**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра «Информационных технологий»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К Проектному Практикуму

НА ТЕМУ: Разработка программного обеспечения

«Канбан Доска»

Студент Д.А. Штриков

Руководитель проектного практикума А. Мокин

**Москва 2025 г.**

|  |  |
| --- | --- |
|  |  |

СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc199460854)

[ВВЕДЕНИЕ 3](#_Toc199460855)

[1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ 4](#_Toc199460856)

[2. АРХИТЕКТУРА СИСТЕМЫ 6](#_Toc199460857)

[ЗАКЛЮЧЕНИЕ 15](#_Toc199460858)

# ВВЕДЕНИЕ

**Актуальность проекта**

**В условиях динамично развивающегося бизнес-ландшафта организации всё чаще сталкиваются с необходимостью оптимизации рабочих процессов. Канбан-методология, зародившаяся в производственной системе Toyota, доказала свою эффективность в различных сферах - от IT-разработки до маркетинговых кампаний. Визуализация workflow через Канбан-доску позволяет достичь сразу нескольких ключевых преимуществ:**

**- Повышение прозрачности всех этапов работы**

**- Оптимизация распределения ресурсов**

**- Своевременное выявление "узких мест"**

**- Улучшение командного взаимодействия**

**Целью данного проектного практикума является разработка программного обеспечения «Канбан Доска» для эффективного управления задачами в рамках рабочих процессов, таких как разработка ПО, организация проектов или управление командными заданиями. Для достижения этой цели поставлены следующие задачи:**

1. **Проектирование архитектуры системы: разработка структуры базы данных, API-эндпоинтов и пользовательских ролей.**
2. **Реализация функционала:**
   * **Создание CRUD-операций для управления проектами, колонками и задачами.**
   * **Внедрение JWT-аутентификации для безопасного доступа.**
   * **Оптимизация работы с базой данных (MS SQL Server) через SQLAlchemy.**

# 1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

**1.1 Техническое задание**

**Функциональные требования**

1. **Управление сущностями:**
   * **CRUD операций для проектов (досок)**
   * **CRUD операций для колонок**
   * **CRUD операций для задач**
   * **Управление пользователями проектов (добавление/удаление)**
   * **Логирование действий с задачами**
2. **Дополнительные возможности:**
   * **Получение списка задач по колонке с возможностью фильтрации**
   * **Получение списка всех проектов**
   * **Просмотр логов задач**
3. **Авторизация через JWT-токены**

**1.2 Нефункциональные требования**

* **Время отклика API ≤ 100 мс**
* **Поддержка 100+ одновременных подключений**

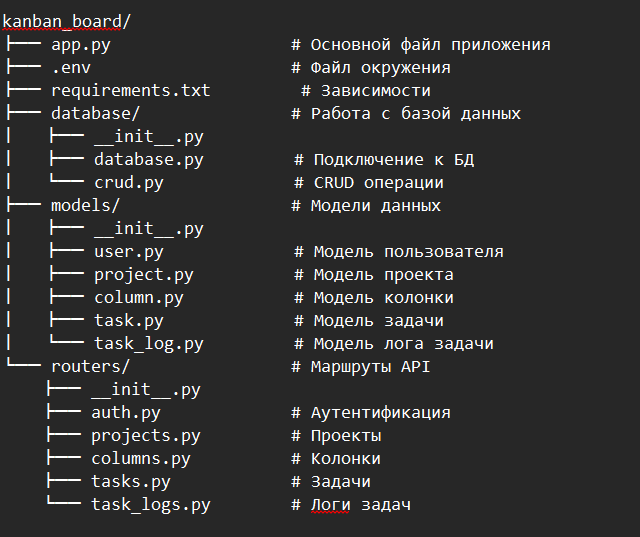
**1.2 Постановка задачи**

**1.Проект направлен на создание RESTful API для управления Kanban-доской, включая:**

* **CRUD-операции для ключевых сущностей**:
  + **Проекты (доски)** – создание, редактирование, удаление и просмотр досок.
  + **Колонки** – управление статусами задач (например, "To Do", "In Progress", "Done").
  + **Задачи** – добавление, перемещение между колонками, изменение приоритета, назначение исполнителей.
* **Безопасность и авторизация**:
  + **JWT-аутентификация** (JSON Web Tokens) для защиты API.
  + Регистрация и вход пользователей с валидацией данных.
  + Разграничение прав доступа (например, только автор задачи может её удалить).
* **Работа с базой данных**:
  + Использование **SQLAlchemy ORM** для взаимодействия с **MS SQL Server**.
  + Оптимизированные запросы для быстрого отклика API.
  + Поддержка транзакций для надежности операций.

