## Аннотация

В пояснительной записке описывается Web API для управления личными финансами, реализованного на базе ASP.NET Core. Приложение предоставляет пользователям возможность учета расходов, установки бюджетов и анализа финансовых привычек. В записке представлены архитектура системы, описание основных компонентов API, а также примеры использования.

**Содержание**

[Аннотация 2](#_Toc1)

[Введение 4](#_Toc2)

[**1 Назначение и область применения** 5](#_Toc3)

[**2 Постановка задачи** 6](#_Toc4)

[**3 Описание программы** 8](#_Toc5)

[**3.1 Общие сведения** 8](#_Toc6)

[**3.2 Функциональное назначение** 9](#_Toc7)

[**3.3 Описание логической структуры** 12](#_Toc8)

[**3.4 Используемые технические средства** 20](#_Toc9)

[**3.5 Вызов и загрузка** 20](#_Toc10)

[**3.6 Входные и выходные данные**  20](#_Toc11)

[**4 Программа и методика испытаний** 21](#_Toc12)

[**4.1 Объекты испытаний** 21](#_Toc13)

[**4.2 Цель испытаний** 21](#_Toc14)

[**4.3 Требования к программе** 21](#_Toc15)

[**4.4 Методы испытаний** 21](#_Toc16)

[**4.5 Тестовый пример** 21](#_Toc17)

[**5 Руководство оператора** 23](#_Toc18)

[**5.1 Выполнение программы** 23](#_Toc19)

[**5.2 Сообщение оператору** 23](#_Toc20)

[**Заключение** 24](#_Toc21)

[**Приложение А** 25](#_Toc22)

[Текст программы 25](#_Toc23)

[**Источники, использованные при разработке** 27](#_Toc24)

[1. ГОСТ Р ИСО\_МЭК 25051-2017 Требования к качеству готового к использованию программного продукта (RUSP) и инструкции по тестированию. 27](#_Toc25)

[**Internet – ресурсы** 27](#_Toc26)

## Введение

Web API, предназначено для предоставления пользователям возможности отслеживать свои расходы, устанавливать финансовые цели и получать аналитические данные о своих финансовых привычках. Разработка Web API осуществлялась на основании:

1. Курсовое задание;
2. Техническое задание (при наличии).

**1 Назначение и область применения**

Назначением разработанного Web API является предоставление программного интерфейса для приложений, позволяющих пользователям эффективно управлять своими личными финансами. Web API предоставляет функциональность для учета расходов, планирования бюджетов и анализа финансовых привычек. Область применения Web API – мобильные и веб-приложения для персонального финансового менеджмента.

Существующие аналоги Web API:

* CoinKeeper (https://about.coinkeeper.me/) – Мобильное приложение для отслеживания трат и аналитики;
* Cashew (https://cashewapp.web.app/) – Мобильное приложение для отслеживания трат и аналитики.

**2 Постановка задачи**

Необходимо разработать Web API для системы управления личными финансами, обеспечивающего пользователям возможность учета расходов, планирования бюджетов и анализа финансового состояния. Разработанный API должен соответствовать следующим требованиям:

* Предоставление функциональности CRUD (Create, Read, Update, Delete) для именованных денежных хранилищ;
* Предоставление функциональности CRUD для именованных категорий расходов;
* Реализация функциональности добавления, получения и удаления транзакций;
* Обеспечение авторизации и контроля доступа, гарантирующего привязку данных к пользователю и исключающего доступ других пользователей;
* Реализация регистрации пользователей с валидацией данных (ФИО, email, пароль, соответствующий требованиям безопасности: длина, спецсимволы, заглавные буквы).

API должен обеспечивать надежность работы посредством:

* Корректной обработки ошибок при запросе несуществующих данных;
* Предотвращения добавления дубликатов данных;
* Валидации входных данных (формат, диапазоны, логическая корректность) и отклонения запросов при обнаружении ошибок.

Входные и выходные данные представлены в формате JSON с кодировкой UTF-8.

Применяемые методы: При разработке API будут использоваться стандартные методы проектирования RESTful API, включая использование HTTP-методов (GET, POST, PUT, DELETE) для выполнения операций с ресурсами.

**3 Описание программы**

**3.1 Общие сведения**

Данный подраздел посвящен представлению общих сведений о разработанном Web API приложении, предназначенном для управления личными финансами. Здесь будет представлена информация об идентификации программного продукта, используемых технологиях и требованиях к программному окружению для его корректного функционирования.

