**Министерство науки и высшего образования Российской Федерации**

**РОССИЙСКИЙ ХИМИКО-ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ**

**имени Д.И. Менделеева**

**Кафедра информационных компьютерных технологий**

*Дата регистрации\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Регистрационный № \_\_\_\_\_\_\_\_\_\_*

Дисциплина «Технологии программирования»

КУРСОВАЯ

**«Выполнение парсинга сайта с использованием языка программирования Python и его библиотек.»**

Выполнил (а): студент

2 курса, группа Кс-20

Шишко Роман Геннадьевич

Зачетная книжка № 18420

Проверил: Васецкий А.М.

Москва 2020

Оглавление

[Введение 3](#_Toc43279582)

[1. Алгоритм решения задачи 4](#_Toc43279583)

[2. Описание интерфейса 9](#_Toc43279584)

[3. Листинг 18](#_Toc43279585)

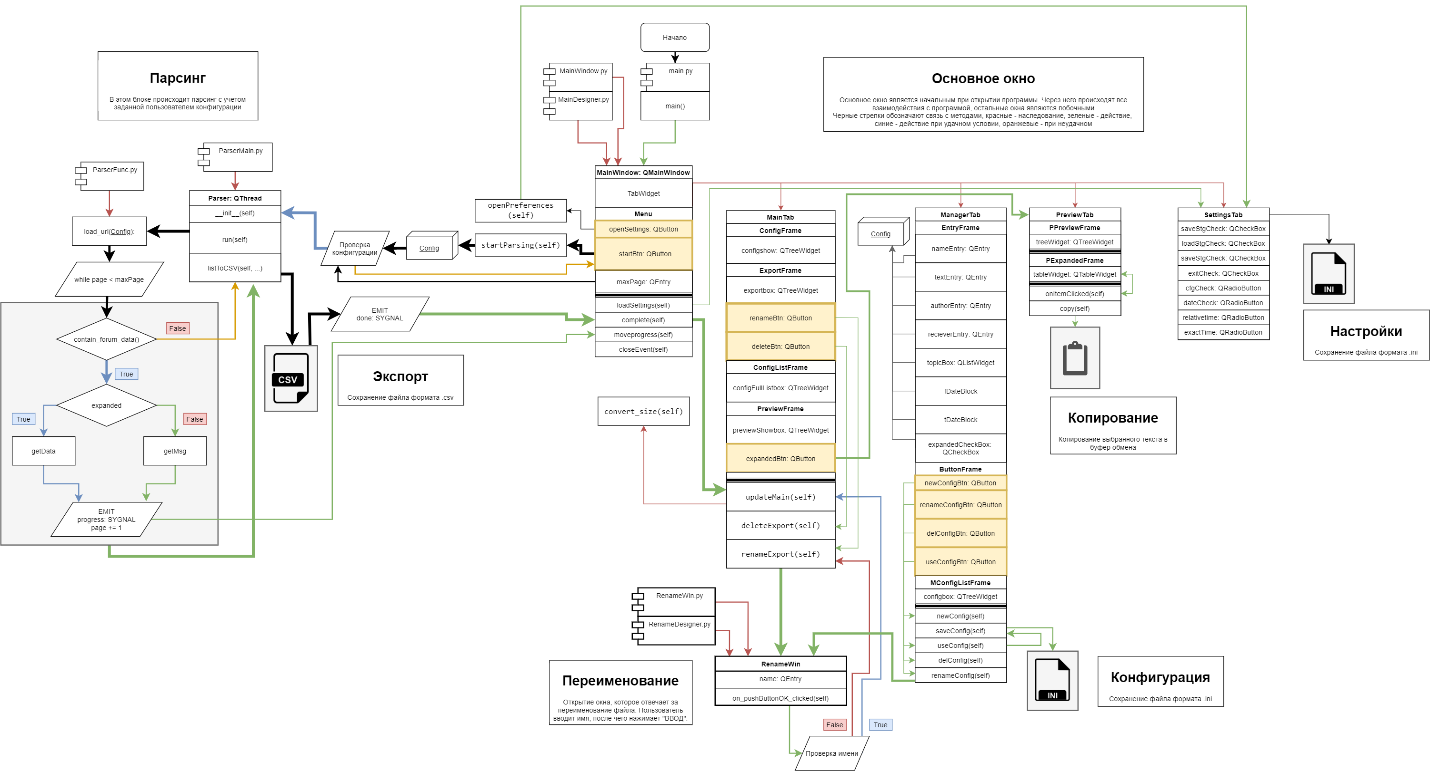
# Введение

Для парсинга какого-либо сайта, сперва определяется способ, который будет применяться для получения информации. В данном случае, в технических требованиях к заданию указано, что парсинг должен производиться по заданным пользователем поисковым запросам с использованием внутренней поисковой системы сайта. Помимо этого, программа должна быть снабжена: пользовательским интерфейсом, файлом конфигурации, файлом (файлами) вывода результатов поиска.

В данной работе будут использоваться файлы \*.ini, в качестве файлов конфигурации, и .csv, в качестве файлов результатов поиска.

