Санкт–Петербургское государственное бюджетное профессиональное образовательное учреждение

«Колледж информационных технологий»

ОТЧЕТ

по производственной практике

**ПМ.01 Разработка модулей программного обеспечения для компьютерных систем**

Специальность 09.02.07 Информационные системы и программирование (программист)

Выполнил

студент гр. 493 \_\_\_\_\_\_\_\_А.Д. Сидоров

Согласовано  
ООО «Омега» \_\_\_\_\_\_\_\_\_С.В. Литвиненко

Руководитель производственной практики \_\_\_\_\_\_\_\_\_\_Н.В. Романовская

Санкт–Петербург   
2021

СОДЕРЖАНИЕ

[Введение 3](#_Toc131080475)

[1. Предметная область 5](#_Toc131080476)

[2. Техническое задание 6](#_Toc131080477)

[3. Формирование алгоритмов разработки программных модулей в соответствии с техническим заданием 7](#_Toc131080478)

[4. Разработка программных модулей в соответствии с техническим заданием 10](#_Toc131080479)

[5. Выполнение отладки программных модулей с использованием специализированных программных средств 15](#_Toc131080480)

[6. Выполнение тестирования программных модулей 18](#_Toc131080481)

[7. Осуществление рефакторинга и оптимизации программного кода 26](#_Toc131080482)

[8. Разработка модулей программного обеспечения для мобильных платформ 33](#_Toc131080483)

[Заключение 35](#_Toc131080484)

[Список использованных источников 37](#_Toc131080485)

[Приложение 38](#_Toc131080486)

# Введение

Целью производственной практики является освоение общих компетенций:

1. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам;
2. Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности;
3. Планировать и реализовывать собственное профессиональное и личностное развитие;
4. Работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством, клиентами;
5. Осуществлять устную и письменную коммуникацию на государственном языке с учётом особенностей социального и культурного контекста;
6. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей;
7. Содействовать сохранению окружающей среды, ресурсосбережению, эффективно действовать в чрезвычайных ситуациях;
8. Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания уровня физической подготовленности;
9. Использовать информационные технологии в профессиональной деятельности;
10. Пользоваться профессиональной документацией на государственном и иностранном языке;
11. Планировать предпринимательскую деятельность в профессиональной сфере.

Также, целью производственной практики является освоение профессиональных компетенций:

1. Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием;
2. Разрабатывать программные модули в соответствии с техническим заданием;
3. Выполнять отладку программных модулей с использованием специализированных программных средств;
4. Выполнять тестирование программных модулей;
5. Осуществлять рефакторинг и оптимизацию программного кода;
6. Разрабатывать модули программного обеспечения для мобильных платформ.

# Предметная область

На производственной практике был выбор предметных областей для прохождения практики, и мной была выбрана предметна область «Магазин котиков», как показано на рисунке 1.1.



Рисунок 1.1 – Выбранная предметная область

Выполненная работа находится по адресу <https://github.com/AntonSidorov1/InterShipOooOmega>.

Необходимо было разработать систему, в которой присутствует база данных и Web API к этой базе данных. Также, есть часть, которая не связана с заданием, но связана с освоением одной из компетенций – Мобильное приложение под созданное Web API.

В данной работе были разработаны база данных, Web API и мобильное приложение. Необходимые навыки для работы были получены при выполнении.

# Техническое задание

Для выполнения задания требуется спроектировать базу данных, в которой будет храниться информация о пользователях (логин, пароль, роль в системе) и о позициях котиков (пол, возраст, цвет, порода, цена, дата добавления и дата изменения). Спроектированную базу данных необходимо разработать и интегрировать с ней приложение. Для реализации логики приложения необходимо разработать API, в соответствии с следующим заданием:

Необходимо разработать API для магазина по продаже котиков (каждый котик представляет из себя позицию на сайте, поэтому дальше в тексте обращаться к ним будем как к позиции)

Список методов для обычного пользователя:

* Получение позиции;
* Покупка позиции.

Список методов для администрации:

* Создать позицию;
* Отредактировать позицию;
* Удалить позицию.

Поскольку система будет работать в многопользовательском режиме, и в ней присутствуют роли, каждая из которых имеет свой набор функций, должна быть предусмотрена авторизация, которая проходит успешно при правильном логине и пароле.

# Формирование алгоритмов разработки программных модулей в соответствии с техническим заданием

В данном разделе описываются алгоритмы, которые я разработал в соответствии с выбранной предметной областью. Эти алгоритмы представлены в таблице 1.1.

Таблица 1.1 – Разработанные алгоритмы.

|  |  |
| --- | --- |
| **Название алгоритма** | **Изображение алгоритма** |
| Алгоритм авторизации | Изображение выглядит как диаграмма  Автоматически созданное описание |
| Алгоритм регистрации |  |
| Алгоритм добавления котика |  |
| Алгоритм удаления котика |  |
| Алгоритм покупки позиции котиков |  |
| Алгоритм добавление позиции котика | Изображение выглядит как диаграмма  Автоматически созданное описание |

# Разработка программных модулей в соответствии с техническим заданием

Данный раздел описывает модули, которые я создал, среди которых присутствует база данных, API.

* 1. Проектирование базы данных

Диаграмма базы данных представлена на рисунке 4.1

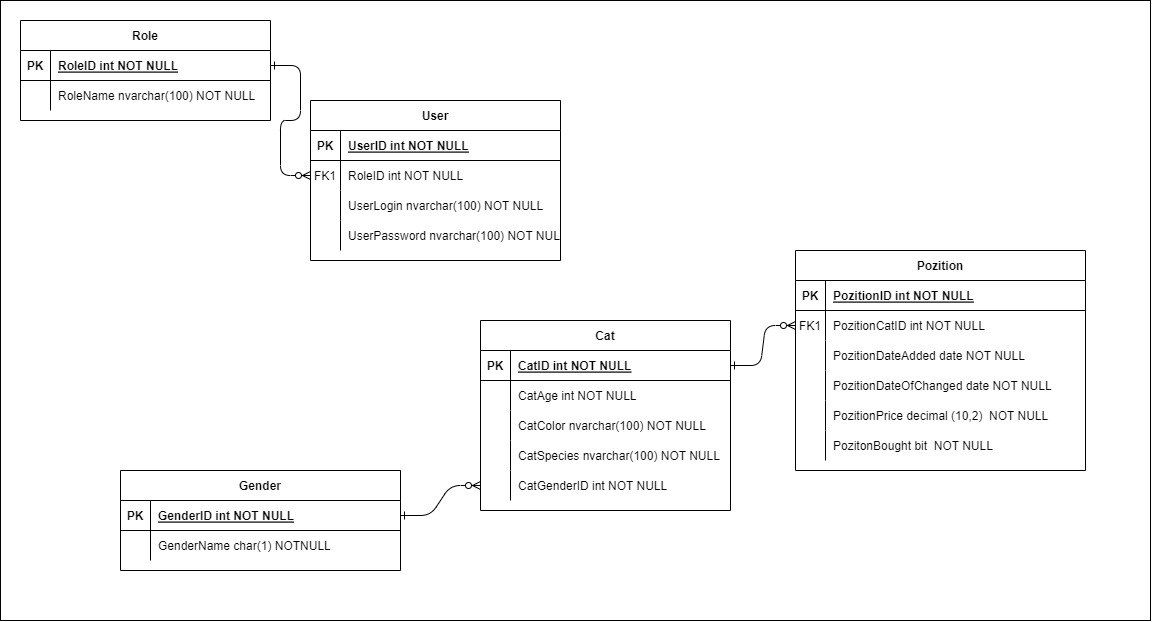


Рисунок 4.1 – Диаграмма базы данных

В данной диаграмме присутствуют таблицы, описание которых представлено в приложении 1.

