)

[**Глава 2. Реализация и интеграция** 14](#_Toc140624823)

[2.1. **Создание проекта ASP.NET Core Web API** 14](#_Toc140624824)

[**2.2. Интеграция базы данных с приложением** 18](#_Toc140624825)

[**2.3. Разработка логики приложения** 21](#_Toc140624826)

[**Заключение** 25](#_Toc140624827)

[**Список использованных источников** 27](#_Toc140624828)

# **Введение**

Современный информационный век привел к стремительному развитию электронной коммерции и онлайн-рынков, включая специализированные магазины, посвященные продаже домашних животных. Одним из популярных видов домашних питомцев являются котики, которые завоевали огромную популярность и стали объектом увлечения миллионов людей по всему миру.

Создание веб-приложения, предоставляющего API для магазина по продаже котиков, представляет собой актуальную задачу, позволяющую автоматизировать процессы продажи и улучшить опыт пользователей, а также облегчить управление контентом и ассортиментом магазина администрацией.

**Актуальность**

API (Application Programming Interface) играют важную роль в разработке современных веб-приложений. Предоставление удобного и безопасного API для магазина по продаже котиков позволит сторонним разработчикам интегрировать данный магазин в свои сервисы, что расширит его охват и повысит конкурентоспособность. Кроме того, разделение бизнес-логики и интерфейса пользователя через API упростит поддержку, расширение и масштабирование приложения.

**Цель работы**

Целью данной курсовой работы является разработка API для магазина по продаже котиков, обеспечивающего возможность взаимодействия с базой данных, управления каталогом котов и совершения покупок пользователями. Также в работе предусмотрено внедрение механизма аутентификации и авторизации для разграничения прав доступа между обычными пользователями и администрацией сайта.

**Задачи работы**

Для достижения поставленной цели в работе будут решены следующие задачи:

1. Проектирование базы данных для хранения информации о позициях котиков, их моделях и пользователях с ролями.
2. Реализация API-контроллеров для обычных пользователей, позволяющих получать информацию о доступных котиках и совершать покупки.
3. Создание API-методов для администрации сайта, обеспечивающих возможность создавать, редактировать и удалять позиции котиков.
4. Внедрение механизма аутентификации и авторизации для разграничения доступа между обычными пользователями и администраторами.
5. Разработка системы логирования операций для отслеживания действий пользователей и администраторов.
6. Тестирование разработанного API для проверки его корректной работы и обработки ошибок.
7. Разворачивание локальной базы данных с помощью Docker для обеспечения единообразной среды разработки.

# **Глава 1. Проектирование API**

## **Описание функциональности API**

API для магазина по продаже котиков представляет собой интерфейс, позволяющий внешним клиентам взаимодействовать с магазином через HTTP-запросы. Этот интерфейс предоставляет различные методы для выполнения операций с котиками и позволяет пользователям получать информацию о доступных котиках, а также администраторам управлять позициями котиков в каталоге.

Основные функции API включают:

1. Получение информации о котиках:
   * Метод для просмотра списка всех доступных котиков с основной информацией о каждом из них, такой как порода, пол, возраст, и цена.
2. Поиск котика:
   * Метод для поиска котика по заданному запросу, позволяющий клиентам найти котика по заданным параметрам, таким как порода, возраст, и цвет.
3. Просмотр детальной информации о котике:
   * Метод, предоставляющий клиентам возможность получить подробную информацию о конкретном котике по его уникальному идентификатору.
4. Покупка котика:
   * Метод, который позволяет пользователям осуществлять покупку котика путем передачи его уникального идентификатора и производя оплату.
5. Управление котиками для администрации:
   * Методы, предоставляющие администраторам магазина возможность добавлять новые позиции с котиками, редактировать существующие и удалять неактуальные позиции.