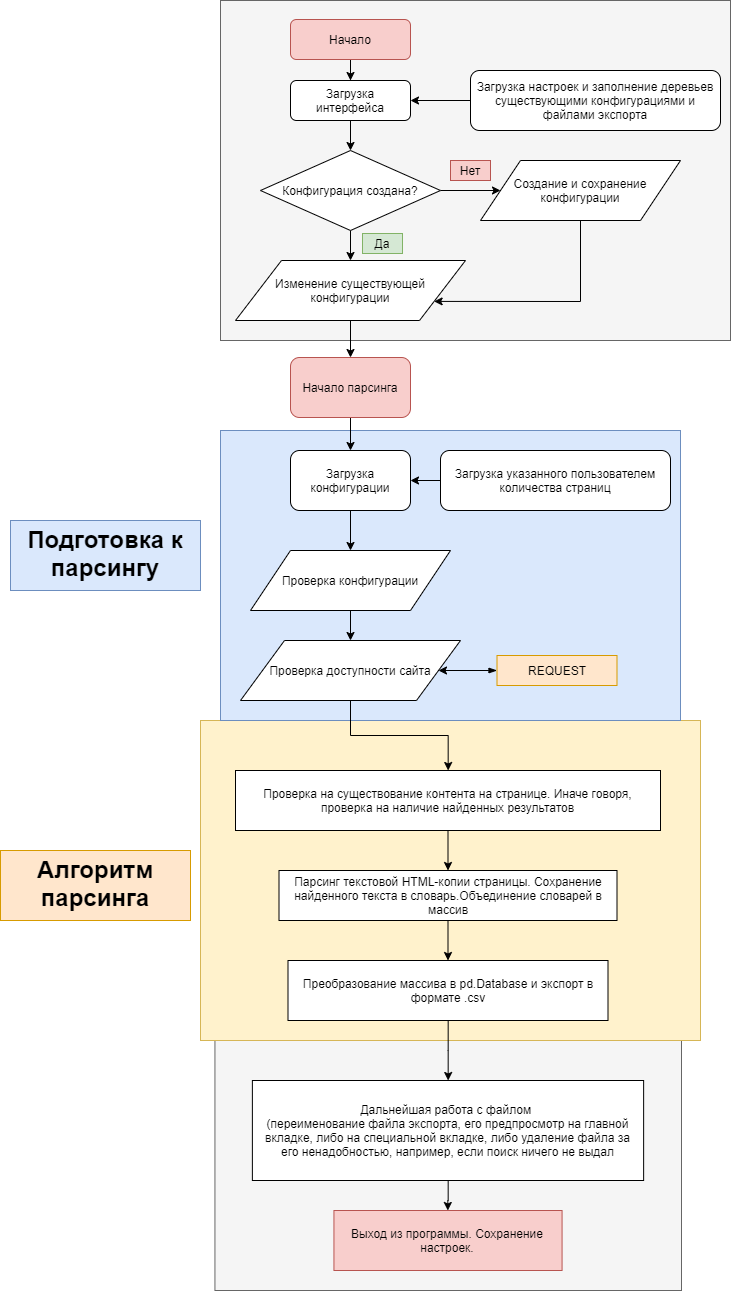
# 1. Алгоритм решения задачи