* 1. Разработка базы данных

База данных была разработана на PostgreSQL 13.3. Диаграмма базы данных представлена на рисунке 4.2.



Рисунок 4.2 – Диаграмма созданной базы данных

В этих таблицах хранятся данные, над которыми будут производиться операции в приложениях, согласно реализованной логике. Для связи приложений с базой данных используется API. Таблицы базы данных представлены в приложении 1.

* 1. Разработка Web API

В данном подразделе описаны созданные мной API-функции. API разработано было в приложениях Visual Studio 2022, Visual Studio 2019 и Rider. Тип проекта – .NET ASP.NET Core Web Application / Web API. Язык программирования – C#. Версия dotnet – 7.0.

В API охвачены все таблицы базы данных.

Входные данные, которые «Объект» передаются в Json-формате (в теле запроса), в котором указаны параметры данного объекта. Остальные в строке URL-ссылке (если указано место данного параметра), или в теле запроса (в противном случае).

Выходные данные, которые «Объект» или «массив …» передаются в Json-формате, в котором указаны параметры объекта (в первом случае) или элемента массива (во втором случае, если это массив объектов), а остальные передаются, как значение. Также, для некоторых функций указано Headers для токена, что означает, необходимость авторизации для выполнения данной функции.

Для запросов используются Http-методы:

* Get – Получение информации;
* Post – Добавление информации;
* Put – Обновление информации;
* Patch – Частичное обновление информации;
* Delete – Удаление информации.

Это из стандарта REST API. Здесь, каждый метод соответствует методу в CRUD (Create, Read, Update, Delete):

* Get – Create;
* Post – Read;
* Put – Update;
* Patch – Update;
* Delete – Delete.
  + 1. API для Авторизации

Авторизация включает в себя ввод логина и пароля и получение токена для пользования другими функциями, как показано на рисунке 4.3.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4.3 – API для авторизации

Описание единственной функции представлено в приложении 2.1.1, а программный код – в приложении 2.2.1.

* + 1. API для работы с пользователями

Список функций представлен на рисунке 4.4.

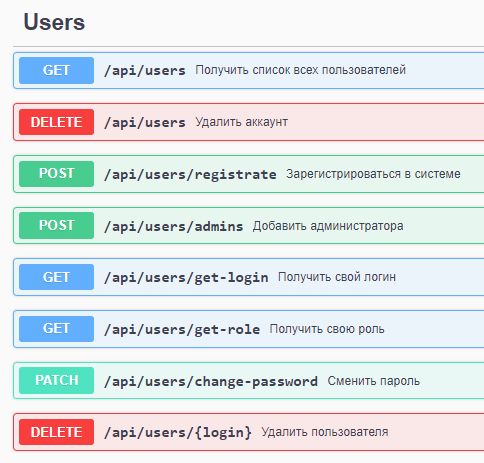


Рисунок 4.4 – API для работы с пользователями

Описание функций представлено в приложении 2.1.2, а программный код – в приложении 2.2.2.

* + 1. API для работы с полами котиков

Список функций представлен на рисунке 4.5.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4.5 – API для работы с полами котиков котиками

Описание функций представлено в приложении 2.1.3, а программный код – в приложении 2.2.3.

* + 1. API для работы с котиками

Список функций представлен на рисунке 4.6.

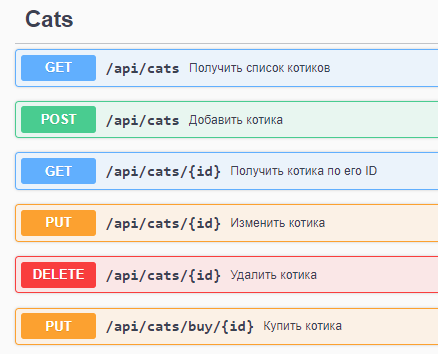


Рисунок 4.6 – API для работы с полами котиков котиками

Описание функций представлено в приложении 2.1.4, а программный код – в приложении 2.2.4.

# Выполнение отладки программных модулей с использованием специализированных программных средств

Отла́дка — этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки. Чтобы понять, где возникла ошибка, приходится: узнавать текущие значения переменных; выяснять, по какому пути выполнялась программа.

Сделаем отладку функции POST api/cats/ для добавления котика. Необходимые условия – пользователь авторизирован в системе, его роль – администратор, данные о котиках заполнены. При отладке предполагается, что все эти условия соблюдены.

Первый шаг представлен на рисунке 5.1.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5.1 – первый шаг

Как видно из рисунка, на этом шаге проверяется пол котика на его существование в базе данных (он хранится отдельной таблицей). В данном случае пол «Ж», и он существует (метод cat.GetGenderID() возвращает 1). Значит данный шаг выполняется успешно.

Второй шаг представлен на рисунке 5.2.

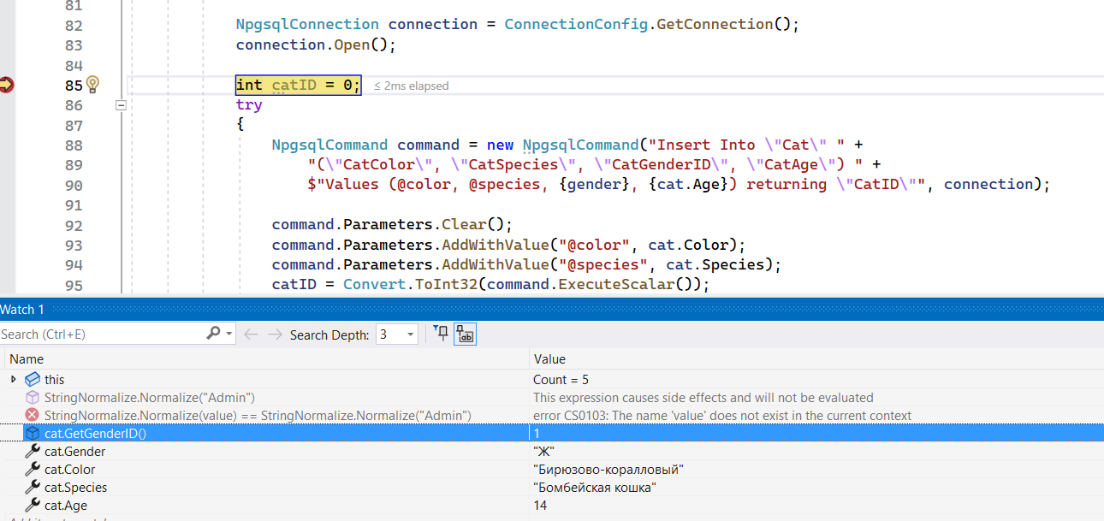


Рисунок 5.2 – Второй шаг

Как видно из рисунка, на этом шаге было открыто соединение с базой данных. Сдесь происходит первая транзакция в базе данных – Вставка записи в таблицу котиков с получением ID новой записи. Здесь в таблицу вводится цвет котика (в данном случае бирюзово-коралловый), порода (в данном случае бомбейская кошка), возраст (в данном случае 14) и ID пола (в данном случае 1 – ID пола «Ж»). То есть, данный шаг, тоже, проходит успешно.

Третий шаг представлен на рисунке 5.3.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5.3 – Третий шаг

На этом шаге выполняется ещё одно добавление, но теперь, уже в таблицу позиций котиков. Здесь берётся ID котика, которое вычислилось на предыдущем шаге, как ID добавленного котика (в данном случае 8). Также, в таблицу позиций добавляется цена (в данном случае 450). При этом даты добавления и изменения позиции котика вычисляется автоматически на уровне базы данных и равняется текущему времени выполнения отладки.

