|  |
| --- |
| **МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ** |
| **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕХНОЛОГИЙ И УПРАВЛЕНИЯ ИМЕНИ К.Г. РАЗУМОВСКОГО (ПЕРВЫЙ КАЗАЧИЙ УНИВЕРСИТЕТ)»**  **(ФГБОУ ВО «МГУТУ ИМ. К.Г. РАЗУМОВСКОГО (ПКУ)»)** |
|  |
| **УНИВЕРСИТЕТСКИЙ КОЛЛЕДЖ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ** |

|  |  |
| --- | --- |
|  | ДОПУСКАЕТСЯ К ЗАЩИТЕ |
|  | Председатель ПЦК специальности 09.02.07 Информационные системы и программирование |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Е.В. Вернер |
|  | «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г. |

|  |
| --- |
| **ДИПЛОМНЫЙ ПРОЕКТ** |
|  |
| на тему: Разработка информационной системы по геймификации и управлению задачами (на примере приложения «Todoist») |
|  |
| студента группы 090207-9о-пр-21/2  специальности 09.02.07 Информационные системы и программирование |
| Хохлова Дмитрия Алексеевича |

|  |  |  |
| --- | --- | --- |
| Студент |  | Д.А. Хохлов |
| Руководитель |  | А. А. Марков |

|  |  |  |
| --- | --- | --- |
| Дата защиты «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2025 г. | | |
| Оценка:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
| Председатель ГЭК |  | П.Р. Сафиканов |

Москва

2025

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc200315817)

[1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 6](#_Toc200315818)

[1.1 Исследование предметной области 6](#_Toc200315819)

[1.1.1 Терминология в предметной области 6](#_Toc200315820)

[1.1.2 Автоматизация процессов 6](#_Toc200315821)

[1.1.3 Анализ схожих информационные системы 8](#_Toc200315822)

[1.1.4 Обоснование актуальности 15](#_Toc200315823)

[1.2 Анализ и выбор инструментальных средств 15](#_Toc200315824)

[1.2.1 Критерии выбора 15](#_Toc200315825)

[1.2.2 Выбранные инструменты 16](#_Toc200315826)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 18](#_Toc200315827)

[2.1 Спецификация программного изделия 18](#_Toc200315828)

[2.1.1 Постановка задачи 18](#_Toc200315829)

[2.1.2 Наименование программы 18](#_Toc200315830)

[2.1.3 Функциональное назначение 19](#_Toc200315831)

[2.1.4 Требования к функциональным характеристикам 19](#_Toc200315832)

[2.1.5 Требования к составу и параметрам технических средств 19](#_Toc200315833)

[2.2 Проектирование программного изделия 20](#_Toc200315834)

[2.2.1 Диаграмма вариантов использования 20](#_Toc200315835)

[2.2.2 Архитектура информационной системы 21](#_Toc200315836)

[2.2.3 Защита информационной системы 23](#_Toc200315837)

[2.2.4 Описание используемых библиотек 23](#_Toc200315838)

[2.3 Разработка программного изделия 25](#_Toc200315839)

[2.3.1 Описание разработки 25](#_Toc200315840)

[2.3.2 Трудности и решения 30](#_Toc200315841)

[2.3.3 Перспективы усовершенствования информационной системы 31](#_Toc200315842)

[2.4 Тестирование ИС 32](#_Toc200315843)

[ЗАКЛЮЧЕНИЕ 36](#_Toc200315844)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 39](#_Toc200315845)

[ПРИЛОЖЕНИЕ А 42](#_Toc200315846)

# ВВЕДЕНИЕ

Дипломный проект (далее ДП) посвящен разработке информационной системы по геймификации и управлению задачами (на примере приложения «Todoist»).

Внедрение геймификации в информационные системы становится все более популярным стратегическим подходом в современном бизнесе. Геймификация представляет собой использование игровых элементов и механик в рабочей среде для повышения мотивации, вовлеченности и участия пользователей. Она предлагает новые возможности для создания стимулирующей и привлекательной рабочей среды, способствующей достижению лучших результатов.

Геймификация является относительно новым понятием, поэтому необходимо определить его и описать основные принципы. Геймификация базируется на использовании игровых элементов, таких как цели, правила, рейтинги, достижения и награды, для создания стимулирующей и привлекательной рабочей среды. Она основывается на теории мотивации, психологии игр и дизайна игр.

Актуальность темы обусловлена растущей потребностью бизнеса в инновационных методах управления персоналом, повышения продуктивности и лояльности сотрудников, а также в усилении вовлеченности пользователей в цифровых сервисах.

Объектом дипломного проекта является управление задачами.

Предмет дипломного проекта информационная система по управлению задачами.

Целью данного дипломного проекта является разработка удобной, интуитивно понятной и увлекательной информационной системы для эффективного управления и распределения задач с использованием элементов геймификации.

Исходные данные: функционал приложения «Todoist», документация разработчика приложений, работающих на С#, документация разработчика ботов для Telegram.

В задачи данного дипломного проекта входит:

* анализ предметной области;
* рассмотрения аналогов;
* проектирование архитектуры информационной системы;
* создание интерфейса;
* реализация основных функций.

Теоретическая значимость обусловлена cозданием подходов к использованию to-do листов как инструмента личной эффективности.

Практическая значимость обусловлена использование геймификации в тайм-менеджменте мотивирует пользователя выполнять задачи за счет игровых элементов (баллы, награды), что повышает вовлеченность и дисциплинирует в следовании намеченным целям.

Круг рассматриваемых проблем:

* изучение предметной области;
* создание информационной системы;
* геймификация управления задачами.

Структура ДП состоит из введения, двух глав (теоретическая, практическая), заключения, списка используемых источников, а также приложения с программным кодом.

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## Исследование предметной области

## Терминология в предметной области

To-Do List– инструмент для фиксации задач, которые необходимо выполнить. В современных приложениях (например, Todoist) туду-листы включают функции напоминаний, категоризации и интеграции с другими сервисами.

Управление задачами – процесс планирования, организации и контроля выполнения задач, направленный на достижение поставленных целей. В цифровых системах оно реализуется через создание списков дел, установку сроков, приоритезацию и делегирование.

Геймификация– применение игровых механик (баллы, награды) в неигровых контекстах для повышения мотивации пользователей. В системах управления задачами (как Todoist) она может выражаться в виде достижений за выполненные задачи или рейтинговой системы."

Продуктивность – показатель эффективности выполнения задач, часто измеряемый соотношением результата к затраченному времени. Приложения вроде Todoist помогают повышать продуктивность за счёт структурирования работы и снижения когнитивной нагрузки.

Информационная система **(**далее **ИС)** – это программно-аппаратный комплекс, предназначенный для сбора, хранения, обработки, поиска и передачи данных с целью поддержки управленческих и бизнес-процессов.

## Автоматизация процессов

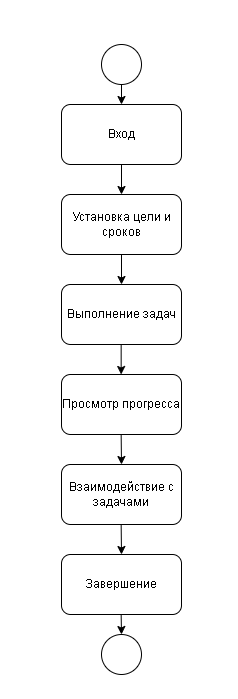
ИС направлена на автоматизацию управления задачами, и повышение продуктивности выполнения этих задач с помощью геймификации.

Ключевые процессы автоматизации:

* создание, редактирование, категоризация и удаление задач;
* отслеживание статусов, сроков;
* cбор и обработка статистики по выполнению задач;
* **мотивационные механики;**
* **социальное взаимодействие.**

ИС позволит пользователям удобно распределять задачи, отслеживать прогресс и повышать эффективность работы за счет игровых элементов. Система автоматизирует рутинные процессы, предоставляет аналитику продуктивности и поддерживает вовлеченность через мотивационные механизмы. Это создаст удобную и стимулирующую среду для управления личными и командными задачами.

Данный процесс представлен в виде схемы на рисунке 1.



1. Бизнес-процесс

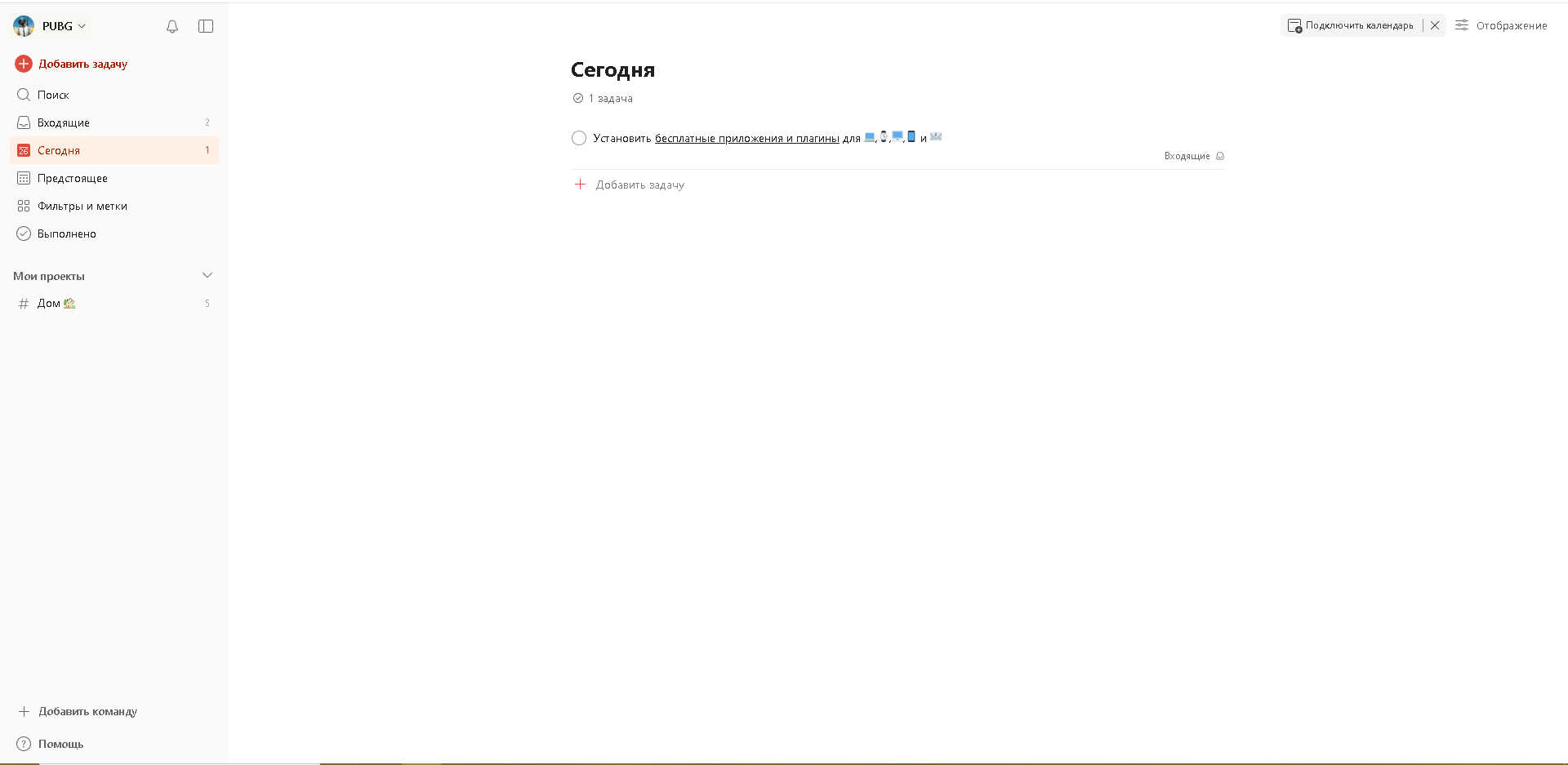
## Анализ схожих информационные системы

В данный момент времени существует множество информационных систем с уклоном на To-Do List. Для анализа были выбраны самые популярные и востребованные информационные системы.

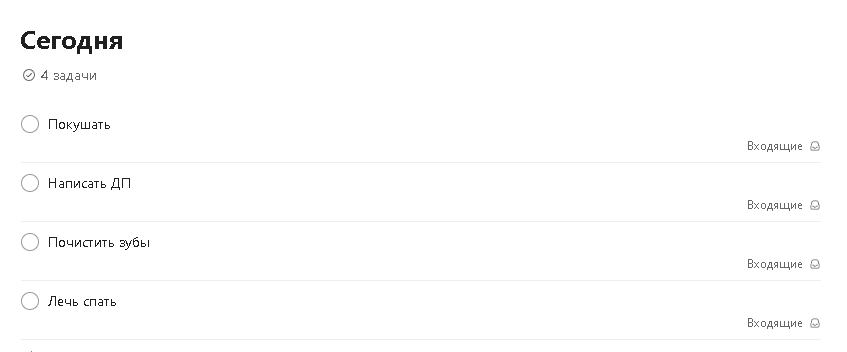
Todoist – это **инструмент для управления задачами и проектами**. Он предлагает множество функций, которые помогают пользователям организовывать свою работу и личные дела.

В Todoist можно создавать задачи, устанавливать сроки, добавлять метки и приоритеты, а также делегировать задачи другим пользователям. Приложение поддерживает создание подзадач и проектов, что позволяет разбивать большие задачи на более мелкие и управляемые части.

