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](#_Toc131003647)

[1. Предметная область 4](#_Toc131003648)

[2. Техническое задание 5](#_Toc131003649)

[3. Формирование алгоритмов разработки программных модулей в соответствии с техническим заданием 6](#_Toc131003650)

[4. Разработка программных модулей в соответствии с техническим заданием 9](#_Toc131003651)

[5. Выполнение отладки программных модулей с использованием специализированных программных средств 14](#_Toc131003652)

[6. Выполнение тестирования программных модулей 15](#_Toc131003653)

[7. Осуществление рефакторинга и оптимизации программного кода 23](#_Toc131003654)

[8. Разработка модулей программного обеспечения для мобильных платформ 24](#_Toc131003655)

[Заключение 25](#_Toc131003656)

[Приложение 26](#_Toc131003657)

# Введение

# Предметная область

На производственной практике в моей организации 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>.

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

# Техническое задание

# 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

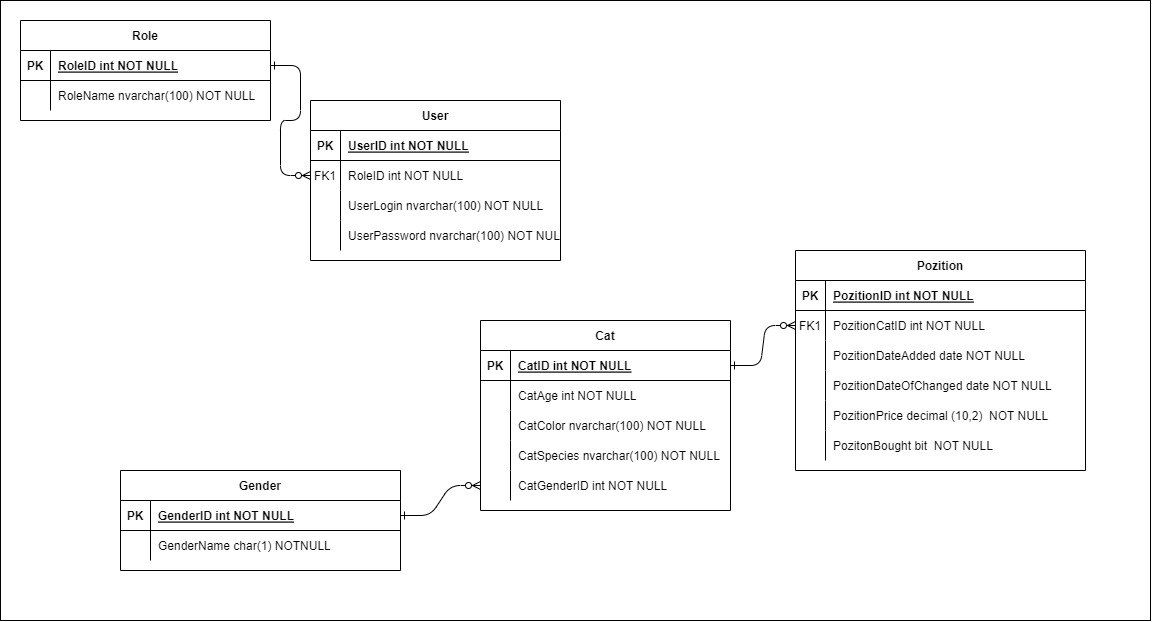


Рисунок 4.1 – Диаграмма базы данных

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.

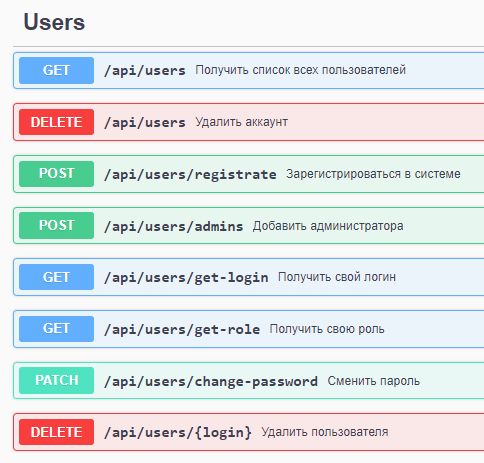


Рисунок 4.4 – API для работы с пользователями

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.