И последний шаг представлен на рисунке 5.4.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5.4 – Четвёртый шаг

То что выполнение программы перешло на return true – это означает, что выполнение программы было успешно, и котик был добавлен в базу данных полностью. Перед этим было закрыто соединение с базой данных и методом GetCatListFromBD() был выведен список котиков из базы данных. Как раз в окне контрольных значений показано, что был добавлен котик с ID=8 – метод GetCatFromID(8). То, есть вся подпрограмма была выполнена успешно.

# Выполнение тестирования программных модулей

В данном разделе описываются методы тестирования разработанных программных модулей.

Разработанное программное обеспечение является информационной системой, как и, практически, в любой другой информационной системе, присутствует серверная и клиентская части. Серверная часть представлена базой данной и API, служащем для взаимодействия клиентских приложений с базой данных.

* 1. Тестирование разработанного API с использованием Postman

Поскольку, в данной информационной системе присутствует API, логично протестировать его функции в Postman.

Postman — это платформа API, позволяющая разработчикам проектировать, создавать, тестировать и повторять свои API.

Тестируемые запросы в Postman представлены на рисунке 6.1.

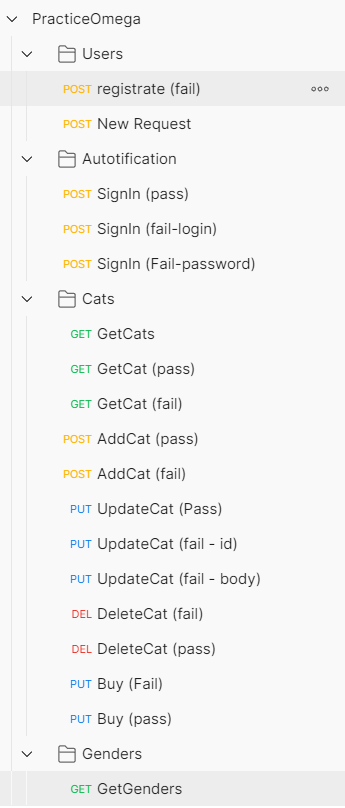


Рисунок 6.1 – Postman

Есть обозначения с фигурными скобками, которые я использую для сокращения. Эти обозначения ниже рассмотрены на примере «id»:

* {{id}}/cat – id является переменной, которая служит для сокращения написания URL-ссылки. Значения этих переменных указаны ниже;
* Cat/{id}/cat – id является параметром в строке. В Postman, вместо данного обозначения пишется значение параметра без фигурных скобок. Это значение указано в виде, как id=n, где n – значение параметра, пишущиеся, вместо id в фигурных скобках.

Переменные:

* Cat – https://localhost:44312//api;
* Users – {{cat}}/users;
* Autothication – {{cat}}/autotification;
* Cats – {{cat}}/cats;
* Genders – {{cat}}/cat-genders.

Тестовые методы были сделаны в Postman на языке JavaScript.

JavaScript — мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией спецификации ECMAScript. JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений.

Результаты выполнения тестовых методов:

* Pass – Удачное выполнение;
* Fail – Неудачное выполнение.

Был протестирован базовый путь для поиска ролей, добавления позиций котиков и для покупки этих позиций. Где требуется авторизация, считаем, что она была произведена, причём пользователя с нужной ролью. В данных тестовых примерах эти подробности не описываются, а только параметры запросов API и тестовых методов, написанных на JavaScript. Будут тестироваться методы с корректными данными и некорректными данными.

Статус-коды, которые могут быть в моём API:

* 200 – OK – Успех;
* 204 – No Content – Не найдено содержимое;
* 401 – Unautorized – Пользователь не авторизирован;
* 404 – Not Found – не найдено;
* 409 – Conflict – Конфликт.

Тестирование методов в Postman представлено в таблице 6.1.

Таблица 6.1 – Тестирование функций API в Postman

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Метод для передачи запроса** | **Запросы API с описанием** | **Входные данные с комментарием** | **Результат выполнения** | **Тестовые методы и их результат** | **Результат выполнения тестового метода** |
| POST | {{Autothification}}/sign-in – Войти в систему | {  "login": "anton",  "password": "password"  } – Верный логин и пароль | {  “authtoken”:”<Токен>”  } | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 401", function () {  pm.response.to.have.status(401);  }); | Fail |
| {  "login": "anton123",  "password": "password"  } – Несуществующий логин | Ошибка: данный логин не существует | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Fail |
| pm.test("Status code is 401", function () {  pm.response.to.have.status(401);  }); | Pass |
| {  "login": "anton",  "password": "123"  } – Неверный пароль | Ошибка: неверный пароль | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Fail |
| pm.test("Status code is 401", function () {  pm.response.to.have.status(401);  }); | Pass |
| POST | {{Users}}/registrate – Зарегистрироваться в системе | {  "login": "anton12345",  "password": "12345"  } – Несуществующий ещё логин | True | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 409", function () {  pm.response.to.have.status(409);  }); | Fail |
| {  "login": "anton",  "password": "password"  } – существующий логин | False | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Fail |
| pm.test("Status code is 409", function () {  pm.response.to.have.status(409);  }); | Pass |
| GET | {{Cats}} – Получить список котиков | Нет | [  {  "id": 1,  "dateUpdated": "2023-03-27T20:48:02.771323",  "dateAdded": "2023-03-27T20:48:02.771323",  "age": 12,  "color": "Красный",  "species": "Тойгер",  "gender": "ж",  "price": 120.00  },  {  "id": 2,  "dateUpdated": "2023-03-27T20:48:25.476708",  "dateAdded": "2023-03-27T20:48:25.476708",  "age": 12,  "color": "Розовый",  "species": "Тойгер",  "gender": "ж",  "price": 450.00  },  …  ] | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| GET | {{Cats}}/{id} – Получить котика по его ID | Id = 6 – существует | {  "id": 6,  "dateUpdated": "2023-03-29T10:03:50.047178",  "dateAdded": "2023-03-29T10:03:50.047178",  "age": 15,  "color": "Бирюзовый",  "species": "Американский кёрл",  "gender": "м",  "price": 900.00  } | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 204", function () {  pm.response.to.have.status(204);  }); | Fail |
| Id = 3 – не существует | Error 204 (No Content) | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Fail |
| pm.test("Status code is 204", function () {  pm.response.to.have.status(204);  }); | Pass |
| POST | {{Cats}} – Добавить котика | {  "age": 14,  "color": "red",  "species": "toyger",  "gender": "ж",  "price": 120  } – Пол существующий | True | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 409", function () {  pm.response.to.have.status(409);  }); | Fail |
| {  "age": 14,  "color": "red",  "species": "toyger",  "gender": "в",  "price": 120  } – Пол несуществующий | False | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Fail |
| pm.test("Status code is 409", function () {  pm.response.to.have.status(409);  }); | Pass |
| PUT | {{Cats}}/{id} – Обновить котика с его ID | Id = 6 – существует | true | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| {  "age": 14,  "color": "red",  "species": "toyger",  "gender": "ж",  "price": 120  } – Пол существующий | pm.test("Status code is 409", function () {  pm.response.to.have.status(409);  }); | Fail |
| Id = 3 – не существует | false | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Fail |
| {  "age": 14,  "color": "red",  "species": "toyger",  "gender": "ж",  "price": 120  } – Пол существующий | pm.test("Status code is 409", function () {  pm.response.to.have.status(409);  }); | Pass |
| Id = 6 – существует | false | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Fail |
| {  "age": 14,  "color": "red",  "species": "toyger",  "gender": "в",  "price": 120  } – Пол несуществующий | pm.test("Status code is 409", function () {  pm.response.to.have.status(409);  }); | Pass |
| DELETE | {{Cats}}/{id} – Удалить котика с его ID | ID = 10 – Существующий | True | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Fail |
| pm.test("Status code is 404", function () {  pm.response.to.have.status(404);  }); | Pass |
| ID = 3 – Несуществующий | False | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Fail |
| pm.test("Status code is 404", function () {  pm.response.to.have.status(404);  }); | Pass |
| PUT | {{Cats}}/Buy/{id} – Купить котика | ID = 11 – ID существующий | True | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Fail |
| pm.test("Status code is 404", function () {  pm.response.to.have.status(404);  }); | Pass |
| ID = 3 – Не существует | False | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Fail |
| pm.test("Status code is 404", function () {  pm.response.to.have.status(404);  }); | Pass |
| GET | {{Genders}} – Получить список полов | Нет | [  {  "id": 1,  "name": "ж"  },  {  "id": 2,  "name": "м"  }  ] | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Fail |