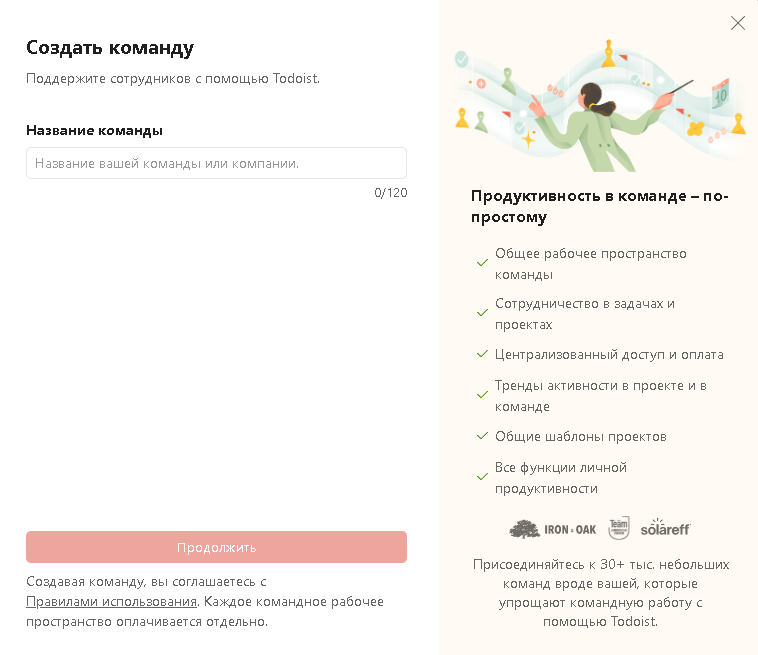
Данная информационная система изображена на рисунках 2 – 5.



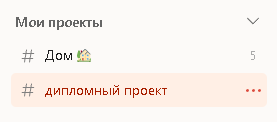
1. Начальная страница Todoist



1. Список дел в Todoist



1. Возможность добавления команды в Todoist

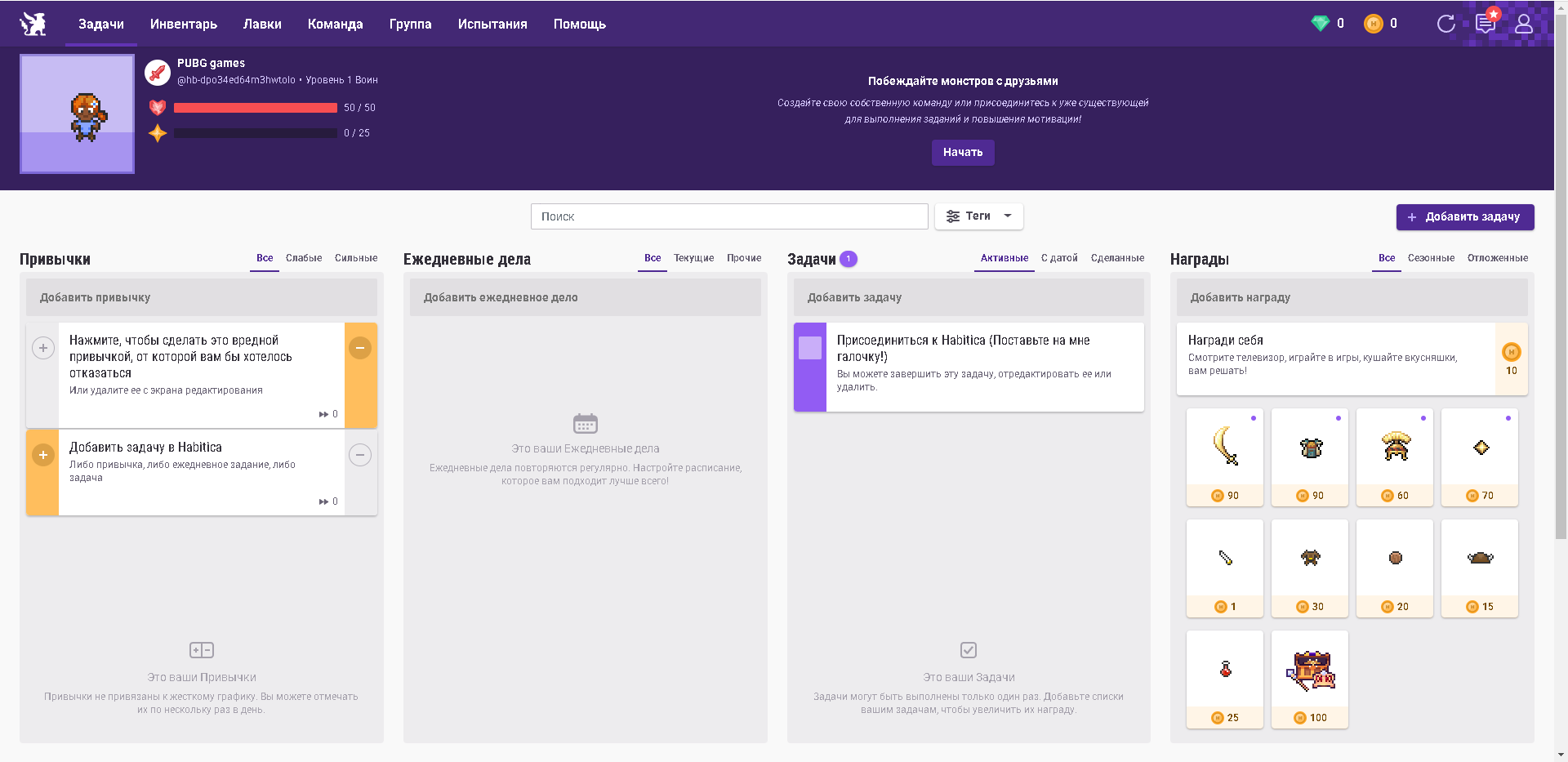


1. Возможность деления задач на проекты в Todoist

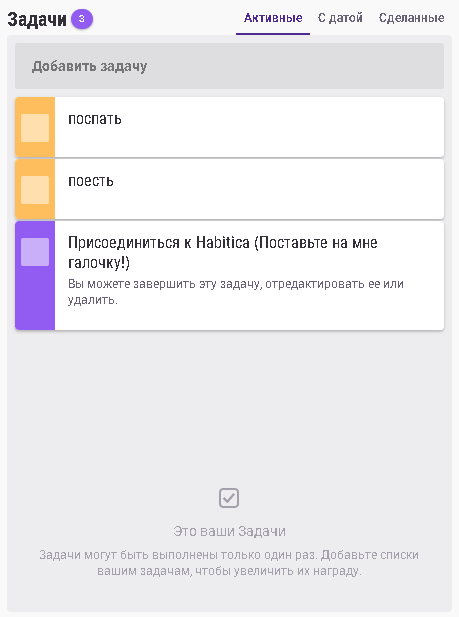
**Habitica** – это **информационная система для привычек и производительности, которое превращает жизнь пользователя в ролевую игру**

Она позволяет пользователям создавать список задач, которые они хотят выполнить, например, учёба или уборка, а затем вознаграждает их очками опыта или золотом за выполнение этих задач.

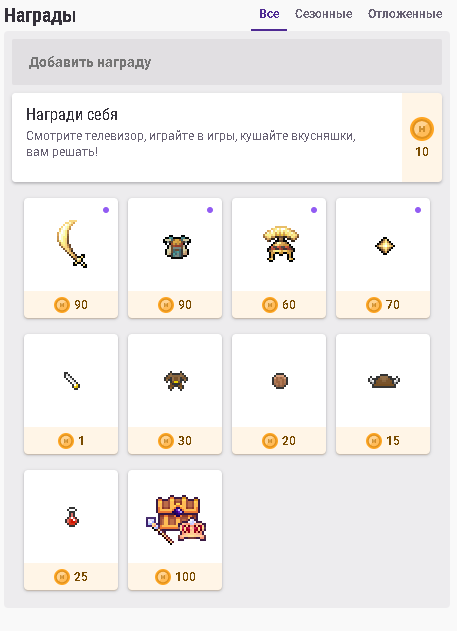
Данная информационная система изображена на рисунках 6 – 8.



1. Начальная страница **Habitica**

****

1. Список дел в **Habitica**



1. Список наград в **Habiticaэ**

После изучения данных информационных систем были выявлены плюсы и минусы каждой системы. Плюсы и минусы можно рассмотреть в таблице 1.

1. Сравнение информационных систем

|  |  |  |
| --- | --- | --- |
| **Категория** | **Todoist** | **Habitica** |
| Плюсы | * **Профессиональный интерфейс** – чёткий, быстрый, без лишнего; * **Г**ибко**сть** – проекты, метки, фильтры, шаблоны; * **Интеграции** (Google Календарь, Slack и др.); * **Надёжность** – работает без багов; * **Коллаборация** – удобно для команд. | * **Мотивация через игру** – прокачка персонажа, квесты; * **Вовлекает** – делает рутину веселее; * **Социальность** – групповые квесты с друзьями; * **Привычки** – удобно трекать регулярные дела; * **Бесплатно** – базовая версия полноценна. |
| Минусы | * **Можно использовать только через сайт;** * **Скучноват** – нет «вау»-мотивации; * **Геймификация слабая** (только Karma); * **Для привычек не идеален** – нет глубокой аналитики. | * **Нужно устанавливать приложение;** * **Выглядит непрофессионально** – пиксельный дизайн; * **Сложные проекты неудобны** – нет нормальных подзадач; * **Баговат** – мобильное приложение глючит; * **Мало интеграций** – только базовые; * **Отвлекает** – можно увлечься игрой, а не делами. |
| Для кого? | * Кому **важна эффективность**, а не мотивация; * Для **работы/учебы;** * Если нужны **сложные проекты.** | * Тем, кому **скучно** вести списки дел; * Для **привычек и личных целей;** * Если любите **игры и соревнование.** |

В Информационной системе, разработанной входе ДП взаимодействие с пользователем будет проходить с помощью мессенджера «Telegram». Благодаря кроссплатформенности, пользователи смогут получать уведомления и работать с системой как с компьютера, так и со смартфона, без необходимости установки дополнительного ПО. «Telegram» обеспечивает быструю доставку сообщений, поддержку ботов для автоматизации задач, а также высокий уровень безопасности.

Так же преимуществом данной информационной системы является легкая в понимание геймификация. Возможность создавать команды и соревноваться между собой. Приятный интерфейс для пользователей, который побуждает заниматься тайм-манежем.

## Обоснование актуальности

Актуальность темы обусловлена растущей потребностью бизнеса в инновационных методах управления персоналом, повышения продуктивности и лояльности сотрудников, а также в усилении вовлеченности пользователей в цифровых сервисах.

Сегодня все больше людей ищут способы не просто планировать дела, а делать это с удовольствием. Обычные списки задач быстро надоедают, а приложения вроде Todoist хоть и удобные, но слишком сухие и деловые. С другой стороны, геймифицированные сервисы, такие как Habitica, добавляют азарта, но часто выглядят слишком по-детски или отвлекают от реальных дел.

## Анализ и выбор инструментальных средств

## Критерии выбора

Основными критерия выбора для разработки информационной системы являются:

* функциональные возможности;
* безопасность системы;
* возможность создания удобного для пользователя интерфейса;
* масштабируемость;
* доступность в регионе.

## Выбранные инструменты

Анализ инструментов для разработки информационной системы представлен в таблице 2.

1. Выбор инструментов

| **Инструменты** | **Плюсы** | **Минусы** |
| --- | --- | --- |
| C# | Высокая производительность, хорошая поддержка OOP, отличная интеграция с Windows | Меньшая кроссплатформенность, заточенность под Windows |
| Java | Кроссплатформенность, широкое сообщество, используется в Android | Медленнее запускается, требует больше памяти |
| Visual Studio | Мощная IDE, отличная отладка, поддержка .NET | Тяжеловесность, привязанность к Windows |
| Eclipse | Быстрая, открытая IDE, хороша для Java | Меньше функционала из коробки по сравнению с VS |
| PostgreSQL | Надежность, поддержка расширений, хорошо работает с большими данными | Сложность настройки, не самая простая кривая обучения |
| MySQL | Простота, широкая поддержка хостингов, быстрее на малых объемах данных | Меньше возможностей масштабирования, не такие строгие типы |
| pgAdmin4 | Удобный графический интерфейс для PostgreSQL, продвинутый функционал | Более сложный интерфейс для новичков |
| phpMyAdmin | Простой интерфейс для управления MySQL | Ограниченный функционал по сравнению с pgAdmin4 |
| Docker | Контейнеризация, переносимость, легкость в развертывании | Не всегда нужен весь функционал, требуется знание Dockerfile |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| VirtualBox | Полноценная виртуализация, поддержка разных ОС | Большая нагрузка на систему, медленнее запускается |
| Git | Распределённая система контроля версий, гибкость | Сложность для новичков, требует понимания ветвления |
| SVN | Централизованная система контроля версий, проще концептуально | Менее гибкая, плохо масштабируется на большие проекты |
| GitHub | Интеграция с Git, поддержка CI/CD, популярность | Ограничения в приватных репозиториях без оплаты |
| Bitbucket | Интеграция с Git, приватные репозитории бесплатно | Меньшая популярность по сравнению с GitHub |

Исходя из данного анализа под критерии инструментов для разработки информационной системы подходят:

* C#;
* Visual Studio;
* PostgresSQL;
* PgAdmin4;
* Docker;
* Git;
* Github.