Разработанное Web API приложение имеет условное обозначение “SmartWallet”. Полное наименование программы: “Web API для мониторинга и управления личными финансами”. Данное наименование отражает основную функциональность и предназначение программного продукта.

Для корректной работы Web API “SmartWallet” требуется наличие следующего программного обеспечения:

* Операционная система: Web API может быть развернут на любой операционной системе, поддерживающей платформу .NET Core. Рекомендуется использование серверных операционных систем, таких как Windows Server или Linux (например, Ubuntu, CentOS), для обеспечения стабильной и надежной работы;
* .NET Runtime: Для запуска Web API требуется установленная среда выполнения .NET Core Runtime версии 8.0.0. Данная среда обеспечивает выполнение скомпилированного кода Web API;
* Веб-сервер: Web API требует наличия веб-сервера для обработки HTTP-запросов. Рекомендуется использование Kestrel в качестве веб-сервера. Kestrel - это высокопроизводительный веб-сервер, который может быть использован как самостоятельно, так и в связке с обратным прокси-сервером, таким как Nginx или Apache;
* СУБД (Система управления базами данных): Для хранения данных о пользователях, транзакциях, категориях и других сущностях используется реляционная СУБД. В качестве СУБД используется PostgreSQL. Необходима установленная и настроенная СУБД с соответствующими правами доступа для Web API.

Web API “SmartWallet” разработан с использованием следующих языков программирования и технологий:

* C#: Основной язык программирования, используемый для написания логики Web API, контроллеров, моделей данных и других компонентов. C# является объектно-ориентированным языком программирования, разработанным Microsoft, и широко используется для создания веб-приложений на платформе .NET;
* .NET Core: Web API разработан на базе .NET Core, платформы для разработки приложений от Microsoft. .NET Core предоставляет широкий набор библиотек и инструментов для создания масштабируемых и надежных веб-приложений.

**3.2 Функциональное назначение**

Основное назначение Web API “SmartWallet” заключается в предоставлении централизованного и унифицированного интерфейса для доступа к данным о финансовых операциях пользователей, что позволяет разработчикам создавать удобные и функциональные приложения для личного финансового планирования и анализа.

Более конкретно, Web API решает следующие классы задач:

* Учет доходов и расходов: Web API предоставляет функциональность для добавления, редактирования и удаления информации о доходах и расходах пользователей. Каждая транзакция может быть связана с определенной категорией (например, “продукты”, “транспорт”, “развлечения”), что позволяет пользователям отслеживать структуру своих расходов;
* Анализ финансовых данных: Web API предоставляет функциональность для анализа данных о расходах пользователей. Это позволяет пользователям выявлять тренды, оптимизировать свои расходы и принимать более обоснованные финансовые решения;
* Управление денежными хранилищами: Web API предоставляет функциональность для управления информацией о денежных хранилищах пользователей (например, “наличные”, “банковский счет”, “кредитная карта”). Пользователи могут добавлять, редактировать и удалять информацию о своих хранилищах, а также отслеживать остаток средств в каждом хранилище;
* Авторизация и аутентификация пользователей: Web API обеспечивает безопасный доступ к данным пользователей посредством реализации механизмов авторизации и аутентификации. Это гарантирует, что только авторизованные пользователи имеют доступ к своим финансовым данным.

Несмотря на широкий спектр решаемых задач, Web API “SmartWallet” имеет определенные функциональные ограничения на применение:

* Отсутствие интеграции с банковскими сервисами: Web API не предоставляет прямой интеграции с банковскими сервисами для автоматического получения информации о транзакциях. Пользователи должны вводить информацию о своих доходах и расходах вручную или импортировать данные из других источников;
* Ограниченные возможности анализа данных: Web API предоставляет базовые возможности анализа данных, такие как просмотр категоризованных трат за месяц. Однако, он не предоставляет продвинутых функций анализа данных, таких как прогнозирование финансовых показателей или выявление аномальных транзакций;
* Отсутствие поддержки мультивалютности: Web API не поддерживает работу с несколькими валютами. Все финансовые операции должны быть представлены в одной валюте;
* Зависимость от стабильного интернет-соединения: Для работы Web API требуется стабильное интернет-соединение. В случае отсутствия интернет-соединения, пользователи не смогут получить доступ к своим финансовым данным или выполнить какие-либо операции;
* Ограничения по масштабируемости: Web API может иметь ограничения по масштабируемости при большом количестве пользователей или объеме данных. В этом случае может потребоваться дополнительная оптимизация и масштабирование инфраструктуры.