# Осуществление рефакторинга и оптимизации программного кода

В данном разделе описываются изменения программного кода, в результате которых, функционал API не изменяется, но увеличивается производительность API и читаемость кода, используя следующие методы:

* Рефа́кторинг (англ. refactoring), или перепроектирование кода, переработка кода, равносильное преобразование алгоритмов — процесс изменения внутренней структуры программы, не затрагивающий её внешнего поведения и имеющий целью облегчить понимание её работы. В основе рефакторинга лежит последовательность небольших эквивалентных (то есть сохраняющих поведение) преобразований;
* Оптимизация кода — различные методы преобразования кода ради улучшения его характеристик и повышения эффективности.

То есть, это – изменения программного кода, неизменяющие поведения и функциональности программы, но улучшающие её работу и упрощающих создание программного кода. Применение этих методов описано далее.

**Осуществление изменений программного кода**

1. Изменения программного кода работы с пользователями

Программный код контроллера Web API для работы с пользователями представлен в приложении 2.2.2. Ниже представлен его кусок, отвечающий за добавление пользователей:

/// <summary>

/// Зарегистрироваться в системе

/// </summary>

/// <returns></returns>

[HttpPost("registrate")]

[AllowAnonymous]

public ActionResult Registrate([FromBody]User user)

{

return UserList.CreateUsersFromDB().AddUser(user, 1) ? Ok(true) : Conflict(false);

}

/// <summary>

/// Добавить администратора

/// </summary>

/// <returns></returns>

[HttpPost("Admins")]

[Authorize(Roles = "Admin")]

public ActionResult AddAdmin([FromBody]User user)

{

try

{

string name = User.Identity.Name ?? "";

if (!UserList.CreateUsersFromDB().HaveLogin(name))

{

throw new Exception("Данный пользователь не существует в системе");

}

return UserList.CreateUsersFromDB().AddUser(user, 2) ? Ok(true) : Conflict(false);

}

catch (Exception ex)

{

return NotFound(ex.Message);

}

}

Есть одинаковые части программного кода, которые похожи – UserList.CreateUsersFromDB().AddUser. Отличия, только во втором параметре – ID роли. Изменим код следующим образом (вынесем строку в отдельный метод):

/// <summary>

/// Добавить пользователя в систему с ролью roleID

/// </summary>

/// <param name="user"></param>

/// <param name="roleID"></param>

/// <returns></returns>

private ActionResult AddUser(User user, int roleID)

{

return UserList.CreateUsersFromDB().AddUser(user, roleID) ? Ok(true) : Conflict(false);

}

/// <summary>

/// Зарегистрироваться в системе

/// </summary>

/// <returns></returns>

[HttpPost("registrate")]

[AllowAnonymous]

public ActionResult Registrate([FromBody]User user)

{

return AddUser(user, 1);

}

/// <summary>

/// Добавить администратора

/// </summary>

/// <returns></returns>

[HttpPost("Admins")]

[Authorize(Roles = "Admin")]

public ActionResult AddAdmin([FromBody]User user)

{

try

{

string name = User.Identity.Name ?? "";

if (!UserList.CreateUsersFromDB().HaveLogin(name))

{

throw new Exception("Данный пользователь не существует в системе");

}

return AddUser(user, 2);

}

catch (Exception ex)

{

return NotFound(ex.Message);

}

}

Также, везде есть строка UserList.CreateUsersFromDB(). В данном случае она нигде ничем не отличается. Тоже вынесем её в отдельный метод:

/// <summary>

/// Получить список всех пользователей

/// </summary>

/// <returns></returns>

[HttpGet]

[Authorize(Roles = "Admin")]

public ActionResult Get()

{

try

{

string name = User.Identity.Name ?? "";

if (!GetUsers().HaveLogin(name))

{

throw new Exception("Данный пользователь не существует в системе");

}

return Ok(GetUsers());

}

catch (Exception ex)

{

return NotFound(ex.Message);

}

}

/// <summary>

/// Получить список всех пользователей

/// </summary>

/// <returns></returns>

private UserList GetUsers() => UserList.CreateUsersFromDB();

/// <summary>

/// Добавить пользователя в систему с ролью roleID

/// </summary>

/// <param name="user"></param>

/// <param name="roleID"></param>

/// <returns></returns>

private ActionResult AddUser(User user, int roleID)

{

return GetUsers().AddUser(user, roleID) ? Ok(true) : Conflict(false);

}

/// <summary>

/// Зарегистрироваться в системе

/// </summary>

/// <returns></returns>

[HttpPost("registrate")]

[AllowAnonymous]

public ActionResult Registrate([FromBody]User user)

{

return AddUser(user, 1);

}

/// <summary>

/// Добавить администратора

/// </summary>

/// <returns></returns>

[HttpPost("Admins")]

[Authorize(Roles = "Admin")]

public ActionResult AddAdmin([FromBody]User user)

{

try

{

string name = User.Identity.Name ?? "";

if (!GetUsers().HaveLogin(name))

{

throw new Exception("Данный пользователь не существует в системе");

}

return AddUser(user, 2);

}

catch (Exception ex)

{

return NotFound(ex.Message);

}

}

И, наконец, обратим внимание на строки GetUsers().HaveLogin(name), которые тоже повторяются и нигде ничем не отличаются, и вынесем их в отдельный метод:

/// <summary>

/// Получить список всех пользователей

/// </summary>

/// <returns></returns>

[HttpGet]

[Authorize(Roles = "Admin")]

public ActionResult Get()

{

try

{

string name = User.Identity.Name ?? "";

if (!HaveLogin(name))

{

throw new Exception("Данный пользователь не существует в системе");

}

return Ok(GetUsers());

}

catch (Exception ex)

{

return NotFound(ex.Message);

}

}

/// <summary>

/// Получить список всех пользователей

/// </summary>

/// <returns></returns>

private UserList GetUsers() => UserList.CreateUsersFromDB();

/// <summary>

/// Существует ли пользователь с логином name в системе

/// </summary>

/// <param name="name"></param>

/// <returns></returns>

private bool HaveLogin(string name) => GetUsers().HaveLogin(name);

/// <summary>

/// Добавить пользователя в систему с ролью roleID

/// </summary>

/// <param name="user"></param>

/// <param name="roleID"></param>

/// <returns></returns>

private ActionResult AddUser(User user, int roleID)

{

return GetUsers().AddUser(user, roleID) ? Ok(true) : Conflict(false);

}

/// <summary>

/// Зарегистрироваться в системе

/// </summary>

/// <returns></returns>

[HttpPost("registrate")]

[AllowAnonymous]

public ActionResult Registrate([FromBody]User user)

{

return AddUser(user, 1);

}

/// <summary>

/// Добавить администратора

/// </summary>

/// <returns></returns>

[HttpPost("Admins")]

[Authorize(Roles = "Admin")]

public ActionResult AddAdmin([FromBody]User user)

{

try

{

string name = User.Identity.Name ?? "";

if (!HaveLogin(name))

{

throw new Exception("Данный пользователь не существует в системе");

}

return AddUser(user, 2);

}

catch (Exception ex)

{

return NotFound(ex.Message);

}

}

Полный вариант изменённого программного кода для работы с пользователями представлен в приложении 3.1. Там описываемые строки везде заменены на вызов методов, которые были описаны.

