

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
1. ПРЕДМЕТНАЯ ОБЛАСТЬ	4
2. ТЕХНИЧЕСКОЕ ЗАДАНИЕ	5
3. ФОРМИРОВАНИЕ АЛГОРИТМОВ РАЗРАБОТКИ ПРОГРАММНЫХ МОДУЛЕЙ В СООТВЕТСТВИИ С ТЕХНИЧЕСКИМ ЗАДАНИЕМ.....	6
4. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ В СООТВЕТСТВИИ С ТЕХНИЧЕСКИМ ЗАДАНИЕМ.....	9
5. ВЫПОЛНЕНИЕ ОТЛАДКИ ПРОГРАММНЫХ МОДУЛЕЙ С ИСПОЛЬЗОВАНИЕМ СПЕЦИАЛИЗИРОВАННЫХ ПРОГРАММНЫХ СРЕДСТВ.....	12
6. ВЫПОЛНЕНИЕ ТЕСТИРОВАНИЯ ПРОГРАММНЫХ МОДУЛЕЙ.....	13
7. ОСУЩЕСТВЛЕНИЕ РЕФАКТОРИНГА И ОПТИМИЗАЦИИ ПРОГРАММНОГО КОДА.....	21
8. РАЗРАБОТКА МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ МОБИЛЬНЫХ ПЛАТФОРМ	22
ЗАКЛЮЧЕНИЕ.....	23
ПРИЛОЖЕНИЕ	24

ВВЕДЕНИЕ

1. 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.

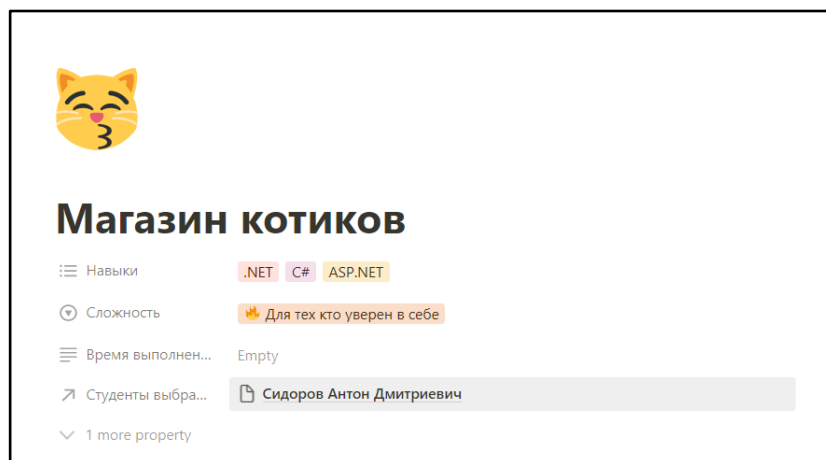


Рисунок 1.1 – Выбранная предметная область

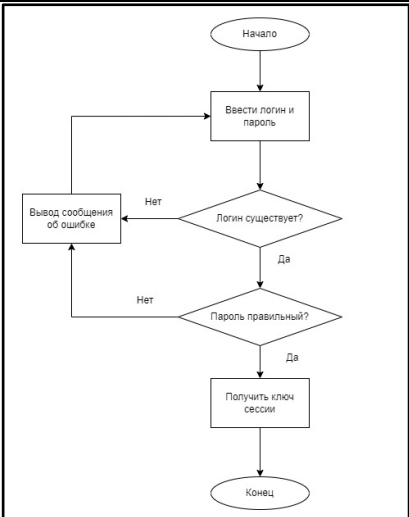
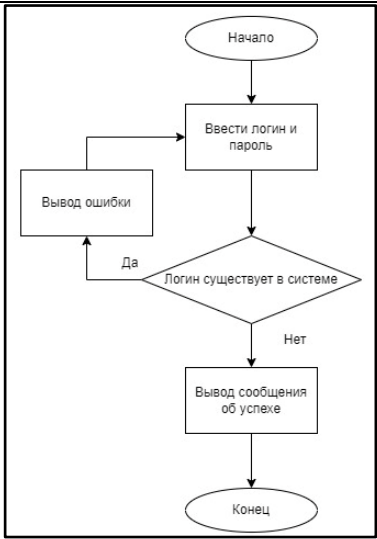
Выполненная работа находится по адресу
<https://github.com/AntonSidorov1/InterShipOooOmega>.

2. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

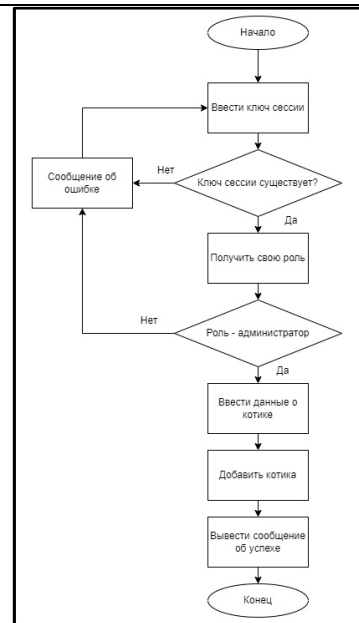
3. ФОРМИРОВАНИЕ АЛГОРИТМОВ РАЗРАБОТКИ ПРОГРАММНЫХ МОДУЛЕЙ В СООТВЕТСТВИИ С ТЕХНИЧЕСКИМ ЗАДАНИЕМ

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

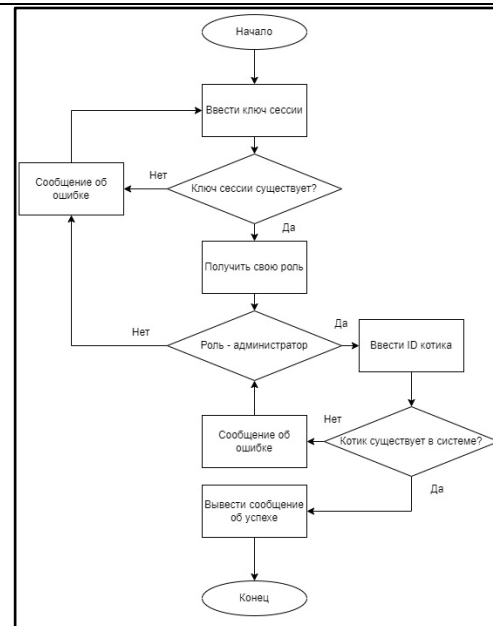
Таблица 1.1 – Разработанные алгоритмы.

Название алгоритма	Изображение алгоритма
Алгоритм авторизации	 <pre> graph TD Start([Начало]) --> Input[Ввести логин и пароль] Input --> L1{Логин существует?} L1 -- Нет --> Error1[Вывод сообщения об ошибке] Error1 --> Input L1 -- Да --> L2{Пароль правильный?} L2 -- Нет --> Error1 L2 -- Да --> Session[Получить ключ сессии] Session --> End([Конец]) </pre>
Алгоритм регистрации	 <pre> graph TD Start([Начало]) --> Input[Ввести логин и пароль] Input --> L1{Логин существует в системе} L1 -- Да --> Error2[Вывод ошибки] Error2 --> Input L1 -- Нет --> Success[Вывод сообщения об успехе] Success --> End([Конец]) </pre>