# ПРАКТИЧЕСКАЯ ЧАСТЬ

## Спецификация программного изделия

## Постановка задачи

Разработка информационной системы по геймификации и управлению задачами (на примере приложения «Todoist»).

Исходные данные: функционал приложения «Todoist», документация разработчика приложений, работающих на С#, документация разработчика ботов для Telegram.

Постановка задачи: изучить функциональные возможности приложения «Todoist» и на их основе разработать распределённое приложение с серверной частью на С# и клиентским интерфейсом в виде Telegram-бота, позволяющее пользователям создавать и выполнять задачи с использованием системы очков, наград, клановой и рейтинговой систем для повышения мотивации и вовлечённости.

## Наименование программы

Наименование информационной системы — «TaskSlayer» — выбрано не случайно и несёт в себе как функциональную, так и эмоциональную нагрузку. Название состоит из двух частей: **«Task»** (задача) и **«Slayer»** (разрушитель, победитель), что в совокупности отражает основную идею и философию приложения — помочь пользователю «разрушать» накопившиеся задачи, эффективно справляться с делами и достигать целей с чувством удовлетворения и победы. Такой подход делает само выполнение задач не рутиной, а своеобразной игрой, в которой каждый пользователь — герой, сражающийся с хаосом и прокрастинацией.

## Функциональное назначение

Информационная система «**TaskSlayer**», предназначенная для управления задачами**,** превращения рутинного планирование в увлекательную игру. Система предназначена для повышения продуктивности пользователей за счет мотивационных механик, визуального прогресса и элементов соревнования.

## Требования к функциональным характеристикам

ИС должна работать на всех устройства поддерживающих мессенджер Telegram. Корректно отвечать на запросы пользователя.

## Требования к составу и параметрам технических средств

Минимальные характеристики телефона для полноценной работоспособности:

* операционная система Android версия 6.0 или выше;
* операционная система iOS версия 11.0 или выше;
* 100 МБ внутренней памяти;
* доступ к интернету.

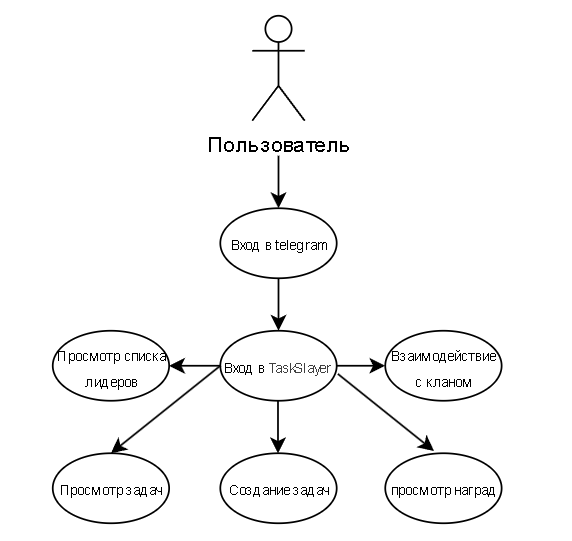
Минимальные характеристики Персонального Компьютера для полноценной работоспособности:

* **операционная система** Microsoft Windows 7 и выше, macOS 11 (Big Sur) и выше;
* **процессор** Intel, AMD или Apple Silicon Processor;
* **оперативная память** не менее 4 ГБ;
* **Жёсткий диск** не менее 10 ГБ свободного дискового пространства;

## Проектирование программного изделия

## Диаграмма вариантов использования

На рисунке 9 представлена диаграмма, отражающая ключевые сценарии взаимодействия пользователя с системой, включая вход в мессенджер Telegram, управление задачами в приложении TaskSlayer и взаимодействие с игровым кланом. Данная схема визуализирует основные пользовательские потоки и демонстрирует логическую связь между различными модулями системы.



1. Диаграмма вариантов

## Архитектура информационной системы

Серверная часть ИС обрабатывает запросы клиентской части (с использованием jwt токенов), организует защиту и хранение данных. Код написан на языке программирования C# с использованием таких подходов как SOLID и Domain-Driven Design (DDD). Для хранения данных был использован postgresSQL.

**SOLID** – это пять ключевых принципов объектно-ориентированного проектирования, обеспечивающих гибкость, масштабируемость и поддержку кода:

* **SRP (Single Responsibility Principle)** каждый класс должен иметь только одну причину для изменения. Один класс равен одной ответственности;
* **OCP (Open/Closed Principle)** классы должны быть открыты для расширения, но закрыты для изменения. Новая функциональность добавляется через наследование/интерфейсы без модификации существующего кода;
* **LSP (Liskov Substitution Principle)** объекты подклассов должны заменять объекты родительских классов, не нарушая работу программы. наследники не должны "ломать" поведение базового класса.
* **ISP (Interface Segregation Principle)** много специализированных интерфейсов лучше одного общего. Клиенты не должны зависеть от методов, которые они не используют.
* **DIP (Dependency Inversion Principle** зависимости должны строиться на абстракциях, а не на конкретных реализациях. Модули высокого уровня не зависят от модулей низкого уровня напрямую.

**DDD** – это подход к разработке ПО, ориентированный на глубокое моделирование предметной области (Domain) для точного отражения бизнес-процессов в коде.

Клиентская часть ИС создает запросы и отправляет на серверную часть ИС. Реализует команды для пользователя и графический интерфейс. Программный код написан на C# с использование м подхода SOLID. Для взаимодействия с серверной часть используется NSwag.

NSwag – это современный и гибкий инструмент, предназначенный для автоматической генерации клиентского кода и документации API на основе спецификаций OpenAPI (ранее известного как Swagger). Он предоставляет разработчикам возможность не только создавать описания API в формате OpenAPI на основе существующего ASP.NET Core-приложения, но и генерировать типобезопасный клиентский код на различных языках программирования, включая C# и TypeScript. Это значительно упрощает интеграцию фронтенда с бэкендом, так как позволяет автоматически создавать классы и методы, отражающие структуру и логику веб-сервиса, без необходимости вручную описывать каждый запрос и ответ.

Достоинства решения:

* **высокая модульность и читаемость кода** благодаря применению SOLID и DDD;
* **гибкость архитектуры** – систему легко расширять и модифицировать;
* **надёжная система аутентификации** с использованием JWT;
* **использование PostgreSQL** как отказоустойчивой и производительной СУБД;
* **минимизация ручного труда** за счёт NSwag – генерация клиентского кода и API-документации автоматизирована;
* **поддержка типобезопасности** в клиент-серверном взаимодействии;
* **быстрое масштабирование и сопровождение** благодаря разделению по слоям и внедрению абстракций.

Недостатки:

* **зависимость от сторонних библиотек** (NSwag, JWT), что требует постоянного контроля версий и совместимости;
* **более высокий порог входа для разработчиков** из-за использования DDD и строгих архитектурных подходов;
* **повышенное время разработки** на начальных этапах из-за детального моделирования предметной области;
* **чувствительность к неверному управлению временными зонами** при работе с типами DateTimeOffset и PostgreSQL;
* **сложности при отладке авто-сгенерированного кода**, особенно при ошибках сериализации или конфликтах моделей.

## Защита информационной системы

Защита информационной системы осуществлена с помощью:

* взаимодействия клиента и сервера с помощью jwt токенов;
* двухфакторной авторизации в Telegram;
* проверки на подозрительную активность (50 сообщений в минуту).

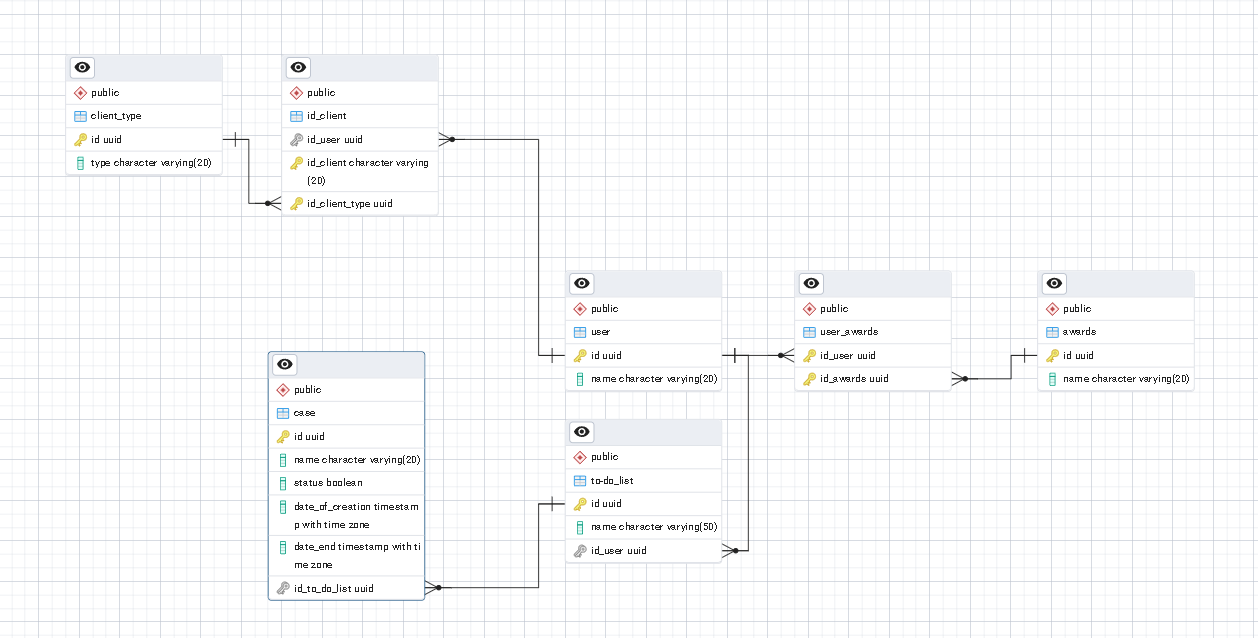
## Описание используемых библиотек

* **Microsoft.AspNetCore.Authentication.JwtBearer** middleware для ASP.NET Core, позволяющий приложению принимать OpenID Connect bearer токены;
* **Microsoft.AspNetCore.Diagnostics** middleware для обработки исключений, отображения страниц ошибок и диагностической информации в ASP.NET Core;
* **Microsoft.AspNetCore.Http.Extensions** общие методы расширения для HTTP-абстракций, заголовков, запросов/ответов и состояния сессии в ASP.NET Core;
* **Microsoft.EntityFrameworkCore** ORM (Object-Relational Mapper) для .NET, поддерживающий LINQ-запросы, отслеживание изменений, обновления и миграции схемы. Работает с PostgreSQL;
* **Microsoft.EntityFrameworkCore.Tools** инструменты Entity Framework Core для консоли диспетчера пакетов NuGet в Visual Studio;
* **Microsoft.Extensions.DependencyInjection.Abstractions** абстракции для внедрения зависимостей (Dependency Injection) в .NET;
* **Microsoft.VisualStudio.Azure.Containers.Tools.Targets** файлы для интеграции инструментов Visual Studio для работы с контейнерами;
* **Npgsql.EntityFrameworkCore.PostgreSQL** провайдер PostgreSQL для Entity Framework Core;
* **Swashbuckle.AspNetCore** инструменты для автоматической генерации документации API (Swagger) в ASP.NET Core;
* **System.IdentityModel.Tokens.Jwt** (Legacy) библиотека предоставляет функционал для создания, сериализации, валидации и обработки JSON Web Tokens (JWT):
* Microsoft.Extensions.ApiDescription.Client пакет для генерации кода и взаимодействия с описанием API в формате OpenAPI в .NET-проектах;
* Microsoft.Extensions.DependencyInjection реализация внедрения зависимостей (Dependency Injection) в проектах на .NET, обеспечивающая слабую связанность компонентов;
* Microsoft.Extensions.Hosting инфраструктура для управления временем жизни приложений, поддержка жизненного цикла приложений (консольных, веб-приложений и сервисов);
* Microsoft.Extensions.Http предоставляет HttpClientFactory, упрощающий конфигурацию, управление временем жизни и расширение HTTP-клиентов в .NET-приложениях;
* Newtonsoft.Json популярная высокопроизводительная библиотека для сериализации и десериализации объектов в JSON-формат;
* NSwag.ApiDescription.Client часть инструментария NSwag, генерирует клиентский код на основе OpenAPI-описания, облегчая интеграцию клиент-серверного взаимодействия;
* Telegram.Bot библиотека, упрощающая создание ботов Telegram на .NET, предоставляя удобные обёртки для API Telegram;
* Telegram.Bots.Extensions.Polling надстройка над Telegram.Bot для организации long polling – способа получения обновлений с сервера Telegram;
* Microsoft.VisualStudio.Azure.Containers.Tools.Targets внутренние цели MSBuild, необходимые для поддержки разработки и развертывания контейнеров в Visual Studio (предварительная версия).

