

# NashSGO

авторы:

Никита Суриков @syricoff

Михаил Рыбалков @erytw

## Цель

Целью проекта является реализация бота на python с использованием aiogram для работы с telegram, неофициального API для Сетевого Города - netschoolapi. База данных - sqlite, взаимодействие с ней через sqlalchemy

## Возможности бота

- Авторизация школьника в системе СГО
- Просмотр расписания
- Просмотр предстоящих домашних заданий
- Просмотр оценок за день
- Создание отчетов об успеваемости

## Реализация

### 1. Работа с Сетевым Городом

В папке sgo находятся 5 файлов. 2 из них — служебные, незаменимы при отладке(салатовые), остальные три необходимы для работы проекта, при этом импортируется извне только data\_wrapper:

```
|— constants.py
|— data_wrapper.py
|— netschool.py
|— personal.py
|— tmp.py
```

Назначения файлов:

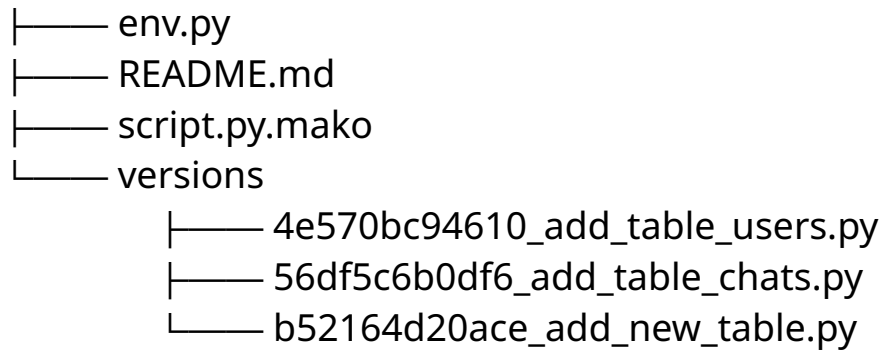
- **constants.py** — хранит ссылки, список символов, базовые сообщения об ошибках
- **netschool.py** — Работает с сетевым городом посредством библиотеки netschoolapi, методы его класса возвращают объекты типов библиотеки, Авторизация во всех методах происходит с помощью декоратора, для всех данных можно установить временные границы с помощью аргументов. На данный момент используются только стокковые границы, связанные с datetime.datetime.now()
- **data\_wrapper.py** — Прослойка между ботом и netschool. Преобразует данные в человекочитаемый формат, формирует ошибки. Высокий уровень кастомизации благодаря работе с переменными из constants и возможности выбирать границы. Импортируется извне, упрощая

работу не знакомого с вышеописанными файлами пользователя, не требуя разбираться в коде.

## 2. Работа с базой данных

В папке migrations находятся 3 файла и папка versions. Так же в корне проекта лежит файл alembic.ini в котором лежат конфигурации системы alembic.

migrations



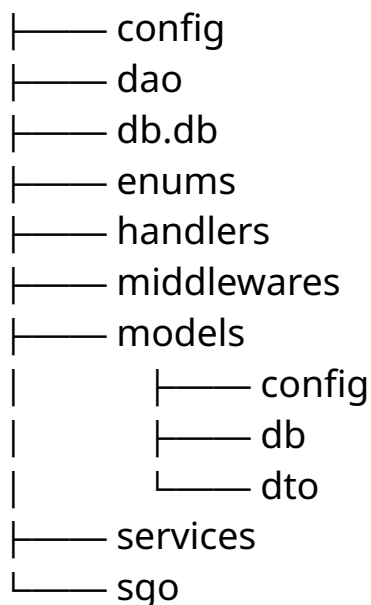
Назначения файлов:

- **env.py** — основной файл, в котором происходит работа с базой данных и создание миграций
- **script.py.mako** — шаблон для создания новых скриптов с таблицами
- **/versions** — папка, в которой лежат сгенерированные миграции. На данный момент там лежат пустые миграции, нужные для автосоздания БД и таблиц в ней.
- **README.md** — описание как работать с alembic