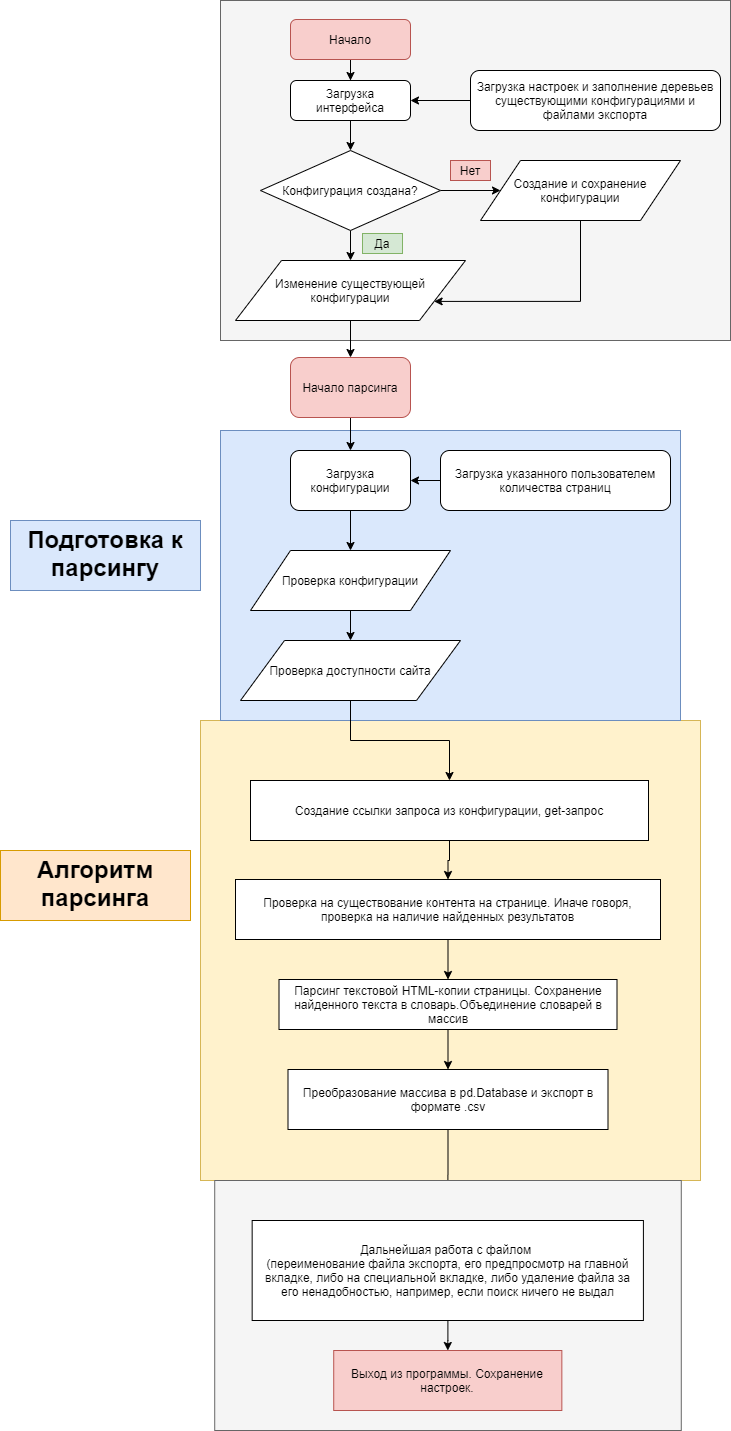
Полный алгоритм изложен в данной блок-схеме (приложение 1).