## Разработка программного изделия

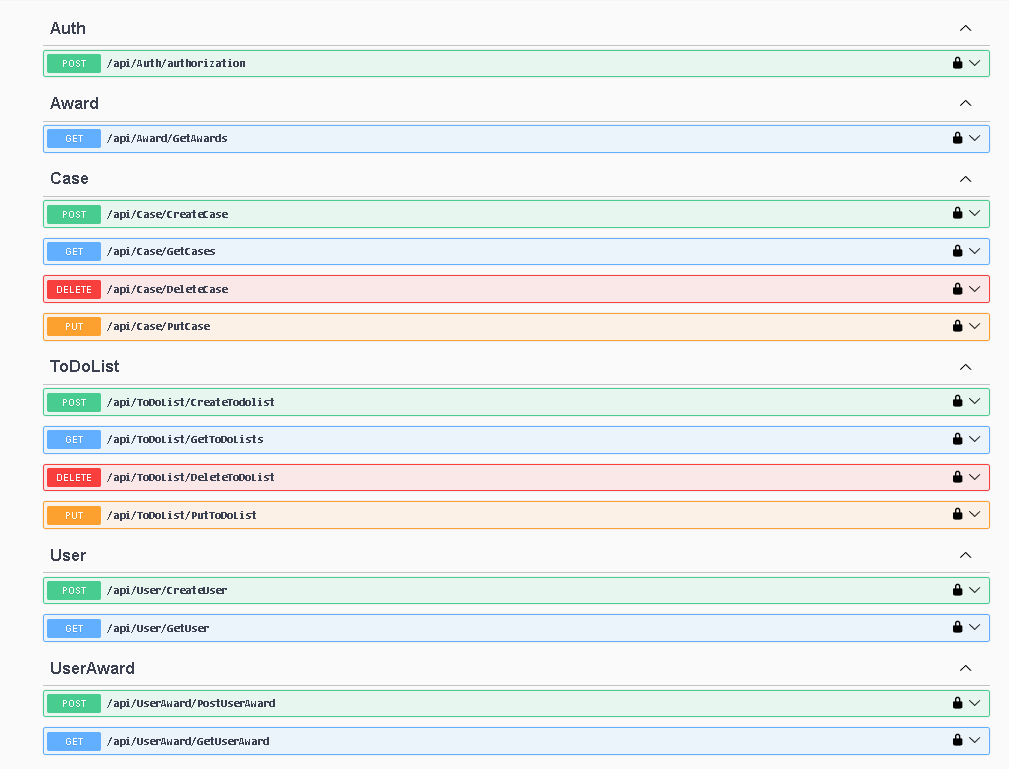
## Описание разработки

Разработка ИС началась с проектирования и реализации базы данных с помощью инструментов pgAdmin4. Диаграмма базы данных изображена на рисунке 10.



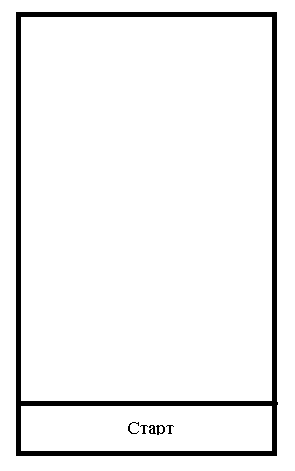
1. Диаграмма базы данных

Разработка серверной часть началась с реализации контроллеров и взаимодействия этих контроллеров с базой данных. С помощью **Microsoft.EntityFrameworkCore были сгенерированы классы базы данных для взаимодействия с PostgreSQL. Проверка работоспособности осуществлялась через Swager. Реализованные контроллеры в swager изображены на рисунке 11.**

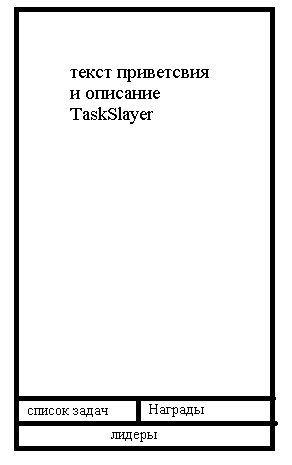
****

1. **Контроллеры серверной части в Swagger**

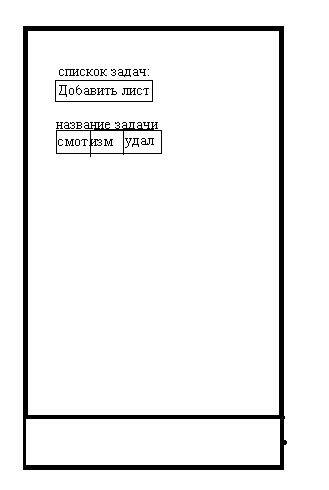
**Разработка клиентской части началась с проектирования макетов интерфейса, изображенных на рисунках.12 – 16.**

****

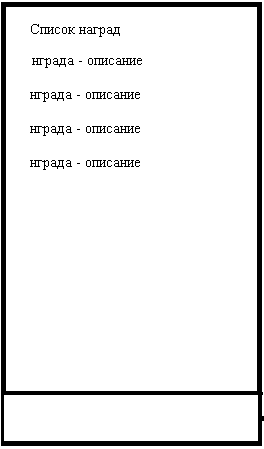
1. **Макет начального экрана**

****

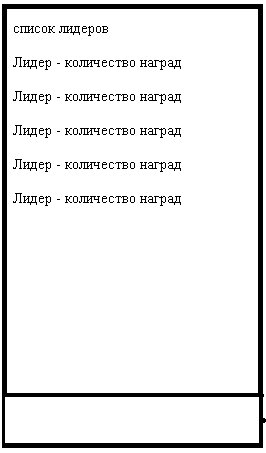
1. **Макет главного меню**

****

1. **Макет списка задач**



1. Макет списка наград



1. Макет списка лидеров

После этапа проектирования и разработки макетов пользовательского интерфейса было выполнено подключение сервиса Telegram для реализации взаимодействия с пользователями. В рамках этой задачи была произведена настройка бота с использованием официального инструмента Telegram – BotFather. С его помощью был создан уникальный бот, назначено имя, задано описание и сгенерирован токен доступа, необходимый для последующей интеграции. Это стало отправной точкой для реализации функционала автоматизированного общения с пользователями через мессенджер.

## Трудности и решения

Во время разработки ИС появлялись определённые ошибки, которые были успешно решены. Трудности и решения представлены в таблице 3.

1. Трудности и решения

|  |  |
| --- | --- |
| **Трудность** | **Решение** |
| Неверное определение временной зоны при работе с DateTimeOffset и PostgreSQL | Приведение всех значений к UTC через ToUniversalTime() и согласование формата времени с ожидаемым API (например, yyyy-MM-ddTHH:mm:ssZ) |
| Ошибки сериализации и десериализации JSON при использовании NSwag и Newtonsoft.Json | Настройка JsonSerializerSettings, ручная корректировка моделей DTO и проверка соответствия типов полям JSON |
| Повышенная сложность архитектуры из-за применения DDD | Разделение проекта на слои (Domain, Application, Infrastructure), использование шаблонов проектирования и внедрение модульного тестирования для упрощения отладки |
| Ошибки при генерации клиентского кода через NSwag (например, при изменении структуры API) | Регулярное обновление OpenAPI-описания и перегенерация клиентов, применение версионирования API |
| Относительно высокий порог входа для новых участников команды | Подготовка внутренней документации, схем архитектуры, и проведение коротких вводных сессий |
| Долгая настройка авторизации и аутентификации с использованием JWT | Использование встроенных возможностей ASP.NET Core (JwtBearerOptions), тестирование через Postman и Swagger |
| Конфликты зависимостей сторонних библиотек | Контроль версий через NuGet, настройка PackageReferences и использование binding redirects при необходимости |

Продолжение таблицы 3

|  |  |
| --- | --- |
| Ошибки времени выполнения из-за авто-сгенерированного кода | Отладка через трассировку сетевых запросов (например, с помощью Fiddler, Postman или встроенного логгера), проверка сериализованных объектов в runtime |
| Необходимость поддержки типобезопасности между сервером и клиентом | Генерация строго типизированных моделей с помощью NSwag, проверка соответствия с DTO на серверной стороне |

## Перспективы усовершенствования информационной системы

Перспективы усовершенствования информационной системы заключаются в дальнейшем расширении функциональности, повышении удобства использования, производительности и отказоустойчивости, а также в обеспечении более глубокой интеграции с внешними сервисами. Одним из ключевых направлений развития является внедрение системы ролей и прав доступа, что позволит гибко разграничивать возможности пользователей в зависимости от их должностных обязанностей и уровня доверия. Это особенно важно при работе в командной среде или в условиях предприятия, где данные требуют дополнительной защиты.

Также целесообразно рассмотреть возможность внедрения системы уведомлений – как внутренних (в рамках клиентского интерфейса), так и внешних (например, через e-mail или мессенджеры, такие как Telegram). Это обеспечит своевременное информирование пользователей о наступлении сроков задач, изменениях статусов или появлении новых назначений, тем самым улучшая вовлечённость и контроль над рабочими процессами.

С точки зрения технической реализации перспективным направлением является масштабирование системы с использованием облачных технологий, таких как контейнеризация через Docker и развертывание в облаке (например, с применением Azure или AWS). Это позволит не только повысить отказоустойчивость и гибкость системы, но и упростить поддержку и развертывание новых версий.

В части пользовательского интерфейса возможно внедрение более современного фронтенд-фреймворка (например, React или Blazor WebAssembly), что обеспечит лучшую отзывчивость и интерактивность клиентской части. Также стоит рассмотреть реализацию мобильного приложения, синхронизирующегося с основной системой через API, что расширит доступность и удобство использования на различных платформах.

Кроме того, перспективным шагом станет интеграция с системами аналитики и мониторинга, такими как Prometheus или Grafana, что позволит отслеживать поведение системы, фиксировать узкие места и принимать своевременные меры для улучшения производительности. На уровне бизнес-логики возможна реализация механизма автоматической приоритизации задач или рекомендаций на основе анализа активности пользователя и статистики выполнения.

Таким образом, дальнейшее развитие информационной системы может быть направлено как на техническое усовершенствование и повышение производительности, так и на расширение бизнес-функциональности, что обеспечит её долгосрочную актуальность, адаптивность и удобство для конечного пользователя.

## Тестирование ИС

Тестирование ИС представлено в таблице 4.

1. Тестирование **TaskSlayer**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название теста** | **Шаги выполнения** | **Ожидаемый результат** | **Пройден?** |
| 1 | Регистрация нового пользователя | 1) ввести /start;  2) Telegram автоматически регистрирует пользователя. | Пользователь добавлен в базу, выводится приветствие и краткая инструкция. | Да |
| 2 | Создание новой задачи | 1) Отправить /addtask. 2. Ввести описание задачи и срок. | Задача сохранена, отображается подтверждение и список текущих задач. | Да |
| 3 | Просмотр списка задач | 1) отправить /tasks. | Отображается список всех невыполненных задач. | Да |
| 4 | Завершение задачи | 1) отправить /done:  2) выбрать задачу из списка. | Задача помечена как выполненная, начисляются баллы, отображается достижение (если применимо). | Да |
| 5 | Начисление очков за выполненные задачи | 1) выполнить несколько задач:  2) отправить /profile. | Количество очков увеличилось, обновился уровень пользователя (если достигнут порог). | Да |
| 6 | Получение достижений | 1) завершить 5, 10 и 20 задач. | После каждого достижения бот присылает награду (медаль, стикер и т.д.). | Да |
| 7 | Удаление задачи | 1) отправить /deletetask. 2. Выбрать задачу. | Задача удалена из списка, бот подтверждает действие. | Да |
| 8 | Просмотр профиля | 1) отправить /profile. | Отображаются баллы, уровень, количество выполненных задач, достижения. | Да |

Продолжение таблицы 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 9 | Таблица лидеров | 1) ввести команду /leaderboard. | Отображается топ-10 пользователей по количеству очков. | Да |
| 10 | Попытка накрутки очков (многократное завершение одной задачи) | 1) выполнить одну задачу, затем снова отправить /done. | Система не даёт выполнить задачу повторно, очки не начисляются. | Да |
| 11 | Атака на бот-API с массовыми запросами | 1) эмулировать сотни запросов на /done или /addtask. | Telegram блокирует спам, бот временно ограничивает доступ. | Да |
| 12 | Ошибка при вводе команды | 1) ввести /taks (опечатка). | Бот сообщает об ошибке и предлагает список доступных команд. | Да |
| 13 | Поведение при отсутствии задач | 1) отправить /tasks после удаления всех задач. | Бот сообщает, что список задач пуст. | Да |
| 14 | Добавление задачи без описания | 1) ввести /addtask и отправить пустое сообщение. | Бот сообщает об ошибке и предлагает повторить ввод. | Да |
| 15 | Завершение задачи другим пользователем | 1) попробовать завершить задачу, созданную другим аккаунтом. | Бот сообщает, что у пользователя нет прав на эту задачу. | Да |

Продолжение таблицы 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 16 | Работа с несколькими пользователями одновременно | 1) выполнить команды с разных аккаунтов. | Каждому пользователю отображаются только его задачи и данные. | Да |
| 17 | Обновление уровня пользователя | 1) набрать необходимое количество очков. | При достижении нового уровня отображается сообщение о повышении. | Да |

# ЗАКЛЮЧЕНИЕ

В результате выполнения ДП была спроектирована и разработана информационная система по геймификации и управлению задачами (на примере приложения «Todoist»).

