МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное   
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

(Самарский университет)

Институт информатики и кибернетики  
Кафедра технической кибернетики

Отчёт по проекту Система управления задачами

по курсу   
Технологии сетевого программирования

Группа 6302-010302D

Студенты Брылёв Д.В

Уржунцев А.С.

(*подпись*)

Самара 2025

**Название: "Система управления задачами"**

Описание: Приложение для работы со своими задачами. Планирование рабочего дня, запись планов на день, неделю и т.д.

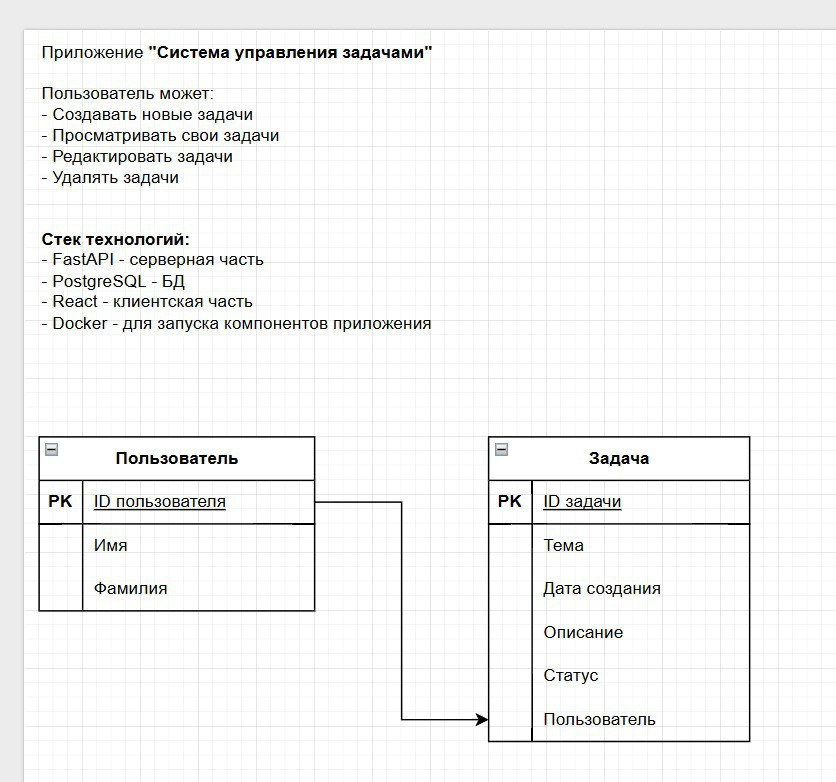
Пользователь может:

- создавать новые задачи

- просматривать задачи

- редактировать задачи

- удалять задачи



**Стек технологий**

Разработка на языке python.

Backend:

- FastAPI - серверная часть

- MongoDB - БД

- Docker - для запуска компонентов приложения

Frontend:

- React - клиентская часть

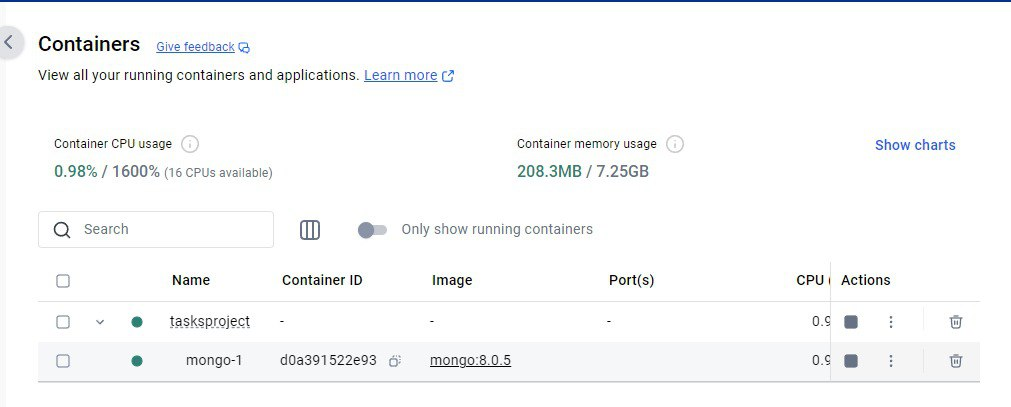
- Docker - для запуска компонентов приложения

**Репозиторий GitHub**

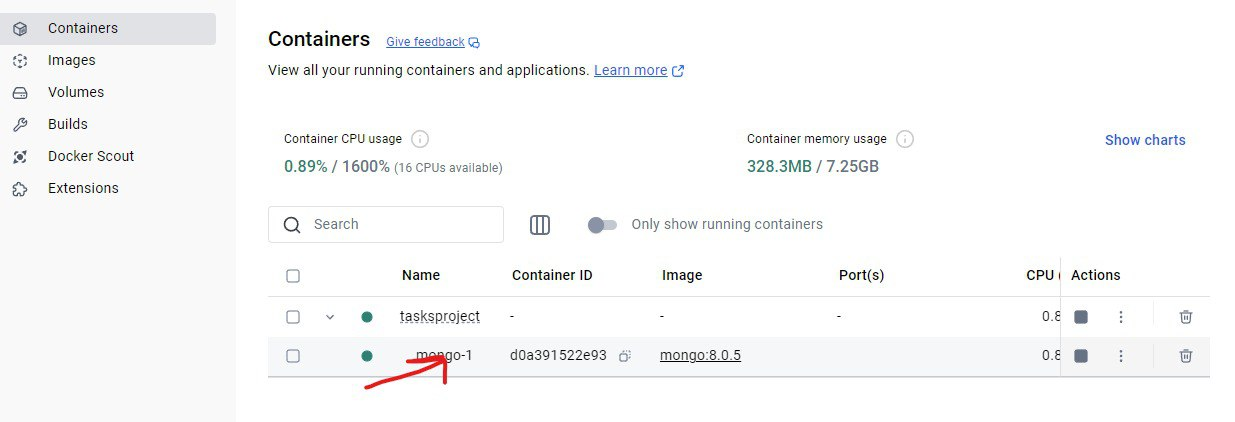
https://github.com/dimokvip/TSP-project

**Лабораторная работа 1**

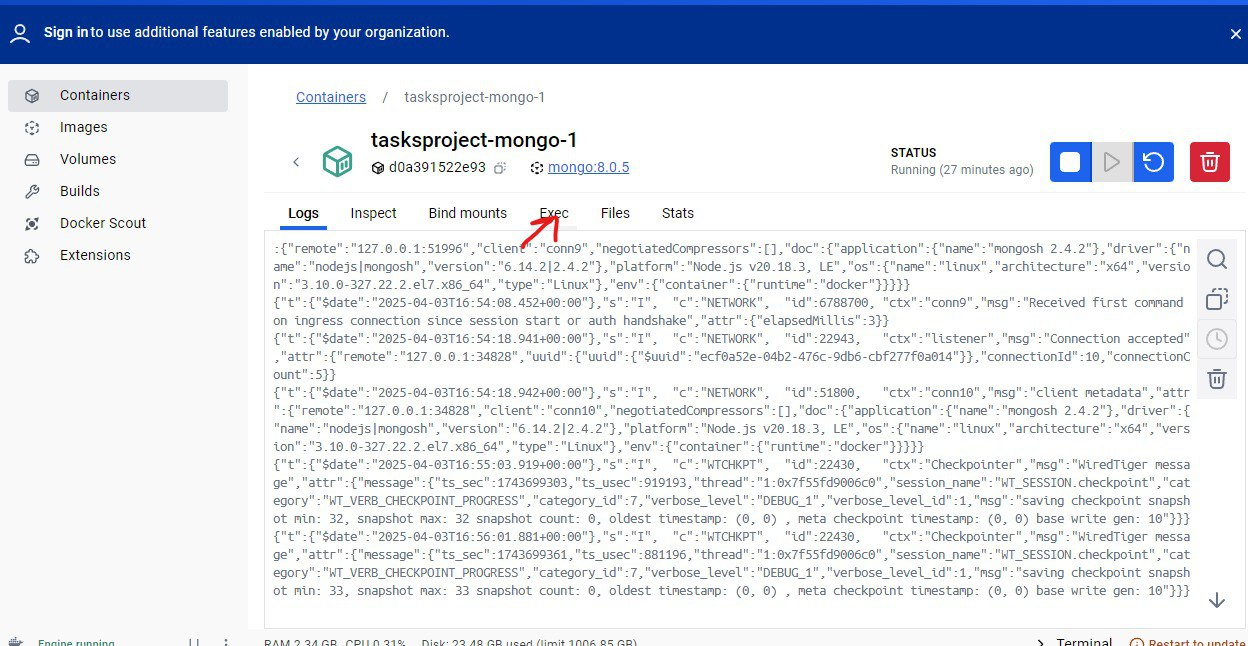
Был установлен Docker, в котором была развёрнута база данных. За image был взят PostgreSQL БД разворачивается с помощью Docker.



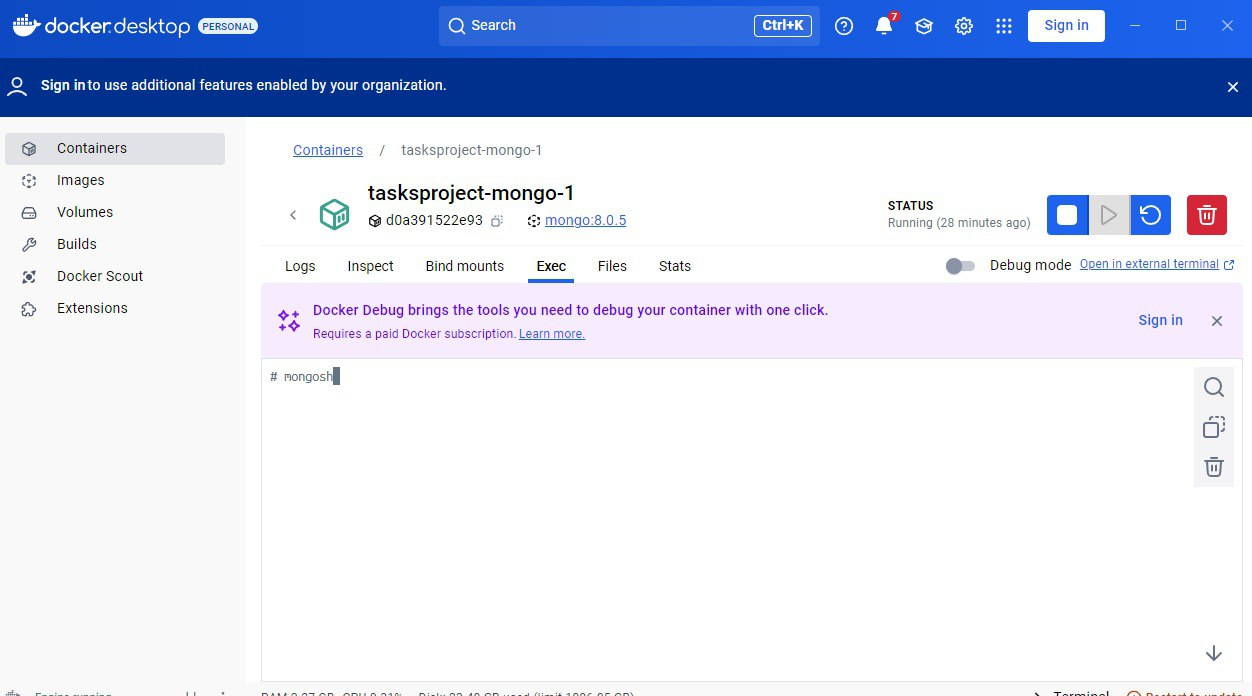
Реализация модели пользователя:

Для этого переходим в контейнер 

Переходим в терминал



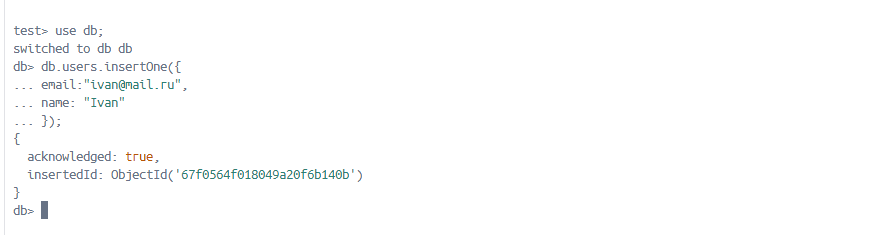
Вводим команду:



