ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Старший преподаватель		Е.О. Шумова
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №8
Описание классов и порождение объектов

по курсу: ОБЪЕКТНО ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	4134к		Столяров Н.С.
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

Цель работы

Научиться на практике применять паттерны проектирования.

Название проекта: WhatsApp Cloud API BOT

Суть его заключается в автоматизации отправки и получения сообщений от клиентов с последующей загрузкой итоговых данных в БД (1С).

Что реализовано

1. Собственное API для взаимодействия с ботом через 1С
2. Админ-панель для обслуживания и дебага
3. Локальная БД для хранения данных до последующего чтения их из 1С или иных действий
4. Набор методов для взаимодействия с API WhatsApp Cloud (META)

Сущности

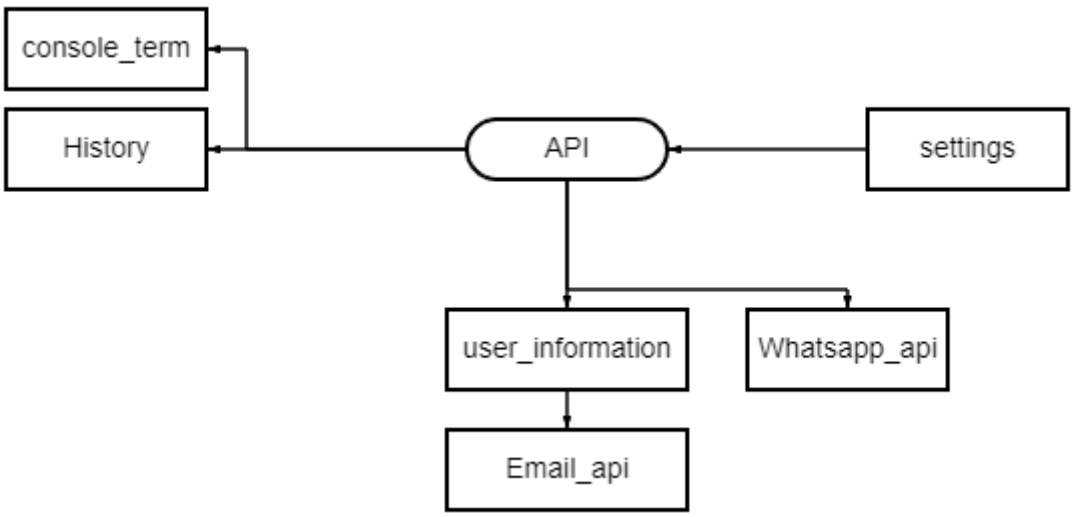
- Клиент

Phone (номер телефона)	Confirm (подтвердил ли пользователь свою запись)	unsubscribe (отписался ли пользователь от расслки)

- Пользователь (для авторизации в админ-панель)

Id (индификатор пользователя)	name (имя пользователя)	password (пароль)	key (ключ для шифрования сессии)

Принципиальная схема работы API



Набор запросов для взаимодействия с API

- ### Настраиваемый запрос

[GET or POST]

`<server-addr>/custom_request?token=<1>&phone=<2>&sample=<3>&params={[<4>]}`

1. токен
2. номер телефона (без +7 и 8)
3. название шаблона
4. параметры шаблона. Передаются в виде json массива {["hello", "world", "test"]}

Вернёт "OK"

- ### Уведомление для отзыва

[GET or POST]

`<server-addr>/review?token=<1>&phone=<2>&name=<3>&date=<4>`

1. токен
2. номер телефона (без +7 и 8)
3. ФИО клиента
4. дата посещения (не обязательно)

Вернёт "OK"

- ### Уведомление о приёме

[GET or POST]

`<server-addr>/appointment?token=<1>&phone=<2>&doctor=<3>&date=<4>&time=<5>`

1. токен
2. номер телефона (без +7 и 8)
3. ФИО доктора
4. дата приём
5. время приёма