В ходе выполнения ДП были решены следующие задачи:

* проведен анализ предметной области, в ходе которого изучены ключевые аспекты, структура и взаимосвязи исследуемой сферы. Определены основные проблемы и тенденции, что позволило сформировать базу для дальнейших решений и рекомендаций;
* рассмотрены аналоги информационных систем, проведен сравнительный анализ их функциональности, архитектуры и особенностей реализации. Выявлены ключевые преимущества и недостатки, что позволило определить оптимальные решения для проектируемой системы;
* спроектирована архитектура информационной системы, включающая модульную структуру, интерфейсы взаимодействия и ключевые компоненты. Определены технологии реализации, что обеспечивает масштабируемость, надежность и безопасность системы;
* реализован пользовательский интерфейс ИС, обеспечивающий интуитивную навигацию и удобство взаимодействия. Интерфейс адаптирован под требования пользователей и корректно отображается на различных устройствах;
* реализованы основные функции информационной системы, обеспечивающие выполнение ключевых задач предметной области. Функционал протестирован на соответствие техническим требованиям и пользовательским ожиданиям;
* выполнено тестирование системы, подтвердившее корректность работы всех модулей и соответствие требованиям. Устранены выявленные ошибки, обеспечена стабильная работа ИС в различных условиях;
* реализована защита информационной системы, соответствующая современным стандартам безопасности. Обеспечена безопасность данных и контроль несанкционированного доступа. Принятые меры гарантируют надежную работу системы в условиях потенциальных угроз.

Разработанная ИС позволяет пользователям:

* взаимодействовать с ботом в Telegram;
* создавать новые задачи;
* взаимодействовать с задачами. Удалять, редактировать, изменять статуса;
* получать награды за выполнение задачи;
* просматривать список наград;
* взаимодействовать с кланами. Создавать, просматривать, присоединяться;
* просматривать список лидеров;

Для создания ИС использовалась среда разработки Visual Studio. VS предоставляет весь спектр нужных инструментов для отладки, установки пакетов и взаимодействия с git. ИС написана на языке C#. Данный язык относится к объектно-ориентированным, что благоприятно влияет на структуру кода, обеспечивая его модульность, масштабируемость и легкость поддержки.

В процессе разработки были выявлены следующие трудности:

* подключение к удалённой базе данных;
* способ защиты данных между клиентом с сервером;
* ошибки отображения задач и списка лидеров;

В будущем ИС может быть модернизирована добавлением улучшенной защиты данных, новых фронтенд приложений, увеличением функционала.

В процессе разработки информационной системы были успешно решены поставленные задачи, что позволило создать полнофункциональное решение для геймификации управления задачами. Система интегрирует удобный пользовательский интерфейс, надежную архитектуру и современные механизмы мотивации, обеспечивая комфортное взаимодействие пользователей с их задачами. Несмотря на возникшие технические сложности, все ключевые функции были реализованы, а система готова к дальнейшему развитию и расширению функциональных возможностей.

Разработанное решение демонстрирует эффективное сочетание технологий C# и платформы Telegram, предоставляя пользователям интуитивно понятный инструмент для повышения продуктивности. В перспективе система может быть дополнена новыми функциями, улучшенными механизмами безопасности и поддержкой дополнительных платформ, что расширит ее применение и повысит удобство использования.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Информационная система Википедия // — URL: [https://ru.wikipedia.org/wiki/Информационная\_система](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) (дата обращения: 22.05.2025).
2. Веб-приложение Википедия // URL: [https://ru.wikipedia.org/wiki/Веб-приложение](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) (дата обращения: 22.05.2025).
3. Клиент-сервер Википедия // — URL: [https://ru.wikipedia.org/wiki/Клиент\_–\_сервер](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_%E2%80%94_%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80) (дата обращения: 22.05.2025).
4. Архитектура программного обеспечения Википедия // URL: [https://ru.wikipedia.org/wiki/Архитектура\_программного\_обеспечения](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) (дата обращения: 23.05.2025).
5. SOLID-принципы проектирования // Хабр — URL: https://habr.com/ru/articles/261773/ (дата обращения: 24.05.2025).
6. Domain-Driven Design (DDD) краткий обзор // Хабр — URL: https://habr.com/ru/companies/pvs-studio/articles/431198/ (дата обращения: 24.05.2025).
7. Геймификация: как игровые механики повышают продуктивность // РБК Тренды — URL: https://trends.rbc.ru/trends/education/61e6f53f9a79470f5e2d2790 (дата обращения: 23.05.2025).
8. Todoist API documentation // Official site — URL: https://developer.todoist.com/ (дата обращения: 25.05.2025).
9. Telegram Bot API documentation // Telegram — URL: https://core.telegram.org/bots/api (дата обращения: 25.05.2025).
10. Firebase documentation // Google Developers — URL: https://firebase.google.com/docs (дата обращения: 25.05.2025).
11. JWT.io: JSON Web Token Introduction // — URL: https://jwt.io/introduction (дата обращения: 26.05.2025).
12. Microsoft Docs – ASP.NET Core Overview // — URL: <https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core> (дата обращения: 26.05.2025).
13. NSwag документация // — URL: <https://github.com/RicoSuter/NSwag> (дата обращения: 26.05.2025).
14. Microsoft Entity Framework Core documentation // — URL: <https://learn.microsoft.com/en-us/ef/core/> (дата обращения: 26.05.2025).
15. PostgreSQL официальная документация // — URL: <https://www.postgresql.org/docs/> (дата обращения: 27.05.2025).
16. Шаблон MVC (Model-View-Controller) Википедия // — URL: <https://ru.wikipedia.org/wiki/Model-View-Controller> (дата обращения: 27.05.2025).
17. Принципы ООП Википедия // — URL: [https://ru.wikipedia.org/wiki/Объектно-ориентированное\_программирование](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) (дата обращения: 27.05.2025).
18. C# Programming Guide // Microsoft Docs — URL: <https://learn.microsoft.com/en-us/dotnet/csharp/> (дата обращения: 28.05.2025).
19. Clean Architecture в .NET // Хабр — URL: https://habr.com/ru/companies/otus/articles/515342/ (дата обращения: 28.05.2025).
20. Telegram Bot Development Guide // Medium — URL: https://medium.com/better-programming/build-a-telegram-bot-using-c-and-asp-net-core-5fbf227c32a7 (дата обращения: 28.05.2025).
21. UX/UI дизайн в разработке приложений // Tilda Education — URL: https://edu.tilda.cc/uxdesign (дата обращения: 28.05.2025).
22. Геймификация в управлении персоналом // HR-портал — URL: https://hr-portal.ru/article/geymifikatsiya-v-upravlenii-personalom (дата обращения: 29.05.2025).
23. Подходы к тестированию программного обеспечения // Википедия — URL: [https://ru.wikipedia.org/wiki/Тестирование\_программного\_обеспечения](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) (дата обращения: 29.05.2025).
24. Тестирование функциональности ботов // Хабр — URL: https://habr.com/ru/articles/539136/ (дата обращения: 29.05.2025).
25. OAuth 2.0 Overview // OAuth.net — URL: https://oauth.net/2/ (дата обращения: 29.05.2025).
26. Капча и защита от ботов // Википедия — URL: <https://ru.wikipedia.org/wiki/CAPTCHA> (дата обращения: 29.05.2025).
27. Защита данных в Telegram Bot // Dev.to — URL: https://dev.to/aaronksaunders/secure-telegram-bot-development-using-nodejs-1e98 (дата обращения: 30.05.2025).
28. Microsoft.IdentityModel.Tokens JWT Tokens // Microsoft Docs — URL: <https://learn.microsoft.com/en-us/dotnet/api/system.identitymodel.tokens.jwt> (дата обращения: 30.05.2025).
29. Telegram bots против атак ботов // Хабр — URL: https://habr.com/ru/articles/749578/ (дата обращения: 30.05.2025).
30. Entity Framework Core PostgreSQL Provider // — URL: https://www.npgsql.org/efcore/index.html (дата обращения: 31.05.2025).
31. Принцип инверсии зависимостей (DIP) // Википедия — URL: [https://ru.wikipedia.org/wiki/Принцип\_инверсии\_зависимостей](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%D0%B8%D0%BD%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B8_%D0%B7%D0%B0%D0%B2%D0%B8%D1%81%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D0%B5%D0%B9) (дата обращения: 31.05.2025).
32. Программирование Telegram бота на C# // GitHub Guide — URL: <https://github.com/TelegramBots/Telegram.Bot> (дата обращения: 31.05.2025).

# ПРИЛОЖЕНИЕ А

Program

using task\_service.Infrastructure.Middlewares;

using task\_service.Infrastructure;

using task\_service.Application;

using task\_service.Domain;

using Microsoft.OpenApi.Models;

namespace task\_service.Presentation

{

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddApplication();

builder.Services.AddDomain();

builder.Services.AddInfrastructure(builder.Configuration);

builder.Services.AddControllers();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "Task Service API", Version = "v1" });

// Добавляем поддержку JWT в Swagger

c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

Description = "JWT Authorization header using the Bearer scheme. Example: \"Bearer {token}\"",

Name = "Authorization",

In = ParameterLocation.Header,

Type = SecuritySchemeType.ApiKey,

Scheme = "Bearer"

});

c.AddSecurityRequirement(new OpenApiSecurityRequirement

{

{

new OpenApiSecurityScheme

{

Reference = new OpenApiReference

{

Type = ReferenceType.SecurityScheme,

Id = "Bearer"

}

},

Array.Empty<string>()

}

});

});

var app = builder.Build();

app.UseMiddleware<ExceptionHandlingMiddleware>();

//if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

}

}

}

UserController

using task\_service.Application.DTOs;

using Microsoft.AspNetCore.Mvc;

using task\_service.Application.Interfaces.Services;

using task\_service.Application.Interfaces;

using Microsoft.AspNetCore.Authorization;

using System.Security.Claims;

namespace task\_service.Presentation.Controllers

{

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

[ApiController]

public class UserController : ControllerBase

{

private readonly IUserService \_userService;

private readonly ITokenService \_tokenService;

public UserController(IUserService userService, ITokenService tokenService)

{

\_userService = userService;

\_tokenService = tokenService;

}

[HttpPost("CreateUser")]

[ProducesResponseType(typeof(string), StatusCodes.Status200OK)]

[Produces("text/plain")]

public async Task<ActionResult> CreateUser(UserDTO userDTO)

{

var user = await \_userService.CreateUser(userDTO);

var token = \_tokenService.GenerateToken(userDTO);

return Content(token, "text/plain");

}

[Authorize]

[HttpGet("GetUser")]

[ProducesResponseType(typeof(UserDTO), StatusCodes.Status200OK)]

public async Task<ActionResult> GetUser()

{

var userDTO = new UserDTO

{

Id = User.FindFirstValue("client\_id"),

type\_id = User.FindFirstValue("client\_type"),

};

return Ok(userDTO);

}

}

}

UserAwardController

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using System.Security.Claims;

using task\_service.Application.DTOs;

using task\_service.Application.DTOs.AwardDTO;

using task\_service.Application.Interfaces.Services;

using task\_service.Domain.Entities;

namespace task\_service.Presentation.Controllers

{

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

[ApiController]

public class UserAwardController:ControllerBase

{

private readonly IUserAwardServices \_userAwardServices;

private readonly IUserService \_userService;

private readonly IAwardServices \_awardServices;

public UserAwardController(IUserAwardServices userAwardServices, IUserService userService,IAwardServices awardServices)

{

\_userAwardServices = userAwardServices;

\_userService = userService;

\_awardServices=awardServices;

}

[Authorize]

[HttpPost("PostUserAward")]

public async Task<IActionResult>PostUserAward(Guid idAward)

{

await \_userAwardServices.PutUserAward(await UserFromToken(), await \_awardServices.GetAward(idAward));

return Ok();

}

[Authorize]

[HttpGet("GetUserAward")]

[ProducesResponseType(typeof(ICollection<ReturnAwardDTO>), StatusCodes.Status200OK)]

public async Task<IActionResult> GetUserAward()

{

return Ok(await \_userAwardServices.GetUserAward(await UserFromToken()));

}

private async Task<User> UserFromToken()

{

var userDTO = new UserDTO

{

Name = "Костыль",

Id = User.FindFirstValue("client\_id"),

type\_id = User.FindFirstValue("client\_type"),

};

return await \_userService.GetUser(userDTO);

}

}

}

ToDoListController

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using System.Security.Claims;

using task\_service.Application.DTOs;

using task\_service.Application.DTOs.ToDoListDTO;

using task\_service.Application.Interfaces.Services;

using task\_service.Domain.Entities;

namespace task\_service.Presentation.Controllers

{

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

[ApiController]

public class ToDoListController : ControllerBase

{

private readonly IToDoListService \_toDoListService;

private readonly IUserService \_userService;

public ToDoListController(IToDoListService toDoListService, IUserService userService)

{

\_toDoListService = toDoListService;

\_userService = userService;

}

[Authorize]

[HttpPost("CreateTodolist")]

public async Task<IActionResult> CreateTo\_Do\_List(NewToDoListDTO toDoListDTO)

{

User user = await UserFromToken();

await \_toDoListService.CreateToDoList(toDoListDTO, user);

return Ok();

}

[Authorize]

[HttpGet("GetToDoLists")]

[ProducesResponseType(typeof(ICollection<ReturnToDoListsDTO>), StatusCodes.Status200OK)]

public async Task<IActionResult> GetToDoLists()

{

User user = await UserFromToken();

ICollection<ReturnToDoListsDTO> toDoLists = await \_toDoListService.GetToDoLists(user);

return Ok(toDoLists);

}

[Authorize]

[HttpDelete("DeleteToDoList")]

public async Task<IActionResult> DeleteToDoList(Guid idToDoList)

{

await \_toDoListService.DeleteToDoList(idToDoList);

return Ok();

}

[Authorize]

[HttpPut("PutToDoList")]

public async Task<IActionResult> PutToDoList(PutToDoListDTO putToDoListDTO)

{

await \_toDoListService.PutToDoList(putToDoListDTO);

return Ok();

}

private async Task<User> UserFromToken()