**3.3 Описание логической структуры**

Ниже представлена схема базы данных приложения.

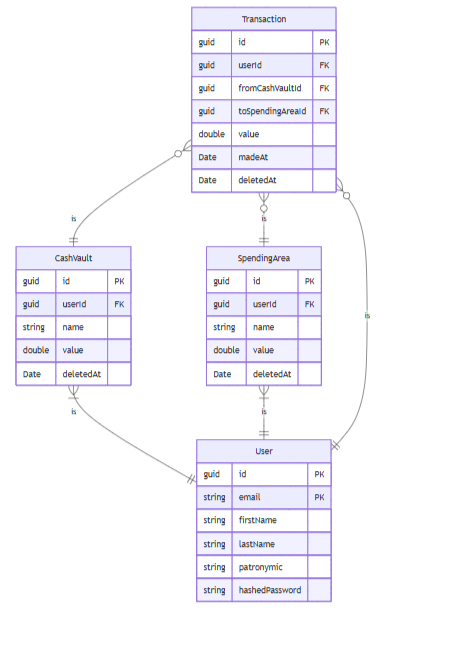


Рисунок 1 - Диаграмма базы данных

Ниже представлены диаграммы последовательности для разных конечных точек Web API.

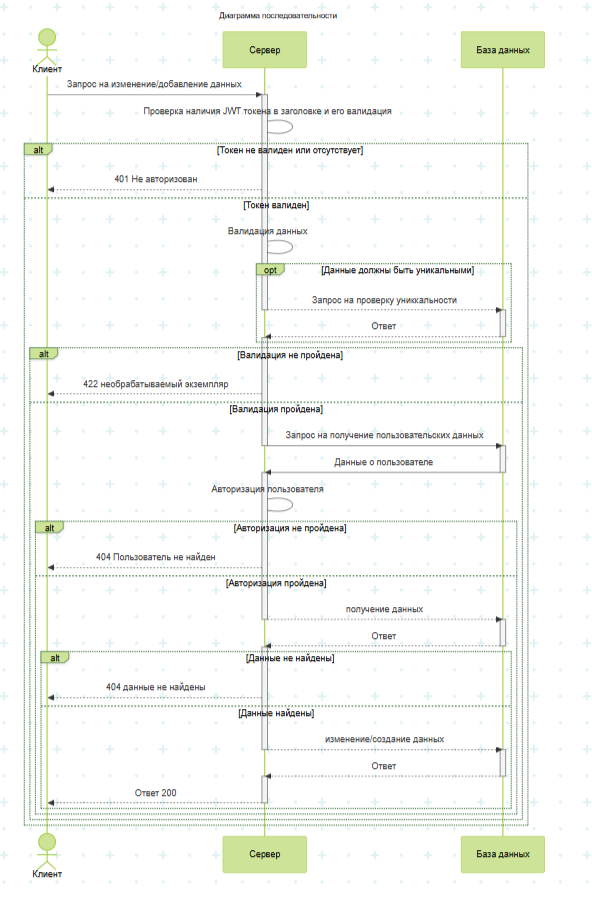


Рисунок 2 - Взаимодействия клиента, сервера и базы данных при запросе на добавление/обновление данных