Вернёт "OK"

- Уведомление о дне рождении

[GET or POST]

```
<server-addr>/happy_birthday_client?token=<1>&phone=<2>&name=<3>&date=<4>
```

1. токен
2. номер телефона (без +7 и 8)
3. ФИО клиента
4. дата др (в формате день.месяц)

Вернёт "OK"

- Получение данных с базы

[GET or POST]

```
<server-addr>/get_all_data?token=<1>&type=<2>
```

1. токен
2. Тип запрашиваемой информации

   confirm - люди которые подтвердили запись

   unsubscribe - люди котоые отписались от рассылок

Вернёт json { 'phones': [<list>] } или если нет людей то None

---

- Получить данные из базы о конкретном номере

[GET or POST]

```
<server-addr>/get_data?token=<1>&phone=<2>
```

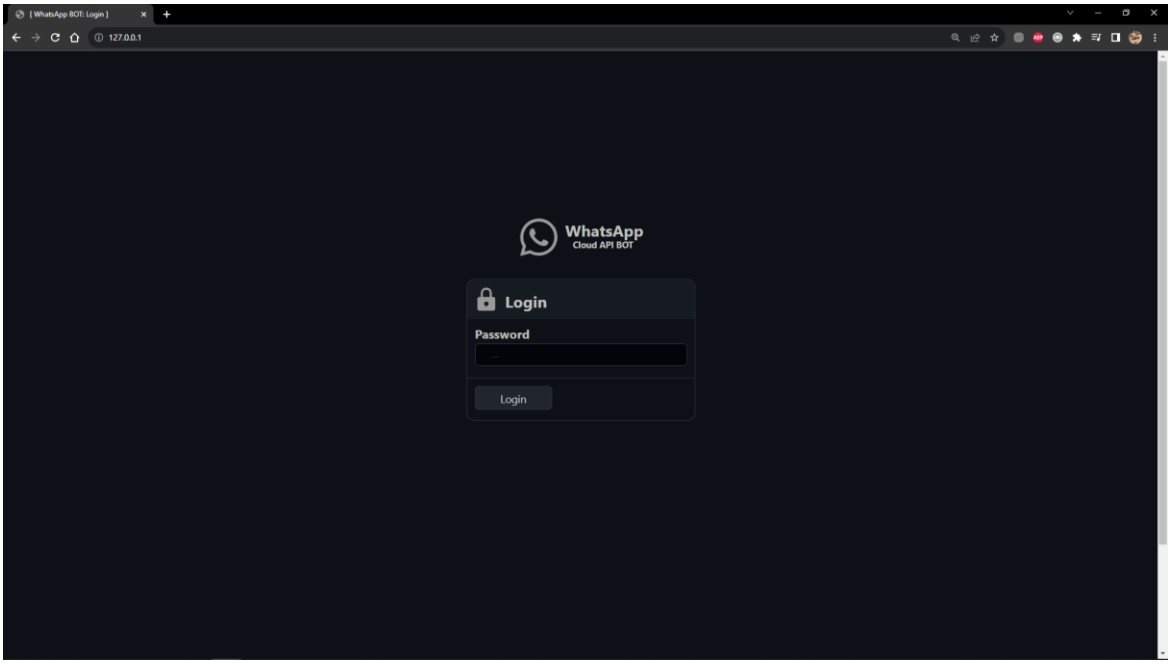
1. токен
2. номер телефона (без +7 и 8)