{

var userDTO = new UserDTO

{

Name = "Костыль",

Id = User.FindFirstValue("client\_id"),

type\_id = User.FindFirstValue("client\_type"),

};

return await \_userService.GetUser(userDTO);

}

}

}

CaseController

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using task\_service.Application.DTOs.CaseDTO;

using task\_service.Application.Interfaces.Services;

using task\_service.Domain.Entities;

namespace task\_service.Controllers

{

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

[ApiController]

public class CaseController:ControllerBase

{

private readonly ICaseService \_caseService;

private readonly IToDoListService \_toDoListService;

public CaseController(ICaseService caseService,IToDoListService toDoListService)

{

\_caseService = caseService;

\_toDoListService = toDoListService;

}

[Authorize]

[HttpPost("CreateCase")]

public async Task<IActionResult> CreateCase(NewCaseDTO newCaseDTO, Guid idToDoList)

{

ToDoList toDoList = await \_toDoListService.GetToDoList(idToDoList);

await \_caseService.CreateCase(newCaseDTO, toDoList);

return Ok();

}

[Authorize]

[HttpGet("GetCases")]

[ProducesResponseType(typeof(ICollection<ReturnCaseDTO>), StatusCodes.Status200OK)]

public async Task<IActionResult> GetCases(Guid idToDoList)

{

ToDoList toDoList = await \_toDoListService.GetToDoList(idToDoList);

ICollection<ReturnCaseDTO> returnCaseDTO = await \_caseService.GetCases(toDoList);

return Ok(returnCaseDTO);

}

[Authorize]

[HttpDelete("DeleteCase")]

public async Task<IActionResult> DeleteCase(Guid idCase)

{

await \_caseService.DeleteCase(idCase);

return Ok();

}

[Authorize]

[HttpPut("PutCase")]

public async Task<IActionResult> PutCase(PutCaseDTO putCase)

{

await \_caseService.PutCase(putCase);

return Ok();

}

}

}

AwardController

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using task\_service.Application.DTOs.AwardDTO;

using task\_service.Application.Interfaces.Services;

namespace task\_service.Presentation.Controllers

{

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

[ApiController]

public class AwardController:ControllerBase

{

private readonly IAwardServices \_awardServices;

public AwardController(IAwardServices awardServices)

{

\_awardServices = awardServices;

}

[Authorize]

[HttpGet("GetAwards")]

[ProducesResponseType(typeof(ICollection<ReturnAwardDTO>), StatusCodes.Status200OK)]

public async Task<IActionResult> GetAwards()

{

return Ok(await \_awardServices.GetAwards());

}

}

}

AuthController

using Microsoft.AspNetCore.Mvc;

using task\_service.Application.DTOs;

using task\_service.Application.Interfaces;

using task\_service.Application.Interfaces.Services;

namespace ttask\_service.Presentation.Controllers

{

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

[ApiController]

public class AuthController: ControllerBase

{

private ITokenService \_tokenService;

private IUserService \_userService;

public AuthController(ITokenService tokenService,IUserService userService)

{

\_tokenService = tokenService;

\_userService = userService;

}

[HttpPost("authorization")]

[ProducesResponseType(typeof(string), StatusCodes.Status200OK)]

[Produces("text/plain")]

public async Task<IActionResult> authorization(UserDTO authorizationDTO )

{

await \_userService.GetUser(authorizationDTO);

var token = \_tokenService.GenerateToken(authorizationDTO);

return Content(token, "text/plain");

}

}

}

UserService

using task\_service.Application.Interfaces.Services;

using task\_service.Domain.Interfaces.IRepository;

using task\_service.Application.DTOs;

using task\_service.Application.Validators;

using task\_service.Domain.Entities;

using task\_service.Application.Interfaces;

namespace task\_service.Application.Services

{

public class UserService : IUserService

{

private readonly IUserRepository \_userRepository;

private readonly IIdClientRepository \_idClientRepository;

private readonly IClientTypeRepository \_clientTypeRepository;

private readonly ITokenService \_tokenService;

public UserService(

IUserRepository userRepository,

IIdClientRepository idClientRepository,

IClientTypeRepository clientTypeRepository,

ITokenService tokenService)

{

\_userRepository = userRepository;

\_idClientRepository = idClientRepository;

\_clientTypeRepository = clientTypeRepository;

\_tokenService = tokenService;

}

public async Task<User> CreateUser(UserDTO userDTO)

{

if (DtoValidator.HasEmptyValues(userDTO))

throw new ArgumentException("При создании пользователя поступили пустые данные");

try

{

return await GetUser(userDTO);

}

catch{ }

var user = CreateNewUser(userDTO);

var clientType = await \_clientTypeRepository.GetByTypeAsync(userDTO.type\_id)

?? throw new KeyNotFoundException("Тип клиента не найден");

await \_userRepository.AddAsync(user);

var idClient = new IdClient

{

IdClient1 = userDTO.Id,

IdUser = user.Id,

IdClientType = clientType.Id

};

await \_idClientRepository.AddAsync(idClient);

return user;

}

public async Task<User> GetUser(UserDTO userDTO)

{

if (DtoValidator.HasEmptyValues(userDTO))

throw new ArgumentException("При авторизации пользователя поступили пустые данные");

ClientType clientType = await \_clientTypeRepository.GetByTypeAsync(userDTO.type\_id)

?? throw new KeyNotFoundException("Тип клиента не найден");

IdClient idClient = await \_idClientRepository.GetAsync(clientType.Id,userDTO.Id)

?? throw new KeyNotFoundException("id типа клиента не найден");

User user = await \_userRepository.GetAsync(idClient.IdUser)

?? throw new KeyNotFoundException("Пользователь не найден");

return user;

}

private User CreateNewUser(UserDTO userDTO)

{

return new User { Name = userDTO.Name };

}

}

}

UserAwardServices

using task\_service.Application.DTOs.AwardDTO;

using task\_service.Application.Interfaces.Services;

using task\_service.Domain.Entities;

using task\_service.Domain.Interfaces.IRepository;

namespace task\_service.Application.Services

{

public class UserAwardServices : IUserAwardServices

{

private readonly IUserAwardRepository \_userRepository;

public UserAwardServices(IUserAwardRepository userRepository)

{

\_userRepository = userRepository;

}

public async Task<ICollection<ReturnAwardDTO>> GetUserAward(User user)

{

ICollection<ReturnAwardDTO> returnAwardDTOs = new List<ReturnAwardDTO>();

foreach (Award award in await \_userRepository.getAwardsAsync(user))

{

returnAwardDTOs.Add(new ReturnAwardDTO

{

id = award.Id,

name = award.Name

});

}

return returnAwardDTOs;

}

public async Task PutUserAward(User user, Award award)

{

await \_userRepository.PutAsync(user, award);

}

}

}

ToDoListService

using task\_service.Application.DTOs.ToDoListDTO;

using task\_service.Application.Interfaces.Services;

using task\_service.Application.Validators;

using task\_service.Domain.Entities;

using task\_service.Domain.Interfaces.IRepository;

namespace task\_service.Application.Services

{

internal class ToDoListService : IToDoListService

{

private readonly IToDoListRepository \_toDoListRepository;

private readonly ICaseRepository \_caseRepository;

public ToDoListService(IToDoListRepository toDoListRepository,ICaseRepository caseRepository)

{

\_toDoListRepository = toDoListRepository;

\_caseRepository = caseRepository;

}

public async Task<ToDoList> CreateToDoList(NewToDoListDTO toDoListDTO, User user)

{

if (DtoValidator.HasEmptyValues(toDoListDTO))

throw new ArgumentException("Списка дел поступили пустые данне");

ToDoList toDoList = CreateNewToDoList(toDoListDTO);

toDoList.IdUser = user.Id;

toDoList.IdUserNavigation = user;

await \_toDoListRepository.AddAsync(toDoList);

return toDoList;

}

public async Task DeleteToDoList(Guid id)

{

ToDoList toDoList = await \_toDoListRepository.GetAsync(id);

foreach (Case c in toDoList.cases)

\_toDoListRepository.DeleteAsync(c.Id);

await \_toDoListRepository.DeleteAsync(id);

}

public async Task<ICollection<ReturnToDoListsDTO>> GetToDoLists(User user)

{

ICollection<ReturnToDoListsDTO> returnToDoListsDTO = new List<ReturnToDoListsDTO>();

foreach (var toDoList in user.ToDoLists)

{

returnToDoListsDTO.Add(new ReturnToDoListsDTO{

Id = toDoList.Id.ToString(),

Name = toDoList.Name

});

}

return returnToDoListsDTO;

}

public async Task PutToDoList(PutToDoListDTO putToDoListDTO)

{

ToDoList toDoList = await \_toDoListRepository.GetAsync(putToDoListDTO.Id);

toDoList.Name = putToDoListDTO.Name;

await \_toDoListRepository.PutAsync(toDoList);

}

public async Task<ToDoList> GetToDoList(Guid id)

{

return await \_toDoListRepository.GetAsync(id);

}

private ToDoList CreateNewToDoList(NewToDoListDTO toDoListDTO)

{

return new ToDoList { Name = toDoListDTO.Name};

}

}

}

CaseService

using task\_service.Application.DTOs.CaseDTO;

using task\_service.Application.Interfaces.Services;

using task\_service.Application.Validators;

using task\_service.Domain.Entities;

using task\_service.Domain.Interfaces.IRepository;

namespace task\_service.Application.Services

{

public class CaseService : ICaseService

{

private readonly ICaseRepository \_caseRepository;

public CaseService(ICaseRepository caseRepository)

{

\_caseRepository = caseRepository;

}

public async Task CreateCase(NewCaseDTO caseDTO, ToDoList toDoList)

{

if (DtoValidator.HasEmptyValues(caseDTO))

throw new ArgumentException("дело поступило пустым");

Case @case = CreateCase(caseDTO);

@case.id\_to\_do\_list = toDoList.Id;

@case.id\_to\_do\_listNavigation = toDoList;

await \_caseRepository.AddAsync(@case);

}

public async Task DeleteCase(Guid id)

{

await \_caseRepository.DeleteAsync(id);

}

public async Task<ICollection<ReturnCaseDTO>> GetCases(ToDoList toDoList)

{

ICollection<ReturnCaseDTO> returnCaseDTOs = new List<ReturnCaseDTO>();

foreach (var @case in toDoList.cases)

{

returnCaseDTOs.Add(new ReturnCaseDTO {

Id = @case.Id,

DateEnd=@case.DateEnd,

Name=@case.Name,

Status=@case.Status,

DateOfCreation=@case.DateOfCreation,

});

}

return returnCaseDTOs;

}

public async Task PutCase(PutCaseDTO PutCaseDTO)

{

Case @case = await \_caseRepository.GetAsync(PutCaseDTO.id);

@case.Name = PutCaseDTO.Name;

@case.Status = PutCaseDTO.Status;

@case.DateEnd = PutCaseDTO.DateEnd;

await \_caseRepository.PutAsync(@case);

}

private Case CreateCase(NewCaseDTO newCaseDTO)

{

return new Case

{

Name = newCaseDTO.Name,

Status = false,

DateOfCreation = newCaseDTO.DateOfCreation,

DateEnd = newCaseDTO.DateEnd,

};

}

}

}

AwardServices

using task\_service.Application.DTOs.AwardDTO;

using task\_service.Application.Interfaces.Services;

using task\_service.Domain.Entities;

using task\_service.Domain.Interfaces.IRepository;

namespace task\_service.Application.Services

{

public class AwardServices : IAwardServices

{

private readonly IAwardRepository \_awardRepository;

public AwardServices(IAwardRepository awardRepository)

{

\_awardRepository = awardRepository;

}

public async Task<Award> GetAward(Guid id)

{

return await \_awardRepository.GetAsync(id);

}

public async Task<ICollection<ReturnAwardDTO>> GetAwards()

{

ICollection<Award> Awards = await \_awardRepository.GetAllAsync();

ICollection<ReturnAwardDTO> returnAwardDTOs = new List<ReturnAwardDTO>();

foreach (Award award in Awards)

{

returnAwardDTOs.Add(new ReturnAwardDTO

{

name = award.Name,

id = award.Id,

});

}

return returnAwardDTOs;

}

}

}

JwtService

using Microsoft.Extensions.Options;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using task\_service.Application.DTOs;

using task\_service.Application.Interfaces;

namespace task\_service.Infrastructure.JWT

{

public class JwtService : ITokenService

{

private readonly JwtSettings \_settings;

public JwtService(IOptions<JwtSettings> settings) => \_settings = settings.Value;

public string GenerateToken(UserDTO userDTO)

{

var claims = new List<Claim>

{

new("client\_id", userDTO.Id),

new("client\_type", userDTO.type\_id)

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_settings.Secret));

var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(

issuer: \_settings.Issuer,

audience: \_settings.Audience,

claims: claims,

expires: DateTime.UtcNow.AddMinutes(\_settings.ExpiryMinutes),

signingCredentials: credentials

);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

}

ExceptionHandlingMiddleware

using Microsoft.Extensions.Logging;

using System.Net;

using task\_service.Application.DTOs;

using Microsoft.AspNetCore.Http;

namespace task\_service.Infrastructure.Middlewares

{

public class ExceptionHandlingMiddleware

{

private readonly RequestDelegate \_next;

private readonly ILogger<ExceptionHandlingMiddleware> \_logger;

public ExceptionHandlingMiddleware(RequestDelegate next, ILogger<ExceptionHandlingMiddleware> logger)

{

\_next = next;

\_logger = logger;