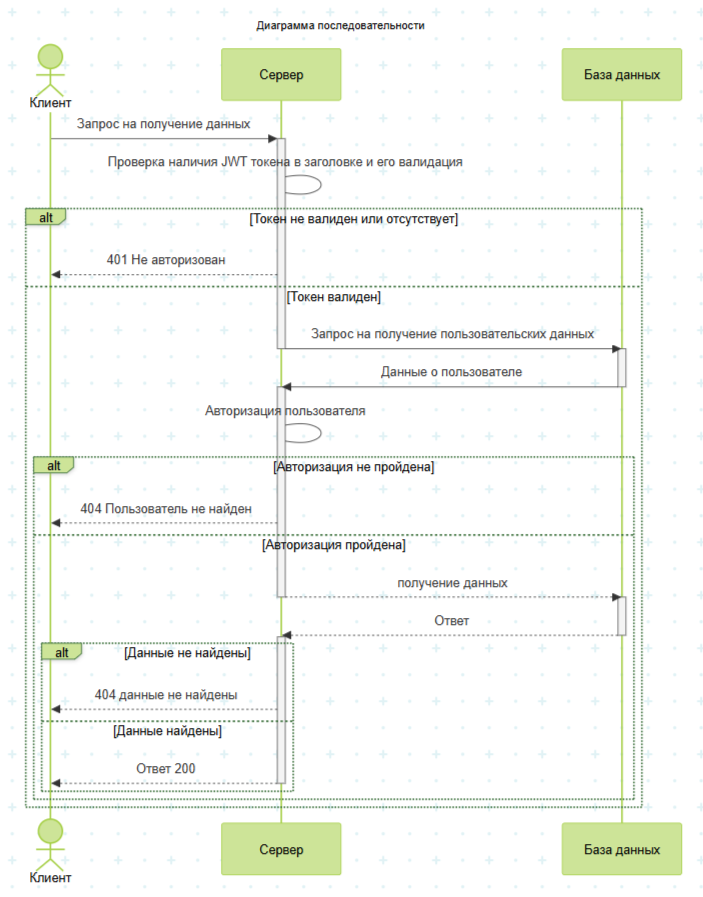


Рисунок 3 - Взаимодействия клиента, сервера и базы данных при запросе данных

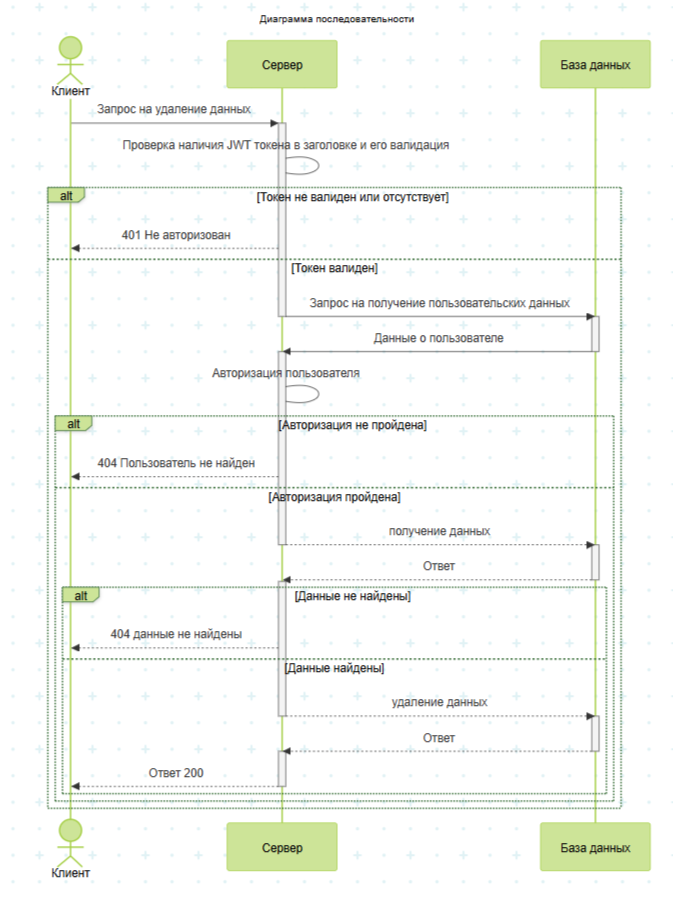


Рисунок 4 - Взаимодействия клиента, сервера и базы данных при запросе на удаление данных