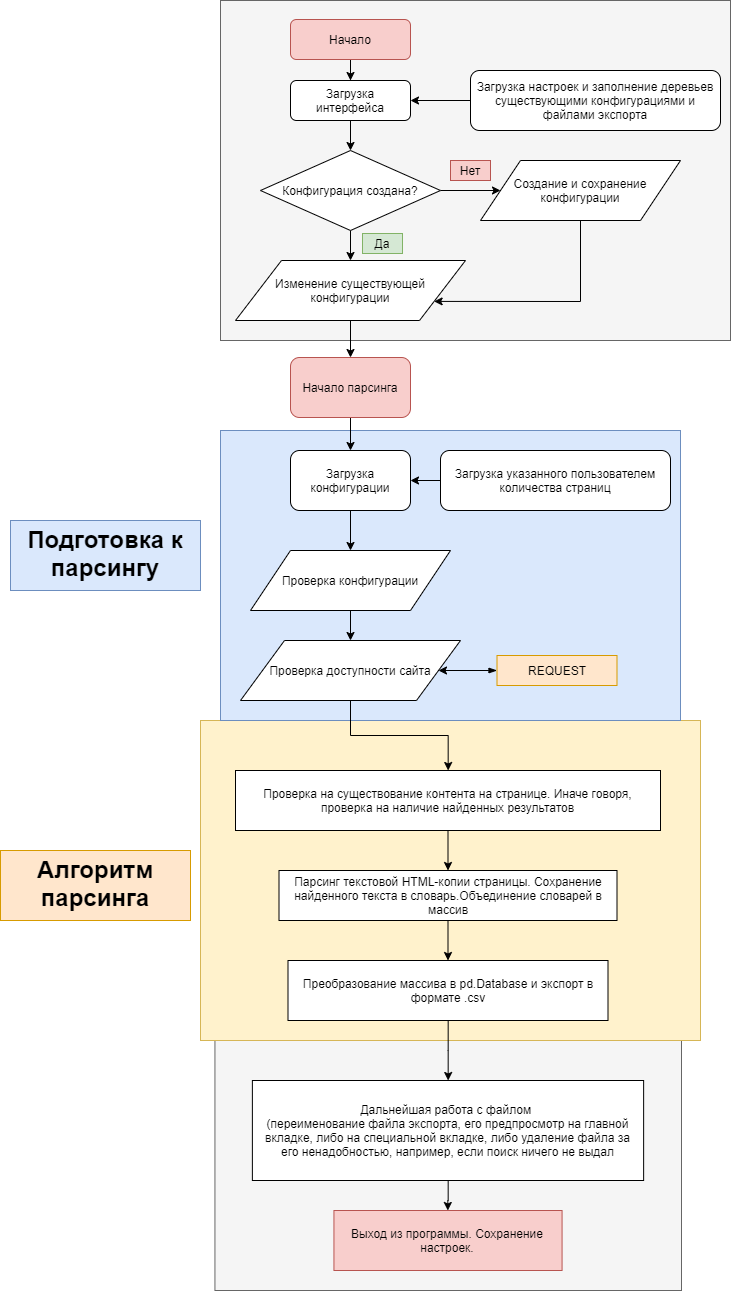


При первом запуске парсера пользователь не будет иметь ни конфигурации, ни экспортированных файлов результата парсинга. Таким образом, первое действие пользователя – зайти в «Менеджер конфигураций» и создать новую конфигурацию поискового запроса. Далее, пользователь волен изменить некоторые поля конфигурации прямо на главном экране. Следующим действием будет выбор конфигурации (кнопка «Использовать») и выбор количества страниц, которые будут парситься. При нажатии на кнопку «Начать парсинг» запускается главный алгоритм парсера. По окончании работы, заполнится индикатор прогресса и выведется уведомление пользователю.

Более подробное описание в принципиальной блок-схеме (приложение 2):