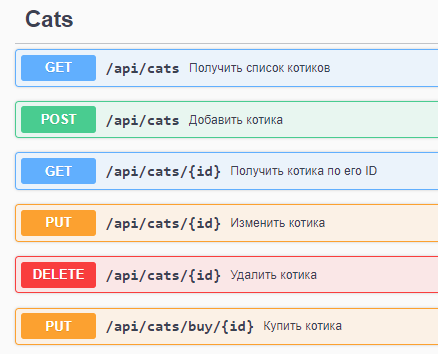


Рисунок 4.6 – API для работы с полами котиков котиками

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

# 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.

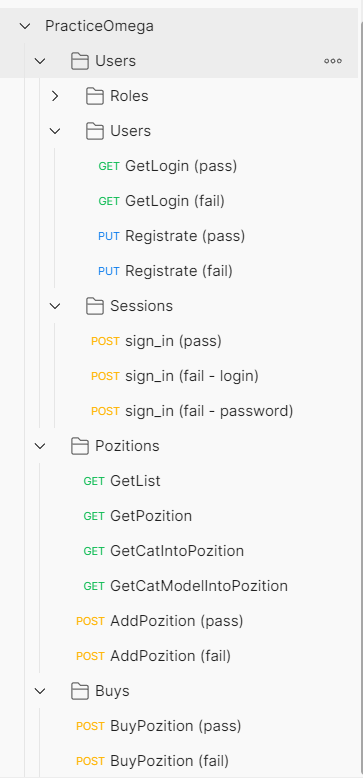


Рисунок 6.1 – Postman

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:44302/cats/api>;
* Roles – {{cat}}/users/Roles;
* Users – {{cat}}/users/Accounts;
* Sessions – {{cat}}/users/Sessions;
* Pozitions – {{cat}}/positions/Pozitions;
* Buy – {{cat}}/Buys.

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.

В моём API в основном присутствуют 2 кода ошибок: 200 (Успешное выполнение) и 500 (Провал).

Был протестирован базовый путь для поиска ролей, добавления позиций котиков и для покупки этих позиций. 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** |
| Get | {{Roles}}/RolesList – Получить список ролей | Отсутствуют | [  {  "id": 1,  "name": "Клиент"  },  {  "id": 2,  "name": "Администратор"  }  ] | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| Get | {{Roles}}/{id}/RoleName – Получить название роли по её ID | Id = 3  //Роль с данным id не существует | Error 500 | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Fail |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Pass |
| Id = 2  //Роль с данным id существует | Клиент | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| Put | {{Roles}}/RoleID – Получить ID роли по её названию | {  "role": "123"  }  //Роль с данным названием не существует | Error 500 | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Fail |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Pass |
| {  "role": "администратор"  }  //Роль с данным названием существует | 2 | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| Get | {Roles}/SessionRole – Получить роль пользователя по его ключу сессии | Session = 99788682342588528420  //Данная сессия существует | {  "id": 2,  "name": "Администратор"  } | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| Session = 12345678912345678900  Данная сессия не существует | Error 500 | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Fail |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Pass |
| Get | {{Users}}/LoginFromSession – Получить логин пользователя по его ключу сессии | Session = 99788682342588528420  // Данный ключ сессии существует | user | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| Session = 01234567890123456789  // Данный ключ сессии не существует | null | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| Put | {{Users}}/Registrate – Зарегистрироваться в системе | {  "login": "12345",  "password": "12345"  }  // Логин ещё не существует | True | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| {  "login" : "anton",  "password" : "password"  }  // Логин уже существует | false | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| Post | {{Sessions}}/SignIn | {  "login" : "anton",  "password" : "123"  }  // Правильный логин и пароль | 71074358591389817763 | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| {  "login" : "asdvafvadfv",  "password" : "123"  }  // Несуществующий логин | null | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| {  "login" : "anton",  "password" : "password"  }  // Неверный пароль | null | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| Get | {{Pozitions}}/List – Получить список позиций | Отсутствуют | [  {  "dateAdded": "2023-03-17T00:00:00",  "dateOfChanged": "2023-03-17T00:00:00",  "id": 2,  "cost": 150.00,  "catID": 3  },  {  "dateAdded": "2023-03-17T00:00:00",  "dateOfChanged": "2023-03-17T00:00:00",  "id": 3,  "cost": 180.40,  "catID": 3  },  …  } | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| Get | {{Pozitions}}/{id}/Get – получить позицию по её ID | Id = 3 | {  "dateAdded": "2023-03-17T00:00:00",  "dateOfChanged": "2023-03-17T00:00:00",  "id": 3,  "cost": 180.40,  "catID": 3  } | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| Get | {{Pozitions}}/{id}/Cat – получить котика в позиции позицию по её ID | Id = 3 | {  "id": 3,  "age": 15,  "modelID": 2  } | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| Get | {{Pozitions}}/{id}/CatModel – получить котика в позиции позицию по её ID | Id = 3 | {  "color": "Красный",  "gender": "ж",  "species": "Американский кёрл",  "id": 2,  "colorID": 2,  "genderID": 10,  "speciesID": 1  } | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| Post | {{Pozition}}/Add – Добавление позиции | Session=99788682342588528420  {  “catID” : 3,  “cost” : 400  }  //Пользователь с данной сессией – администратор | true | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| Session=72634502140285742755  {  “catID” : 3,  “cost” : 400  }  //Пользователь с данной сессией – клиент | false | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| Post | {{Buy}}/BuyPozition/{id} | Id = 3  Session=72634502140285742755  //Пользователь с данной сессией – клиент | true | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | Fail |
| Id = 3  Session=99788682342588528420  //Пользователь с данной сессией – администратор | false | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); | Pass |
| pm.test("Status code is 500", function () {  pm.response.to.have.status(500);  }); | 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.

Применение этих методов описано далее.

# Разработка модулей программного обеспечения для мобильных платформ

# Заключение

# 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**