ГУАП

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ

Разработка приложения для организации взаимодействия объектов при заданных критериях

по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.	4134к		Столяров Н.С.
		подпись, дата	инициалы, фамилия

Санкт-Петербург
2023

Содержание

1. Постановка задачи

Данная курсовая работа по программированию направлена на разработку системы классов, которая будет описывать СМС уведомления. Основной целью проектирования является создание программного продукта, который позволит уведомлять пользователей о записи куда либо где будет установлено это ПО.

1.1 Анализ предметной области

1.1.1. Предметной областью является отправка СМС уведомлений клиентам. Основными сущностями предметной области являются клиенты, комманда, и шаблон.

1.1.2. Словарь предметной области:

- СМС - Short Message Service (служба коротких сообщений)
- USSD
- AT

1.1.3. Функциональные требования:

- Логирование всего что происходит в системе (для анализа ошибок)
- Отправка СМС уведомлений
- Связь с 1С через запросы
- Отправка USSD запросов (например для получения баланса)
- Обеспечение безопасности при общении с 1С

2 Разработка классов

Для разработки иерархии классов мы начнем с выделения основных сущностей предметной области и определения классов, описывающих эти сущности. Затем мы определим управляющие классы и интерфейсные классы для организации взаимодействия между ними и с внешней средой. Ниже приведена детальная разработка иерархии классов

2.1 Иерархия классов

2.1.1. Комманда (Command):

- command (текст комманды для отправки)
- end (ожидаемое окончание)
- priority (приоритет)

Методы:

- Конструктор для создания

2.1.2. Клиент (User):

- phone (Номер телефона клиента)
- checked (флаг. Ответил ли пользователь на уведомление)

Методы:

- Конструктор для создания

2.1.3. Шаблон (Template):

- text(тест шаблона)
- args (переменные для подставления)

- default (массив значений по умолчанию)

Методы:

- Конструктор для создания

2.2 Управляющие классы:

2.2.1. Templates (Шаблоны):

- Поля:
  - Список всех шаблонов
- Методы:
  - Методы для добавления шаблонов
  - Метод для использования шаблона (подставки текста)

2.2.2. user_information (База клиентов):

- Поля:
  - Список всех клиентов
- Методы:
  - Методы для добавления/удаления клиентов
  - Метод для изменения статуса ответа клиента

2.2.3. SIM800L (Модуль для управления СМС):

- Поля:
  - Список всех комманд
- Методы:
  - Методы для выполнения АТ комманд
  - Методы для автоматизированной отправки сообщений

2.2 Патерны проектирования:

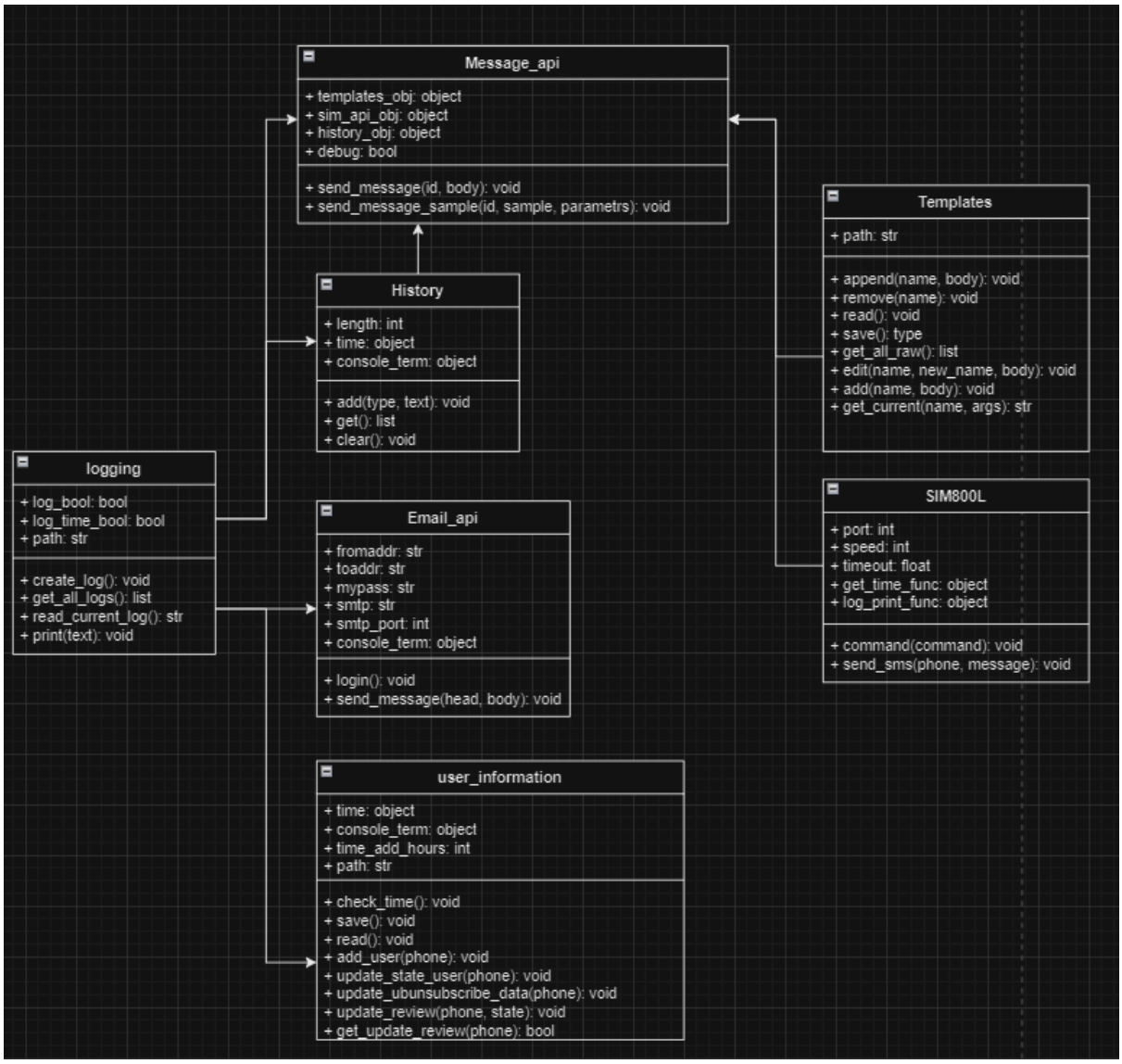
Наблюдатель (Observer):

Паттерн наблюдатель может быть полезен для оповещения пользователей о новых письмах, таким образом, они могут быть уведомлены о появлении новых сообщений в реальном времени.

Фабричный метод (Factory Method):

Для создания экземпляров писем и почтовых ящиков можно использовать фабричный метод, который позволяет инкапсулировать процесс создания объектов.

2.5 Рассмотрим полученную диаграмму:



Эта иерархия классов позволит эффективно управлять данными и операциями, связанными с СМС уведомлениями, а также обеспечит удобный интерфейс для управления всей этой системой.

3.1 Описание интерфейсов

Интерфейс представляет из себя набор различных пунктов меню (написанный через html). Таким образом все возможные действия можно аккуратно скомпоновать. По итогу были сделаны:

- Форма авторизации

- Различные формы для ввода сообщения (каждая форма представляет из себя шаблон сообщения)

- Просмотр текущей базы с клиентами
- Различные формы для просмотра аудита (история сообщений, логи, АТ терминал)

Код интерфейса

Авторизация

```
<!-- форма для авторизации -->
<div class="login_page border">

  <div class="name_area_border" style="width: 300px; height: 51px">
        <img class="icon" width="30" height="30" src="static/img/lock.svg">
        <h2>Login</h2>
  </div>
  <hr class="main_page_hr">

  <!-- поле ввода пароля -->
  <h3 style="margin: 0px 10px">Password</h3>
  <input id="password_input" class="input_border" style="left: 10px; width: 280px; margin-top: 0px;"
type=password placeholder="..." onkeypress="return login_enter(event)">

  <hr class="main_page_hr">

  <!-- кнопка для отправки пароля -->
  <div class="main_page_button" style="width: 100px; margin: 10px" onclick="login()">
        <p align="center">Login</p>
  </div>

</div>
```