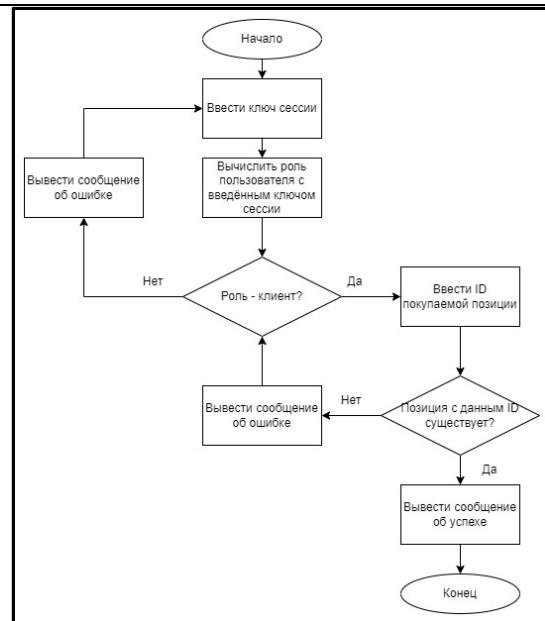
Алгоритм добавления котика



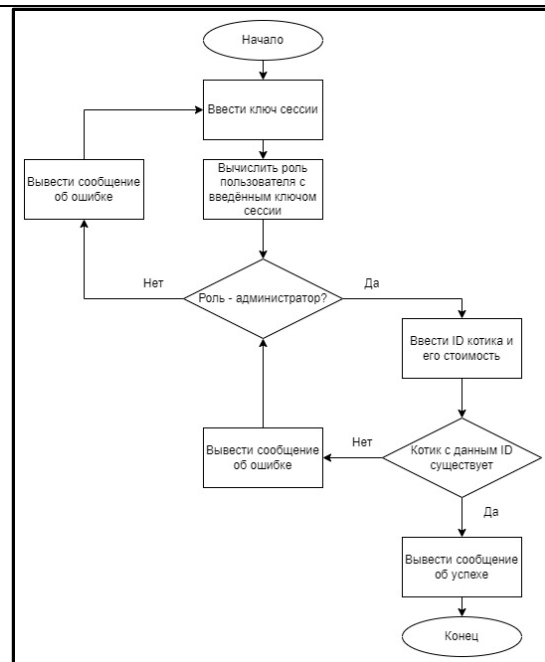
Алгоритм удаления котика



Алгоритм покупки позиции котиков



Алгоритм добавление позиции котика



4. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ В СООТВЕТСТВИИ С ТЕХНИЧЕСКИМ ЗАДАНИЕМ