Вернёт json { "appointment": [None, True, False], "unsubscribe": [None, True] }

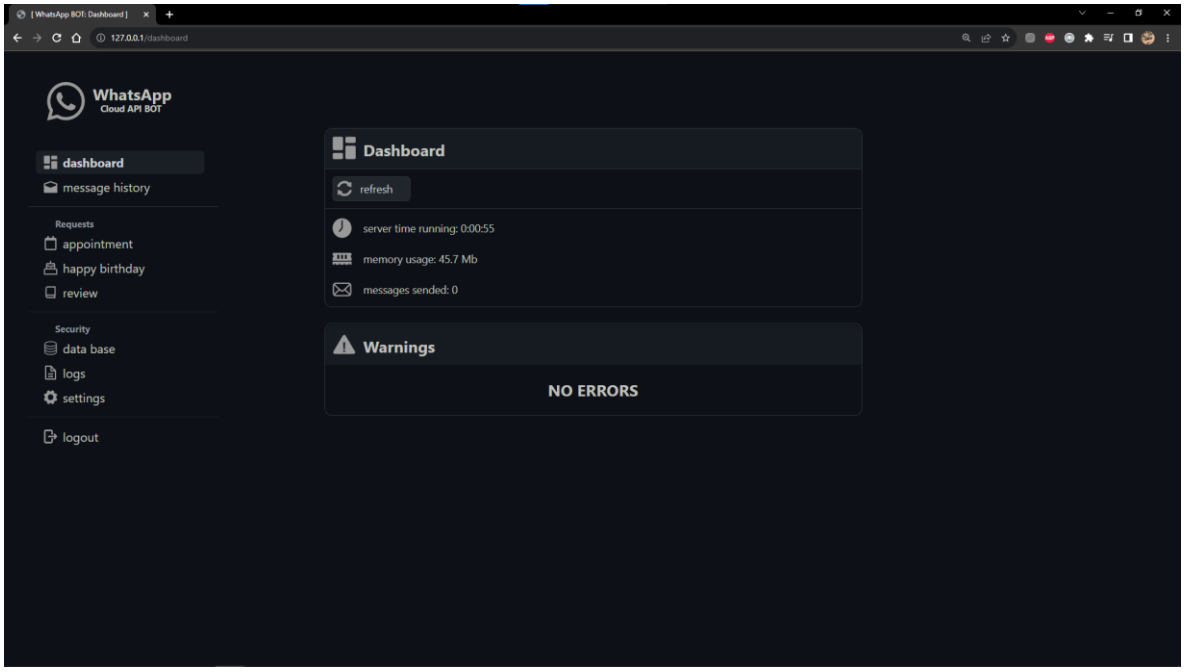
Ошибки которые могут вернуться после запроса

- "ERROR TOKEN" - Неверный токен.
- "WEB DASHBOARD OFF" - Панель отключена (можно включить через файл настроек).
- "SERVER ERROR" - Призыв к избиению создателя бота.