Меню

```
<!-- меню -->
<div class="left_bar">
  <!-- логотип -->
  <div class="logo">
          <img class="icon logo_img" width="50" height="50" src="static/img/logo.svg">
          <h2 style="margin: -50px 60px;">Appointment</h2>
          <h5 style="margin: 45px 70px;">Message BOT</h5>
  </div>
  <!-- список кнопок -->
  <table class="menu_table left_bar_button_p">
          <!-- кнопка: главная панель -->
          <tr><th>
            <div class="left_bar_button" id="left_bar_dashboard" onclick="open_page('dashboard')">
                    <img class="icon" width="18" height="18" src="static/img/dashboard.svg">
                    <p align="left">dashboard</p>
            </div>
          </th></tr>
          <!-- кнопка: история сообщений -->
          <tr><th>
            <div class="left_bar_button" id="left_bar_history" onclick="open_page('history')">
                    <img class="icon" width="18" height="18" src="static/img/history.svg">
                    <p align="left">message history</p>
            </div>
          </th></tr>
          <!-- линия -->
          <tr><th>
            <hr class="left_bar_hr">
          </th></tr>
          <!-- комментарий -->
          <tr><th>
            <p class="left_bar_description" align="left">Requests</p>
          </th></tr>
```

```
        <!-- кнопка: шаблоны -->
        <tr><th>
          <div class="left_bar_button_selected" id="left_bar_templates"
onclick="open_page('templates')">
                <img class="icon" width="18" height="18" src="static/img/template.svg">
                <p align="left">Templates</p>
          </div>
        </th></tr>
        <!-- кнопка: уведомления -->
        <tr><th>
          <div class="left_bar_button_selected" id="left_bar_appointment"
onclick="open_page('appointment')">
                <img class="icon" width="18" height="18" src="static/img/appointment.svg">
                <p align="left">appointment</p>
          </div>
        </th></tr>
        <!-- кнопка: поздравление с днём рождения -->
        <tr><th>
          <div class="left_bar_button" id="left_bar_happy_birthday"
onclick="open_page('happy_birthday')">
                <img class="icon" width="18" height="18" src="static/img/happy birthday.svg">
                <p align="left">happy birthday</p>
          </div>
        </th></tr>
        <!-- кнопка: отзыв -->
        <tr><th>
          <div class="left_bar_button" id="left_bar_review" onclick="open_page('review')">
                <img class="icon" width="18" height="18" src="static/img/review.svg">
                <p align="left">review</p>
          </div>
        </th></tr>
        <!-- линия -->
        <tr><th>
          <hr class="left_bar_hr">
        </th></tr>
```

```
        <!-- комментарий -->
        <tr><th>
          <p class="left_bar_description" align="left">SIM800L</p>
        </th></tr>
        <tr><th>
        <tr><th>
          <div class="left_bar_button" id="left_bar_at_terminal" onclick="open_page('at_terminal')">
                <img class="icon" width="18" height="18" src="static/img/terminal.svg">
                <p align="left">AT terminal</p>
          </div>
        </th></tr>
        <!-- кнопка: USSD запрос -->
        <tr><th>
          <div class="left_bar_button" id="left_bar_ussd_request"
onclick="open_page('ussd_request')">
                <img class="icon" width="18" height="18" src="static/img/ussd.svg">
                <p align="left">USSD request</p>
          </div>
        </th></tr>
        <!-- кнопка: отправить сообщение -->
        <tr><th>
          <div class="left_bar_button" id="left_bar_send_message"
onclick="open_page('send_message')">
                <img class="icon" width="18" height="18" src="static/img/message.svg">
                <p align="left">Send message</p>
          </div>
        </th></tr>
        <!-- линия -->
        <tr><th>
          <hr class="left_bar_hr">
        </th></tr>
        <!-- комментарий -->
        <tr><th>
          <p class="left_bar_description" align="left">Security</p>
```