}

public async Task InvokeAsync(HttpContext httpContext)

{

try

{

await \_next(httpContext);

}

catch (

Exception e)

{

await HandleExceptionAsync(httpContext, e.Message, HttpStatusCode.NotFound, "что то пошло не так");

}

}

private async Task HandleExceptionAsync(HttpContext httpContext, string exMsg, HttpStatusCode httpStatusCode, string message)

{

\_logger.LogError(exMsg);

HttpResponse response = httpContext.Response;

response.ContentType = "application/json";

response.StatusCode = (int)httpStatusCode;

ErrorDTO errorDTO = new ErrorDTO()

{

Message = message,

StatusCode = (int)httpStatusCode

};

string result = errorDTO.ToString();

await response.WriteAsync(result);

}

}

}

DtoValidator

namespace task\_service.Application.Validators

{

using System;

using System.Reflection;

public static class DtoValidator

{

public static bool HasEmptyValues(object dto)

{

if (dto == null) return true;

PropertyInfo[] properties = dto.GetType().GetProperties();

foreach (PropertyInfo property in properties)

{

object value = property.GetValue(dto);

if (property.PropertyType == typeof(string))

{

if (string.IsNullOrWhiteSpace((string)value)) return true;

}

else

if (IsDefaultValue(value, property.PropertyType))

{

return true;

}

}

return false;

}

private static bool IsDefaultValue(object value, Type type)

{

return type.IsValueType

? value.Equals(Activator.CreateInstance(type))

: value == null;

}

}

}

Program

using Microsoft.Extensions.DependencyInjection;

using TaskSlayerfrontendTGBot;

using TaskSlayerfrontendTGBot.ApiClient;

using TaskSlayerfrontendTGBot.BotHandles.Handles;

using TaskSlayerfrontendTGBot.BotHandles.IHandles;

using TaskSlayerfrontendTGBot.Interface.IServices;

using TaskSlayerfrontendTGBot.Services;

using Telegram.Bot;

class Program

{

static async Task Main(string[] args)

{

var serviceProvider = new ServiceCollection()

.AddScoped<App>()

.AddScoped<IBotServices,BotService>()

.AddSingleton<IUserStateService, UserStateService>()

.AddScoped<IHandleUpdate,HandleUpdate>()

.AddScoped<IHandelCommand,HandelCommand>()

.AddScoped<IHandleError,HandleError>()

.AddScoped<IHandelCallbackQuery,HandelCallbackQuery>()

.AddSingleton<IHandleUserState,HandleUserState>()

.AddScoped<TaskServiceApiClient>(\_ => new TaskServiceApiClient(Environment.GetEnvironmentVariable("base\_\_Url"),new HttpClient()))

.AddSingleton<ITelegramBotClient>(\_ => new TelegramBotClient(Environment.GetEnvironmentVariable("Bot\_\_token")))

.BuildServiceProvider();

var app = serviceProvider.GetService<App>();

await app.Run();

Console.ReadLine();

}

}

BotService

using TaskSlayerfrontendTGBot.BotHandles.IHandles;

using TaskSlayerfrontendTGBot.Interface.IServices;

using Telegram.Bot;

namespace TaskSlayerfrontendTGBot.Services

{

internal class BotService: IBotServices

{

private readonly ITelegramBotClient \_botClient;

private readonly IHandleUpdate \_handelUpdate;

private readonly IHandleError \_handleError;

public BotService(ITelegramBotClient botClient, IHandleUpdate handelUpdate,IHandleError handleError)

{

\_botClient = botClient;

\_handelUpdate = handelUpdate;

\_handleError = handleError;

}

public async Task Start()

{

\_botClient.StartReceiving(

updateHandler: \_handelUpdate.HandleUpdateAsync,

errorHandler: \_handleError.HandleErrorAsync

);

}

}

}

UserStateService

using TaskSlayerfrontendTGBot.Interface.IServices;

namespace TaskSlayerfrontendTGBot.Services

{

public class UserStateService : IUserStateService

{

private readonly Dictionary<long, string> \_states = new();

public void SetState(long userId, string state)

{

\_states[userId] = state;

}

public string? GetState(long userId)

{

return \_states.TryGetValue(userId, out var state) ? state : null;

}

public void ClearState(long userId)

{

\_states.Remove(userId);

}

}

}

TaskSlayerfrontendTGBot

using Newtonsoft.Json;

using System.Globalization;

using System.Net.Http.Headers;

using TaskSlayerfrontendTGBot.ApiClient;

using TaskSlayerfrontendTGBot.BotHandles.IHandles;

using TaskSlayerfrontendTGBot.Interface.IServices;

using Telegram.Bot;

using Telegram.Bot.Types;

namespace TaskSlayerfrontendTGBot.BotHandles.Handles

{

internal class HandleUserState : IHandleUserState

{

IUserStateService \_userStateService;

private readonly TaskServiceApiClient \_Api;

public HandleUserState(IUserStateService userStateService, TaskServiceApiClient Api)

{

\_userStateService = userStateService;

\_Api = Api;

}

public async Task HandleUserStateAsync(ITelegramBotClient bot, Update update, CancellationToken cancellationToken)

{

var message = update.Message;

var chatId = message.Chat.Id;

var user = message.From;

var state = \_userStateService.GetState(chatId);

var parts = state.Split(':');

string leftPart = parts[0];

string idTarget = parts[1];

var segments = leftPart.Split('/');

string action = segments[0];

string obj = segments[1];

string property = segments[2];

var userDTO = new UserDTO

{

Name = "Костыль",

Id = user.Id.ToString(),

Type\_id = "Telegram"

};

\_Api.\_httpClient.DefaultRequestHeaders.Authorization =

new AuthenticationHeaderValue("Bearer", await \_Api.AuthorizationAsync(userDTO));

switch (obj)

{

case "ToDoList":

switch(action)

{

case "change":

string userMessagehange = update.Message.Text;

PutToDoListDTO putToDoListDTOChange = new PutToDoListDTO

{

Name = userMessagehange,

Id = Guid.Parse(idTarget)

};

await \_Api.PutToDoListAsync(putToDoListDTOChange);

await bot.SendMessage(chatId, "✅ " + "Список успешно изменён", Telegram.Bot.Types.Enums.ParseMode.Html);

break;

case "add":

string userMessageAdd = update.Message.Text;

NewToDoListDTO newToDoListDTOAdd = new NewToDoListDTO

{

Name = userMessageAdd,

};

await \_Api.CreateTodolistAsync(newToDoListDTOAdd);

await bot.SendMessage(chatId, "✅ " + "Список успешно создан", Telegram.Bot.Types.Enums.ParseMode.Html);

break;

}

break;

case "Case":

if (segments.Length == 3)

{

await bot.SendMessage(

chatId,

"🗓️ <b>Когда ты хочешь завершить задачу?</b>\n✍️ Введи дату в формате <i>ДД.ММ.ГГГГ</i>",

Telegram.Bot.Types.Enums.ParseMode.Html

);

\_userStateService.ClearState(chatId);

\_userStateService.SetState(chatId, $"add/Case/name/"+ update.Message.Text + $":{idTarget}");

return;

}

else

{

if (DateTime.TryParseExact(update.Message.Text, "dd.MM.yyyy", CultureInfo.InvariantCulture, DateTimeStyles.AssumeUniversal, out DateTime parsedDate))

{

var dateEnd = new DateTimeOffset(parsedDate, TimeSpan.Zero); // гарантированно UTC

NewCaseDTO newCaseDTO = new NewCaseDTO

{

Name = segments[3],

Id\_to\_do\_list = Guid.Parse(idTarget),

DateEnd = dateEnd,

DateOfCreation = DateTimeOffset.UtcNow

};

await \_Api.CreateCaseAsync(Guid.Parse(idTarget), newCaseDTO);

await bot.SendMessage(chatId, "✅ " + "Задача успешно создана", Telegram.Bot.Types.Enums.ParseMode.Html);

}

else

{

await bot.SendMessage(chatId, "<b>Не балуйся</b>", Telegram.Bot.Types.Enums.ParseMode.Html);

}

}

break;

}

\_userStateService.ClearState(chatId);

}

}

}

HandleUpdate

using TaskSlayerfrontendTGBot.BotHandles.IHandles;

using TaskSlayerfrontendTGBot.Interface.IServices;

using Telegram.Bot;

using Telegram.Bot.Types;

using Telegram.Bot.Types.Enums;

namespace TaskSlayerfrontendTGBot.BotHandles.Handles

{

internal class HandleUpdate:IHandleUpdate

{

private readonly IHandelCommand \_handelCommand;

private readonly IHandelCallbackQuery \_handelCallbackQuery;

private readonly IUserStateService \_userStateService;

private readonly IHandleUserState \_handleUserState;

public HandleUpdate(IHandelCommand handelCommand, IHandelCallbackQuery handelCallbackQuery, IUserStateService userStateService, IHandleUserState handleUserState)

{

\_handelCommand = handelCommand;

\_handelCallbackQuery = handelCallbackQuery;

\_userStateService = userStateService;

\_handleUserState = handleUserState;

}

public async Task HandleUpdateAsync(ITelegramBotClient bot, Update update, CancellationToken cancellationToken)

{

if (update.Type == UpdateType.CallbackQuery && update.CallbackQuery != null)

{

if (\_userStateService.GetState(update.CallbackQuery.Message.Chat.Id) == null)

{

await \_handelCallbackQuery.HandelCallbackQueryAsync(bot, update, cancellationToken);

}

else

{

await bot.SendMessage(update.CallbackQuery.Message.Chat.Id, "<b>Не балуйся</b>", Telegram.Bot.Types.Enums.ParseMode.Html);

\_userStateService.ClearState(update.CallbackQuery.Message.Chat.Id);

}

return;

}

var message = update.Message;

var chatId = message.Chat.Id;

var user = message.From;

Console.WriteLine($"[{user?.Id}] {user?.FirstName}: {message.Text ?? message.Type.ToString()}");

var state = \_userStateService.GetState(chatId);

if (state != null)

{

await \_handleUserState.HandleUserStateAsync(bot, update, cancellationToken);

return;

}

if (message.Text == "/start")

{

await \_handelCommand.StartCommand(bot, update, cancellationToken);

return;

}

if (message.Text == "/To\_do\_list")

{

await \_handelCommand.To\_do\_listCommand(bot, update, cancellationToken);

return;

}

string welcomeText = $"/help — Не бойся просить помощи! 💙";

await bot.SendMessage(

chatId: chatId,

text: welcomeText,

cancellationToken: cancellationToken

);

return;

}

}

}

HandleError

using TaskSlayerfrontendTGBot.BotHandles.IHandles;

using Telegram.Bot;

using Telegram.Bot.Exceptions;

namespace TaskSlayerfrontendTGBot.BotHandles.Handles

{

internal class HandleError : IHandleError

{

public Task HandleErrorAsync(ITelegramBotClient bot, Exception exception, CancellationToken cancellationToken)

{

var errorMessage = exception switch

{

ApiRequestException apiRequestException => $"Telegram API Error:\n[{apiRequestException.ErrorCode}]\n{apiRequestException.Message}",

\_ => exception.ToString()

};

Console.WriteLine(errorMessage);

return Task.CompletedTask;

}

}

}

TaskSlayerfrontendTGBot

using System.Net.Http.Headers;

using TaskSlayerfrontendTGBot.ApiClient;

using TaskSlayerfrontendTGBot.BotHandles.IHandles;

using Telegram.Bot;

using Telegram.Bot.Types;

using Telegram.Bot.Types.ReplyMarkups;

namespace TaskSlayerfrontendTGBot.BotHandles.Handles

{

internal class HandelCommand : IHandelCommand

{

private readonly ITelegramBotClient \_botClient;

private readonly TaskServiceApiClient \_Api;

public HandelCommand(ITelegramBotClient botClient, TaskServiceApiClient Api)

{

\_botClient = botClient;

\_Api = Api;

}

public async Task StartCommand(ITelegramBotClient bot, Update update, CancellationToken cancellationToken)

{

var message = update.Message;

var chatId = message.Chat.Id;

var user = message.From;

string welcomeText = $"Привет, {user?.FirstName ?? "друг"}! 👋\n" +

$"🎮 Добро пожаловать в TaskSlayer — твой игровой Todo-лист!\n\n" +

$"Ты — герой в мире задач, и каждый чекбокс — это победа над хаосом! 🌟\n\n" +

$"📜 Как это работает?\n\n" +

$"Добавляй задачи, как квесты.\n\n" +

$"Отмечай выполненные задачи — получай опыт и уровни.\n\n" +

$"Зарабатывай достижения и сражайся с прокрастинацией!\n\n" +

$"⚔️ Твои инструменты:\n" +

$"✅ /To\_do\_list — посмотреть списки\n" +

$"📋 /list — показать активные задания\n" +

$"🏆 /stats — твой прогресс и награды\n" +

$"🎯 /goals — долгосрочные миссии\n" +

$"💙/help — Не бойся просить помощи!\n\n" +

$"🔥 Готов стать легендой продуктивности? Тогда в бой!";

var replyKeyboard = new ReplyKeyboardMarkup(new[]

{

new KeyboardButton[] { "/To\_do\_list", "Кнопка 2" },

new KeyboardButton[] { "Кнопка 3" }

})

{

ResizeKeyboard = true

};

await bot.SendMessage(

chatId: chatId,

text: welcomeText,

cancellationToken: cancellationToken,

replyMarkup: replyKeyboard

);

var userDTO = new UserDTO

{

Name = user.Username,

Id = user.Id.ToString(),

Type\_id = "Telegram"