1. Изменения программного кода работы с котиками

Программный код данных функций представлен в приложении 2.2.4.

Рассмотрим кусок данного программного кода:

/// <summary>

/// Добавить котика

/// </summary>

/// <param name="cat"></param>

/// <returns></returns>

[HttpPost]

[Authorize(Roles = "Admin")]

public ActionResult<bool> Add([FromBody] Cat cat)

{

string name = User.Identity.Name ?? "";

if (!UserList.CreateUsersFromDB().HaveLogin(name))

{

return Unauthorized("Ваш логин больше не существует в системе");

}

return Get().CatAdd(cat) ? Ok(true) : Conflict(false);

}

/// <summary>

/// Изменить котика

/// </summary>

/// <param name="cat"></param>

/// <returns></returns>

[HttpPut("{id}")]

[Authorize(Roles = "Admin")]

public ActionResult<bool> Update(int id, [FromBody] Cat cat)

{

string name = User.Identity.Name ?? "";

if (!UserList.CreateUsersFromDB().HaveLogin(name))

{

return Unauthorized("Ваш логин больше не существует в cистеме");

}

return Get().UpdateCat(id, cat) ? Ok(true) : Conflict(false);

}

Здесь, как и в предыдущем примере есть повторяющаяся строка UserList.CreateUsersFromDB().HaveLogin(name). В данном случае она нигде ничем не отличается. Вынесем её в отдельный метод:

/// <summary>

/// Существует ли в системе пользователь с логином name в системе

/// </summary>

/// <param name="name"></param>

/// <returns></returns>

private bool HaveLogin(string name) => UserList.CreateUsersFromDB().HaveLogin(name);

/// <summary>

/// Добавить котика

/// </summary>

/// <param name="cat"></param>

/// <returns></returns>

[HttpPost]

[Authorize(Roles = "Admin")]

public ActionResult<bool> Add([FromBody] Cat cat)

{

string name = User.Identity.Name ?? "";

if (!HaveLogin(name))

{

return Unauthorized("Ваш логин больше не существует в системе");

}

return Get().CatAdd(cat) ? Ok(true) : Conflict(false);

}

/// <summary>

/// Изменить котика

/// </summary>

/// <param name="cat"></param>

/// <returns></returns>

[HttpPut("{id}")]

[Authorize(Roles = "Admin")]

public ActionResult<bool> Update(int id, [FromBody] Cat cat)

{

string name = User.Identity.Name ?? "";

if (!HaveLogin(name))

{

return Unauthorized("Ваш логин больше не существует в cистеме");

}

return Get().UpdateCat(id, cat) ? Ok(true) : Conflict(false);

}

Полный вариант изменённого программного кода представлен в приложении 3.2. Там описываемая строка везде заменена на вызов метода, который был описан.

# Разработка модулей программного обеспечения для мобильных платформ

В данном разделе описывается разработанное мобильное приложение под созданное Web API. Используемый языка программирования для разработки мобильного приложения – Java. Среда разработки – Android Studio. Эмулятор для запуска и тестирования приложения – Pixel 2 – API 28. Операционная система в эмуляторе – Android 9.

Все окна имеют разные компоненты, но есть те, которые присутствуют на каждом окне. Эти компоненты представлены на рисунке 8.1.

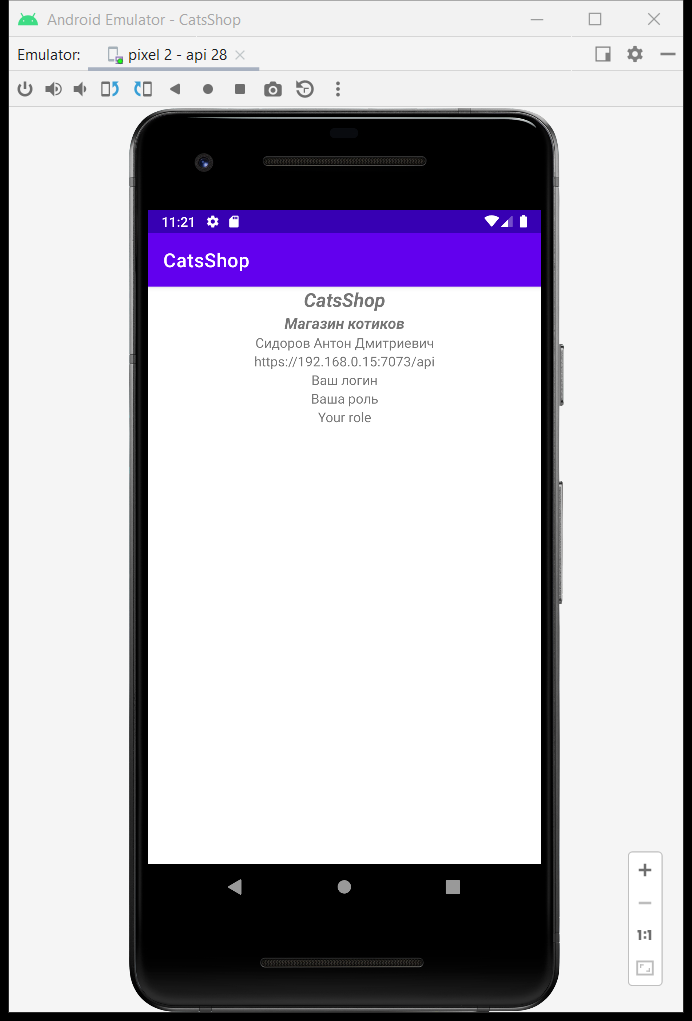


Рисунок 8.1 – Общий макет для всех остальных окон

Среди представленных компонентов присутствуют:

* Название приложения;
* Предметная область, под которую разработано приложение;
* ФИО разработчика приложения;
* URL-ссылка на используемое API;
* «Ваш логин» – Логин пользователя, вошедшего в систему;
* «Ваша роль» – Русскоязычное название роли этого пользователя в системе;
* «Your role» – Англоязычное название роли этого пользователя в системе.

Под пользователем, вошедшем в систему, понимается пользователя, который авторизировался, используя данное приложение, с устройства, на котором данное приложение установлено. При успешной авторизации, пользователю выдаётся токен, который приложение будет хранить в своём буфере и использовать, когда тот необходим.

Исходя из созданного API, пользоваться функциями данной системы может только авторизированный пользователь. Неавторизированный пользователь может, только, просматривать список котиков и их полов, а также выбирать элемент из этого списка. Также, неавторизированный пользователь может авторизироваться в системе (тогда, он станет авторизированным) или зарегистрироваться (тогда он получит логин и пароль для авторизации). Авторизация проходит успешно, только, при правильном вводе логина и пароля. Регистрация пользователя в системе проходит успешно, только, при вводе логина, несуществующего ещё в базе данных.

# Заключение

В ходе производственной практики были освоены следующие компетенции.