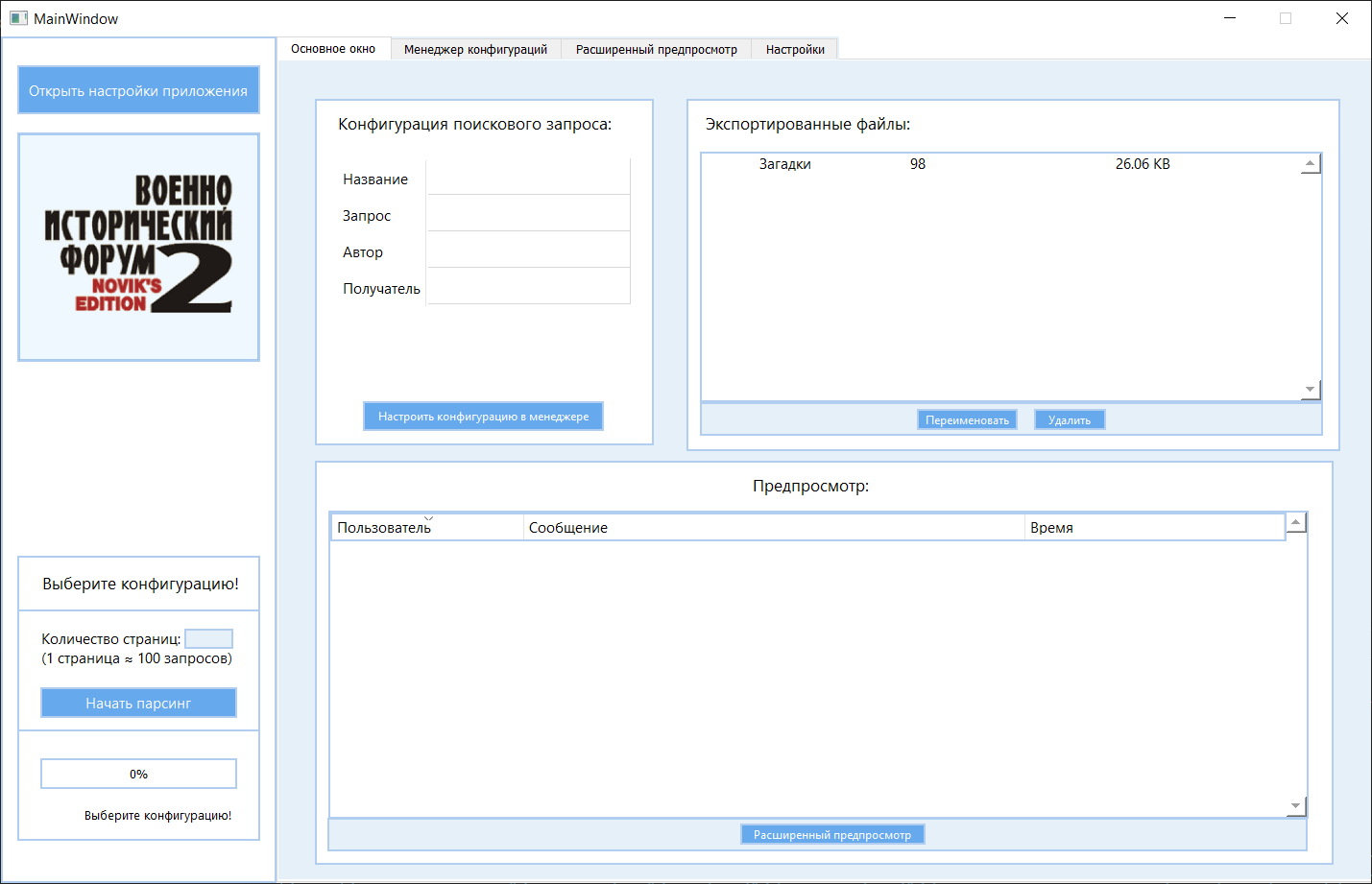