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

4.1. Проектирование базы данных

Диаграмма базы данных представлена на рисунке 4.1

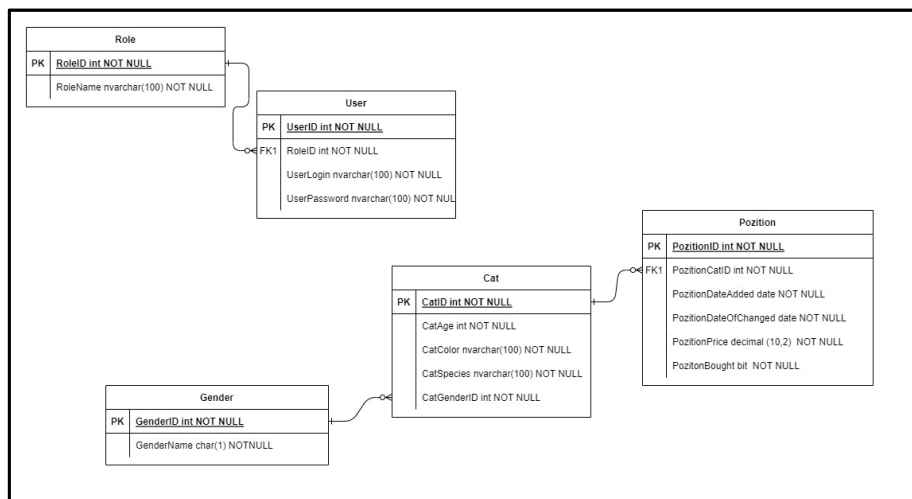


Рисунок 4.1 – Диаграмма базы данных

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

4.2. Разработка базы данных

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

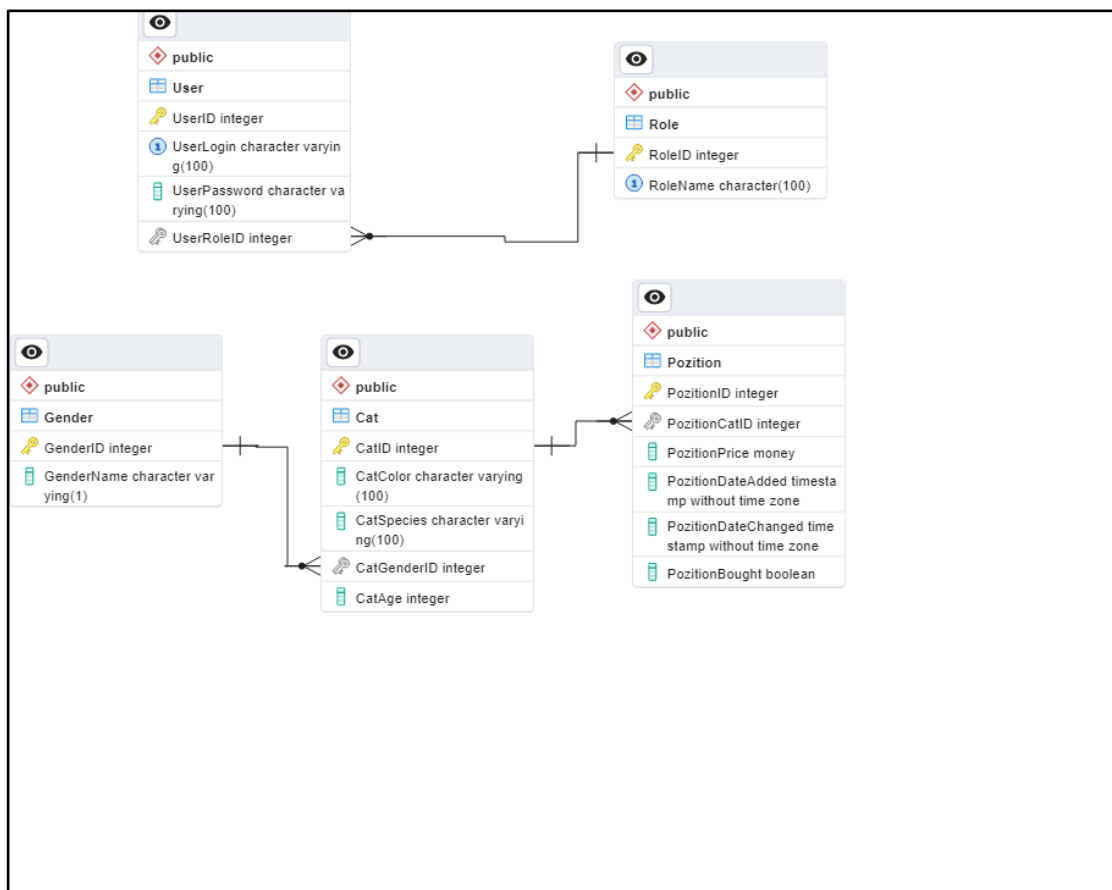


Рисунок 4.2 – Диаграмма созданной базы данных

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.

4.3. Разработка 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 котором указаны параметры данного объекта. Остальные в строке URL-

ссылке (если указано место данного параметра), или в конце ссылки, после знака «?» (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.

Для запросов используются Http-методы:

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

4.3.1. API для строки подключения к базе данных

5. ВЫПОЛНЕНИЕ ОТЛАДКИ ПРОГРАММНЫХ МОДУЛЕЙ С ИСПОЛЬЗОВАНИЕМ СПЕЦИАЛИЗИРОВАННЫХ ПРОГРАММНЫХ СРЕДСТВ

6. 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.

6.1. 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.