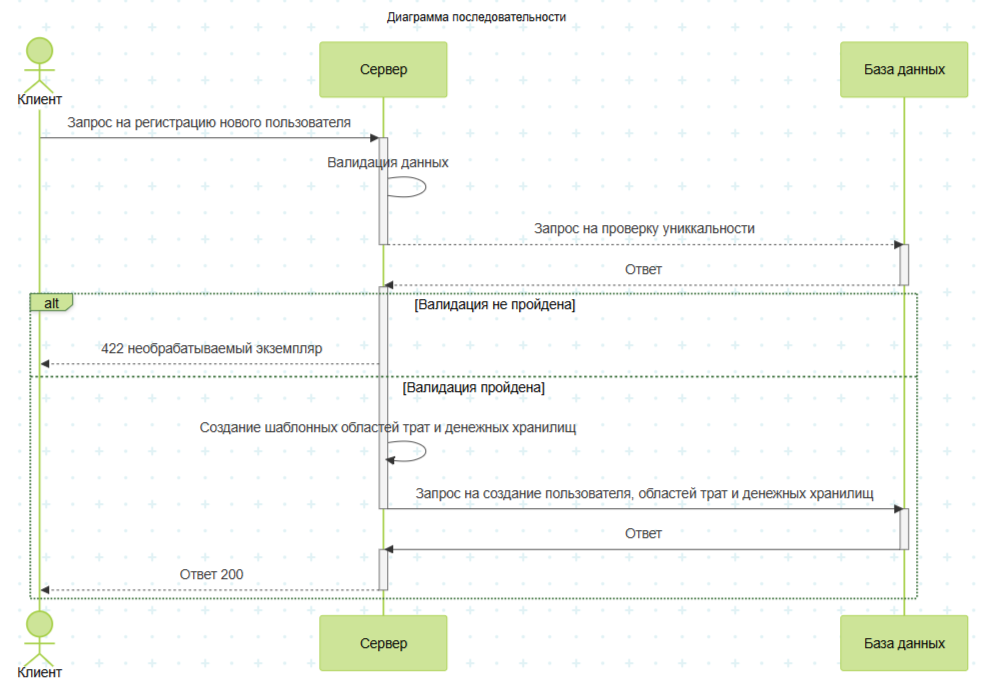


Рисунок 5 - Взаимодействия клиента, сервера и базы данных при запросе на регистрацию пользователя

Ниже представлены некоторые части диаграммы классов всего решения Web API.

В приложении используются паттерны проектирования «Репозиторий» и «Единица работы», а также принцип инверсии зависимостей.

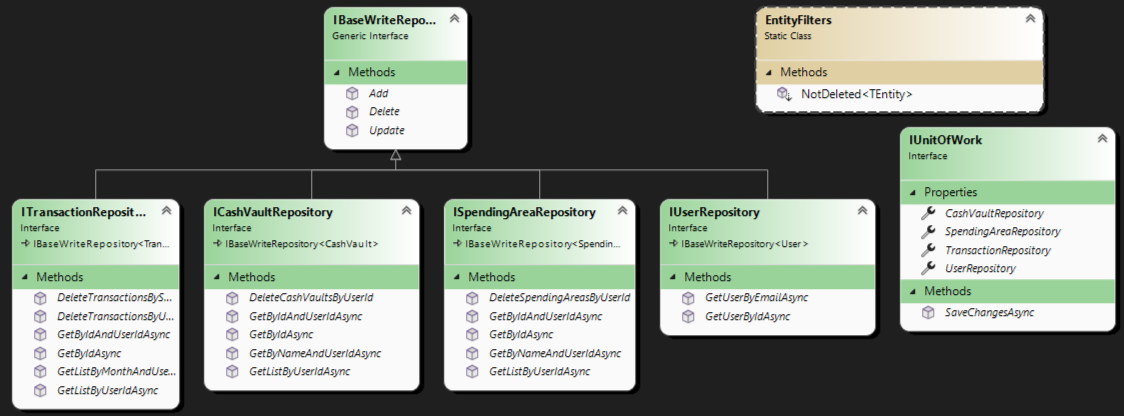


Рисунок 6 - Диаграмма класов проекта "Context.Repository.Contracts"

