Тестовое задание №1 — Проект “Task Tracker”

1. Структура данных

Пользователь (User)

id- уникальный идентификатор

username- имя пользователя

email- электронная почта

role- роль пользователя (например: user или admin)

hashed\_password- хэш пароля

Задача (Task)

id- уникальный идентификатор

title- название задачи

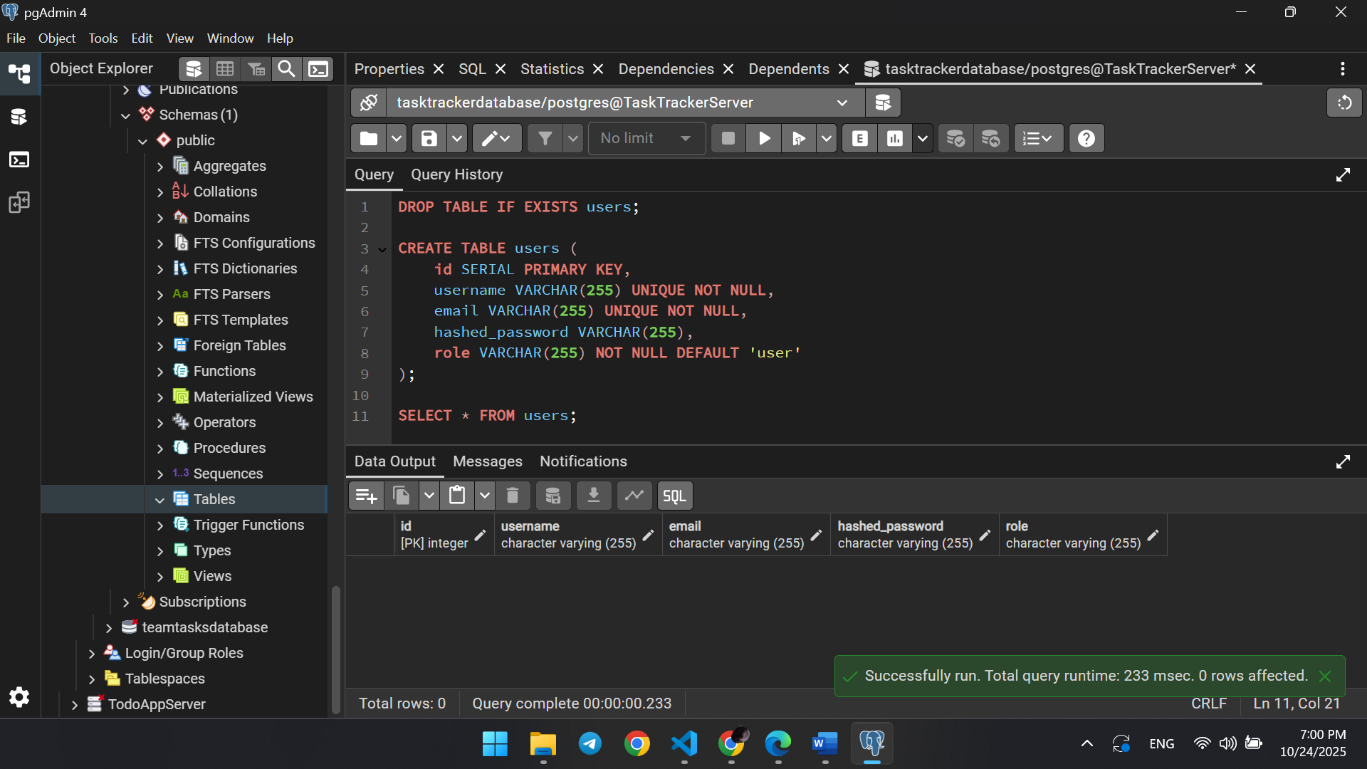
description- описание

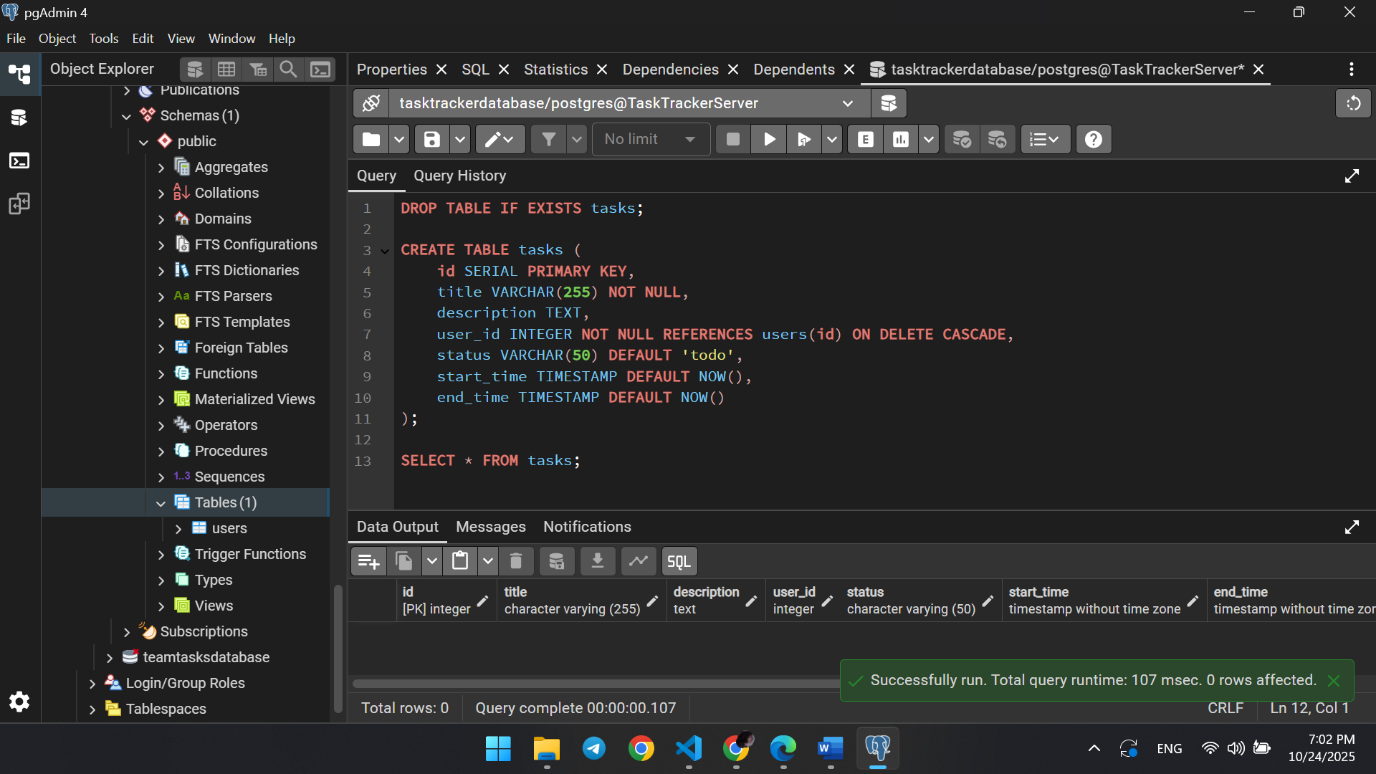
status- состояние (напр: новая*,* в процессе*,* выполнена)

start\_time- дата создания

end\_time- дата выполнения

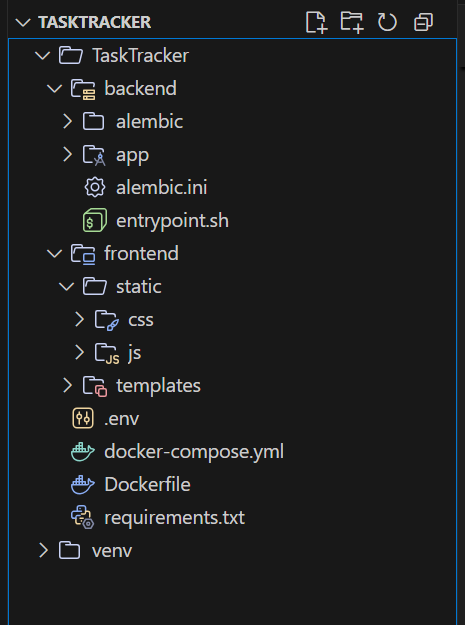
Допустим, мы хотим создать таблицы в PostgreSQL — я бы сделала это через **pgAdmin**. Можно использовать **Alembic** и делать миграции, но лично я предпочитаю создавать таблицы вручную прямо в pgAdmin, так я заодно тренируюсь в написании SQL-запросов.





1. Общая структура проекта

Создаю папку TaskTracker, внутри неё — две папки: Backend и Frontend.  
В Backend размещаю файлы для серверной части — основной файл приложения и модули API.  
В Frontend добавляю статические файлы (CSS, JS) и шаблоны (HTML) для отображения страниц.  
Снаружи создаю Dockerfile и docker-compose.yml, чтобы контейнеризировать проект с помощью Docker.



В качестве фреймворка использую **FastAPI**, базу данных выбираю **PostgreSQL**.  
Для работы с БД применяю **SQLAlchemy** как ORM, а для валидации данных — **Pydantic**.  
Миграции делаю с помощью **Alembic**. Аутентификация реализуется через **JWT**, для **Swagger UI** подключаю **OAuth2Bearer**, а на сайте авторизация работает через **cookie**.  