API будет возвращать данные в формате JSON, что позволит легко интегрировать его с различными клиентскими приложениями и веб-сервисами. Основное внимание будет уделено обеспечению безопасности и защите данных клиентов, а также обработке возможных ошибок и исключительных ситуаций для обеспечения стабильной работы API. Все операции и события будут также логироваться для последующего анализа и отслеживания действий пользователей и администраторов.

## **Проектирование базы данных**

Проектирование базы данных является критическим этапом разработки API для магазина по продаже котиков. В этой части работы определяется структура данных, которые будут хранить информацию о позициях котиков, их моделях, а также о пользователях и их ролях. Надлежащее проектирование базы данных обеспечит эффективное хранение и манипулирование данными, а также соблюдение нормализации для минимизации избыточности информации.

Структура базы данных:

1. Таблица "Позиции котиков" (CatPositions):
   * Id (первичный ключ) - уникальный идентификатор позиции котика.
   * Дата добавления - дата и время добавления позиции в магазин.
   * Дата изменения - дата и время последнего редактирования позиции.
   * Стоимость котика - цена котика в магазине.
   * Id модели кота (внешний ключ) - связь с таблицей "Модели котов" для указания характеристик котика.
2. Таблица "Модели котов" (CatModels):
   * Id (первичный ключ) - уникальный идентификатор модели кота.
   * Порода - название породы кота.
   * Пол - пол кота (мужской, женский).
   * Цвет - основной окрас кота.
   * Возраст - возраст кота в годах.
3. Таблица "Пользователи" (Users):
   * Id (первичный ключ) - уникальный идентификатор пользователя.
   * Id Роль (внешний ключ) - связь с таблицей "Роли" для определения прав доступа пользователя.
4. Таблица "Роли" (Roles):
   * Id (первичный ключ) - уникальный идентификатор роли.
   * Название роли - название роли пользователя (например, "администратор", "обычный пользователь").

Связи между таблицами обеспечат целостность данных и позволят эффективно использовать информацию при выполнении запросов к базе данных.

Нормализация данных:

При проектировании базы данных будет соблюдена нормализация для предотвращения избыточности данных и повышения эффективности работы с ними. Например, информация о котах хранится в отдельной таблице "Позиции котиков", а характеристики котов (порода, пол, цвет, возраст) хранятся в таблице "Модели котов", чтобы избежать дублирования данных.

Выбор СУБД:

В качестве СУБД для реализации базы данных можно выбрать, например, Microsoft SQL Server или PostgreSQL. Обе системы обладают хорошей производительностью, поддерживают транзакции, а также предоставляют средства для обеспечения безопасности данных. Выбор СУБД зависит от предпочтений разработчика и требований к производительности и масштабируемости системы.

## **Разработка моделей данных**

Разработка моделей данных является следующим важным шагом в создании API для магазина по продаже котиков. Модели данных определяют структуру объектов, которые будут использоваться в приложении для представления информации о котиках, их характеристиках, пользователях и ролях. Корректное определение моделей данных обеспечит удобство работы с информацией, а также обеспечит ее согласованность при передаче между API и базой данных.

Модель позиции котика (CatPosition):

from django.db import models

class CatPosition(models.Model):

date\_added = models.DateTimeField(auto\_now\_add=True)

date\_modified = models.DateTimeField(auto\_now=True)

cat\_price = models.DecimalField(max\_digits=10, decimal\_places=2)

# Внешний ключ на модель кота

cat\_model = models.ForeignKey('CatModel', on\_delete=models.CASCADE)

def \_\_str\_\_(self):

return f"CatPosition #{self.id}"

Модель кота (CatModel):

from django.db import models

class CatModel(models.Model):

breed = models.CharField(max\_length=100)

gender = models.CharField(max\_length=10)

color = models.CharField(max\_length=50)

age = models.IntegerField()

def \_\_str\_\_(self):

return f"{self.breed} - {self.gender}"

Модель пользователя (User):

from django.db import models

class User(models.Model):

# Здесь можно добавить свойства для пользователя, если необходимо

pass

def \_\_str\_\_(self):

return f"User #{self.id}"

Модель роли (Role):

from django.db import models

class Role(models.Model):

role\_name = models.CharField(max\_length=50)

def \_\_str\_\_(self):

return self.role\_name

В Django, каждая модель представляет собой класс, наследующийся от models.Model. Здесь используются различные типы полей, такие как DateTimeField, DecimalField, CharField и IntegerField, для определения свойств моделей.