Были освоены общие компетенции:

1. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам;
2. Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности;
3. Планировать и реализовывать собственное профессиональное и личностное развитие;
4. Работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством, клиентами;
5. Осуществлять устную и письменную коммуникацию на государственном языке с учётом особенностей социального и культурного контекста;
6. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей;
7. Содействовать сохранению окружающей среды, ресурсосбережению, эффективно действовать в чрезвычайных ситуациях;
8. Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания уровня физической подготовленности;
9. Использовать информационные технологии в профессиональной деятельности;
10. Пользоваться профессиональной документацией на государственном и иностранном языке;
11. Планировать предпринимательскую деятельность в профессиональной сфере.

Также, были освоены профессиональные компетенции:

1. Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием;
2. Разрабатывать программные модули в соответствии с техническим заданием;
3. Выполнять отладку программных модулей с использованием специализированных программных средств;
4. Выполнять тестирование программных модулей;
5. Осуществлять рефакторинг и оптимизацию программного кода;
6. Разрабатывать модули программного обеспечения для мобильных платформ.

# Список использованных источников

# Приложение

1. **Описание таблиц базы данных**

|  |  |  |  |
| --- | --- | --- | --- |
| **Таблица и её назначение** | **Столбец и его назначение** | **Тип данных в столбце** | **Ограничение в столбце** |
| Role (Роли пользователей в системе) | RoleID (ID роли) | int | Primary key |
| RoleName (Название роли) | Nvarchar(100) | Not Null |
| User (Пользователи в системе) | UserID (ID пользователя) | int | Primary key |
| RoleID (ID роли у пользователя) | int | Not Null,  Foreign key (Role.RoleID) |
| UserLogin (Логин пользователя) | Nvarchar(100) | Not Null, Unique Key |
| UserPassword (Пароль пользователя) | Nvarchar(100) | Not Null |
| CatGender (Пол котика) | CatGenderID (ID пола) | Int | Primary Key |
| CatGenderName (Название пола) | char(1) | Not Null |
| Cat (котик) | CatID (ID котика) | int | Primary key |
| CatSpecies (порода) | Nvarchar(100) | Not Null,  Foreign Key (CatSpecies. CatSpeciesID) |
| CatColor (цвет) | Nvarchar(100) | Not Null |
| CatGenderID (ID пола) | Int | Not Null |
| CatAge (возраст котика) | Decimal(10, 2) | Not Null |
| Pozition (Позиция котика) | PozitionID (ID позиции) | Int | Primary Key |
| PozitionCatID | int | Not Null,  Foreign Key (Cat.CatID) |
| PozitionCost (стоимость котика) | Decimal(10, 2) | Not Null |
| PozitionDateAdded (Дата добавления котика) | Date | Not Null, Default (Now()) |
| PozitionDateOfChanged (Дата изменения котика) | Date | Not Null, Default (Now()) |
| BuyPozitionID – ID клиента | int | Not Null,  Foreign key (Pozition.PozitionID) |
| PozitionBought – Куплена ли позиция | Bit/Boolean | Not NULL |

1. **Web API**
2. **Функции**
3. **Авторизация**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Функция и её назначение** | **Входные данные с их типом** | **Параметры входных данных с их типом** | **Выходные данные с их типом** | **Параметры выходных данных или их элемента с их типом** |
| POST – api/autotification/sign-in | Пользователь (объект) | Login – Логин (текст) | Токен (объект) | Authtoken – токен для аутотификации (текст) |
| Password – Логин (текст) |

1. **Работа с пользователями**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Функция и её назначение** | **Входные данные с их типом** | **Параметры входных данных с их типом** | **Выходные данные с их типом** | **Параметры выходных данных или их элемента с их типом** |
| GET – api/users – Получить список всех пользователей (доступно, только, администратору) | Autorization – токен авторизации (строка в Headers) | | Список пользователей (массив) | Login – Логин (строка) |
| Password – Пароль (строка) |
| RoleRus – Роль на русском (строка) |
| RoleEng – Роль на английском (строка) |
| POST – Api/users/registrate – регистрация пользователя в системе | Пользователь (объект) | Login – Логин (текст) | Успешность (True) или неуспешность (false) | |
| Password – Логин (текст) |
| POST – Api/users/admins – добавление администратора (доступно, только администратору) | Autorization – токен авторизации (строка в Headers) | | Успешность (True) или неуспешность (false) | |
| Пользователь (объект) | Login – Логин (текст) |
| Password – Логин (текст) |
| GET – Api/users/get-logins – получить свой логин (доступно только авторизованному пользователю) | Autorization – токен авторизации (строка в Headers) | | Логин (строка) | |
| GET – api/users/get-role – получить свою роль (доступно только авторизованному пользователю) | Autorization – токен авторизации (строка в Headers) | | Роль (объект) | RoleRus – Роль на русском (строка) |
| RoleEng – Роль на английском (строка) |
| PATCH – api/users/change-password – смена пароля (доступно только авторизованному пользователю) | Autorization – токен авторизации (строка в Headers) | | Успешность (True) или неуспешность (false) | |
| Password – новый пароль (строка) | |
| DELETE – api/users/ – получить свой аккаунт (доступно только авторизованному пользователю) | Autorization – токен авторизации (строка в Headers) | | Успешность (True) или неуспешность (false) | |
| DELETE – api/users/{login} – получить аккаунт с введённым логином(доступно, только администратору) | Autorization – токен авторизации (строка в Headers) | | Успешность (True) или неуспешность (false) | |
| Login – логин пользователя | |

1. **Работа с полами котиков**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Метод, Функция и её назначение** | **Входные данные с их типом** | **Параметры входных данных с их типом** | **Выходные данные с их типом** | **Параметры выходных данных или их элемента с их типом** |
| Get – api/cats-genders – Получить список пород | Отсутствуют | | Пол (объект) | ID – ID пола (число) |
| Name – название пола (текст) |
| Get – api/cats-genders/{id} – Получить пол по его ID | ID – ID пола (число) | | Пол (объект) | ID – ID пола (число) |
| Name – название пола (текст) |
| Get – api/cats-genders/by-name/{name} | Name – название пола (текст) | | Пол (объект) | ID – ID пола (число) |
| Name – название пола (текст) |

1. **Работа с котиками и их позициями**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Функция и её назначение** | **Входные данные с их типом** | **Параметры входных данных с их типом** | **Выходные данные с их типом** | **Параметры выходных данных или их элемента с их типом** |
| Get – api/cats – список котиков | Отсутствуют | | Список котиков (массив) | ID – ID котика (число) |
| DateAdded – Дата добавления (дата и время) |
| DateUpdated – Дата изменения (дата и время) |
| Age – возраст (число) |
| Color – цвет (текст) |
| Species – Порода (текст) |
| Gender – Пол (текст) |
| Price – Цена (число) |
| Get – api/cats/{id} – получить котика по его ID | ID – ID котика (число) | | Котик (объект) | ID – ID котика (число) |
| DateAdded – Дата добавления (дата и время) |
| DateUpdated – Дата изменения (дата и время) |
| Age – возраст (число) |
| Color – цвет (текст) |
| Species – Порода (текст) |
| Gender – Пол (текст) |
| Price – Цена (число) |
| POST – api/cats – Добавление котика (доступно, только, администратору) | Autorization – токен авторизации (строка в Headers) | | Успешность (True) или неуспешность (false) | |
| Данные котика (объект) | Age – возраст (число) |
| Color – цвет (текст) |
| Species – Порода (текст) |
| Gender – Пол (текст) |
| Price – Цена (число) |
| PUT – api/cats/{id} – Изменение котика (доступно, только, администратору) | Autorization – токен авторизации (строка в Headers) | | Успешность (True) или неуспешность (false) | |
| ID – ID обновляемого котика (число) | |
| Новые данные котика (объект) | Age – возраст (число) |
| Color – цвет (текст) |
| Species – Порода (текст) |
| Gender – Пол (текст) |
| Price – Цена (число) |
| DELETE – api/cats/{id} – Изменение котика (доступно, только, администратору) | Autorization – токен авторизации (строка в Headers) | | Успешность (True) или неуспешность (false) | |
| ID – ID обновляемого котика (число) | |
| PUT – api/cats/{id} – Покупка котика (доступно, только, клиенту) | Autorization – токен авторизации (строка в Headers) | | Успешность (True) или неуспешность (false) | |
| ID – ID покупаемого котика (число) | |