Для фронтенда использую **HTML**, **CSS** и **JavaScript**.

1. Короткое описание логики авторизации и работы с задачами.

Авторизация работает так: пользователь входит в систему или регистрируется. После успешного входа или регистрации ему выдаётся JWT-токен, который кодирует данные пользователя.Когда нужно получить данные пользователя, токен декодируется — так через зависимости можно определить, кто именно вошёл в систему.Для этого используется OAuth2 Bearer, который помогает понять, является ли пользователь администратором или обычным пользователем.

Пароли в базе данных хранятся в зашифрованном (хэшированном) виде — это делается для безопасности, чтобы реальные пароли нигде не отображались.

После входа пользователь видит только свои задачи. Он может создавать задачи с указанием названия, описания, даты начала, даты окончания, а также статуса — например, в процессе или выполнено.Пользователь может просматривать детали задачи, редактировать её или удалять, но только свои собственные задачи.

1. Пример структуры API

post - /users/create-user - регистрация  
post - /users/login - вход через форму  
post - /users/api/login - вход через API, получение токена  
get - /users/logout - выход  
get - /tasks/ - просмотр дашборда с задачами  
post - /tasks/create - создание задачи через форму  
get - /tasks/{task\_id} - просмотр деталей задачи  
post - /tasks/{task\_id}/edit - обновление задачи через форму  
post - /tasks/{task\_id}/mark-done - отметить задачу как выполненную  
post - /tasks/{task\_id}/delete - удалить задачу