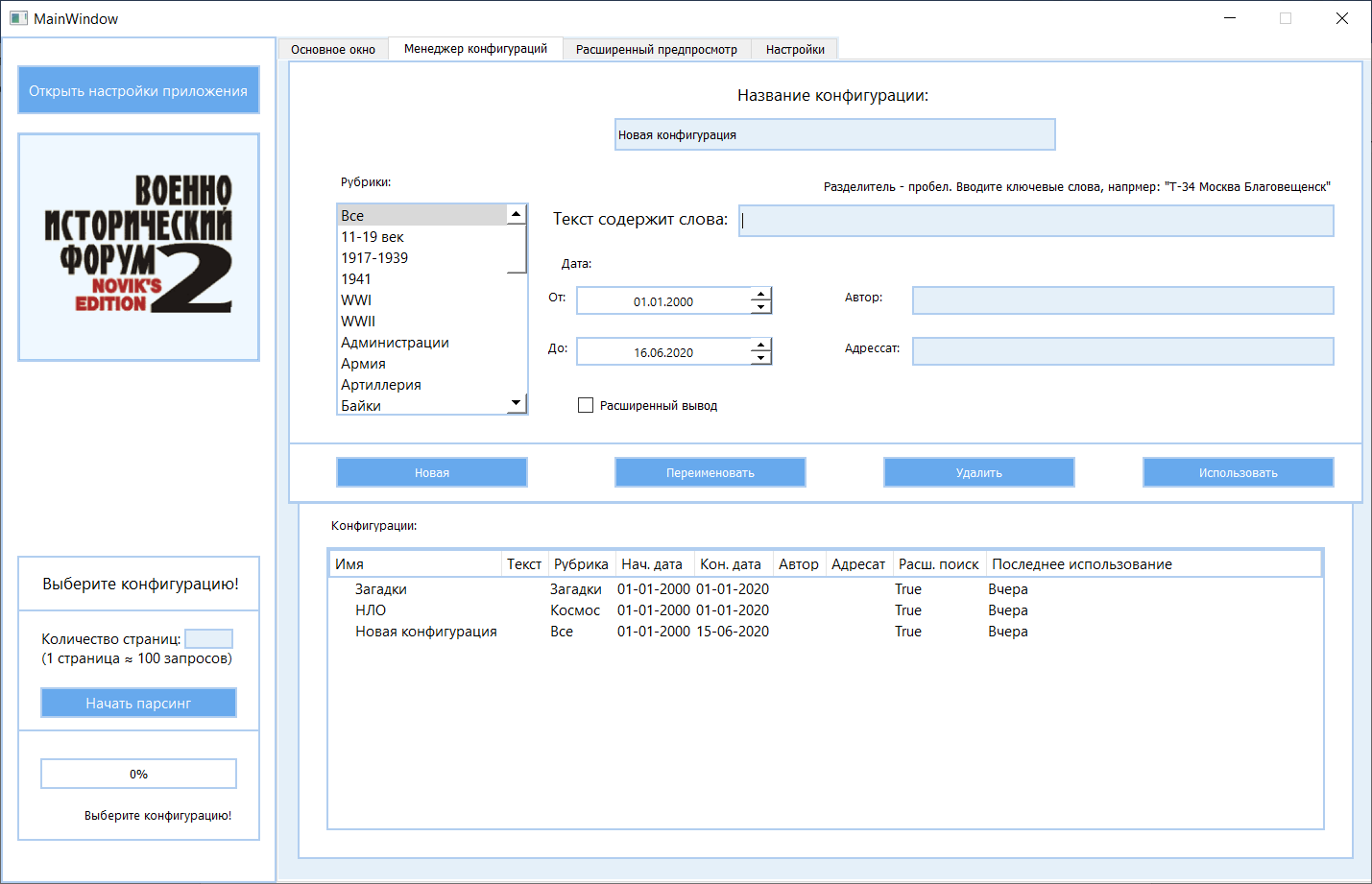


# 2. Описание интерфейса