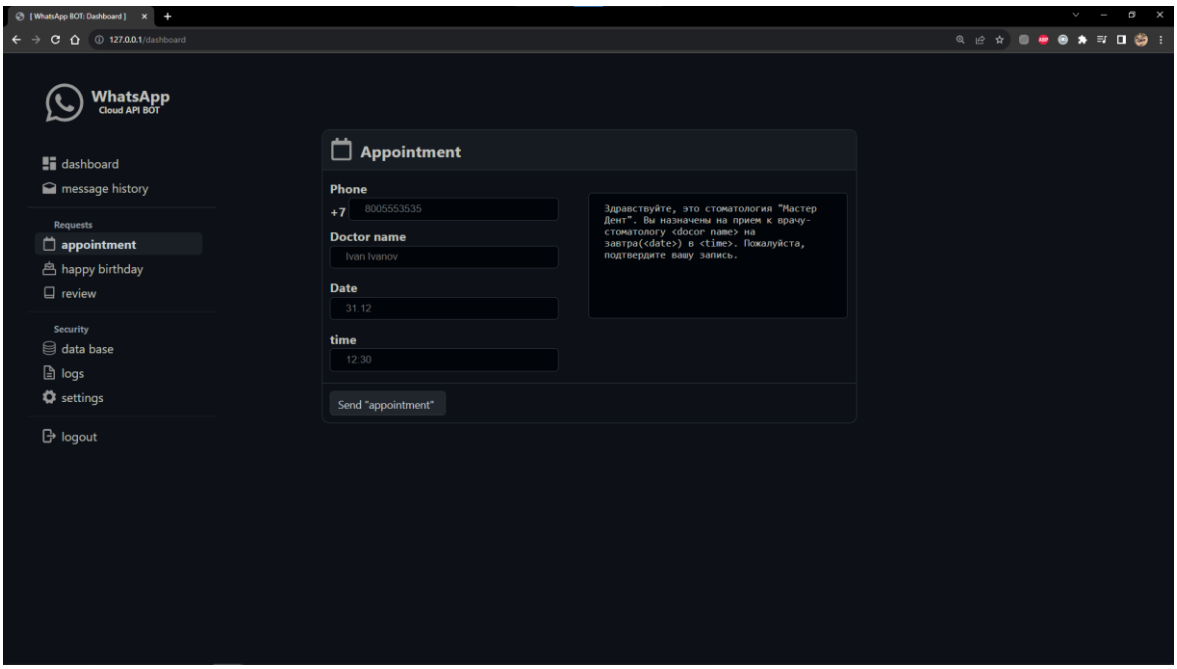
Скриншоты



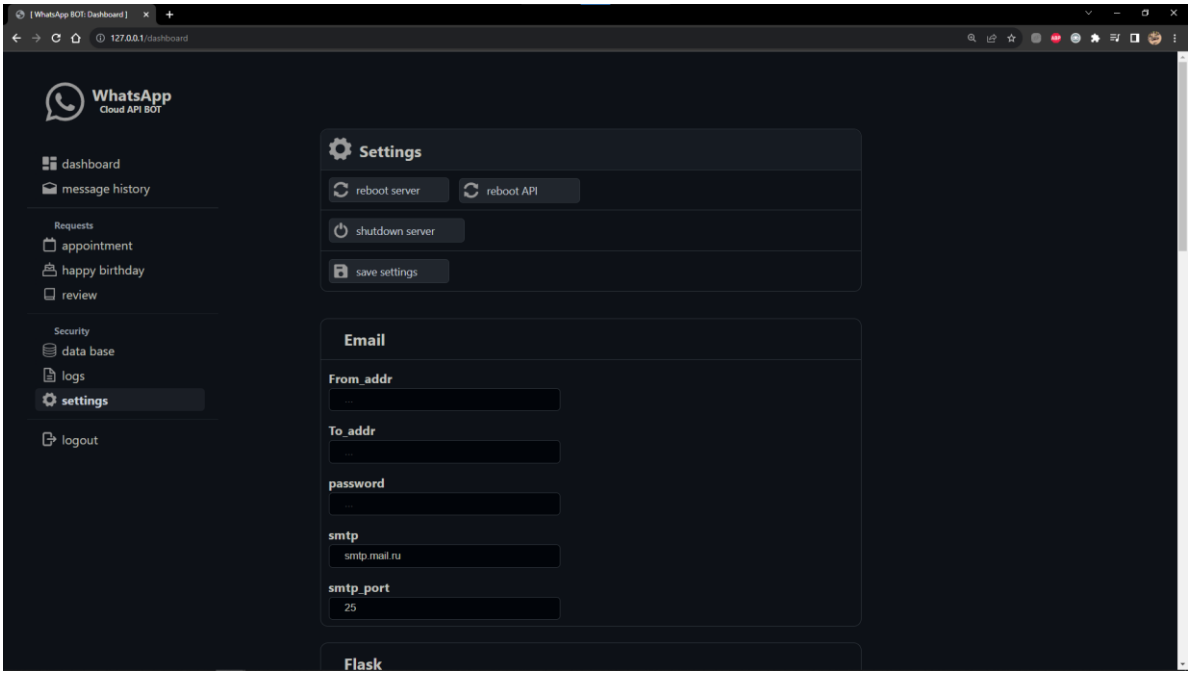
Страница авторизации



Главная страница (время работы сервера, объём загруженной оперативной памяти, количество отправленных сообщений и список непредвиденных ошибок)