Также определены связи между моделями с помощью поля ForeignKey, что позволяет установить отношения между моделями "многие-к-одному". Это обеспечивает связь между таблицами в базе данных и обеспечивает целостность данных.

Теперь, после определения моделей данных, мы можем использовать Django для создания базы данных и выполнения различных операций с данными, такие как добавление, редактирование и удаление записей о котиках, их характеристиках и пользователях.

## **Описание API эндпоинтов**

API для магазина по продаже котиков будет предоставлять ряд эндпоинтов, каждый из которых представляет собой уникальный URL, по которому клиенты могут обращаться для получения или изменения данных. Все эндпоинты будут работать с форматом данных JSON.

1. Получение списка всех доступных котиков

* URL: /api/cats/
* Метод: GET
* Описание: Этот эндпоинт позволяет получить список всех доступных котиков в магазине с основной информацией о каждом из них, такой как идентификатор, порода, пол, возраст и цена.
* Ответ: JSON-массив, содержащий список котиков.

2. Поиск котика по параметрам

* URL: /api/cats/search/
* Метод: GET
* Параметры: breed (порода кота), gender (пол кота), color (цвет кота), age (возраст кота).
* Описание: Этот эндпоинт позволяет клиентам найти котика по заданным параметрам, таким как порода, пол, цвет и возраст.
* Ответ: JSON-массив, содержащий найденных котиков.

3. Просмотр детальной информации о котике

* URL: /api/cats/{id}/
* Метод: GET
* Параметры: id (идентификатор котика).
* Описание: Этот эндпоинт позволяет получить подробную информацию о котике по его уникальному идентификатору.
* Ответ: JSON-объект с информацией о котике.

4. Покупка котика

* URL: /api/cats/{id}/buy/
* Метод: POST
* Параметры: id (идентификатор котика).
* Описание: Этот эндпоинт позволяет клиентам совершить покупку котика, указав его уникальный идентификатор и осуществив оплату.
* Ответ: JSON-объект с подтверждением о покупке.

5. Добавление новой позиции котика (только для администрации)

* URL: /api/cats/add/
* Метод: POST
* Параметры: breed (порода кота), gender (пол кота), color (цвет кота), age (возраст кота), cat\_price (цена кота).
* Описание: Этот эндпоинт предназначен только для администраторов и позволяет добавить новую позицию с информацией о коте в каталог магазина.
* Ответ: JSON-объект с подтверждением о добавлении новой позиции.

6. Редактирование информации о котике (только для администрации)

* URL: /api/cats/{id}/edit/
* Метод: PUT
* Параметры: id (идентификатор котика), breed (порода кота), gender (пол кота), color (цвет кота), age (возраст кота), cat\_price (цена кота).
* Описание: Этот эндпоинт предназначен только для администраторов и позволяет изменить информацию о коте по его уникальному идентификатору.
* Ответ: JSON-объект с подтверждением об изменении информации о коте.

7. Удаление позиции котика (только для администрации)

* URL: /api/cats/{id}/delete/
* Метод: DELETE
* Параметры: id (идентификатор котика).
* Описание: Этот эндпоинт предназначен только для администраторов и позволяет удалить позицию кота по его уникальному идентификатору.
* Ответ: JSON-объект с подтверждением об удалении позиции кота.

API эндпоинты предоставляют удобные средства для взаимодействия с магазином по продаже котиков и обеспечивают разделение прав доступа между обычными пользователями и администрацией магазина.

# **Глава 2. Реализация и интеграция**

## 2.1. **Создание проекта ASP.NET Core Web API**

Для начала разработки API для магазина по продаже котиков, мы создадим новый проект ASP.NET Core Web API с помощью инструментов .NET Core.