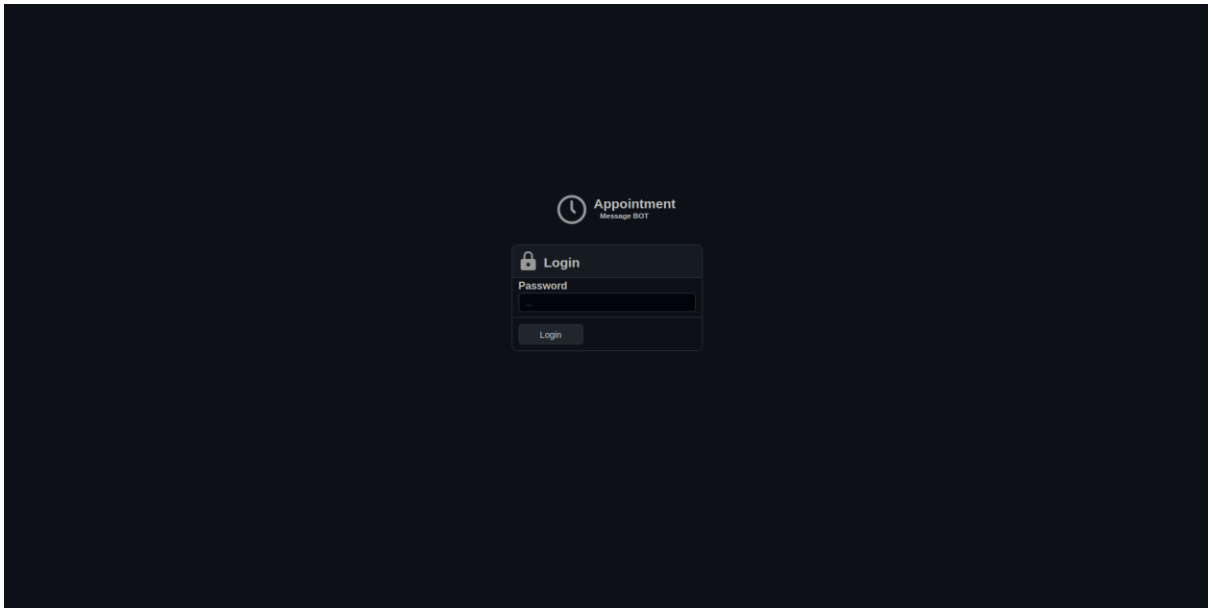
```
</th></tr>
<!-- кнопка: база данных -->
<tr><th>
  <div class="left_bar_button" id="left_bar_data_base" onclick="open_page('data_base')">
        <img class="icon" width="18" height="18" src="static/img/data base.svg">
        <p align="left">data base</p>
  </div>
</th></tr>
<!-- кнопка: логи -->
<tr><th>
  <div class="left_bar_button" id="left_bar_logs" onclick="open_page('logs')">
        <img class="icon" width="18" height="18" src="static/img/log.svg">
        <p align="left">logs</p>
  </div>
</th></tr>
<!-- кнопка: настройки -->
<tr><th>
  <div class="left_bar_button" id="left_bar_settings" onclick="open_page('settings')">
        <img class="icon" width="18" height="18" src="static/img/settings.svg">
        <p align="left">settings</p>
  </div>
</th></tr>
<!-- линия -->
<tr><th>
  <hr class="left_bar_hr">
</th></tr>
<!-- кнопка: выход -->
<tr><th>
  <div class="left_bar_button" id="left_bar_data_base" onclick="logout()">
        <img class="icon" width="18" height="18" src="static/img/logout.svg">
        <p align="left">logout</p>
  </div>
</th></tr>
```

```
  </table>
</div>
```