Пример одного из запросов. Отправка клиенту уведомления о записи



Страница настройки серверной части

Листинг и описание важных элементов

Локальная БД. Создаёт и управляет разными потоками для проверки и удаления пользователей у которых просрочилась запись.

```
import os
import json
import threading
import time

def save_dict(dict, name):
    json.dump(dict, open(str(name) + '.json','w'))
```

```
def read_dict(name):
    with open(str(name) + '.json', encoding='utf-8') as fh:
        data = json.load(fh)
    return data

# типа БД для хранения данных о номерах
class user_information():
    def __init__(self, time, console_term, time_add_hours=36, path='base'):
        self.time = time
        self.console_term = console_term

        self.path = 'base'
        self.error_promt = 'USER INFORMATION: '

        self.data = {}
        self.unsubscribe_data = {}
        self.review = {}
        self.happy_birthday = {}

        self.read()

        self.run_check = True
        self.time_add_hours = time_add_hours

        self.thread = threading.Thread(target=self.check_time)
        self.thread.daemon = True
        #self.thread.start()

    def check_time(self):
        self.console_term.print(str(self.error_promt) + "Thread start", 0)
        while self.run_check:
            try:
                ################ Уведомления о приёме
                delete_phone_list = []
                for phone in self.data:
                    if (not self.data[phone][0] and
self.time.check_data_for_base(self.data[phone][1])):
                        delete_phone_list.append(phone)

                for phone in delete_phone_list:
                    self.data.pop(phone)
                    self.console_term.print(str(self.error_promt) + "Deleted
phone " + phone, 0)

                ################# Запись по случаю дня рождения
                delete_phone_list = []
                for phone in self.happy_birthday:
                    if
(self.time.check_data_for_base(self.happy_birthday[phone])):
                        delete_phone_list.append(phone)

                for phone in delete_phone_list:
                    self.happy_birthday.pop(phone)
                    self.console_term.print(str(self.error_promt) + "Deleted
phone happy birthday client " + phone, 0)

                ################# Уведомления о отзывах и их стадиях
                delete_phone_list = []
                for phone in self.review:
                    if
(self.time.check_data_for_base(self.review[phone][1])):
                        delete_phone_list.append(phone)
```

```
                for phone in delete_phone_list:
                    self.review.pop(phone)
                    self.console_term.print(str(self.error_promt) + "Deleted
phone review " + phone, 0)

                # сохраняем
                self.save()

            except Exception as e:
                self.console_term.print(str(self.error_promt) + "Thread => "
+ str(e), 3)

            time.sleep(1)

    def save(self):
        data = {
            'confirm': self.data,
            'unsubscribe': self.unsubscribe_data,
            'review': self.review,
            'happy_birthday': self.happy_birthday
        }

        save_dict(data, self.path)

    def read(self):
        if not os.path.exists(self.path + '.json'):
            self.save()

        else:
            data = read_dict(self.path)

            self.data = data['confirm']
            self.unsubscribe_data = data['unsubscribe']
            self.review = data['review']
            self.happy_birthday = data['happy_birthday']

    # добавление номера в базу
    def add_user(self, phone):
        self.data[phone] = [
            False,
            self.time.get_data_for_base(self.time_add_hours)

        ]
        self.save()

        self.console_term.print(str(self.error_promt) + "Add phone(confirm) "
+ phone, 0)

    # обновление состояния записи у номера
    def update_state_user(self, phone):
        if phone in self.data:
            self.data[phone][0] = True

        self.save()
        self.console_term.print(str(self.error_promt) + "Update
phone(confirm) " + phone, 0)

    # добавление номера в базу отписавшихся от расслок
    def update_ubunsubscribe_data(self, phone):
        self.unsubscribe_data[phone] = True

        self.save()
        self.console_term.print(str(self.error_promt) + "Add
phone(unsubscribe) " + phone, 0)
```

```
    def update_review(self, phone, state=0):
        # state:
        # 0 - предложить пользователю оценить от 0 до 5
        # 1 - дать пользователю ввести отзыв в сообщении
        # -1 - удалить из списка

        if state == -1:
            if phone in self.review:
                self.review.pop(phone)

        elif state in [0, 1]:
            self.review[phone] = [
                state,
                self.time.get_data_for_base(self.time_add_hours)
            ]

            self.console_term.print(str(self.error_promt) + "Update state(" +
str(state) + ") " + phone, 0)

        else:
            self.console_term.print(str(self.error_promt) + "Error state(" +
str(state) + ") " + phone, 3)

        self.save()

    def add_happy_birthday(self, phone):
        self.happy_birthday[phone] = self.time.get_data_for_base(7)

        self.save()

        self.console_term.print(str(self.error_promt) + "Add
phone(happy_birthday) " + phone, 0)

    def get_update_review(self, phone):
        if phone in self.review:
            return self.review[phone][0]

        else:
            return False
```