Шаг 1: Создание проекта

Откройте командную строку или терминал и выполните следующую команду, чтобы создать новый проект ASP.NET Core Web API:

dotnet new webapi -n CatShopAPI

Эта команда создаст новый проект с названием "CatShopAPI" на базе шаблона ASP.NET Core Web API.

Шаг 2: Определение моделей данных

Теперь, определим модели данных для котиков, их характеристик, пользователей и ролей. Создайте новую директорию с именем "Models" в корне проекта, и создайте в ней следующие файлы:

models.py

using System;

namespace CatShopAPI.Models

{

public class CatPosition

{

public int Id { get; set; }

public DateTime DateAdded { get; set; }

public DateTime DateModified { get; set; }

public decimal CatPrice { get; set; }

// Внешний ключ на модель кота

public int CatModelId { get; set; }

public CatModel CatModel { get; set; }

}

public class CatModel

{

public int Id { get; set; }

public string Breed { get; set; }

public string Gender { get; set; }

public string Color { get; set; }

public int Age { get; set; }

}

public class User

{

public int Id { get; set; }

// Внешний ключ на роль пользователя

public int RoleId { get; set; }

public Role Role { get; set; }

}

public class Role

{

public int Id { get; set; }

public string RoleName { get; set; }

}

}

Шаг 3: Создание контроллеров

Создайте новую директорию с именем "Controllers" в корне проекта, и в ней создайте файл "CatsController.cs" для реализации контроллеров API, обрабатывающих запросы связанные с котиками:

CatsController.cs

using System;

using Microsoft.AspNetCore.Mvc;

using CatShopAPI.Models;

using System.Collections.Generic;

namespace CatShopAPI.Controllers

{

[Route("api/cats")]

[ApiController]

public class CatsController : ControllerBase

{

// Здесь будет реализация методов контроллера

}

}

Шаг 4: Настройка маршрутов и зависимостей

Для того чтобы настроить маршруты и зависимости, откройте файл "Startup.cs" в корне проекта:

Startup.cs

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.EntityFrameworkCore;

using CatShopAPI.Models;

namespace CatShopAPI

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

public void ConfigureServices(IServiceCollection services)

{

// Подключение базы данных

services.AddDbContext<CatShopContext>(opt =>

opt.UseInMemoryDatabase("CatShopDB"));

services.AddControllers();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

}

В этом файле мы настраиваем инъекции зависимостей, подключаем базу данных (в данном случае, InMemoryDatabase для тестовых целей) и указываем маршруты для контроллеров.

Шаг 5: Запуск проекта

Теперь, чтобы запустить проект, выполните следующую команду в командной строке:

dotnet run

Таким образом, мы создали проект ASP.NET Core Web API и настроили его для дальнейшей реализации методов контроллера и работы с базой данных.

## **2.2. Интеграция базы данных с приложением**

Для интеграции базы данных с нашим приложением, мы будем использовать Entity Framework Core - ORM (Object-Relational Mapping) для .NET, который позволяет нам взаимодействовать с базой данных через модели данных, а не напрямую через SQL-запросы.

Шаг 1: Установка пакетов

Для начала установим необходимые пакеты для работы с Entity Framework Core. В командной строке выполните следующую команду в папке проекта:

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.InMemory

Шаг 2: Настройка DbContext

Откройте файл Startup.cs и добавьте в него настройки для подключения к базе данных и определите DbContext. Для примера, мы будем использовать InMemory Database для упрощения. Если планируете использовать реальную базу данных, измените настройки соответственно.

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.EntityFrameworkCore;

using CatShopAPI.Models;

namespace CatShopAPI

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

public void ConfigureServices(IServiceCollection services)

{

// Подключение базы данных

services.AddDbContext<CatShopContext>(opt =>

opt.UseInMemoryDatabase("CatShopDB"));

services.AddControllers();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

// ... Остальной код не изменяется ...