};

Console.WriteLine($"Создание иди получение существующего пользователя\n Name = {user.Username}\nId = {user.Id.ToString()}\nType\_id = \"Telegram\"");

await \_Api.CreateUserAsync(userDTO);

}

public async Task To\_do\_listCommand(ITelegramBotClient bot, Update update, CancellationToken cancellationToken)

{

var message = update.Message;

var chatId = message.Chat.Id;

var user = message.From;

string Text = "✨📋 Твои списки задач 📋✨";

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("Добавить список", "add/ToDoList/"+user.Id.ToString()+":"+""),

}

});

await bot.SendMessage(

chatId: chatId,

text: Text,

cancellationToken: cancellationToken,

replyMarkup: inlineKeyboard

);

var userDTO = new UserDTO

{

Name = user.Username,

Id = user.Id.ToString(),

Type\_id = "Telegram"

};

\_Api.\_httpClient.DefaultRequestHeaders.Authorization =

new AuthenticationHeaderValue("Bearer", await \_Api.AuthorizationAsync(userDTO));

ICollection<ReturnToDoListsDTO> ToDoListsDTOs = await \_Api.GetToDoListsAsync();

if (ToDoListsDTOs.Count == 0)

{

await bot.SendMessage(

chatId: chatId,

text: "🚫 списков нет…\n✨ <i>Но никогда не поздно начать!</i>\n➕ Добавь первый прямо сейчас!",

cancellationToken: cancellationToken,

parseMode: Telegram.Bot.Types.Enums.ParseMode.Html

);

}

foreach ( var toDo in ToDoListsDTOs )

{

inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("Просмотреть", "view/ToDoList/"+user.Id.ToString()+":"+toDo.Id),

InlineKeyboardButton.WithCallbackData("Изменить", "change/ToDoList/"+user.Id.ToString()+":"+toDo.Id),

InlineKeyboardButton.WithCallbackData("Удалить", "delete/ToDoList/"+user.Id.ToString()+":"+toDo.Id)

}

});

Text = "📝 " + toDo.Name;

await bot.SendMessage(

chatId: chatId,

text: Text,

cancellationToken: cancellationToken,

replyMarkup: inlineKeyboard

);

}

}

}

}

HandelCallbackQuery

using System.Net.Http.Headers;

using TaskSlayerfrontendTGBot.ApiClient;

using TaskSlayerfrontendTGBot.BotHandles.IHandles;

using TaskSlayerfrontendTGBot.Interface.IServices;

using Telegram.Bot;

using Telegram.Bot.Types;

using Telegram.Bot.Types.ReplyMarkups;

using static System.Net.Mime.MediaTypeNames;

namespace TaskSlayerfrontendTGBot.BotHandles.Handles

{

internal class HandelCallbackQuery : IHandelCallbackQuery

{

private readonly ITelegramBotClient \_botClient;

private readonly TaskServiceApiClient \_Api;

private readonly IUserStateService \_userStateService;

public HandelCallbackQuery(ITelegramBotClient botClient, TaskServiceApiClient Api, IUserStateService userStateService)

{

\_botClient = botClient;

\_Api = Api;

\_userStateService = userStateService;

}

public async Task HandelCallbackQueryAsync(ITelegramBotClient bot, Update update, CancellationToken cancellationToken)

{

var callbackData = update.CallbackQuery.Data;

var callbackQueryId = update.CallbackQuery.Id;

var chatCallbackQueryId = update.CallbackQuery.Message.Chat.Id;

int firstSlash = callbackData.IndexOf('/');

int secondSlash = callbackData.IndexOf('/', firstSlash + 1);

int colonIndex = callbackData.IndexOf(':');

string action = callbackData.Substring(0, firstSlash);

string obj = callbackData.Substring(firstSlash + 1, secondSlash - firstSlash - 1);

string idObj = callbackData.Substring(secondSlash + 1, colonIndex - secondSlash - 1);

string idTarget = callbackData.Substring(colonIndex + 1);

var userDTO = new UserDTO

{

Name = "Костыль",

Id = idObj,

Type\_id = "Telegram"

};

\_Api.\_httpClient.DefaultRequestHeaders.Authorization =

new AuthenticationHeaderValue("Bearer", await \_Api.AuthorizationAsync(userDTO));

switch (action)

{

case "delete":

switch (obj)

{

case "ToDoList":

await \_Api.DeleteToDoListAsync(Guid.Parse(idTarget));

await bot.SendMessage(chatCallbackQueryId, "✅ " + "Список успешно удалён");

break;

case "Case":

await \_Api.DeleteCaseAsync(Guid.Parse(idTarget));

await bot.SendMessage(chatCallbackQueryId, "✅ " + "Задача успешно удалена");

break;

}

break;

case "view":

switch (obj)

{

case "ToDoList":

ICollection<ReturnCaseDTO> CaseDTOs = await \_Api.GetCasesAsync(Guid.Parse(idTarget));

var inlineKeyboard = new Telegram.Bot.Types.ReplyMarkups.InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("Добавить цель", "add/Case/"+idObj.ToString()+":"+idTarget),

}

});

await bot.SendMessage(

chatId: chatCallbackQueryId,

text: "🚀 <b>Твои цели</b>",

cancellationToken: cancellationToken,

replyMarkup: inlineKeyboard,

parseMode: Telegram.Bot.Types.Enums.ParseMode.Html

);

bool thereTasks = false;

foreach (var CaseDTO in CaseDTOs)

{

thereTasks = true;

inlineKeyboard = new Telegram.Bot.Types.ReplyMarkups.InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("Изменить", "change/Case/"+idObj.ToString()+":"+CaseDTO.Id),

InlineKeyboardButton.WithCallbackData("Удалить", "delete/Case/"+idObj.ToString()+":"+CaseDTO.Id)

}

});

string Text = $"📝 <b>{CaseDTO.Name}</b>\n" +

$"📅 До: <i>{CaseDTO.DateEnd:dd.MM.yyyy}</i>\n" +

$"Статус: {(CaseDTO.Status ? "✅ Выполнено" : "❌ Не выполнено")}";

await bot.SendMessage(

chatId: chatCallbackQueryId,

text: Text,

cancellationToken: cancellationToken,

replyMarkup: inlineKeyboard,

parseMode: Telegram.Bot.Types.Enums.ParseMode.Html

);

}

if (!thereTasks)

{

await bot.SendMessage(

chatId: chatCallbackQueryId,

text: "🚫 Целей нет…\n✨ <i>Но никогда не поздно начать!</i>\n➕ Добавь первую прямо сейчас!",

cancellationToken: cancellationToken,

parseMode: Telegram.Bot.Types.Enums.ParseMode.Html

);

}

break;

}

break;

case "change":

switch (obj)

{

case "ToDoList":

await bot.SendMessage(chatCallbackQueryId, "📋 <b>Как назвать этот список?</b>\n✏️ Введите новое имя ниже:", Telegram.Bot.Types.Enums.ParseMode.Html);

\_userStateService.SetState(chatCallbackQueryId, $"change/ToDoList/name:{idTarget}");

break;

}

break;

case "add":

switch(obj)

{

case "ToDoList":

await bot.SendMessage(chatCallbackQueryId, "📋 <b>Как назвать этот список?</b>\n✏️ Введите название ниже:", Telegram.Bot.Types.Enums.ParseMode.Html);

\_userStateService.SetState(chatCallbackQueryId, $"add/ToDoList/name:{idTarget}");

break;

case "Case":

await bot.SendMessage(chatCallbackQueryId, "📋 <b>Как назвать этоу задачу?</b>\n✏️ Введите название ниже:", Telegram.Bot.Types.Enums.ParseMode.Html);

\_userStateService.SetState(chatCallbackQueryId, $"add/Case/name:{idTarget}");

break;

}

break;

}

}

}

}

Git log

Дмитрий@DESKTOP-HM6L3DO MINGW64 ~/Desktop/Diploma-project (main)

$ git log

commit b342782bfc3edfacb378d39cc8e423358ef9e14a (**HEAD** -> **main**, **origin/main**, **orig**

**in/HEAD**)

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Fri Jun 6 13:20:00 2025 +0300

Реализация тг бота

commit 478ece091957c465c9d847907a2e1b0704526e4b

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Fri Jun 6 13:19:22 2025 +0300

Написана часть диплома

commit a7a1e84b9384d2d7be172692437699317abc3e27

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Thu Jun 5 21:05:39 2025 +0300

Реализовано добавление нового пользователя в клиенте

commit c3e6e61896c87fe3bc3a2c0f6a42fb40779fdaf8

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Thu Jun 5 21:02:53 2025 +0300

Исправлена ошибка с возвращаемым типом стринг

commit ec837d6205ca90f445b7e6a4a26e2a1f2d2a71fc

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Thu Jun 5 14:33:09 2025 +0300

Всем методам контроллеров которые возвращают что то прописан тип возвращаемо

го объекта

commit b22759ad75fbe4becc6589d75f38faae9a530bb4

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Thu Jun 5 10:25:29 2025 +0300

Добавлен проект frontend TGBot

commit 43c309d96cebb880f701f5699bc55d5bf0fccf1e

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Thu Jun 5 10:12:48 2025 +0300

backend работает

commit c972ad0b01f3c1d463f8a5786f6b526278f1e327

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Thu Jun 5 10:12:27 2025 +0300

backend работает

commit 6ab79ec21074415efbd02f08640c82f5225149a9

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Wed Jun 4 18:16:07 2025 +0300

Реализован UserAwardController

commit 59396215efb4de5db88855c22a68b9f4b74d82ba

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Wed Jun 4 15:36:57 2025 +0300

реализован award контроллер

commit 5fdc8e9163d1938c00b51935258a7d01860434c4

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Wed Jun 4 14:29:51 2025 +0300

Реализован case контроллер

commit b91a2b6df5e70edb2f83bf5af191b2508ff822d8

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Tue Jun 3 22:49:22 2025 +0300

Реализован ToDoListController

commit e8025ab2e0c771ed311a8afa388c7e882018062e

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Tue Jun 3 18:18:06 2025 +0300

Промежуточное сохранение

commit f766878e229a09ae6cb07bdb71f56ea0bffc3ece

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Tue Jun 3 12:44:00 2025 +0300

Добавлена авторизация(Получения токена). Добавлено получения пользователя по

токену.

commit 83cf408a66b09128892754a8d254c8905b6b61c0

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Mon Jun 2 12:03:24 2025 +0300

добавлены и настроены jwt токены

commit 217f7b49d16cc3a615eb8ffd447578bade2382ad

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Mon Jun 2 10:18:59 2025 +0300

Написана первая часть диплома. Вторая часть на половину. Добавлен титульник

и рамки

commit 221612151103c444fd7aedfe63574d42414e3625

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Mon May 26 21:17:22 2025 +0300

Начал писать первую чать

commit cff2e02d89625834432eb1d8f9eaab3552213868

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Sun May 25 21:32:00 2025 +0300

Исправлено введение

commit b08dac89895ffac695e0149a49dc499778f4b51b

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Sun May 25 20:37:49 2025 +0300

Добавлен файл Диплом. В нем написано введение

commit b555434dffacf3320036abbacfeba1f4ffc3f5c8

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Fri May 23 10:55:26 2025 +0300

Перемещены некоторые файлы и разгружен класс Program

commit 05ef116b150a33ddcc167ba7cebd6521b82520f3

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Fri May 23 10:54:13 2025 +0300

Перемещены некоторые файлы и разгружен класс Program

commit 287402dddd2473bf0c27ef571d78d8a19507c905

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Fri May 23 09:42:52 2025 +0300

Изменен план backeand

commit d565b49e6aad8d968c76ba51e4ee0984f40e7792

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Fri May 23 09:35:26 2025 +0300

Исправлен namespace в IdClientRepository

commit 552c12b3e9ab62eb7a758eb5ffb203edbb44cb3a

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Thu May 22 21:14:13 2025 +0300

Исправлена ошибка с получением строки подключения

commit 2fd46d4a756dae6c45b2a7b46567843f8bce43b2

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Thu May 22 20:00:26 2025 +0300

Проект разбит на слои по принципу DDD. Добавлены репозитории и переписан кла

сс UserService

commit 4277b447c128cf649f3040418db08f1fafb1c93a

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Thu May 22 15:05:22 2025 +0300

Исправлена ошибка с проверкой idClient

commit 6139d6be0423ede2b5f14a8e83607f6eb288d390

Author: Дмитрий <hohlovdmitrii@yandex.ru>

Date: Thu May 22 14:45:50 2025 +0300

Добавлена проверка DTO на Null

commit 92e0ca74a9cc5125b3fa3f036116b8da335f7dc4

Author: dimontururu <hohlovdmitrii@yandex.ru>

Date: Wed Feb 12 12:15:21 2025 +0300

ER Диаграмма

commit a32268bca431f3fe8975790da4d6ca5eb12cbf6b

Author: dimontururu <hohlovdmitrii@yandex.ru>

Date: Sun Feb 2 01:13:15 2025 +0300

отредактирован файл README

commit 53bc7170995c59861d8a21b0a75a04298f1f9b39

Author: dimontururu <hohlovdmitrii@yandex.ru>

Date: Sun Feb 2 00:58:52 2025 +0300

Создание backend приложения

commit 997b661292a18e2cbe5029d7b5289e8bcee3d1f9

Author: dimontururu <159140980+dimontururu@users.noreply.github.com>

Date: Sun Feb 2 00:50:36 2025 +0300