Класс для взаимодействия с WhatsApp cloud API (отправка и чтение сообщений)

```
import requests
import json

class Whatsapp_api():
    def __init__(self, token, phone, console_term, history,
url='https://graph.facebook.com', version='13.0'):
        self.console_term = console_term
        self.history = history

        self.error_promt = 'WHATSAPP API: '

        self.token = token
        self.phone = phone

        self.messages_count = 0
        self.error_messages_count = 0

        # основная ссылка куда будут отправляется все запросы
        self.url = '%s/v%s/%s/' % (url, version, self.phone)
```

```
        # создаём сессию (чисто по приколу)
        self.session = requests.Session()

        # для авторизации
        self.header = {
            'Content-Type': 'application/json',
            'Authorization': 'Bearer %s' % self.token
        }

    # Отправка сообщения по шаблоу
    def send_message_sample(self, id, sample='', parameters=[]):
        # id - id пользователя
        # sample - имя шаблона
        # parameters - параметры шаблона

        try:

            # собираем json для сообщения
            data = {
                'messaging_product': 'whatsapp',
                'to': id,
                'type': 'template',
                'template': {
                    'name': sample,
                    'language': {
                        'code': 'en'
                    },
                    'components': [{
                        'type': 'body',
                        'parameters': parameters
                    }]
                }
            }

            # отправляем запрос
            res = self.session.post(
                self.url + 'messages',
                json=data,
                headers=self.header
            )

            self.console_term.print(str(self.error_promt) + "Send " +
str(sample) + "message to" + str(id) + "\n Return: " + str(res.content), 0)

            if 'error' in json.loads(res.content):
                self.error_messages_count += 1

            else:
                self.messages_count += 1
                self.history.add(0, 'Отправлено сообщение (%s) клиенту %s' %
(sample, id))

            # возвращаем ответ запроса
            return res.content

        except Exception as e:
            self.console_term.print(str(self.error_promt) + str(e), 3)

            return False

    # Изменение статуса сообщения на прочитано (Mark)
    def set_mark_message(self, message_id):
        # message_id - id сообщения. Получается черех вебхук
```

```
try:

    # собираем json
    data = {
        'messaging_product': 'whatsapp',
        'status': 'read',
        'message_id': str(message_id)
    }

    # отправляем запрос
    res = self.session.post(
        self.url + 'messages/',
        json=data,
        headers=self.header
    )

    # возвращаем ответ запроса
    return res.content

except Exception as e:
    self.console_term.print(str(self.error_promt) + str(e), 3)

    return False
```

Будущие цели проекта

- Перенос с WhatsApp на SMS
- Отправка статистики по email
- Собственная шаблонизация сообщений

Выводы

Я научился использовать ООП в настоящих задачах.