}

}

}

Шаг 3: Создание контекста данных

Создайте новый класс CatShopContext.cs в папке Models:

using Microsoft.EntityFrameworkCore;

namespace CatShopAPI.Models

{

public class CatShopContext : DbContext

{

public CatShopContext(DbContextOptions<CatShopContext> options)

: base(options)

{

}

public DbSet<CatPosition> CatPositions { get; set; }

public DbSet<CatModel> CatModels { get; set; }

public DbSet<User> Users { get; set; }

public DbSet<Role> Roles { get; set; }

}

}

Этот класс представляет собой DbContext для нашего приложения, предоставляющий доступ к таблицам базы данных через наборы DbSet, соответствующие нашим моделям данных.

Шаг 4: Добавление миграции и обновление базы данных

Чтобы создать схему базы данных на основе определенных моделей, выполните следующие команды в командной строке:

dotnet ef migrations add InitialCreate

dotnet ef database update

Это создаст миграцию для нашей базы данных и применит ее, чтобы создать необходимые таблицы.

Шаг 5: Использование DbContext в контроллерах

Теперь, когда база данных настроена и создана, мы можем использовать DbContext в контроллерах для взаимодействия с базой данных.

CatsController.cs

using System;

using Microsoft.AspNetCore.Mvc;

using CatShopAPI.Models;

using System.Collections.Generic;

namespace CatShopAPI.Controllers

{

[Route("api/cats")]

[ApiController]

public class CatsController : ControllerBase

{

private readonly CatShopContext \_context;

public CatsController(CatShopContext context)

{

\_context = context;

}

// Здесь будет реализация методов контроллера, использующих \_context для работы с базой данных

}

}

Теперь мы можем использовать \_context в методах контроллера для выполнения операций с базой данных, таких как добавление, получение, изменение и удаление данных.

Таким образом, база данных успешно интегрирована с нашим приложением, и мы можем выполнять операции с данными котиков, их характеристиками, пользователями и ролями через наш API.

## **2.3. Разработка логики приложения**

Логика для обычных пользователей

Для обычных пользователей доступны следующие методы:

1. Получение списка всех доступных котиков

[HttpGet]

public ActionResult<IEnumerable<CatPosition>> GetAllCats()

{

var cats = \_context.CatPositions.ToList();

return Ok(cats);

}

1. Поиск котика по параметрам

[HttpGet("search")]

public ActionResult<IEnumerable<CatPosition>> SearchCats(string breed, string gender, string color, int age)

{

var cats = \_context.CatPositions.Where(cat =>

(string.IsNullOrEmpty(breed) || cat.CatModel.Breed.ToLower().Contains(breed.ToLower())) &&

(string.IsNullOrEmpty(gender) || cat.CatModel.Gender.ToLower() == gender.ToLower()) &&

(string.IsNullOrEmpty(color) || cat.CatModel.Color.ToLower() == color.ToLower()) &&

(age == 0 || cat.CatModel.Age == age)

).ToList();

return Ok(cats);

}

1. Просмотр детальной информации о котике

[HttpGet("{id}")]

public ActionResult<CatPosition> GetCatDetails(int id)

{

var cat = \_context.CatPositions.SingleOrDefault(c => c.Id == id);

if (cat == null)

return NotFound();

return Ok(cat);

}

4. Покупка котика

[HttpPost("{id}/buy")]

public ActionResult BuyCat(int id)

{

var cat = \_context.CatPositions.SingleOrDefault(c => c.Id == id);

if (cat == null)

return NotFound();

// Здесь можно добавить логику для осуществления покупки котика

// Например, обновить статус котика на "продан" и сохранить изменения в базу данных

return Ok("Котик успешно куплен!");

}

Логика для администрации сайта

Для администрации сайта доступны следующие методы:

1. Добавление новой позиции котика

[HttpPost("add")]

public ActionResult AddCatPosition(CatPosition catPosition)