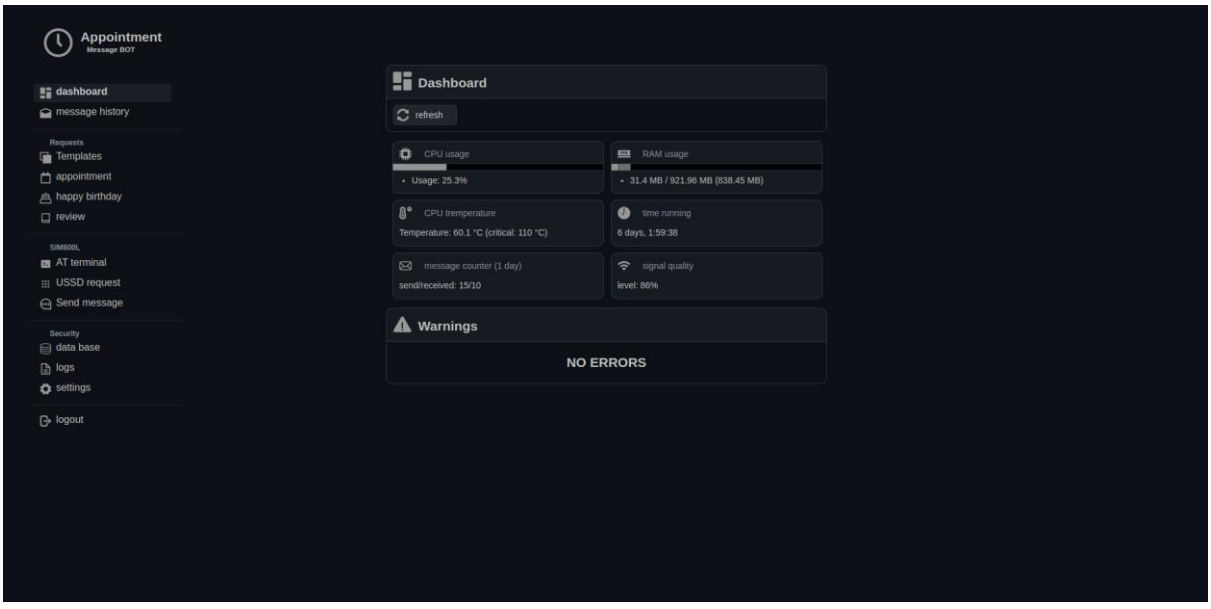
3.2 Разработка методов классов

Тестирование программы

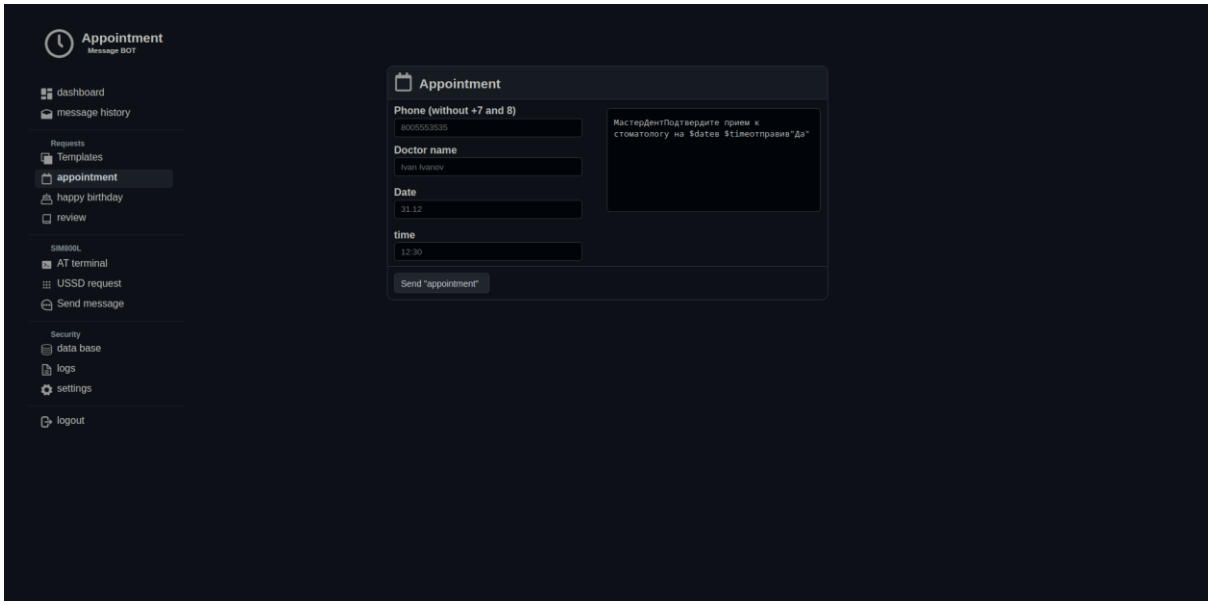