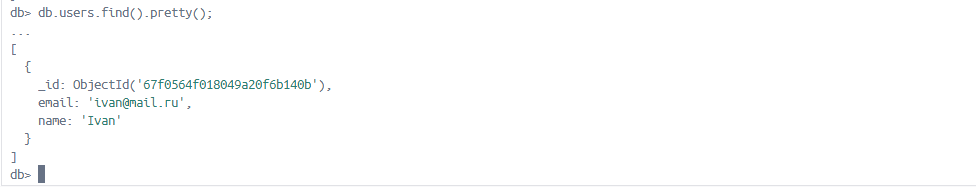
Станет доступна оболочка Postgres. Теперь можно будет вводить команды и получать / добавлять / удалять данные:

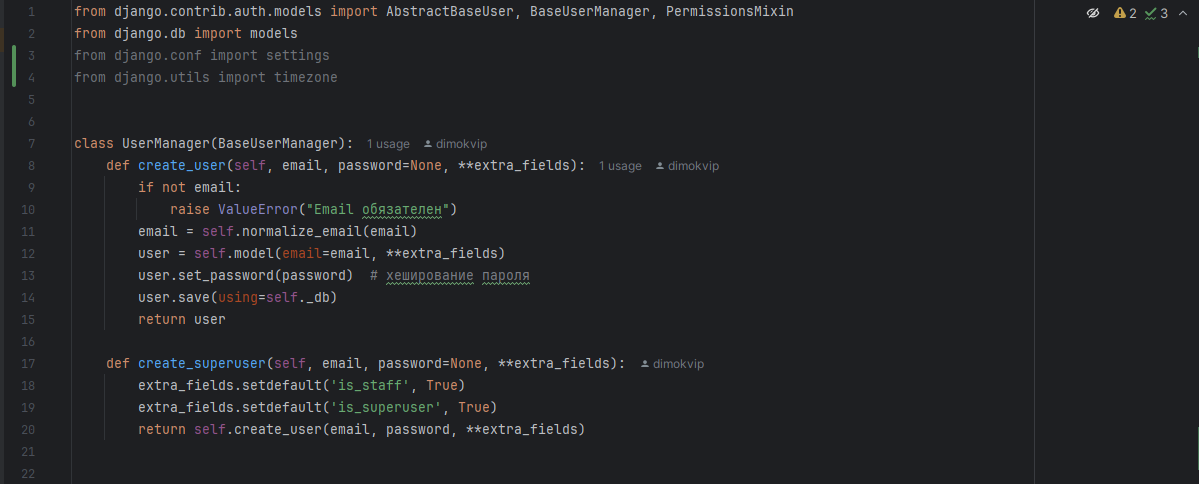


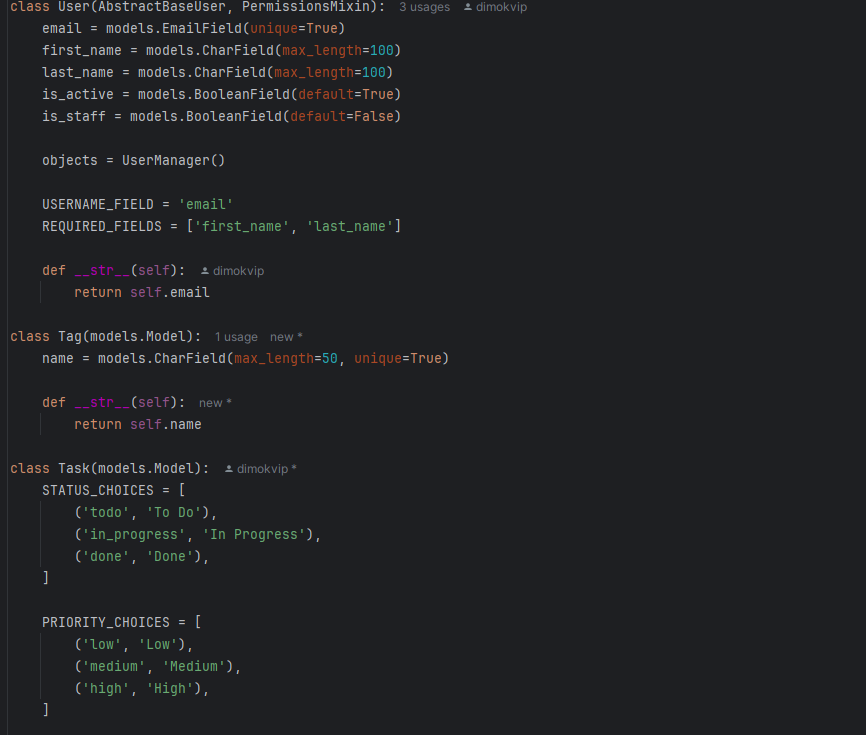
Переходим в БД и создаём пользователя:



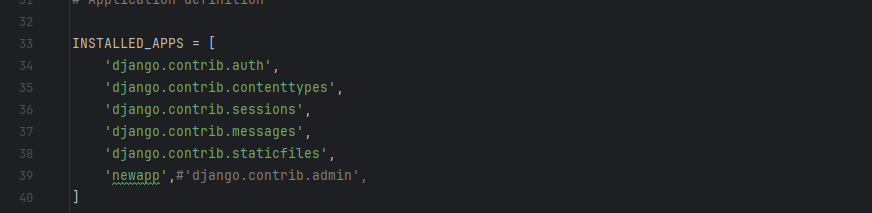
Проверяем, что пользователь создался:









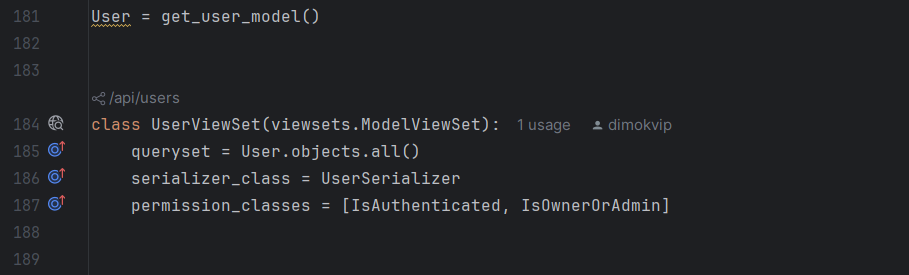


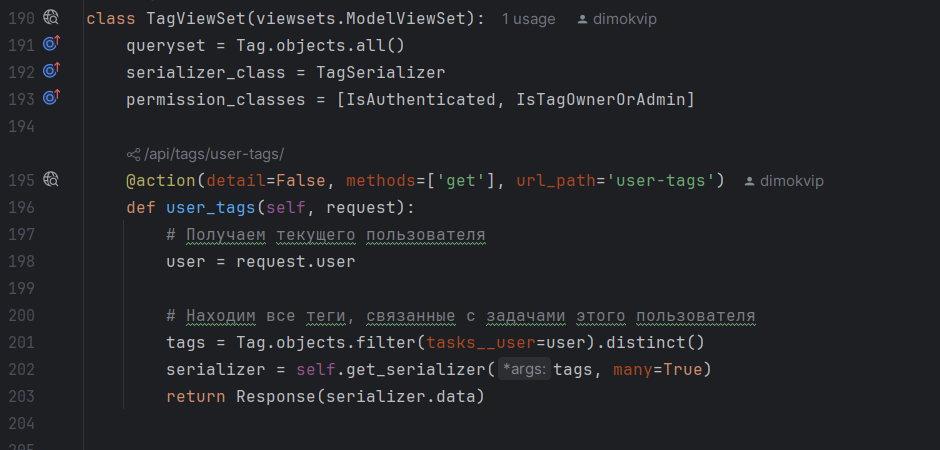


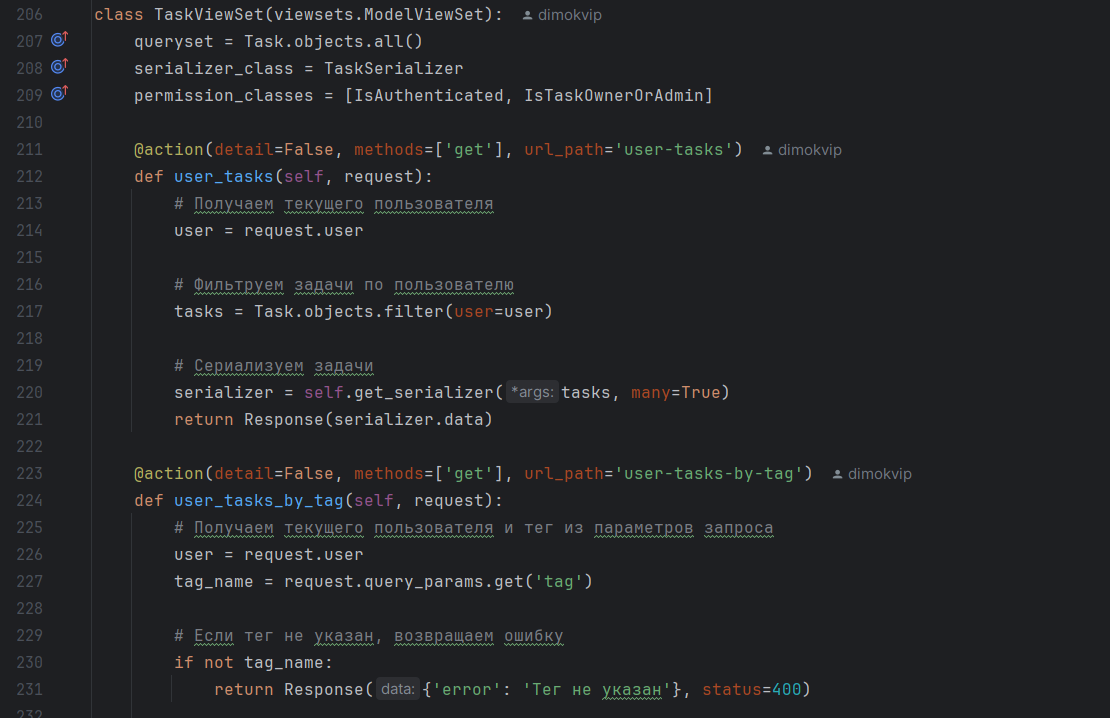
**Лабораторная работа 2**

В данной лабораторной работе мы реализовали CRUD-методы с помощью rest-framework. И дописали своим методы в файле view.py, для того чтобы создать представление.