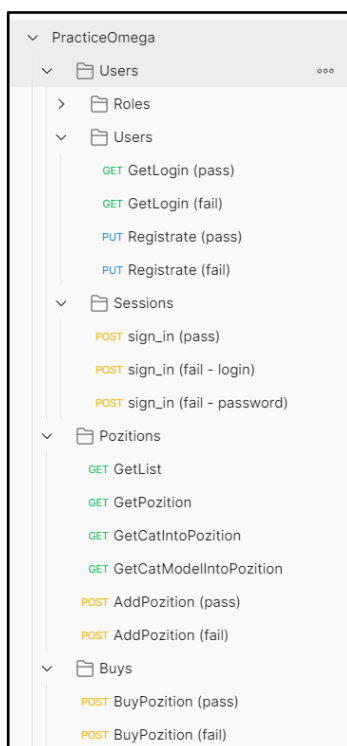


Рисунок 6.1 – Postman

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}/Session Role – 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" }	True	pm.test("Status code is 200", function () { pm.response.to.have .status(200); });	Pass
		{ "login": "anton", "password": "12345" }	True	pm.test("Status code is 500", function () { pm.response.to.have .status(500); });	Fail
		{ "login": "anton", "password": "12345" }	false	pm.test("Status code is 200", function () { pm.response.to.have .status(200); });	Pass
		{ "login": "anton", "password": "12345" }	false	pm.test("Status code is 500", function () { pm.response.to.have .status(500); });	Fail
Post	{{Sessions}}/SignIn	{ "login": "anton", "password": "123" }	71074358591389817763	pm.test("Status code is 200", function () { pm.response.to.have .status(200); });	Pass
		{ "login": "anton", "password": "123" }	71074358591389817763	pm.test("Status code is 500", function () { pm.response.to.have .status(500); });	Fail
		{ "login": "asdvafvadf", "password": "123" }	null	pm.test("Status code is 200", function () { pm.response.to.have .status(200); });	Pass
		{ "login": "asdvafvadf", "password": "123" }	null	pm.test("Status code is 500", function () { pm.response.to.have .status(500); });	Fail
		{ "login": "anton", "password": "123" }	71074358591389817763	pm.test("Status code is 200", function () { pm.response.to.have .status(200); });	Pass
		{ "login": "anton", "password": "123" }	71074358591389817763	pm.test("Status code is 500", function () { pm.response.to.have .status(500); });	Fail

		<pre>" : "password " } // Неверный пароль</pre>		<pre>pm.test("Status code is 500", function () { pm.response.to.have .status(500); });</pre>	Fail
Get	{ { Pozitions } } / List – Получить список позиций	Отсутствуют	<pre>[{ "dateAdded": "2023-03- 17T00:00:00" , "dateOfChan ged": "2023- 03- 17T00:00:00" , "id": 2, "cost": 150.00, "catID": 3 }, { "dateAdded": "2023-03- 17T00:00:00" , "dateOfChan ged": "2023- 03- 17T00:00:00" , "id": 3, "cost": 180.40, "catID": 3 }, ...]</pre>	<pre>pm.test("Status code is 200", function () { pm.response.to.have .status(200); });</pre>	Pass
				<pre>pm.test("Status code is 500", function () { pm.response.to.have .status(500); });</pre>	Fail
Get	{ { Pozitions } } / { id } / Get – получить позицию по её ID	Id = 3	<pre>{ "dateAdded": "2023-03- 17T00:00:00" , "dateOfChan ged": "2023- 03- 17T00:00:00" , "id": 3, </pre>	<pre>pm.test("Status code is 200", function () { pm.response.to.have .status(200); });</pre>	Pass
				<pre>pm.test("Status code is 500", function () { pm.response.to.have .status(500); });</pre>	Fail

			"cost": 180.40, "catID": 3 }		
Get	{ {Pozitions} }/{id}/Cat – получить кота 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 кота 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=9 97886823 42588528 420 { "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=7 26345021 40285742 755 { "catID": 3,	false	pm.test("Status code is 200", function () { pm.response.to.have .status(200); });	Pass
				pm.test("Status code is 500", function () {	Fail

		<code>"cost" : 400 } //Пользователь с данной сессией – клиент</code>		<code>pm.response.to.have .status(500); });</code>	
Post	<code>{{ Buy }}/BuyPosition/{id}</code>	<code>Id = 3 Session=7 26345021 40285742 755 //Пользователь с данной сессией – клиент</code>	true	<code>pm.test("Status code is 200", function () { pm.response.to.have .status(200); });</code>	Pass
				<code>pm.test("Status code is 500", function () { pm.response.to.have .status(500); });</code>	Fail
		<code>Id = 3 Session=9 97886823 42588528 420 //Пользователь с данной сессией – администратор</code>	false	<code>pm.test("Status code is 200", function () { pm.response.to.have .status(200); });</code>	Pass
				<code>pm.test("Status code is 500", function () { pm.response.to.have .status(500); });</code>	Fail

7. 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:

– Рефáкторинг (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.

8. РАЗРАБОТКА МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ МОБИЛЬНЫХ ПЛАТФОРМ

ЗАКЛЮЧЕНИЕ

ПРИЛОЖЕНИЕ

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

Приложение 2. API