1. **Программный код**
2. **Авторизация**

namespace CatsShop

{

/// <summary>

/// Контроллер для авторизации

/// </summary>

[Route("api/[controller]")]

public class AutotificationController : ControllerBase

{

private readonly IAppAuthService \_appAuthService;

public AutotificationController()

{

\_appAuthService = new AppAuthService();

}

/// <summary>

/// Авторизировать пользователя в системе

/// </summary>

/// <returns></returns>

[HttpPost("Sign-In")]

public ActionResult Token([FromBody] User user)

{

\_appAuthService.Users = UserList.CreateUsersFromDB();

try

{

Token? token = \_appAuthService.Authenticate(user);

if (token == null)

{

return Unauthorized();

}

return Ok(token);

}

catch (Exception ex)

{

return Unauthorized(ex.Message);

}

}

}

}

1. **Работа с пользователями**

/// <summary>

/// Контроллер для пользователя

/// </summary>

[Authorize]

[Route("api/[controller]")]

public class UsersController : ControllerBase

{

/// <summary>

/// Получить список всех пользователей

/// </summary>

/// <returns></returns>

[HttpGet]

[Authorize(Roles = "Admin")]

public ActionResult Get()

{

try

{

string name = User.Identity.Name ?? "";

if (!UserList.CreateUsersFromDB().HaveLogin(name))

{

throw new Exception("Данный пользователь не существует в системе");

}

return Ok(UserList.CreateUsersFromDB());

}

catch (Exception ex)

{

return NotFound(ex.Message);

}

}

/// <summary>

/// Зарегистрироваться в системе

/// </summary>

/// <returns></returns>

[HttpPost("registrate")]

[AllowAnonymous]

public ActionResult Registrate([FromBody]User user)

{

return UserList.CreateUsersFromDB().AddUser(user, 1) ? Ok(true) : Conflict(false);

}

/// <summary>

/// Добавить администратора

/// </summary>

/// <returns></returns>

[HttpPost("Admins")]

[Authorize(Roles = "Admin")]

public ActionResult AddAdmin([FromBody]User user)

{

try

{

string name = User.Identity.Name ?? "";

if (!UserList.CreateUsersFromDB().HaveLogin(name))

{

throw new Exception("Данный пользователь не существует в системе");

}

return UserList.CreateUsersFromDB().AddUser(user, 2) ? Ok(true) : Conflict(false);

}

catch (Exception ex)

{

return NotFound(ex.Message);

}

}

/// <summary>

/// Получить свой логин

/// </summary>

/// <returns></returns>

[HttpGet("get-login")]

public IActionResult GetLogin()

{

string name = User.Identity.Name??"";

if (UserList.CreateUsersFromDB().HaveLogin(name))

return Ok(name);

else

return NotFound();

}

/// <summary>

/// Получить свою роль

/// </summary>

/// <returns></returns>

[HttpGet("get-role")]

public IActionResult GetRole()

{

string name = User.Identity.Name??"";

if (UserList.CreateUsersFromDB().HaveLogin(name))

{

Claim? claim = ((ClaimsIdentity)User.Identity).FindFirst(claim => claim.Type == ClaimsIdentity.DefaultRoleClaimType);

Role role = new Role();

if (claim != null)

{

role.RoleEng = claim.Value;

}

var response = new

{

roleEng = role.RoleEng,

roleRus = role.RoleRus

};

return Ok(response);

}

else

{

return NotFound();

}

}

/// <summary>

/// Сменить пароль

/// </summary>

/// <returns></returns>

[HttpPatch("change-password")]

public ActionResult ChangePassword([FromBody] string password)

{

return UserList.CreateUsersFromDB().ChangePassword(User.Identity.Name??"", password) ? Ok(true) : Conflict(false);

}

/// <summary>

/// Удалить пользователя

/// </summary>

/// <returns></returns>

[HttpDelete("{login}")]

[Authorize(Roles = "Admin")]

public ActionResult DropUser(string login)

{

string name = User.Identity.Name ?? "";

if (!UserList.CreateUsersFromDB().HaveLogin(name))

{

return Unauthorized("Ваш логин больше не существует в системе");

}

return UserList.CreateUsersFromDB().DeleteUser(login) ? Ok(true) : NotFound(false);

}

/// <summary>

/// Удалить аккаунт

/// </summary>

/// <returns></returns>

[HttpDelete()]

public ActionResult DropUser()

{

return UserList.CreateUsersFromDB().DeleteUser(User.Identity.Name ?? "") ? Ok(true) : NotFound(false);

}

}

1. **Работа с полами котиков**

/// <summary>

/// Функции для показа полов котиков

/// </summary>

[Route("api/[controller]")]

public class CatsGendersController : ControllerBase

{

/// <summary>

/// Получить список всех полов

/// </summary>

/// <returns></returns>

[HttpGet]

public GendersList Get()

{

return GendersList.CreateGendersListFromDB();

}

/// <summary>

/// Получить пол по его ID

/// </summary>

/// <returns></returns>

[HttpGet("{id}")]

public Gender Get(int id)

{

return Get().GetGender(id);

}

/// <summary>

/// Получить пол по его названию

/// </summary>

/// <returns></returns>

[HttpGet("By-Name/{name}")]

public Gender Get(string name)

{

return Get().GetGender(name);

}

}

1. **Работа с котиками и их позициями**

/// <summary>

/// Список функций для котиков

/// </summary>

[Route("api/[controller]")]

public class CatsController : ControllerBase

{

/// <summary>

/// Получить список котиков

/// </summary>

/// <returns></returns>

[HttpGet]

public CatsList Get()

{

return CatsList.CreateCatsListFromDB();

}

/// <summary>

/// Получить котика по его ID

/// </summary>

/// <param name="id"></param>

[HttpGet("{id}")]

public Cat? Get(int id) => Get().GetCatFromID(id);

/// <summary>

/// Добавить котика

/// </summary>

/// <param name="cat"></param>

/// <returns></returns>

[HttpPost]

[Authorize(Roles = "Admin")]

public ActionResult<bool> Add([FromBody] Cat cat)

{

string name = User.Identity.Name ?? "";

if (!UserList.CreateUsersFromDB().HaveLogin(name))

{

return Unauthorized("Ваш логин больше не существует в системе");

}

return Get().CatAdd(cat) ? Ok(true) : Conflict(false);

}

/// <summary>

/// Изменить котика

/// </summary>

/// <param name="cat"></param>

/// <returns></returns>

[HttpPut("{id}")]

[Authorize(Roles = "Admin")]

public ActionResult<bool> Update(int id, [FromBody] Cat cat)

{

string name = User.Identity.Name ?? "";

if (!UserList.CreateUsersFromDB().HaveLogin(name))

{

return Unauthorized("Ваш логин больше не существует в системе");