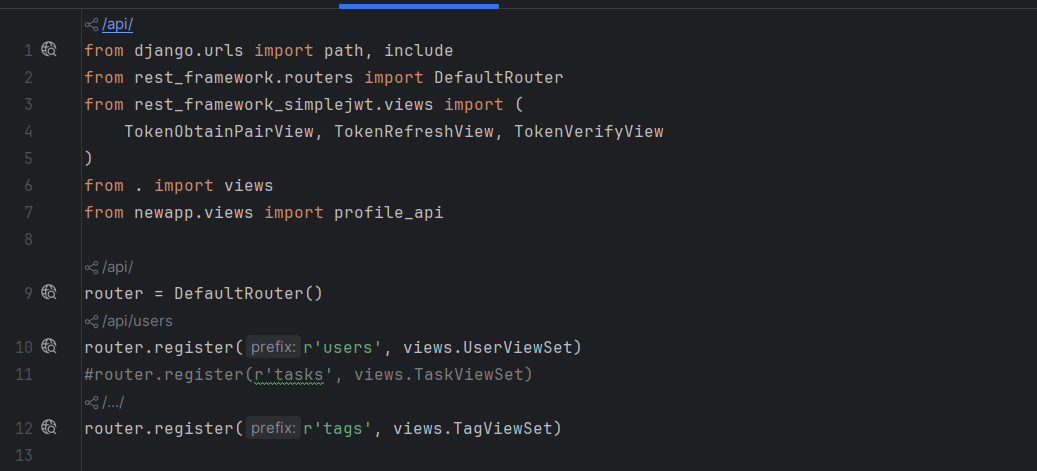
Вот код программы



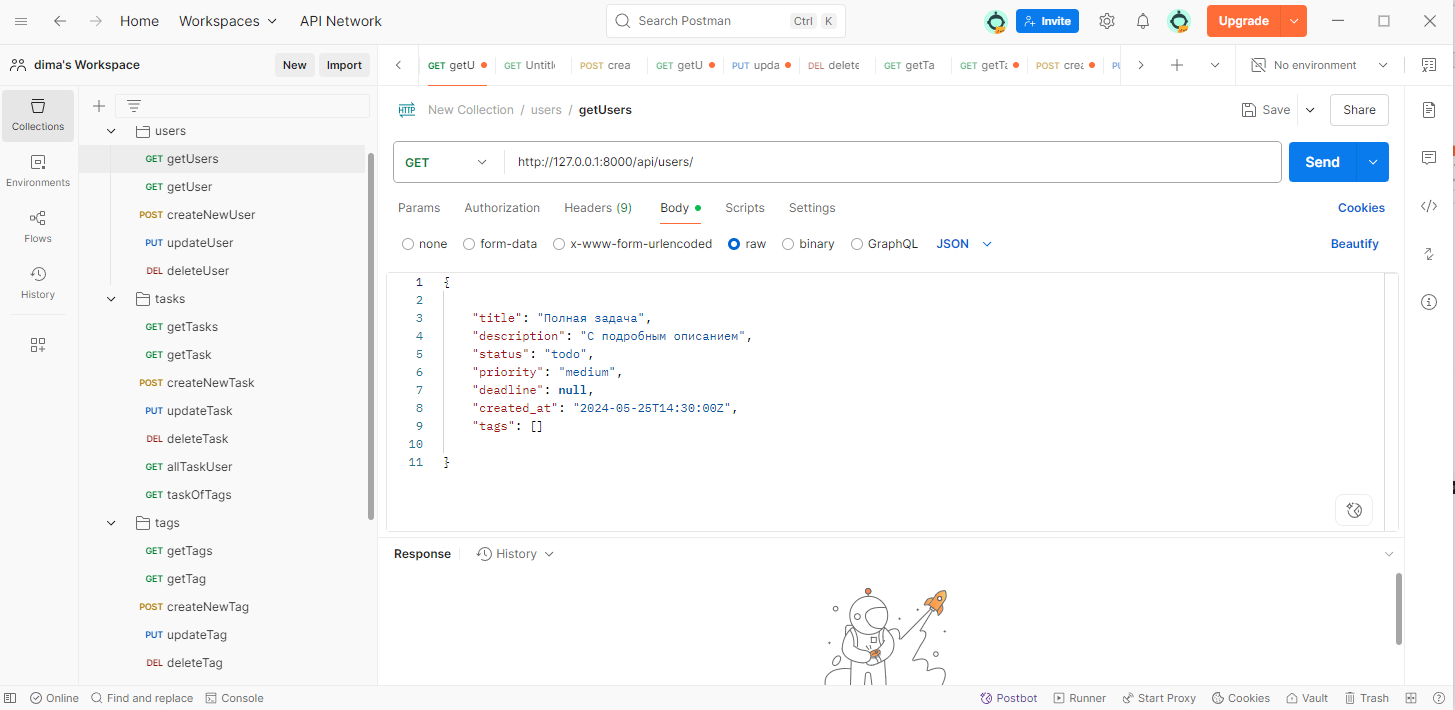


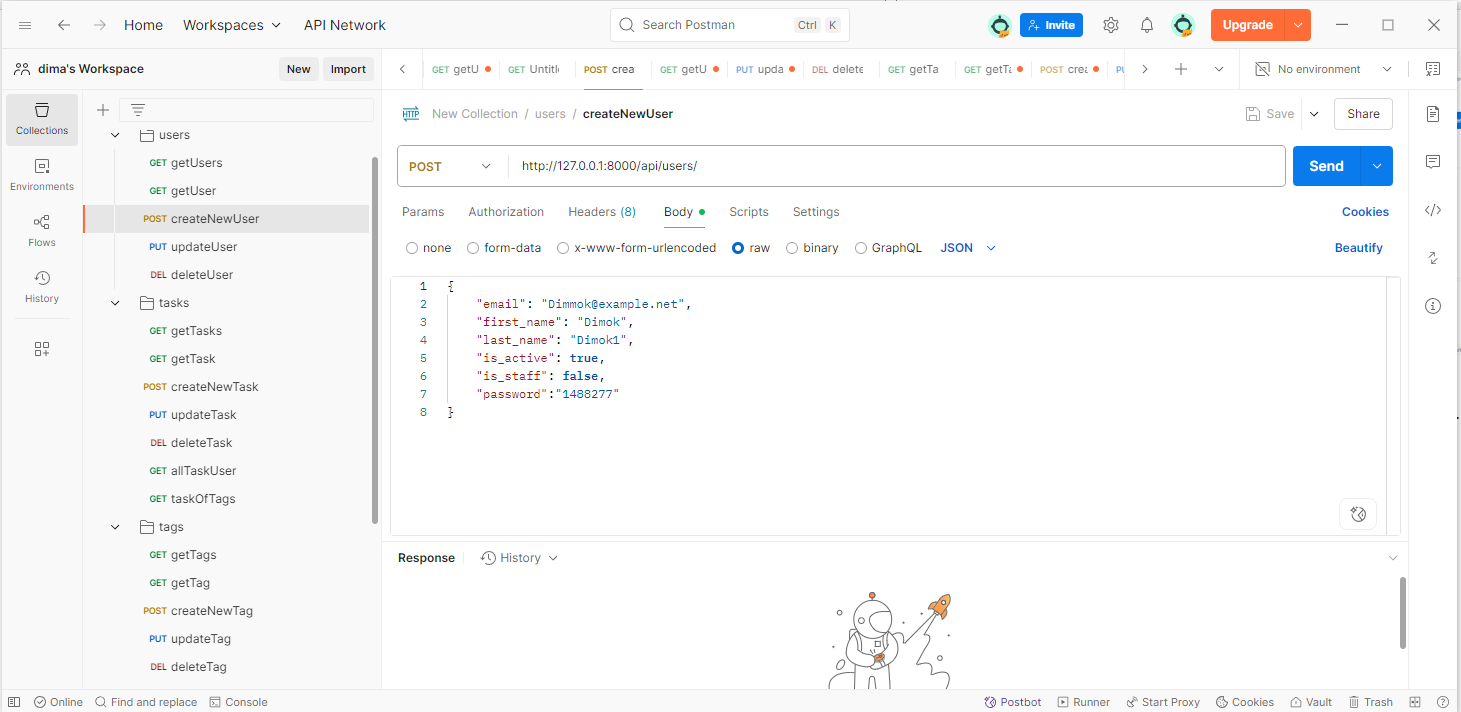


Далее мы настроили маршруты в файле urls.py

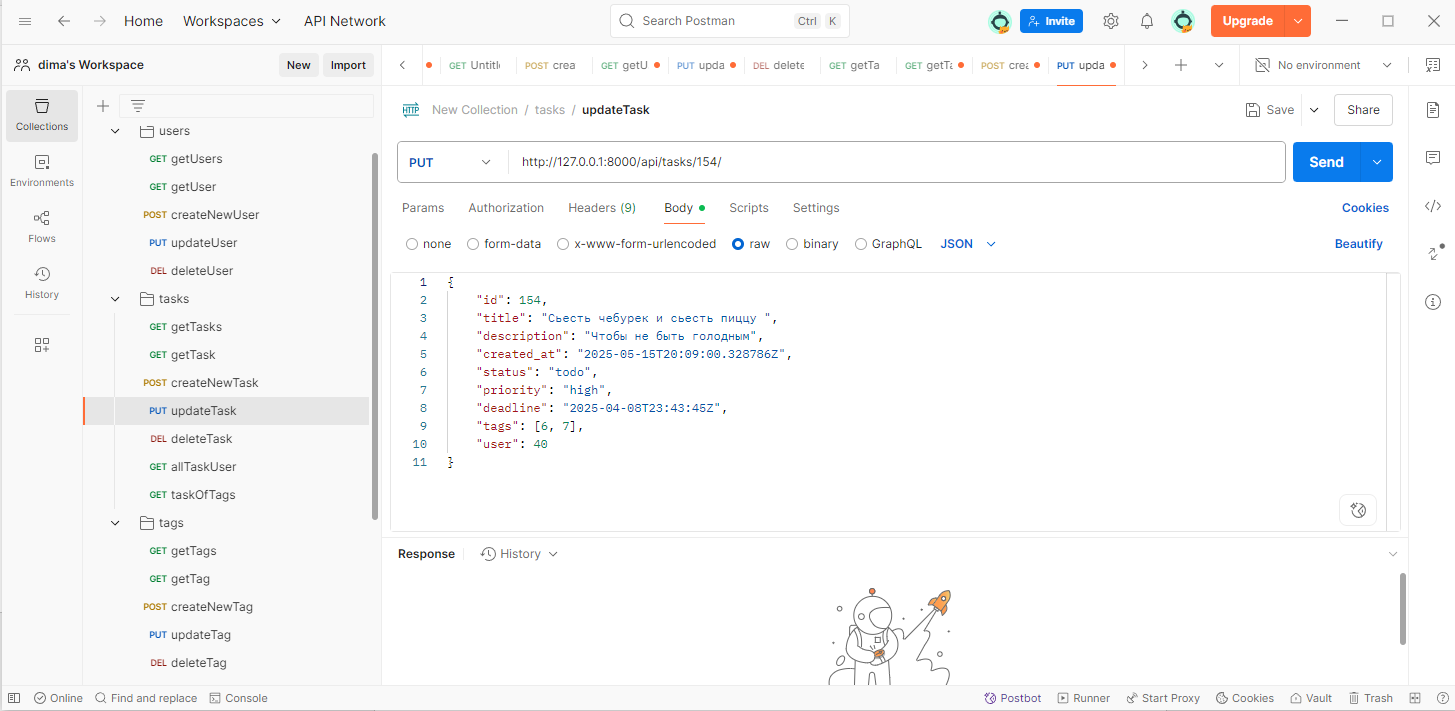


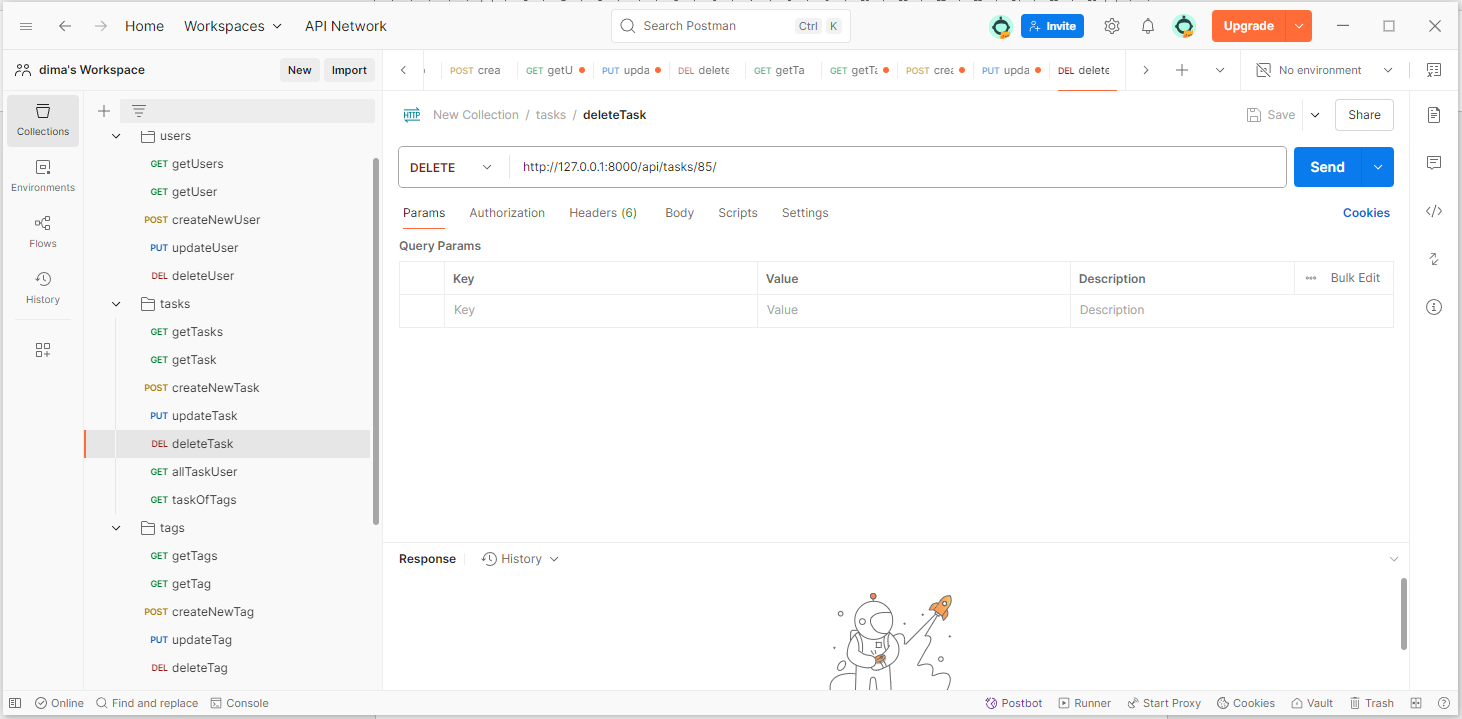
Далее мы протестировали api c помощью Postman. Создали коллекцию и создали основные запросы CRUD методов