Страница авторизации



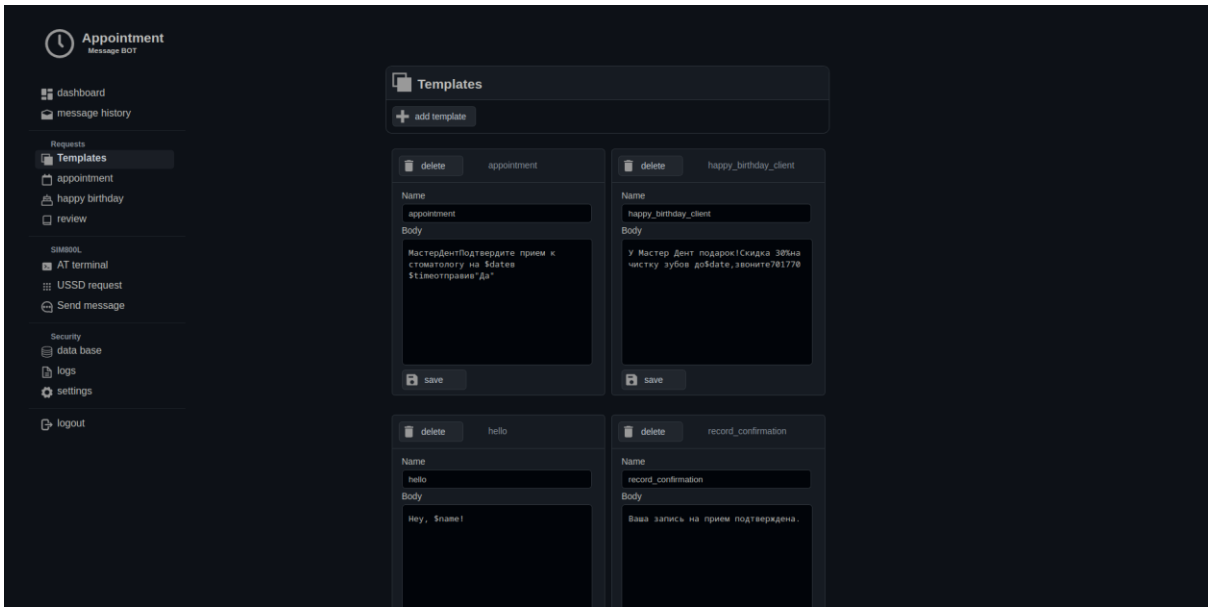
Главная панель



Отправка сообщения с шаблоном (appointment)