## 3. Работа с ботом

Основная работа с ботом происходит в папке app:

app



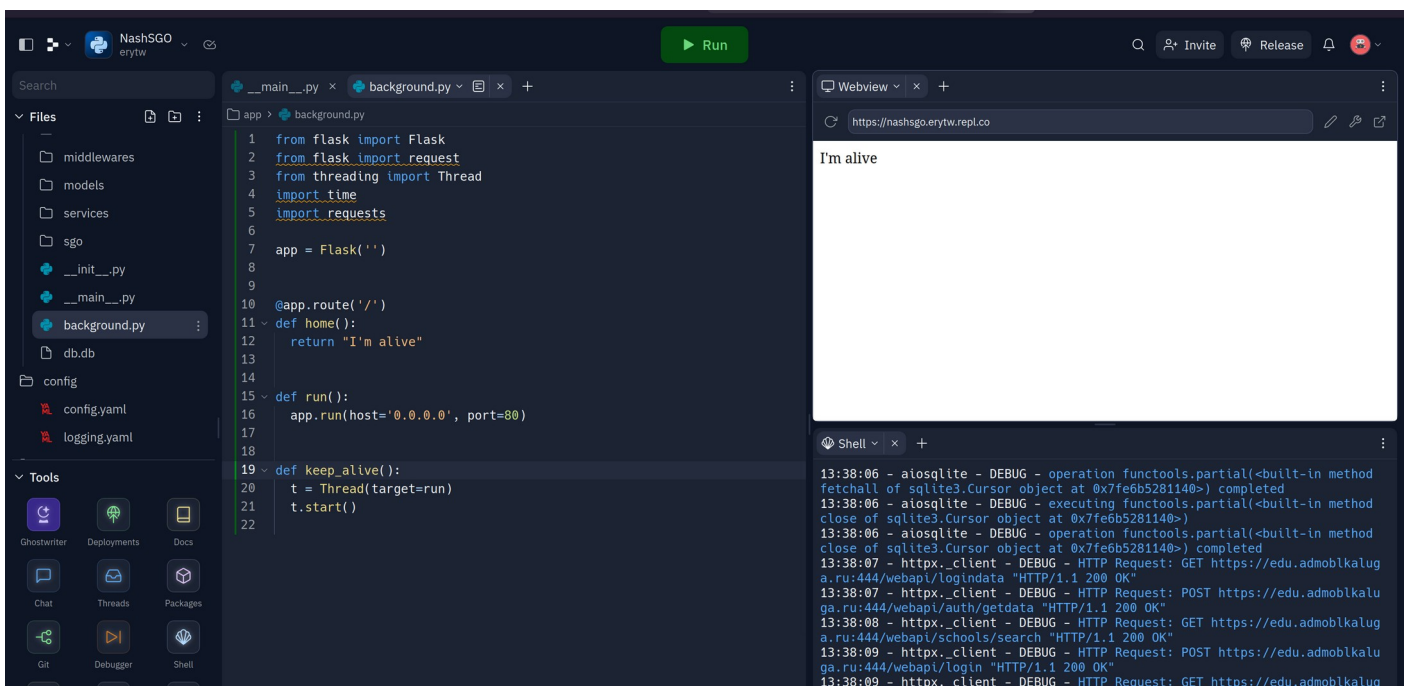
Назначения файлов:

- **config** — папка, в которой происходит работа с конфиг-файлом, полученная информация хранится в dataclass`е и используется при запуске бота
- **dao** — папка в которой хранятся модели работы с базой данных, для каждого типа данных своя модель, наследованная от BaseDao
- **enums** — папка с файлом, содержащим типы групп для работы с telegram api
- **handlers** — папка с основной логикой общения бота с человеком, содержит функции для получения команд пользователя и ответа на них
- **middlewares** — папка с работой мидлваря, который вклинивается в общение бота и пользователя, обрабатывает получаемые данные и добавляет их в базу данных при необходимости
- **models** — содержит папки, в которых лежат dataclass`ы, используемые программой для обмена данными
- **services** — папка с функциями для обмена данных между базой данных и её объектами и программы с dataclass

## Хостинг

В связи с тем, что в ноябре 2022 Heroku убрал бесплатный тариф, теперь хороший хостинг со свободой в плане библиотек и актуальной версией python найти сложно(Не платя деньги, разумеется). Теперь необходимы окольные пути.