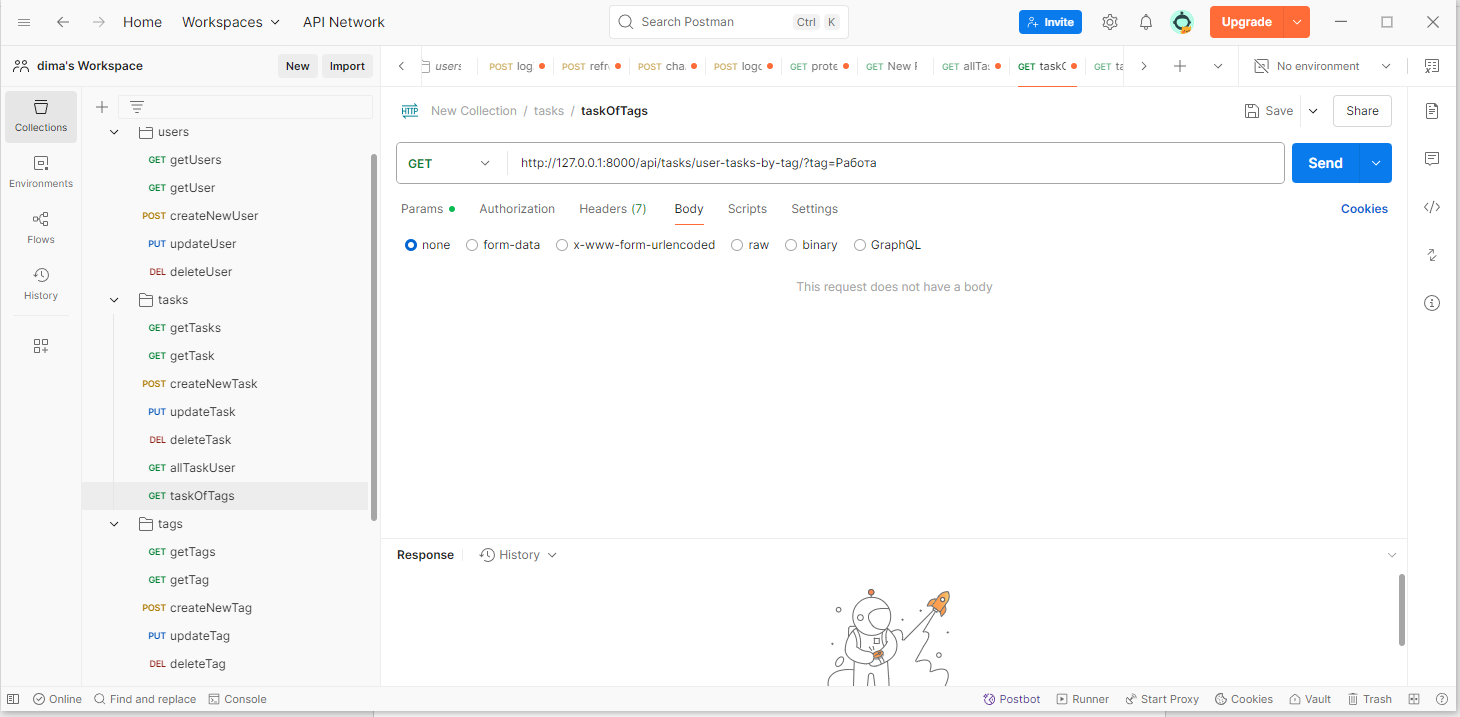


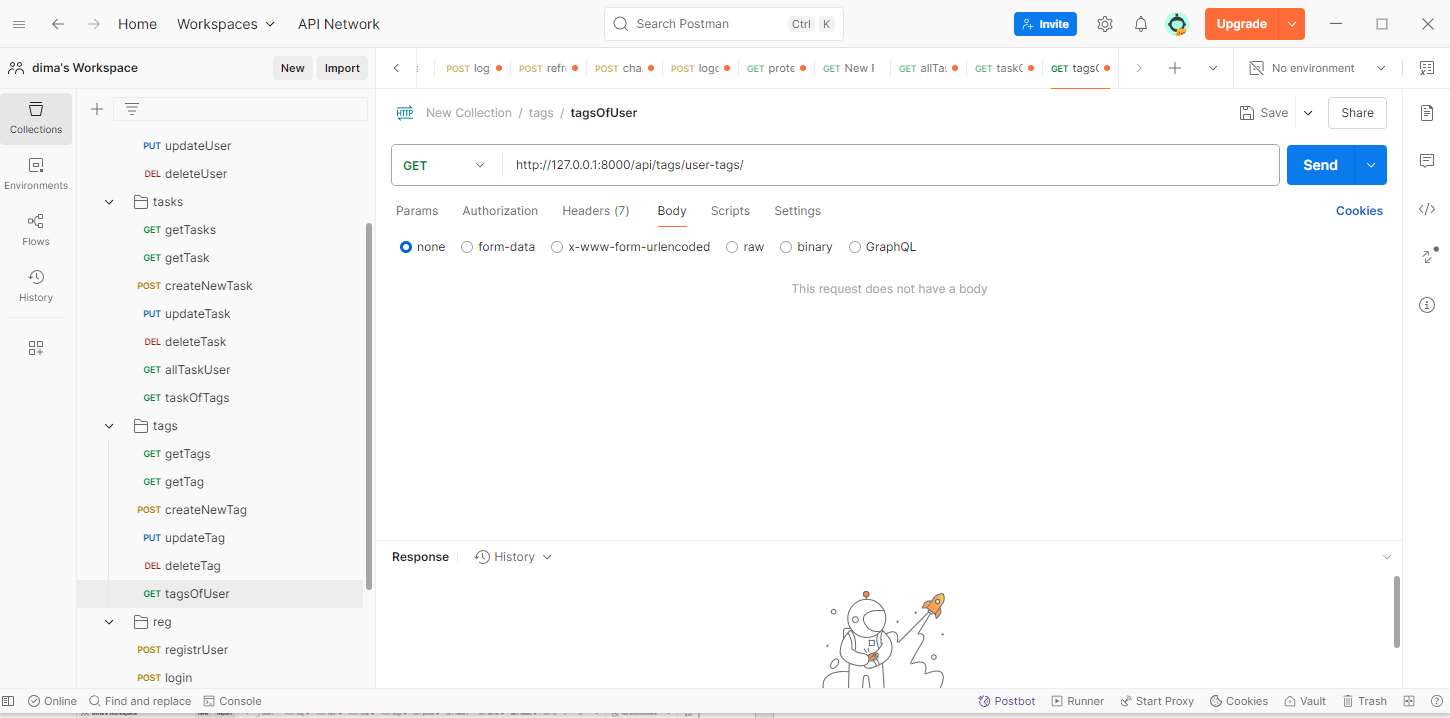










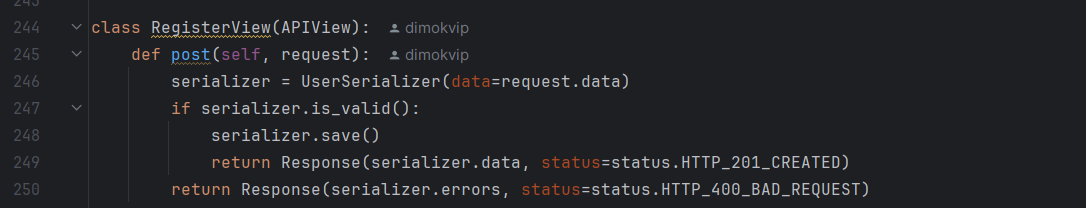


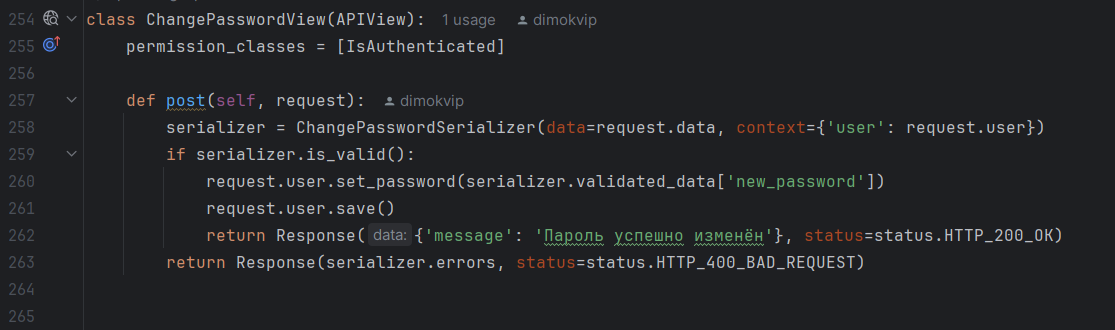
Все методы работоспособны, продемонстрировали это преподавателю.

**Лабораторная работа 3**

В данной лабораторной работе реализовали аутентификацию. А именно регистрацию, и авторизацию пользователя с использованием JWT токенов.

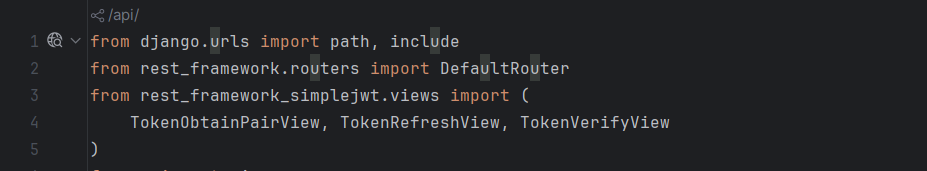
Для регистрации, логина, изменения пароля мы написали вьюшки





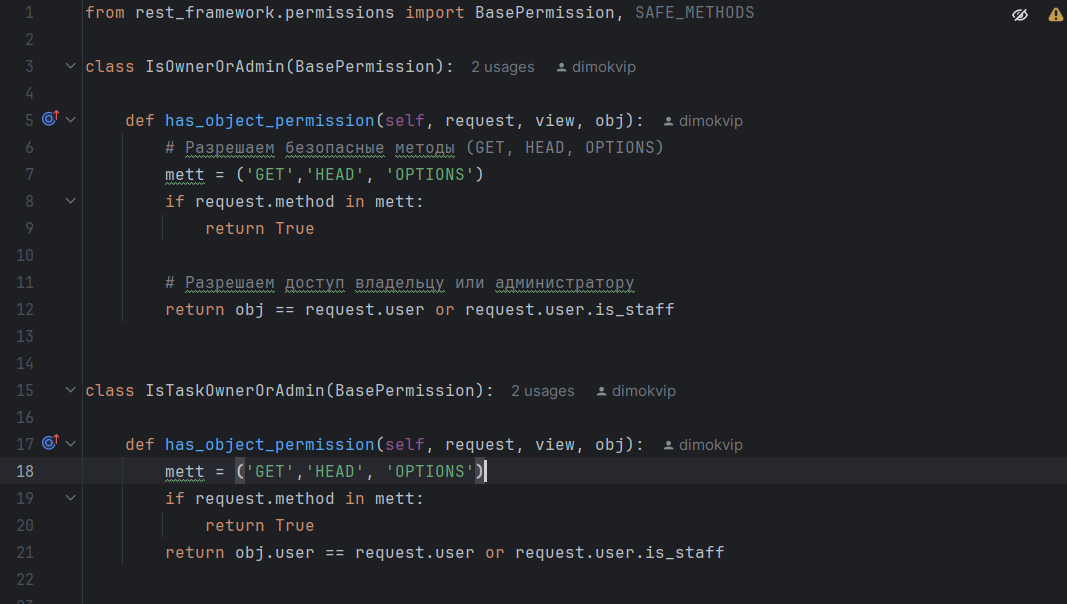


Далее в urls.py прописали пути и описали работу JWT токенов с использованием фреймворка rest\_framework и библиотеки simplejwt. Оттуда мы импортировали функции для работы с JWT токенами