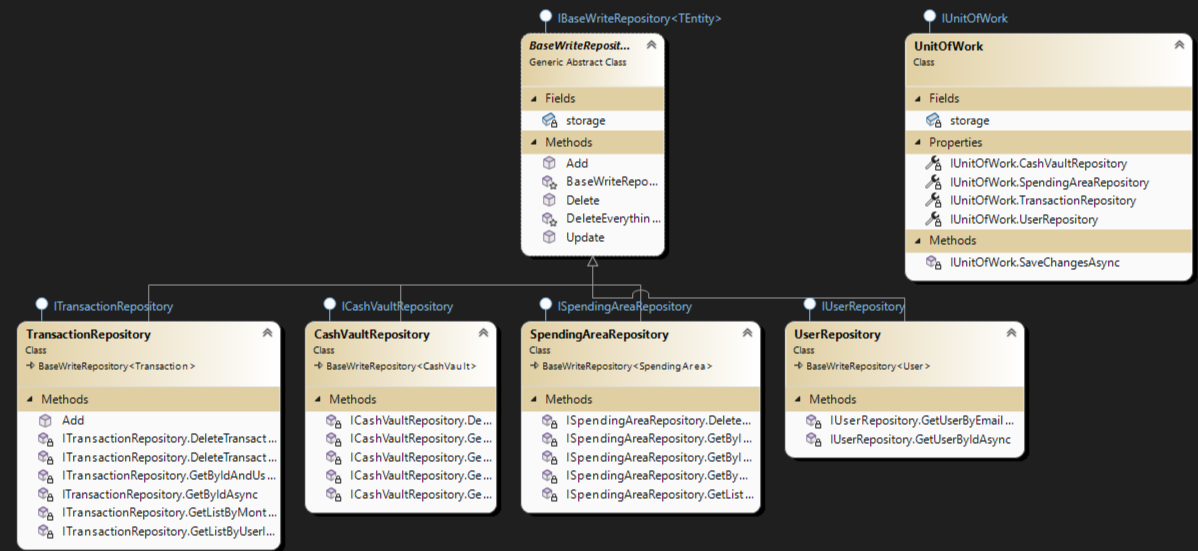


Рисунок 7 - Диаграмма класов проекта "Context.Repository"

Ниже представлена часть структура проекта.

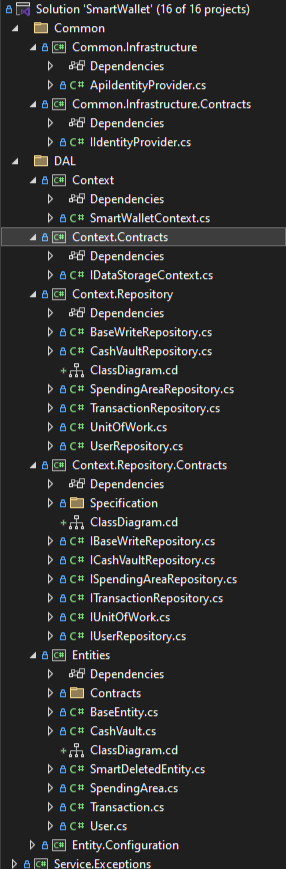


Рисунок 8 - Часть структуры проекта

Ниже представлены конечные точки API.