# 2. АРХИТЕКТУРА СИСТЕМЫ

**2.1 Структура проекта**



**2.2 Технологический стек реализации**

**Для разработки цифрового решения была выбрана следующая технологическая база:**

**1. Python 3.10 - современная версия интерпретируемого языка программирования, обеспечивающая высокую производительность и богатый выбор библиотек для backend-разработки.**

**2. Flask 2.3 - микрофреймворк для создания веб-приложений, отличающийся простотой настройки и гибкостью архитектуры. Позволяет быстро развернуть базовый функционал при минимальных накладных расходах.**

**3. SQLAlchemy 1.4 - мощная ORM-библиотека, предоставляющая:**

**- Удобный интерфейс для работы с реляционными БД**

**- Механизм миграций**

**- Поддержку различных СУБД**

**- Оптимизированные запросы**

**4. Flask-RESTx 1.1 - расширение для Flask, которое:**

**- Упрощает создание RESTful API**

**- Автоматически генерирует Swagger-документацию**

**- Обеспечивает валидацию входных данных**

**5. PyODBC 4.0 - драйвер для подключения к SQL-серверам, поддерживающий:**

**- Работу с MS SQL Server, PostgreSQL, MySQL**

**- Транзакции**

**- Пулинг соединений**

**2.3 Схема базы данных**

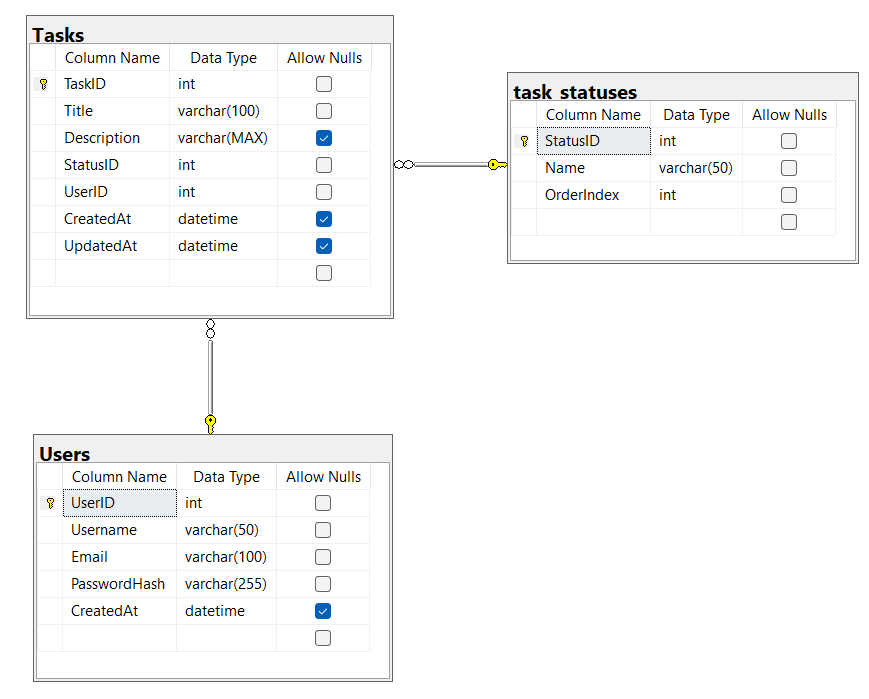


Рисунок 1 - Схема базы данных

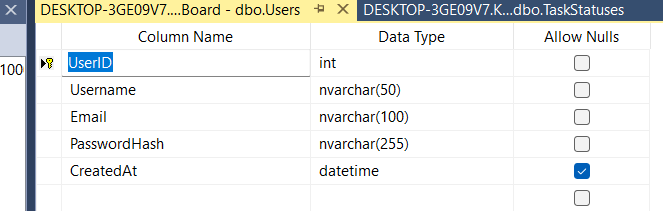
****

Рисунок 2 - Таблица Users (dbo.Users)

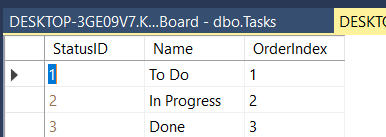
****

Рисунок 3 - Таблица Tasks (dbo.Tasks)

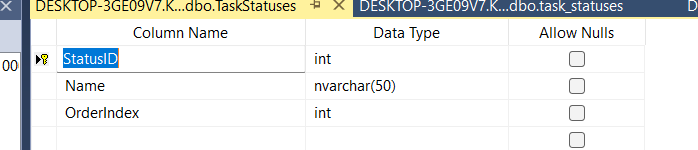


Рисунок 4 - Таблица TaskStatuses (dbo.TaskStatuses)