И прописали пути 

Так же создали ограничение доступа к определенным эндпоинтам. Для этого мы создали файл permission.py и описали права доступа

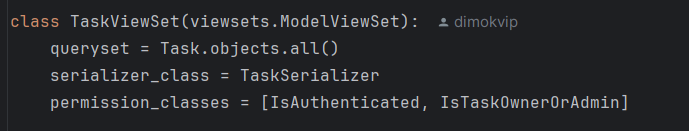




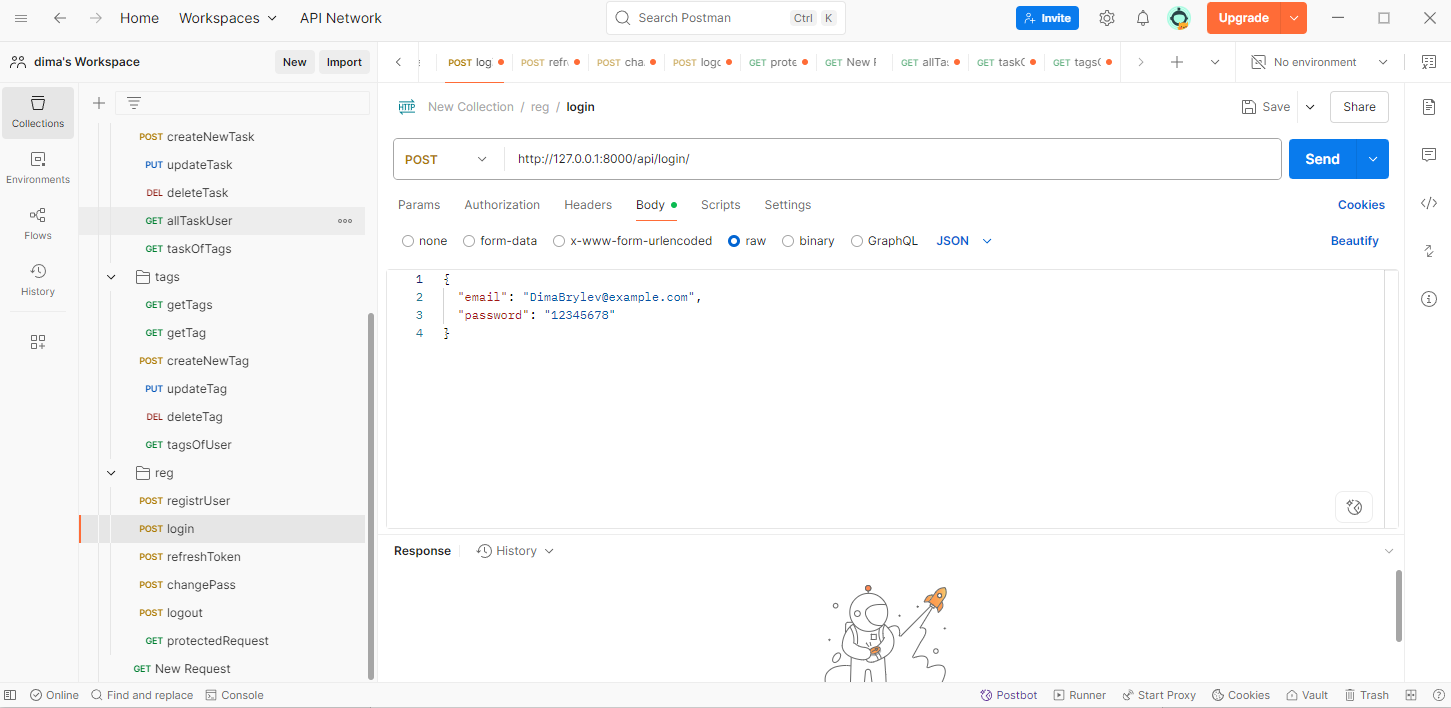
И после каждой вьюшке добавили классы доступа. На примере представлена вьюшка для изменения пароля класс доступа IsAuthenticated



Для создания задачи IsAuthenticated, IsTaskOwnerOrAdmin

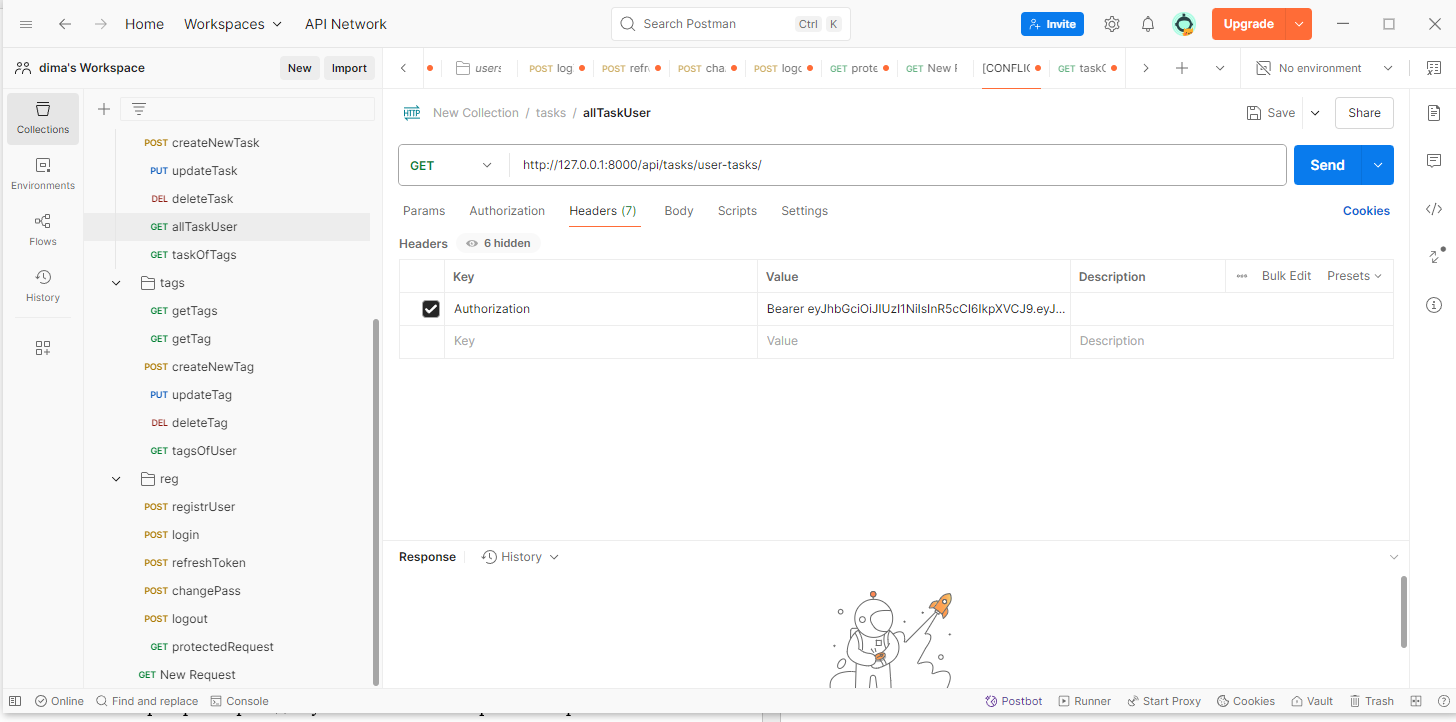


Далее проверили работоспособность с помощью Postman, в коллекцию добавили новую папку reg и добавили в нее необходимые запросы.





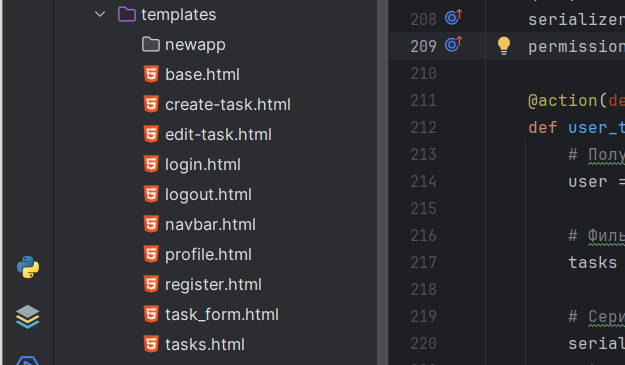
Также проверили права доступа на остальные запросы. Теперь большинство запросов нельзя делать без access токена.



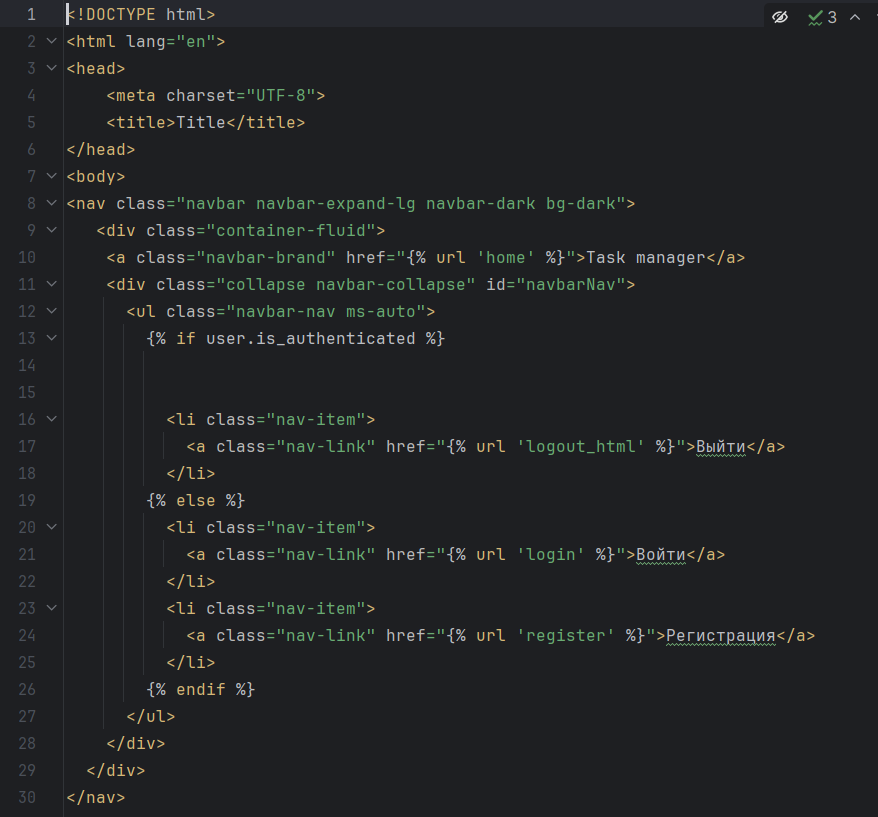
**Лабораторная работа 4**

В данной лабораторной работе мы разработали пользовательский интерфейс создали макеты страниц с помощью шаблонизатора Django разработали страницы регистрации, авторизации и работы с данными.