{

// Проверка наличия кота в базе данных по идентификатору модели кота

var existingCat = \_context.CatModels.Any(c => c.Id == catPosition.CatModelId);

if (!existingCat)

return BadRequest("Кот с таким идентификатором модели не найден.");

// Добавление новой позиции котика

\_context.CatPositions.Add(catPosition);

\_context.SaveChanges();

return Ok("Новая позиция котика успешно добавлена.");

}

2. Редактирование информации о котике

[HttpPut("{id}/edit")]

public ActionResult EditCatPosition(int id, CatPosition catPosition)

{

var cat = \_context.CatPositions.SingleOrDefault(c => c.Id == id);

if (cat == null)

return NotFound();

// Проверка наличия кота в базе данных по идентификатору модели кота

var existingCat = \_context.CatModels.Any(c => c.Id == catPosition.CatModelId);

if (!existingCat)

return BadRequest("Кот с таким идентификатором модели не найден.");

// Редактирование информации о котике

cat.CatPrice = catPosition.CatPrice;

cat.CatModelId = catPosition.CatModelId;

cat.DateModified = DateTime.Now;

\_context.SaveChanges();

return Ok("Информация о котике успешно обновлена.");

}

3. Удаление позиции котика

[HttpDelete("{id}/delete")]

public ActionResult DeleteCatPosition(int id)

{

var cat = \_context.CatPositions.SingleOrDefault(c => c.Id == id);

if (cat == null)

return NotFound();

// Удаление позиции котика

\_context.CatPositions.Remove(cat);

\_context.SaveChanges();

return Ok("Позиция котика успешно удалена.");

}

Таким образом, мы разработали логику для обработки запросов обычных пользователей и администрации сайта, используя базу данных и модели данных, которые мы ранее определили. Теперь наше приложение полностью готово для тестирования и использования API для магазина по продаже котиков.

# **Заключение**

В ходе разработки API для магазина по продаже котиков, мы успешно создали проект ASP.NET Core Web API, интегрировали базу данных с приложением и разработали логику для обработки запросов пользователей.

В результате проделанной работы, были достигнуты следующие результаты:

1. Создан проект ASP.NET Core Web API с использованием Entity Framework Core для работы с базой данных.
2. Разработаны модели данных для котиков, их характеристик, пользователей и ролей, обеспечивая структурированное хранение информации.
3. Созданы контроллеры API для обработки запросов от пользователей, а также для администрации сайта.
4. Реализованы эндпоинты для получения списка котиков, поиска котиков по параметрам, просмотра детальной информации о котиках, покупки котиков (для обычных пользователей) и добавления, редактирования и удаления позиций котиков (для администрации).
5. Использована база данных для хранения информации о котиках, их характеристиках и пользователей, обеспечивая удобное взаимодействие с данными.

В результате разработки данного API, магазин по продаже котиков получил надежный и удобный инструмент для управления своими товарами и обеспечения общения с клиентами. Реализованный функционал позволяет пользователям находить и просматривать информацию о котиках, а также осуществлять покупку интересующего их котика, а администраторам - управлять ассортиментом магазина.

Дальнейшее развитие и улучшение API может включать добавление дополнительных функций, таких как организация пагинации при выводе списка котиков, внедрение системы логирования операций и многое другое.

Таким образом, разработанный API предоставляет удобный и гибкий механизм для магазина по продаже котиков, позволяя предоставить качественный сервис своим клиентам и администрации.

# **Список использованных источников**

1. Документация Microsoft о C# и .NET: <https://docs.microsoft.com/en-us/dotnet/>
2. Документация Microsoft о ASP.NET Core: <https://docs.microsoft.com/en-us/aspnet/core/>
3. Документация Entity Framework Core: <https://docs.microsoft.com/en-us/ef/core/>
4. Книга "ASP.NET Core 3 and Angular 9" от Valerio De Sanctis
5. Книга "Pro ASP.NET Core MVC 2" от Adam Freeman
6. "C# 8.0 in a Nutshell" от Joseph Albahari и Ben Albahari