Редактор шаблонов



Приложение

sim_api.py

```
# модуль для работы с Sim800L через arduino
#
# пример инициализации класса:
#     sms_API = Sim800l("/dev/ttyUSB0", 9600)
#
# пример отправки сообщения:
#     sms_API.read_messages()
#
# пример кода для чтения сообщений:
#     while True:
#         if len(sms_API.messages) > 0:
#             for el in sms_API.messages:
#                 print(el)
#                 sms_API.send_sms(el["phone"], el["body"])
#             sms_API.messages = []

import time
import serial
import binascii
import threading

REMOVE_SYMBOLS = ["\n", "\r"]
AT_HISTORY_LENGTH = 100
MAX_USSD_RESULT = 5

class SIM800L:
    def __init__(self, port, speed=9600, timeout=0.1, get_time_func=None,
log_print_func=None):
        self.commands_queue = []
        self.command_end_flag = False
        self.messages = []
        self.ussd_result = []

        self.signal_quality = 100
        self.check_signal_flag = False
        self.signal_quality_delay = 60
        self.signal_quality_time = time.perf_counter() +
self.signal_quality_delay

        self.at_history = []
        self.get_time_func = get_time_func
        self.terminal_io = None
        self.log_print_func = log_print_func

        self.timeout = timeout

        self.ser = serial.Serial(
            port=port,
            baudrate = speed,
            parity=serial.PARITY_NONE,
            stopbits=serial.STOPBITS_ONE,
            bytesize=serial.EIGHTBITS,
            timeout=timeout
        )

        self.read_thread = threading.Thread(target=self.read_thread_func)
        self.read_thread.start()
```

```
        self.write_thread = threading.Thread(target=self.write_thread_func)
        self.write_thread.start()

        self.configure()
        self.read_messages()

    def append_history(self, text):
        for el in REMOVE_SYMBOLS:
            text = text.replace(el, "")

        self.at_history.append(text)
        while len(self.at_history) > AT_HISTORY_LENGTH:
            self.at_history.pop(0)

        self.log_print_func.print(text, 0)
        if self.terminal_io != None:
            self.terminal_io(text)

    def command(self, command, encode=True):
        self.commands_queue.append(
            (str(command).encode() if encode else command)
        )

    def read_thread_func(self):
        buf = {}
        while True:
            try:
                if self.signal_quality_time <= time.perf_counter() and not
self.check_signal_flag:
                    self.check_signal()

                out = self.ser.readline()
                try:
                    out = out.decode()
                except:
                    pass
                if len(out) > 0:
                    for el in REMOVE_SYMBOLS:
                        out = out.replace(el, "")
                    if (out in ["START", "OK", "ERROR", "> "]):
                        self.command_end_flag = True

                    if len(out) > 0:
                        self.append_history("[%s] >> %s" %
(self.get_time_func(), out))

                    out_splited = out.split(":")
                    if out_splited[0] == "+CMTI":
                        self.read_messages()

                    elif out_splited[0] == "+CUSD":
                        body = out_splited[1].split("\"")[1]
                        try:

self.ussd_result.append(binascii.unhexlify(body).decode("utf-16-be"))
                        except:

self.ussd_result.append(binascii.unhexlify(body).decode())

                        while len(self.ussd_result) > MAX_USSD_RESULT:
```

```
                            self.ussd_result.pop(0)

                    elif out_splited[0] == "+CMGL":
                        buf = {
                            "phone":
binascii.unhexlify(out_splited[1].split(",")[2].replace("\"", "")).decode()
                        }

                    if out_splited[0] == "+CSQ":
                        self.signal_quality =
int(out_splited[1].split(',')[0])
                        self.signal_quality_time = time.perf_counter() +
self.signal_quality_delay
                        self.check_signal_flag = False

                    elif len(buf) > 0:
                        try:
                            buf["body"] =
binascii.unhexlify(out).decode("utf-16-be")
                        except:
                            buf["body"] = binascii.unhexlify(out).decode()
                        self.messages.append(buf.copy())
                        buf = {}

            except Exception as e:
                print("Serial READ THREAD", e)

    def write_thread_func(self):
        while True:
            if len(self.commands_queue) and self.command_end_flag > 0:
                self.append_history("[%s] << %s" % (self.get_time_func(),
self.commands_queue[0].decode()))
                self.ser.write(self.commands_queue[0])
                self.commands_queue.pop(0)
                self.command_end_flag = False

            else:
                time.sleep(self.timeout)

    def reboot(self):
        self.command("AT+CFUN=1,1\r\n")

    def configure(self):
        self.command("ATZ\r\n")
        self.command("AT+CSCS=\"HEX\"\r\n")
        self.command("AT+CSMP=17,168,0,8\r\n")
        self.command("AT+CMGF=1\r\n")
        self.command("AT+GSMBUSY=1\r\n")
        self.check_signal()

    def check_signal(self):
        self.command("AT+CSQ\r\n")
        self.check_signal_flag = True

    def send_sms(self, phone, message):
        self.command("AT+CMGS=\"%s\"\r\n" % phone)
        self.command(binascii.hexlify(str(message).encode('utf-16-be')) +
chr(26).encode(), encode=False)
        self.delete_send()

    def send_ussd(self, code):
```

```
        self.command("AT+CUSD=1,\"%s\"\r\n" % code)

    def read_messages(self):
        self.command("at+cmgl=\"REC UNREAD\"\r\n")
        self.delete_read()

    def delete_send(self):
        self.command("AT+CMGDA=\"DEL SENT\"\r\n")

    def delete_read(self):
        self.command("AT+CMGDA=\"DEL READ\"\r\n")
```

user_information.py

```
import os
import json
import threading
import time

from app.dict_json import *

# 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.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:
            edited = False
            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) + "Delet-
ed phone " + phone, 0)
                    edited = True

                ################ 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) + "Delet-
ed phone review " + phone, 0)
                    edited = True

                # сохраняем
                if edited:
                    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
        }

        save_dict(data, self.path)

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

        else:
            try:
                data = read_dict(self.path)

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

            except:
                self.save()

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

        else:
            return False
```