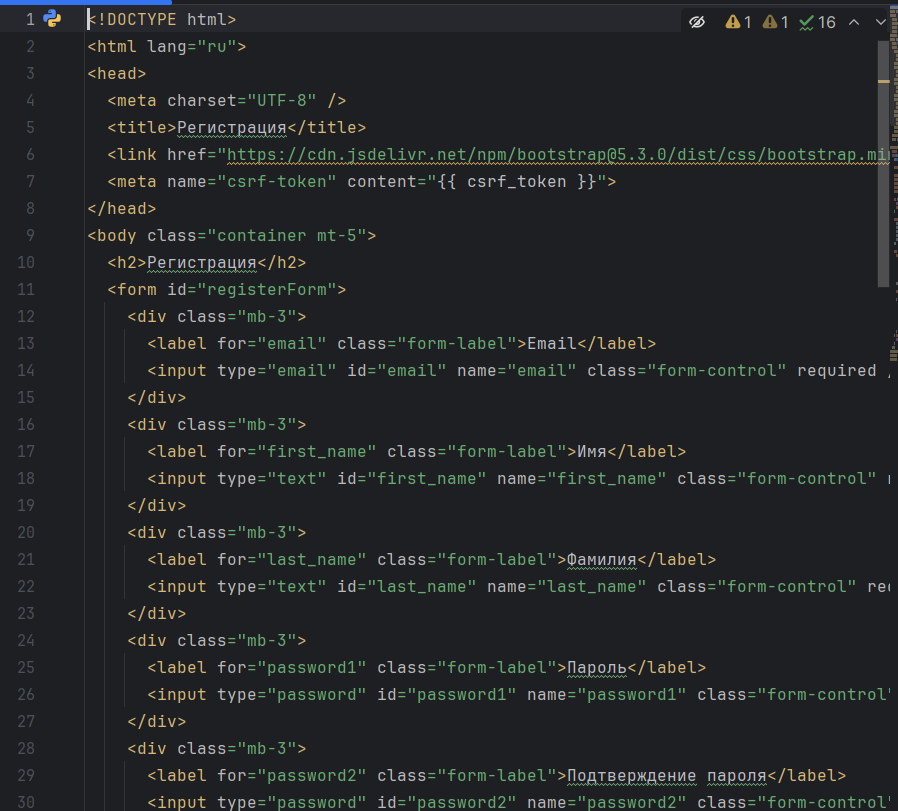
В начале мы создали шаблоны страниц html



Определили базовые параметры и создали динамический nav bar

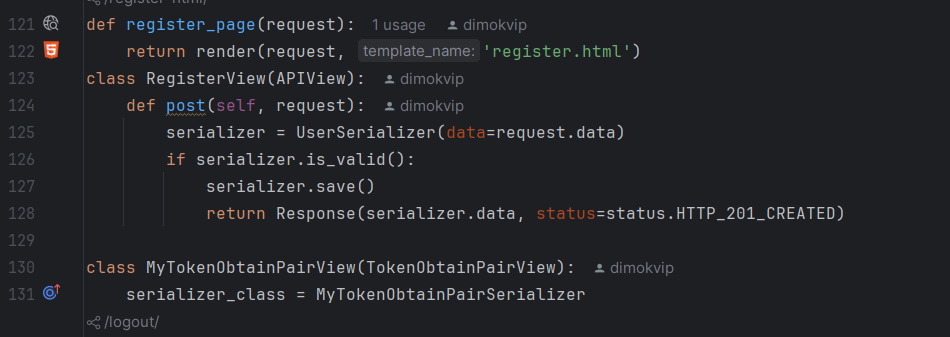


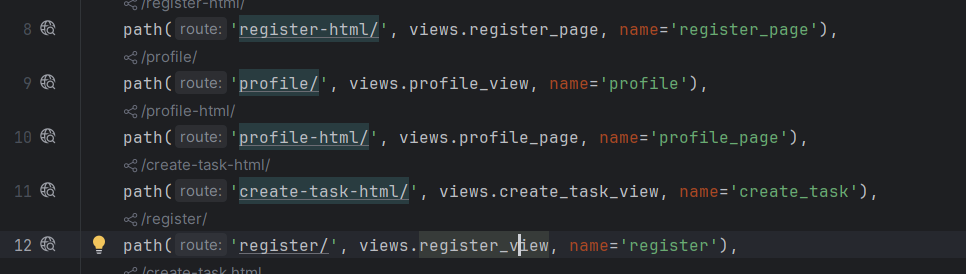
И создали html представление страницы с регистрацией.



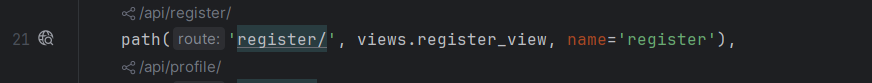
Далее нужно было добавить вьюшки и прописать пути.



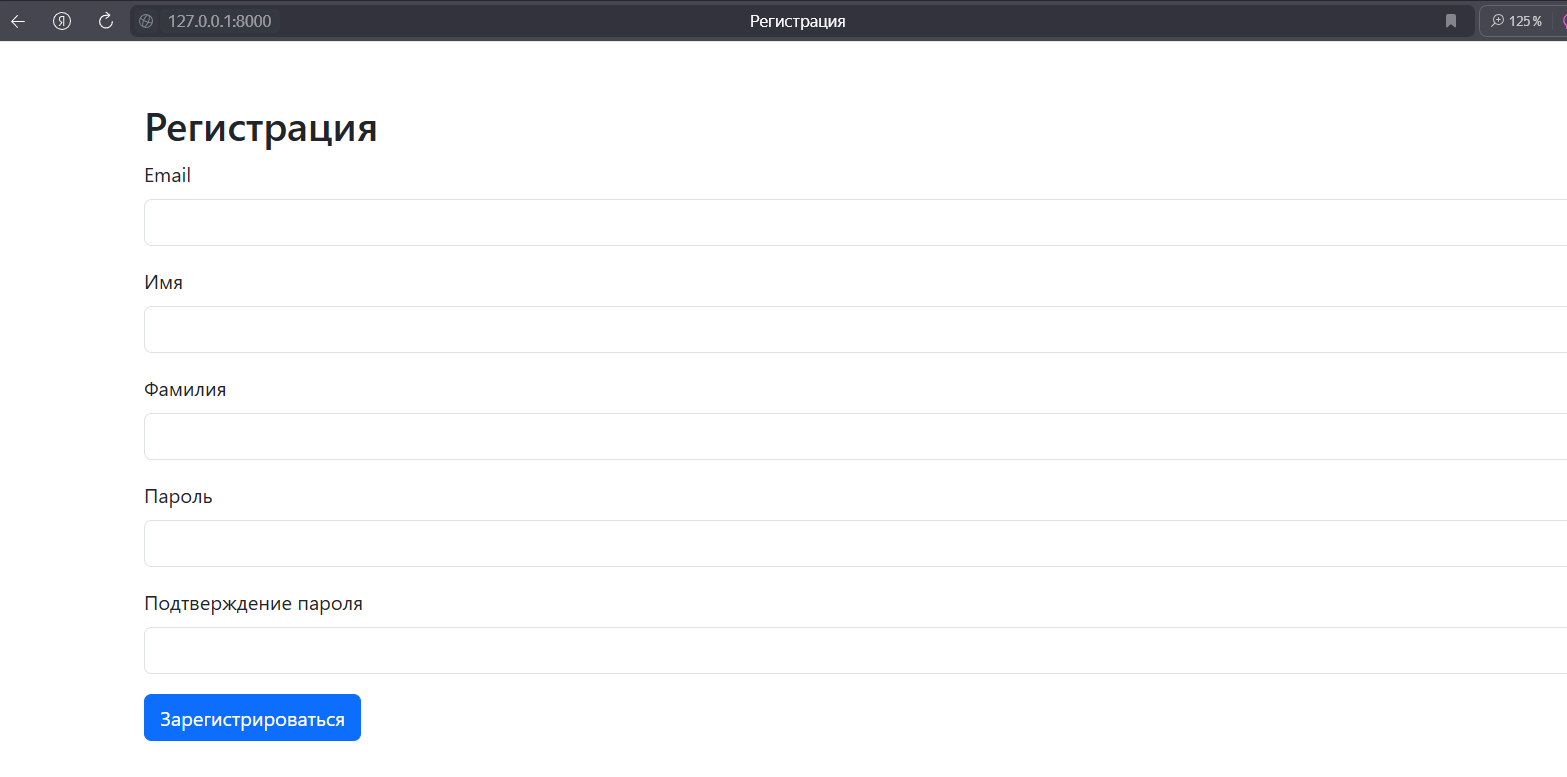


И в файле urls.py прописали путь на html страницу 

И также в файле api\_urls.py прописали путь для api



Запустили сервер и проверили работоспособность регистрации.

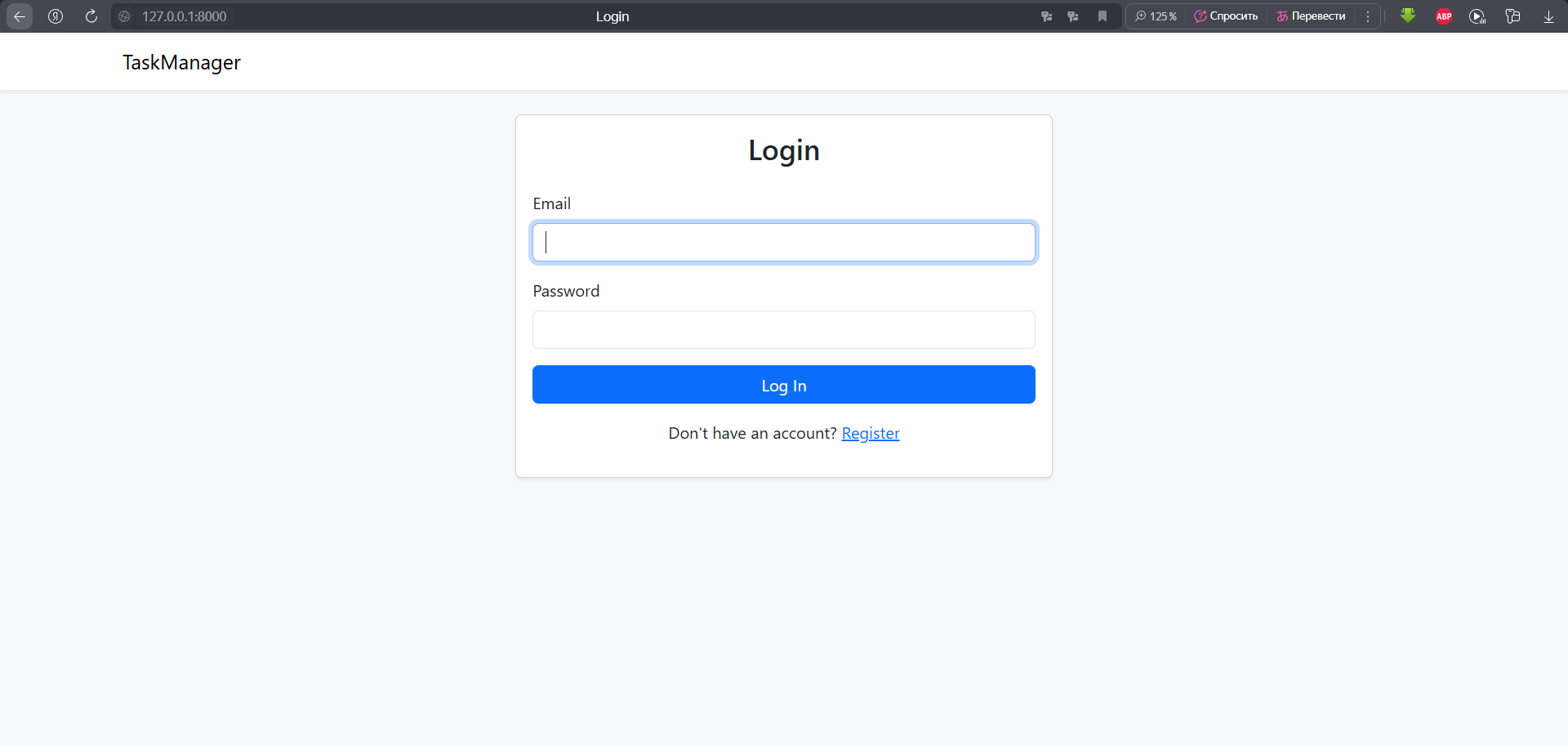
Вот так выглядит страница с регистрацией 

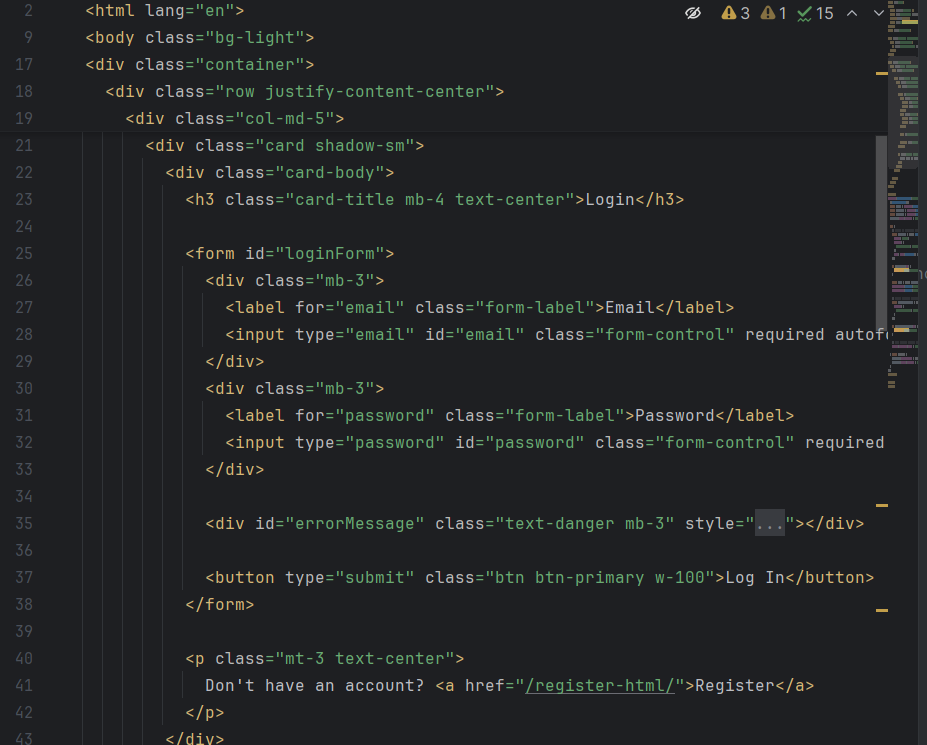
Для того чтобы работала передача данных и последующий переход на другие страницы мы добавили js script fetch запросов