**Пример заполнения**

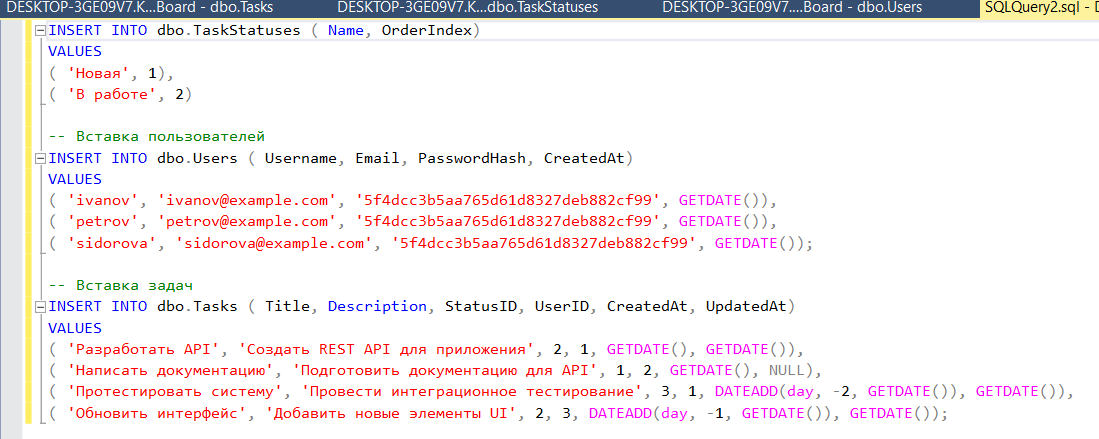
****

Рисунок 5 - Код

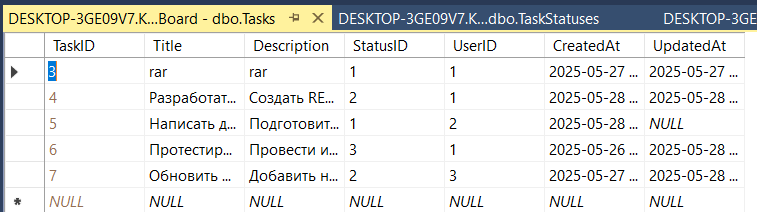
****

Рисунок 6 – Пример: Таблица Tasks (dbo.Tasks)

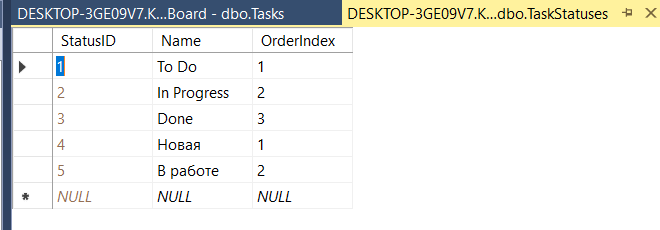
****

Рисунок 7 – Пример: Таблица TaskStatuses (dbo.TaskStatuses)

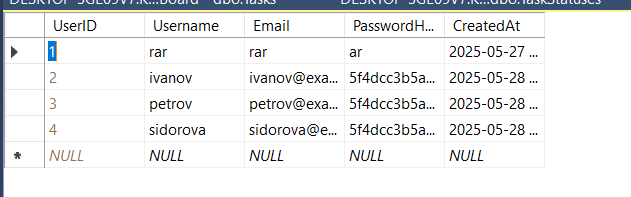
****

Рисунок 8 – Пример: Таблица Users (dbo.Users)

**2.3** **Requirements**

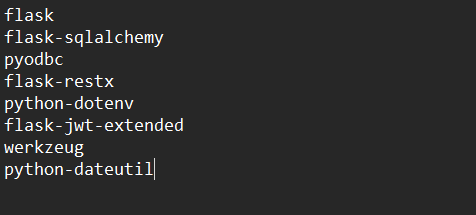


Рисунок 9 – requirements.txt

**1. Flask (основа приложения)**

**Почему выбрали**:

* Минималистичный фреймворк, идеально подходит для создания API
* Гибкость в настройке маршрутов и middleware
* Легковесность (быстрый запуск, низкое потребление ресурсов)
* Широкая экосистема расширений

**Альтернативы**: Django (избыточен для API), FastAPI (могли бы использовать, но Flask более зрелый)

**2. Flask-SQLAlchemy (работа с БД)**

**Почему выбрали**:

* ORM для удобной работы с SQL Server без прямых SQL-запросов
* Поддержка транзакций и сессий
* Автоматическое управление подключениями
* Интеграция с Flask через единую конфигурацию

**Альтернативы**: Raw SQL (менее безопасно), Django ORM (привязка к Django)

**3. PyODBC (подключение к SQL Server)**

**Почему выбрали**:

* Нативный драйвер для MS SQL Server
* Поддержка Windows Authentication (trusted\_connection)
* Совместимость с SQLAlchemy
* Высокая производительность для enterprise-решений