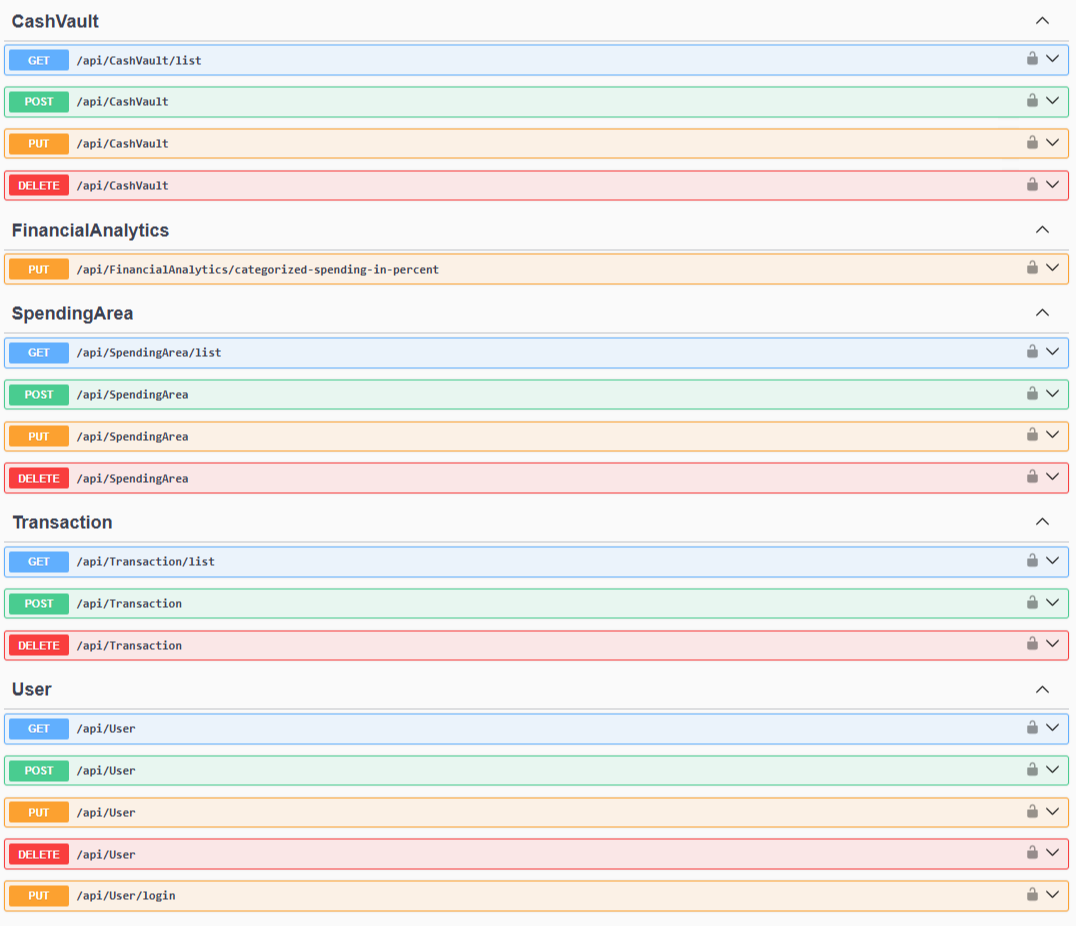


Рисунок 9 - Конечные точки Api

**3.4 Используемые технические средства**

Для работы программы необходим персональный компьютер(сервер) с операционной системой, которая поддерживающей платформу .NET Core.

**3.5 Вызов и загрузка**

Web API “SmartWallet” не вызывается напрямую пользователем, как традиционное приложение. Вместо этого, API вызывается клиентскими приложениями (например, мобильными приложениями или веб-сайтами) посредством отправки HTTP-запросов к определенным URL-адресам (endpoints)(Рисунок 9).

**3.6 Входные и выходные данные**

Web API принимает входные данные в виде HTTP-запросов. Входные данные могут передаваться как в теле запроса (например, в формате JSON), так и в параметрах URL. Характер входных данных зависит от конкретного endpoint API и выполняемой операции.

Входные данные:

* Добавление денежного хранилища/транзакции/области трат;
* Регистрация пользователя;
* Вход в аккаунт;
* Обновление денежного хранилища/области трат/пользователя;
* Удаление денежного хранилища/транзакции/области трат/пользователя;
* Получение данных о денежном хранилище/транзакции/области трат/пользователе.

Выходне данные:

* Список денежных хранилищ/транзакций/областей трат;
* Агрегированные траты в процентах.

**4 Программа и методика испытаний**

**4.1 Объекты испытаний**

Объектом испытаний является Web API приложение «SmartWallet» и его функционал.

**4.2 Цель испытаний**

Испытания проводятся с целью проверки соответствия проекта требованиям, указанным в техническом задании.

**4.3 Требования к программе**

Соответствие программного продукта требованиям, указанным в техническом задании.

**4.4 Методы испытаний**

Испытания проводятся в следующем порядке:

* Тестирование endpoints для управления транзакциями;
* Тестирование endpoints для управления категориями расходов;
* Тестирование endpoints для аутентификации и авторизации;
* Тестирование endpoints для аналитики.

**4.5 Тестовый пример**

Таблица 1 – Тест – кейс 01 «Тест успешной регистрации пользователя»

|  |  |  |  |
| --- | --- | --- | --- |
| Номер тест-кейса | 01 | | |
| Название: | Тест успешной регистрации пользователя | | |
| Действие | | Ожидаемый результат | Результат теста:  пройден |
| Предусловие: | | | |
| Запустить сервер с БД | | Сервер запустился без ошибок | Пройден |
| Шаги теста: | | | |
| Сформировать корректное тело запроса | | Задать данные в соответствии с требованиями Api | Пройден |
| Отправить http запрос по адресу POST <https://localhost:7178/api/User> с телом запроса | | Ответ от сервера со статусом 200 и данными пользователя | Пройден |
| Постусловие: | | | |
| Закрыть приложение, выйти из OS | | Приложение закрыто | Пройден |

Таблица 2 – Тест – кейс 02 «Обновление данных о пользователе»

|  |  |  |  |
| --- | --- | --- | --- |
| Номер тест-кейса | 02 | | |
| Название: | Обновление данных о пользователе | | |
| Действие | | Ожидаемый результат | Результат теста:  пройден |
| Предусловие: | | | |
| Запустить сервер с БД, авторизоваться в системе | | Сервер запустился без ошибок. Авторизация успешно пройдена | Пройден |
| Шаги теста: | | | |
| Сформировать корректное тело запроса | | Задать данные в соответствии с требованиями Api | Пройден |
| Отправить http запрос по адресу PUT https://localhost:7178/api/User | | Ответ от сервера со статусом 200 и данными пользователя | Пройден |
| Постусловие: | | | |
| Закрыть приложение, выйти из OS | | Приложение закрыто | Пройден |

**5 Руководство оператора**

**5.1 Выполнение программы**

Для начала работы необходимо зарегистрироваться в системе по адресу POST <https://localhost:7178/api/User.>

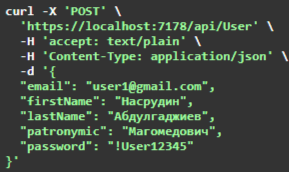


Рисунок 10 - Пример коррекного запроса на регистрацию

Далее нужно войти в аккаунт. Для этого нужно отправить запрос по адресу PUT <https://localhost:7178/api/User/login> в ответе должен быть JWT токен, который необходимо передавать в качестве заголовка аутентификации во все конечные точки, нуждающиеся в авторизации[.](https://localhost:7178/api/User.)