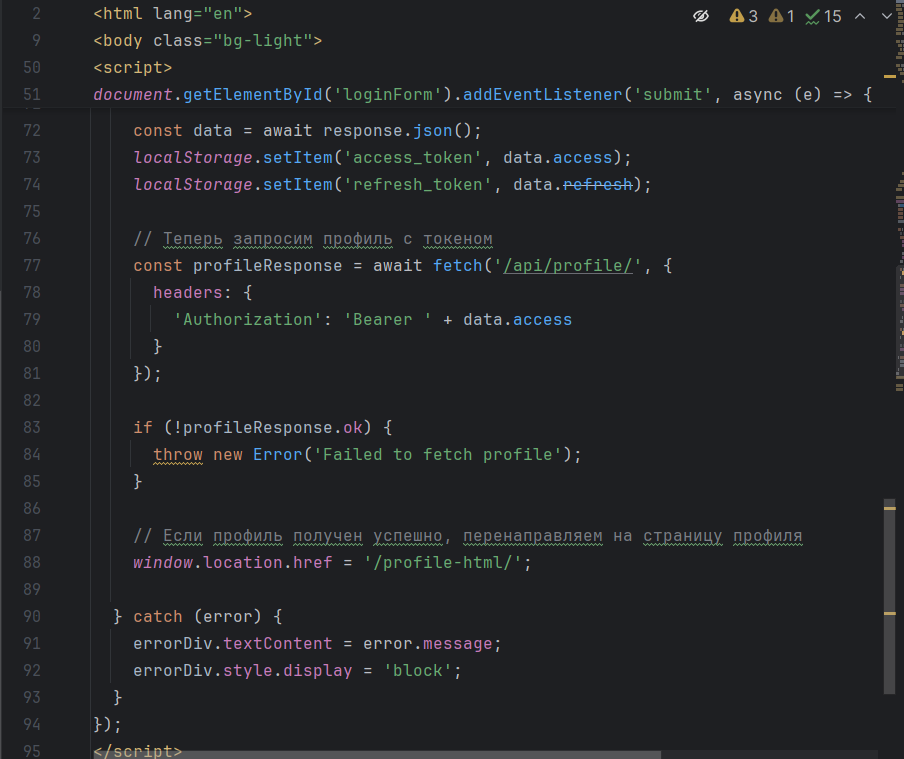


После того как пользователь зарегистрировался его перебрасывает на страницу логина, чтобы он вошел и получил refresh токен

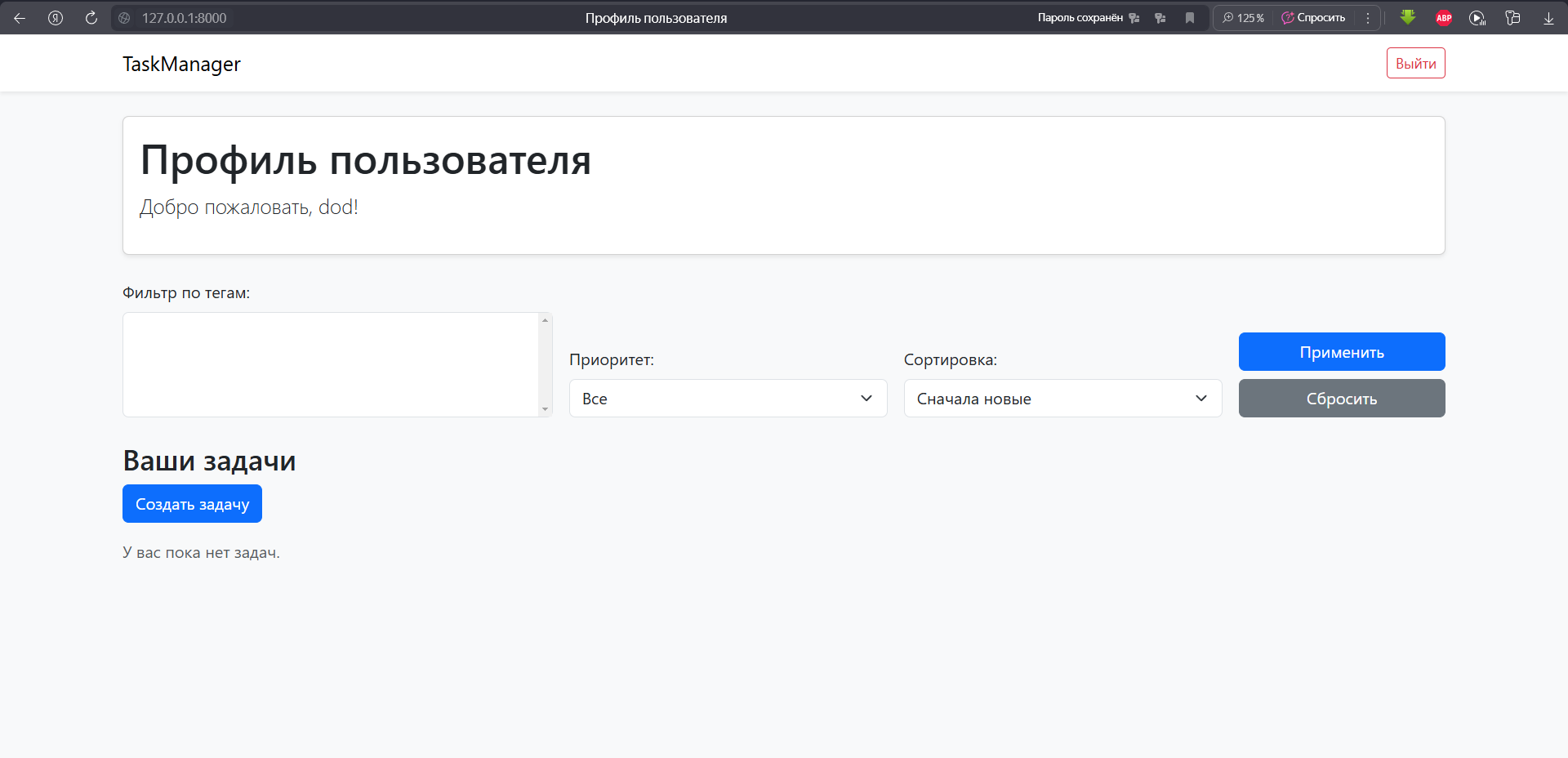




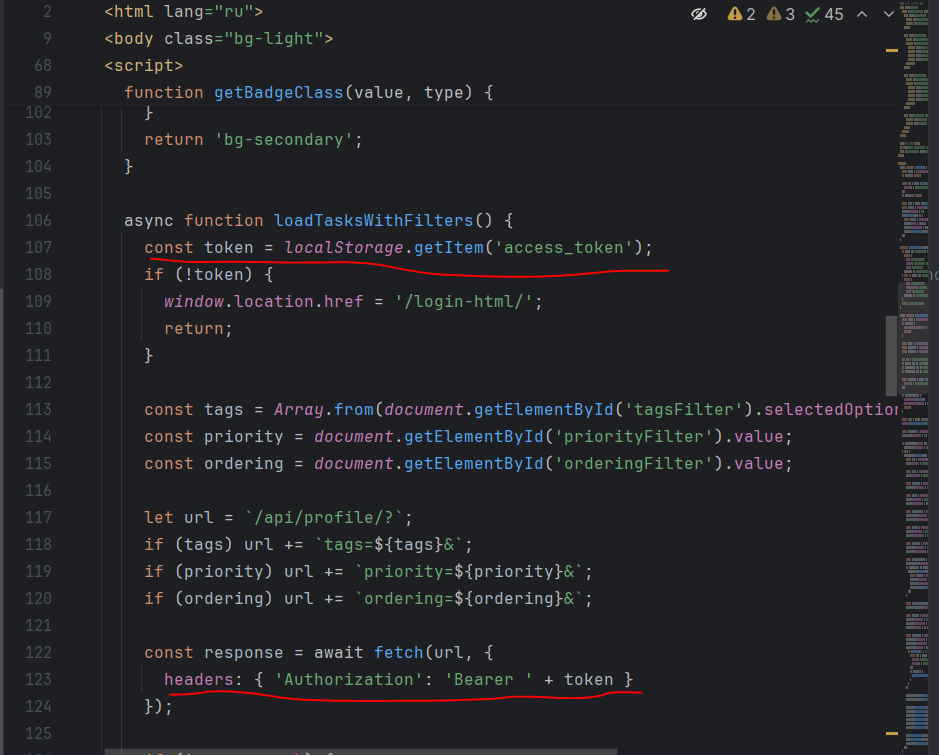




После успешного входа пользователя перебрасывает на страницу профиля, на скрине команда *window*.location.href = '/profile-html/';