Поэтому в качестве сервера было выбрано online-IDE replit.com, бесплатный тариф которого не включает возможность работы кода 24/7, и через пару минут после ухода в idle завершает выполнение программы. Однако решение есть. Было создано и запущено на отдельном потоке пустое flask-приложение:



The screenshot displays the Replit IDE interface for a project named 'NashSGO'. The left sidebar shows a file explorer with directories like 'middlewares', 'models', 'services', 'sgo', and files like '\_\_init\_\_.py', '\_\_main\_\_.py', 'background.py', 'db.db', 'config', 'config.yaml', and 'logging.yaml'. The main editor window shows the code for 'background.py', which is a simple Flask application. The code imports Flask, request, and threading, sets up a Flask app, and defines a home route that returns 'I'm alive'. It also includes a keep\_alive function that runs the app on a separate thread. The right sidebar shows a webview of the application running at 'https://nashsgo.erytw.repl.co', displaying 'I'm alive'. Below the webview is a shell window showing the application's output, including debug messages from aiosqlite and HTTP requests from a client.

```
1 from flask import Flask
2 from flask import request
3 from threading import Thread
4 import time
5 import requests
6
7 app = Flask('')
8
9
10 @app.route('/')
11 def home():
12     return "I'm alive"
13
14
15 def run():
16     app.run(host='0.0.0.0', port=80)
17
18
19 def keep_alive():
20     t = Thread(target=run)
21     t.start()
22
```

Webview: <https://nashsgo.erytw.repl.co>

I'm alive

Shell:

```
13:38:06 - aiosqlite - DEBUG - operation functools.partial(<built-in method fetchall of sqlite3.Cursor object at 0x7fe6b5281140>) completed
13:38:06 - aiosqlite - DEBUG - executing functools.partial(<built-in method close of sqlite3.Cursor object at 0x7fe6b5281140>)
13:38:06 - aiosqlite - DEBUG - operation functools.partial(<built-in method close of sqlite3.Cursor object at 0x7fe6b5281140>) completed
13:38:07 - httpx_client - DEBUG - HTTP Request: GET https://edu.admobkalug.ru:444/webapi/logindata "HTTP/1.1 200 OK"
13:38:07 - httpx_client - DEBUG - HTTP Request: POST https://edu.admobkalug.ru:444/webapi/auth/getdata "HTTP/1.1 200 OK"
13:38:08 - httpx_client - DEBUG - HTTP Request: GET https://edu.admobkalug.ru:444/webapi/schools/search "HTTP/1.1 200 OK"
13:38:09 - httpx_client - DEBUG - HTTP Request: POST https://edu.admobkalug.ru:444/webapi/login "HTTP/1.1 200 OK"
13:38:09 - httpx_client - DEBUG - HTTP Request: GET https://edu.admobkalug.ru:444/webapi/schools/search "HTTP/1.1 200 OK"
```

А так же задача на uptimerobot.com, которая отправляет запрос на наш Flask-сервер <https://nashsgo.erytw.repl.co> каждые 5 минут:

**Edit Monitor**

**Monitor Information**

Monitor Type: HTTP(s)

Friendly Name: NashSGO

URL (or IP): <https://nashsgo.erytw.repl.co>

Monitoring Interval: every 5 minutes

Monitor Timeout: in 30 seconds

Monitor SSL errors: ☒ **PAID**  
Available only in the paid plans. [Upgrade](#)

Enable SSL expiry reminders: ☒ **PAID**  
Available only in the paid plans. [Upgrade](#)

[Advanced Settings \(Optional\)](#) show/hide

**Select "Alert Contacts To Notify"**

Search

Type	Alert Contacts	(hide advanced options)
<input type="checkbox"/>	Mikhail Rybalkov - E-mail - erytw@outlook...	<a href="#">Settings</a>

Do you want to invite someone from your team?

[Invite team member](#)

**Select "Integrations To Notify"**

There are no integrations.

Search

**Select "Maintenance Windows"**

This is a **PAID** Plan feature for auto-disabling/enabling monitoring on pre-defined periods. [Upgrade](#)

[Close](#) [Save Changes](#)

Таким образом, запросы держат всё приложение, в том числе и бота, онлайн, а мы не платим ни копейки, и при этом пользуемся стабильным сервером с быстрым интернетом.

## Результат

Мы получили телеграм-бота с хостингом, предоставляющего пользователю(ученику) большинство необходимых в повседневной жизни функций и информации в удобном формате, позволяя сэкономить ценнейшее время школьника.