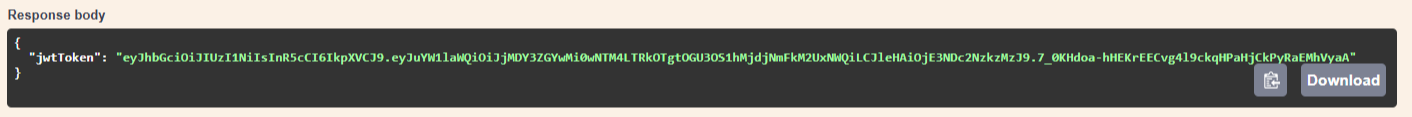


Рисунок 11 - Ответ от сервера, содержащий JWT токен

**5.2 Сообщение оператору**

При отправке запроса от сервера могут придти следующие статус коды:

* 404 – Запрашиваемая сущность не найдена в базе данных;
* 422 – Запрос на обновление/создание сущности не прошел валидацию;
* 401 – В запросе не содержится заголовок авторизации, или пользователь ввёл не верный логин или пароли при попытке входа в аккаунт;
* 200 – Запрос успешно обработан.

**Заключение**

В ходе выполнения курсовой работы был разработан Web API “SmartWallet” для управления личными финансами. Разработанный API предоставляет функциональность для учета расходов, планирования бюджетов, анализа финансовых привычек и управления денежными хранилищами.

Проведенные испытания показали, что Web API соответствует требованиям, изложенным в техническом задании. Web API обеспечивает надежную и безопасную работу с данными пользователей, а также предоставляет удобный интерфейс для клиентских приложений.

Разработанный Web API может быть использован в качестве основы для создания различных приложений для личного финансового планирования, что делает его актуальным и востребованным.

Разработка велась с использованием C#, .NET Core, PostgreSQL, ASP.Net Core.

**Приложение А**

## Текст программы

/// <summary>

/// Сервис финансовой аналитики

/// </summary>

public class FinancialAnalyticsService(IUnitOfWork unitOfWork) : IFinancialAnalyticsService

{

private readonly IUserRepository userRepository = unitOfWork.UserRepository;

private readonly ITransactionRepository transactionRepository = unitOfWork.TransactionRepository;

async Task<(double SpendingAmount, Dictionary<Guid, double> CategorizedSpendingInPercent)> IFinancialAnalyticsService.GetCategorizingSpendingByMonthOfYearAndUserIdAsync(Guid userId,

DateOnly monthOfYear,

CancellationToken token)

{

var user = await userRepository.GetUserByIdAsync(userId, token)

?? throw new EntityNotFoundServiceException($"Пользователь с id = {userId} не найден.");

var source = await transactionRepository.GetListByMonthAndUserIdAsync(userId, monthOfYear, token);

var categorizedTransactions = source.GroupBy(x => x.ToSpendingAreaId).ToList();

var spendingAmount = 0.0;

var categorizedSpending = new Dictionary<Guid, double>();

foreach (var category in categorizedTransactions)

{

categorizedSpending.Add(category.Key, 0.0);

foreach (var transaction in category)

{

spendingAmount += transaction.Value;

categorizedSpending[category.Key] += transaction.Value;

}

}

foreach (var category in categorizedSpending.Keys)

{

categorizedSpending[category] = GetPercentage(spendingAmount, categorizedSpending[category]);

}

return (spendingAmount, categorizedSpending);

}

public static double GetPercentage(double sum, double part, int decimals = 2)

=> sum <= 0.0 ? 0.0 : Math.Round((part / sum) \* 100, decimals);

}

**Источники, использованные при разработке**

1. ГОСТ Р ИСО\_МЭК 25051-2017 Требования к качеству готового к использованию программного продукта (RUSP) и инструкции по тестированию.
2. ЕСПД Единая система программной документации.

**Internet – ресурсы**

1. Введение в C# [Электронный ресурс] – <https://metanit.com/sharp>