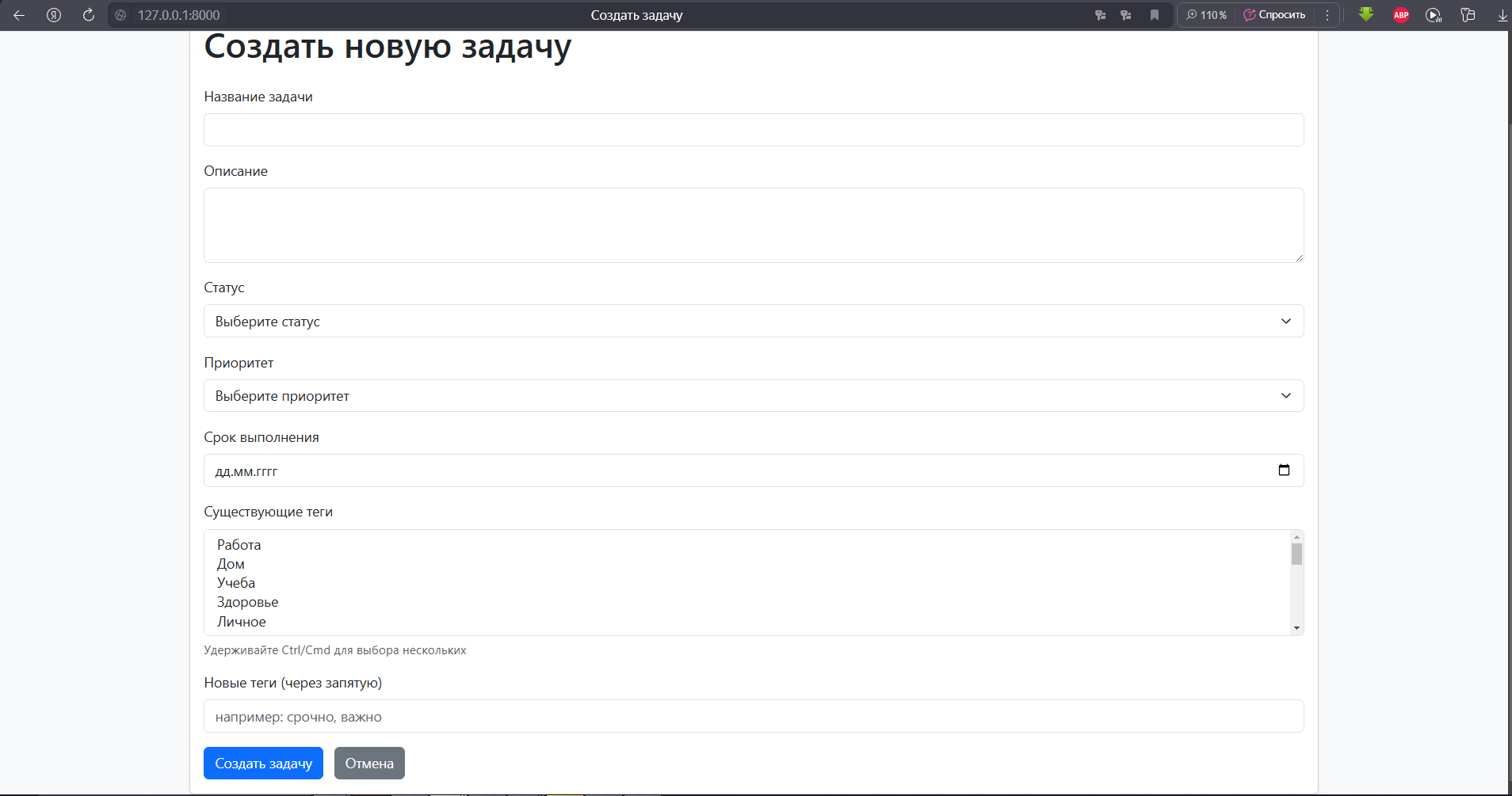
Так выглядит страница пользователя   


Каждое действие на странице сопровождается обработкой access токена

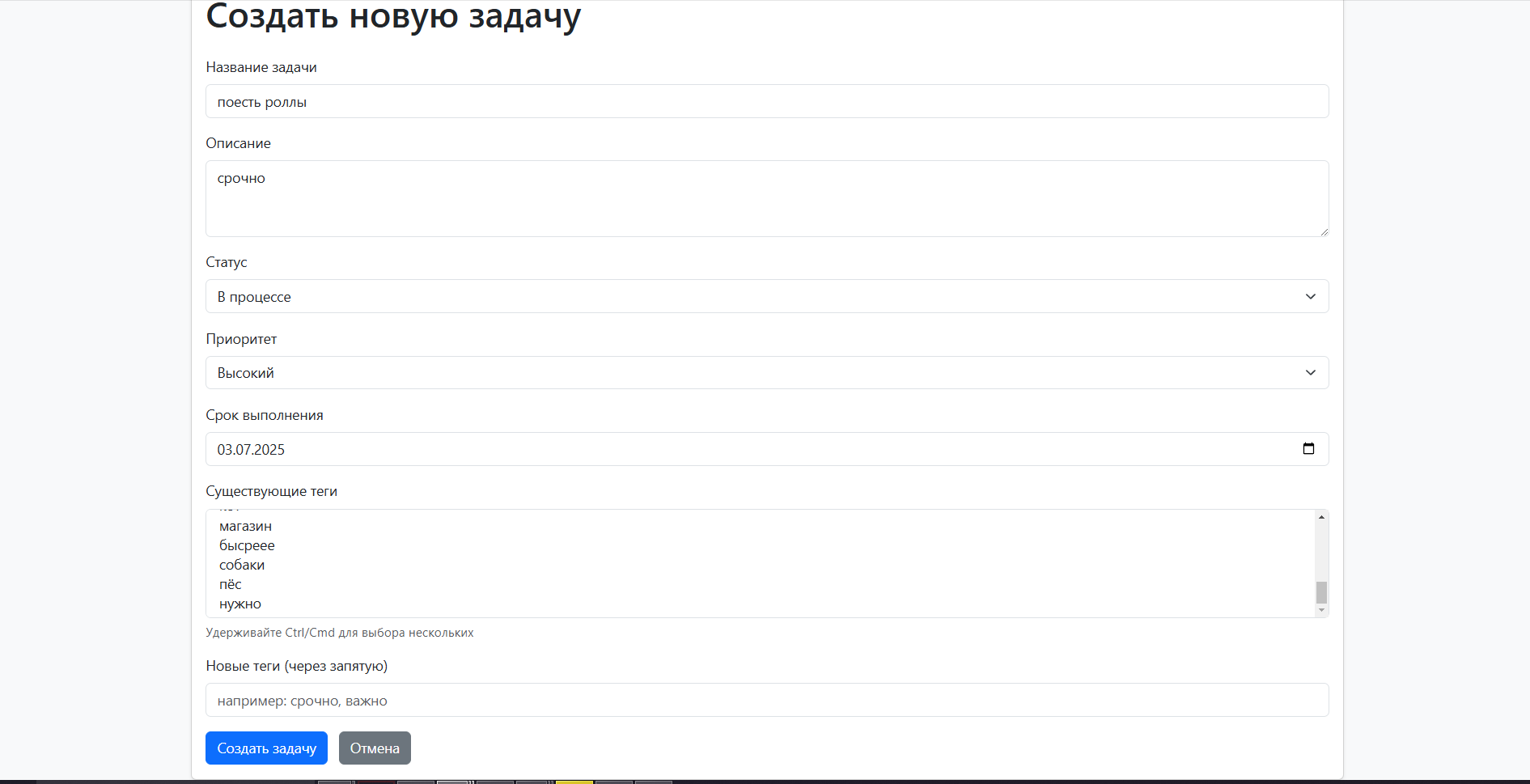


Так же после нажатия кнопки создать задачу, пользователь переходит на страницу создания задачи.

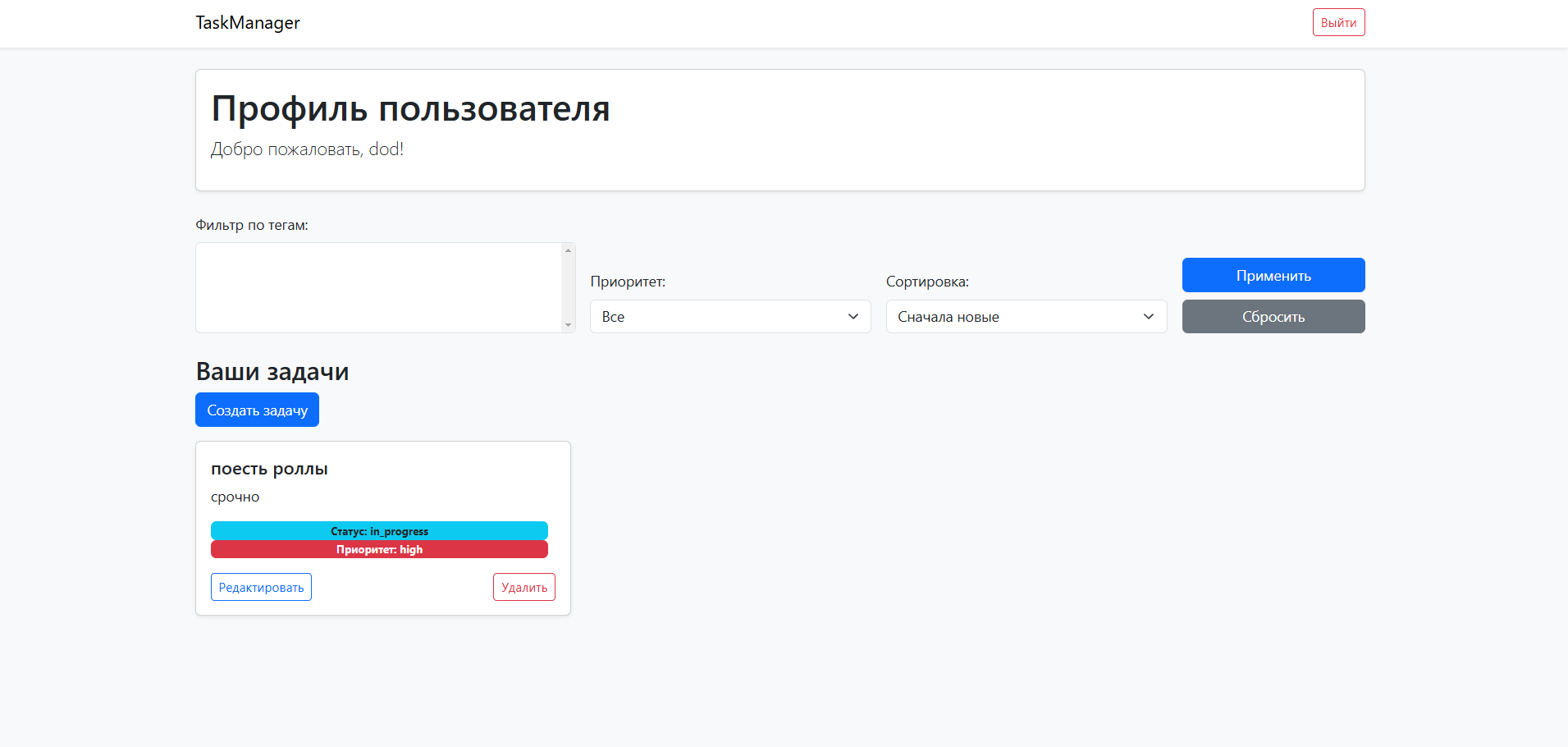
Она выглядит так



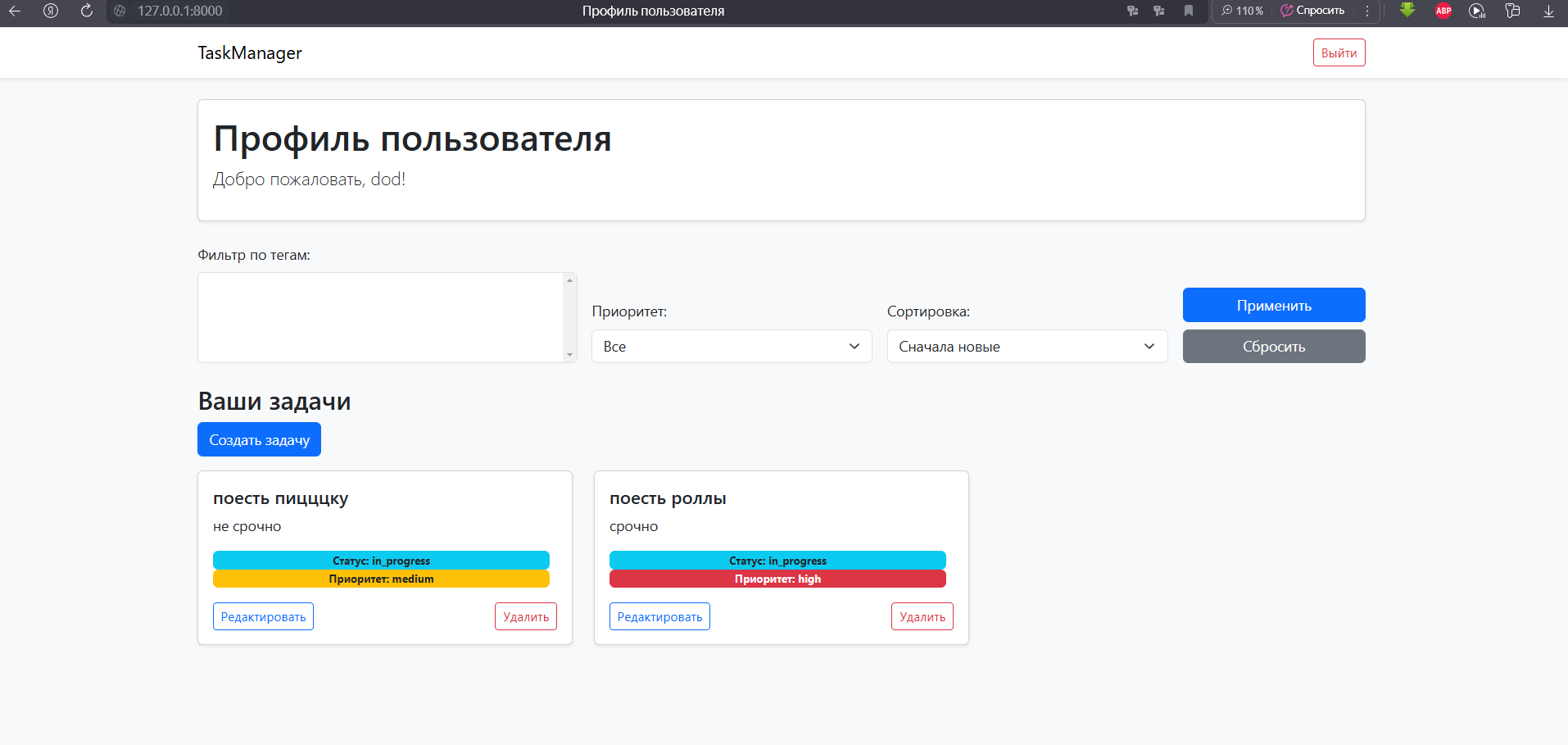
Введя нужные поля создается задача, добавляется в базу данных, и отображается на странице профиля



Теперь наша задача отображается на странице профиля



Также на странице профиля существует фильтр по приоритету и сортировка, а также кнопка удалить фильтр



Ещё есть кнопка выхода, при нажатии которой рефреш токен попадет в блеклист, и сессия закончиться. Код представлен на скрине.