}

return Get().UpdateCat(id, cat) ? Ok(true) : Conflict(false);

}

/// <summary>

/// Удалить котика

/// </summary>

/// <returns></returns>

[HttpDelete("{id}")]

[Authorize(Roles = "Admin")]

public ActionResult<bool> Delete(int id)

{

string name = User.Identity.Name ?? "";

if (!UserList.CreateUsersFromDB().HaveLogin(name))

{

return Unauthorized("Ваш логин больше не существует в системе");

}

return Get().DeleteCat(id) ? Ok(true) : NotFound(false);

}

/// <summary>

/// Купить котика

/// </summary>

/// <returns></returns>

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

[Authorize(Roles = "Client")]

public ActionResult<bool> Buy(int id)

{

string name = User.Identity.Name ?? "";

if (!UserList.CreateUsersFromDB().HaveLogin(name))

{

return Unauthorized("Ваш логин больше не существует с системе");

}

return Get().BuyPozition(id) ? Ok(true) : NotFound(false);

}

}

1. **Рефакторинг и оптимизация программного кода**
2. **Изменение программного кода для работы с пользователями**

/// <summary>

/// Контроллер для пользователя

/// </summary>

[Authorize]

[Route("api/[controller]")]

public class UsersController : ControllerBase

{

/// <summary>

/// Получить список всех пользователей

/// </summary>

/// <returns></returns>

[HttpGet]

[Authorize(Roles = "Admin")]

public ActionResult Get()

{

try

{

string name = User.Identity.Name ?? "";

if (!HaveLogin(name))

{

throw new Exception("Данный пользователь не существует в системе");

}

return Ok(GetUsers());

}

catch (Exception ex)

{

return NotFound(ex.Message);

}

}

/// <summary>

/// Получить список всех пользователей

/// </summary>

/// <returns></returns>

private UserList GetUsers() => UserList.CreateUsersFromDB();

/// <summary>

/// Существует ли пользователь с логином name в системе

/// </summary>

/// <param name="name"></param>

/// <returns></returns>

private bool HaveLogin(string name) => GetUsers().HaveLogin(name);

/// <summary>

/// Добавить пользователя в систему с ролью roleID

/// </summary>

/// <param name="user"></param>

/// <param name="roleID"></param>

/// <returns></returns>

private ActionResult AddUser(User user, int roleID)

{

return GetUsers().AddUser(user, roleID) ? Ok(true) : Conflict(false);

}

/// <summary>

/// Зарегистрироваться в системе

/// </summary>

/// <returns></returns>

[HttpPost("registrate")]

[AllowAnonymous]

public ActionResult Registrate([FromBody]User user)

{

return AddUser(user, 1);

}

/// <summary>

/// Добавить администратора

/// </summary>

/// <returns></returns>

[HttpPost("Admins")]

[Authorize(Roles = "Admin")]

public ActionResult AddAdmin([FromBody]User user)

{

try

{

string name = User.Identity.Name ?? "";

if (!HaveLogin(name))

{

throw new Exception("Данный пользователь не существует в системе");

}

return AddUser(user, 2);

}

catch (Exception ex)

{

return NotFound(ex.Message);

}

}

/// <summary>

/// Получить свой логин

/// </summary>

/// <returns></returns>

[HttpGet("get-login")]

public IActionResult GetLogin()

{

string name = User.Identity.Name??"";

if (HaveLogin(name))

return Ok(name);

else

return NotFound();

}

/// <summary>

/// Получить свою роль

/// </summary>

/// <returns></returns>

[HttpGet("get-role")]

public IActionResult GetRole()

{

string name = User.Identity.Name??"";

if (HaveLogin(name))

{

Claim? claim = ((ClaimsIdentity)User.Identity).FindFirst(claim => claim.Type == ClaimsIdentity.DefaultRoleClaimType);

Role role = new Role();

if (claim != null)

{

role.RoleEng = claim.Value;

}

var response = new

{

roleEng = role.RoleEng,

roleRus = role.RoleRus

};

return Ok(response);

}

else

{

return NotFound();

}

}

/// <summary>

/// Сменить пароль

/// </summary>

/// <returns></returns>

[HttpPatch("change-password")]

public ActionResult ChangePassword([FromBody] string password)

{

return GetUsers().ChangePassword(User.Identity.Name??"", password) ? Ok(true) : Conflict(false);

}

/// <summary>

/// Удалить пользователя

/// </summary>

/// <returns></returns>

[HttpDelete("{login}")]

[Authorize(Roles = "Admin")]

public ActionResult DropUser(string login)

{

string name = User.Identity.Name ?? "";

if (HaveLogin(name))

{

return Unauthorized("Ваш логин больше не существует с cистеме");

}

return GetUsers().DeleteUser(login) ? Ok(true) : NotFound(false);

}

/// <summary>

/// Удалить аккаунт

/// </summary>

/// <returns></returns>

[HttpDelete()]

public ActionResult DropUser()

{

return GetUsers().DeleteUser(User.Identity.Name ?? "") ? Ok(true) : NotFound(false);

}

}

1. **Изменение программного кода для работы с котиками**

/// <summary>

/// Список функций для котиков

/// </summary>

[Route("api/[controller]")]

public class CatsController : ControllerBase

{

/// <summary>

/// Получить список котиков

/// </summary>

/// <returns></returns>

[HttpGet]

public CatsList Get()

{

return CatsList.CreateCatsListFromDB();

}

/// <summary>

/// Получить котика по его ID

/// </summary>

/// <param name="id"></param>

[HttpGet("{id}")]

public Cat? Get(int id) => Get().GetCatFromID(id);

/// <summary>

/// Существует ли в системе пользователь с логином name в системе

/// </summary>

/// <param name="name"></param>

/// <returns></returns>

private bool HaveLogin(string name) => UserList.CreateUsersFromDB().HaveLogin(name);

/// <summary>

/// Добавить котика

/// </summary>

/// <param name="cat"></param>

/// <returns></returns>

[HttpPost]

[Authorize(Roles = "Admin")]

public ActionResult<bool> Add([FromBody] Cat cat)

{

string name = User.Identity.Name ?? "";

if (!HaveLogin(name))

{

return Unauthorized("Ваш логин больше не существует в системе");

}

return Get().CatAdd(cat) ? Ok(true) : Conflict(false);

}

/// <summary>

/// Изменить котика

/// </summary>

/// <param name="cat"></param>

/// <returns></returns>

[HttpPut("{id}")]

[Authorize(Roles = "Admin")]

public ActionResult<bool> Update(int id, [FromBody] Cat cat)

{

string name = User.Identity.Name ?? "";

if (!HaveLogin(name))

{

return Unauthorized("Ваш логин больше не существует в cистеме");

}

return Get().UpdateCat(id, cat) ? Ok(true) : Conflict(false);

}

/// <summary>

/// Удалить котика

/// </summary>

/// <returns></returns>

[HttpDelete("{id}")]

[Authorize(Roles = "Admin")]

public ActionResult<bool> Delete(int id)

{

string name = User.Identity.Name ?? "";

if (!HaveLogin(name))

{

return Unauthorized("Ваш логин больше не существует в cистеме");

}

return Get().DeleteCat(id) ? Ok(true) : NotFound(false);

}

/// <summary>

/// Купить котика

/// </summary>

/// <returns></returns>

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

[Authorize(Roles = "Client")]

public ActionResult<bool> Buy(int id)

{

string name = User.Identity.Name ?? "";

if (!HaveLogin(name))

{

return Unauthorized("Ваш логин больше не существует в системе");

}

return Get().BuyPozition(id) ? Ok(true) : NotFound(false);

}

}