**Альтернативы**: pymssql (устарел), SQLite (не подходит для продакшена)

**4. Flask-RESTx (документирование API)**

**Почему выбрали**:

* Автогенерация Swagger-документации
* Валидация запросов/ответов через модели
* Поддержка пространств имен (Namespaces)
* Интеграция с JWT

**Альтернативы**: FastAPI (встроенная документация), drf-yasg (для Django)

**5. Python-dotenv (управление конфигурацией)**

**Почему выбрали**:

* Безопасное хранение чувствительных данных (JWT-ключ, DB credentials)
* Удобство разработки (разные настройки для dev/prod)
* Совместимость с 12-факторными приложениями

**Альтернативы**: Конфиг-файлы (.py), переменные окружения ОС

**6. Flask-JWT-Extended (аутентификация)**

**Почему выбрали**:

* Поддержка access/refresh токенов
* Гибкие callback-функции
* Интеграция с Flask-контекстом
* Защита эндпоинтов через @jwt\_required()

**Альтернативы**: OAuth2 (сложнее), SessionAuth (не для API)

**7. Werkzeug (вспомогательные утилиты)**

**Почему выбрали**:

* Хеширование паролей (generate\_password\_hash)
* Утилиты для работы с HTTP-запросами
* Входит в состав Flask (не требует отдельной установки)

**8. Python-dateutil (работа с датами и временем)**

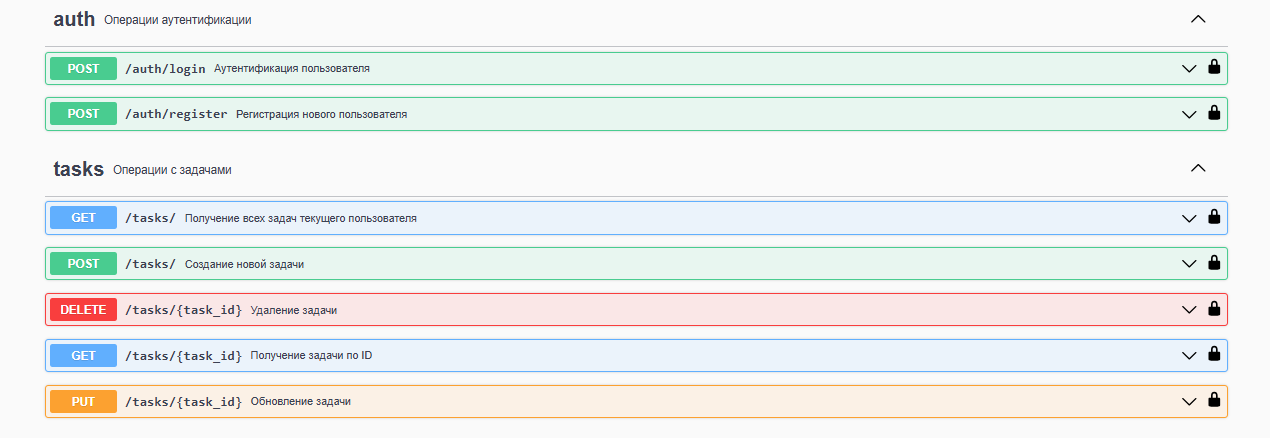
**Почему выбрали:**

* **Упрощенный парсинг дат** из различных строковых форматов без необходимости явного указания шаблона
* **Расширенные возможности работы с датами** (корректное добавление месяцев/лет с учетом разной длины месяцев)
* **Поддержка временных зон** для корректного отображения времени для пользователей из разных регионов
* **Гибкие рекуррентные вычисления** для реализации периодических задач и напоминаний

**Альтернативы:** Arrow (менее распространен в корпоративной среде),

Pendulum (более тяжеловесное решение)

**2.4 API Endpoints**



ЗАКЛЮЧЕНИЕ

**В результате выполнения проектного практикума было разработано программное обеспечение «Канбан Доска», соответствующее всем поставленным требованиям. Выполнены следующие этапы:**

* **Сформулированы функциональные/нефункциональные требования.**
* **Разработана архитектура системы, включая схему базы данных и RESTful API.**
* **Реализованы основные функции: управление задачами, колонками, проектами и пользователями с использованием Flask и SQLAlchemy.**
* **Обеспечена безопасность через JWT-аутентификацию и разграничение прав доступа.**
* **Создана автоматизированная Swagger-документация для удобства интеграции.**

**Цель проекта достигнута, все задачи выполнены в полном объёме.**

**Исходный код:** https://github.com/DmitriiShtrikov/KanbanBoard