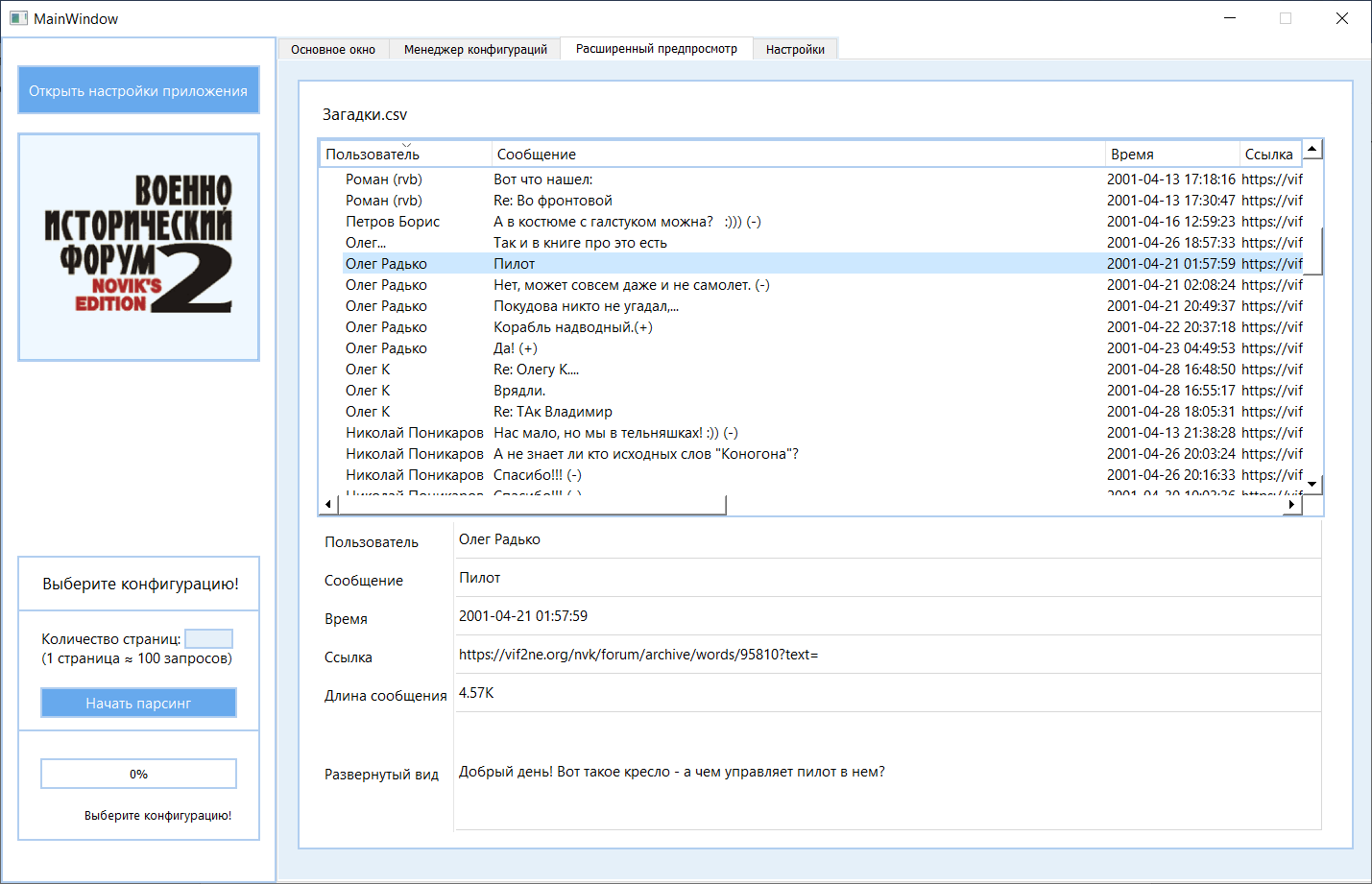
Главная вкладка программы:



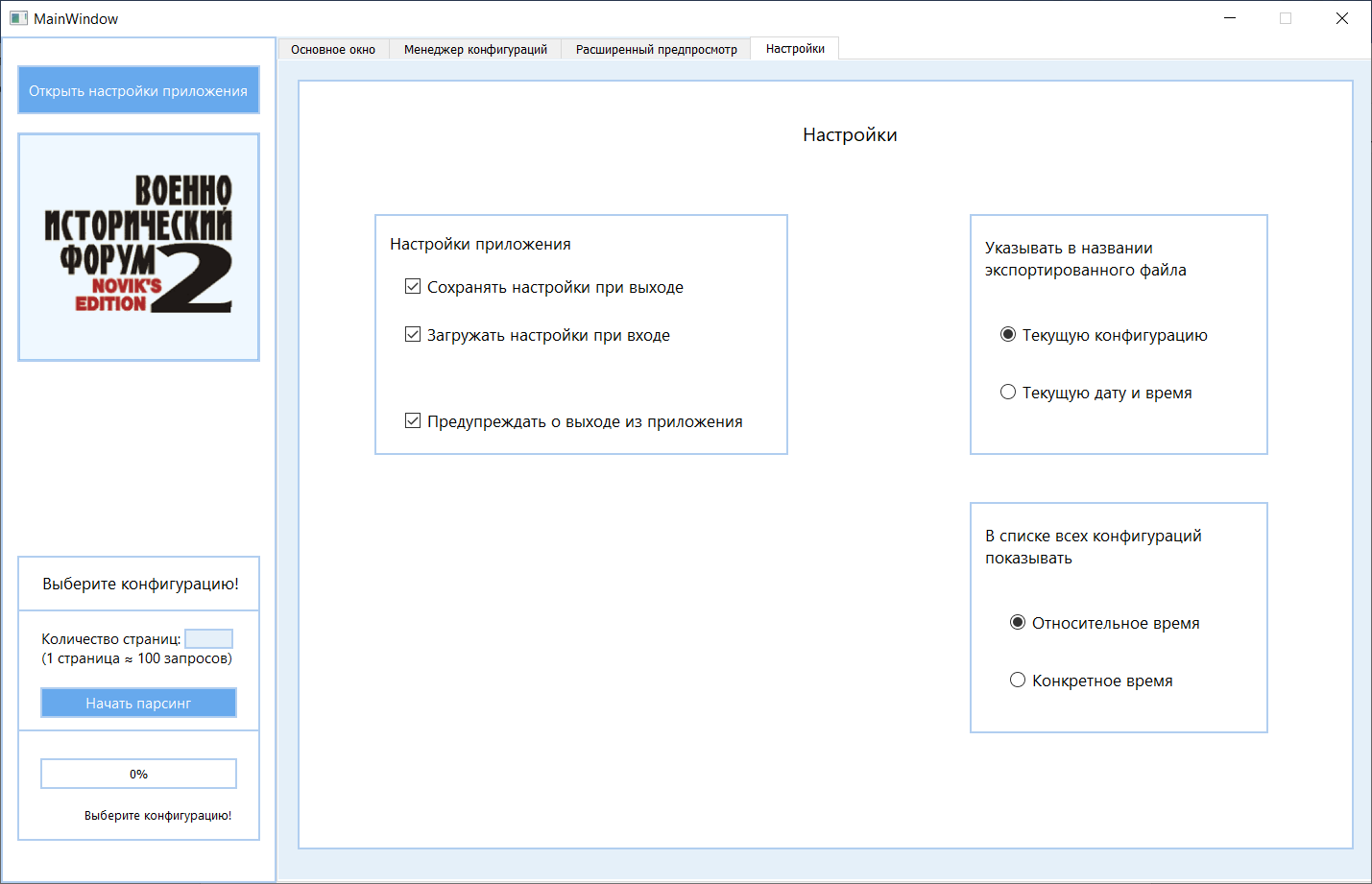
Вкладка настройки конфигурации поискового запроса:



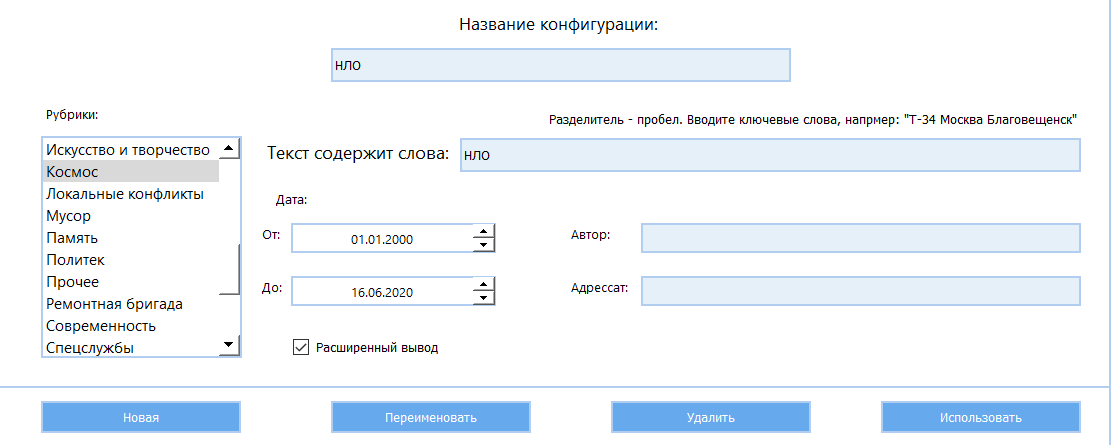
Вкладка предпросмотра:



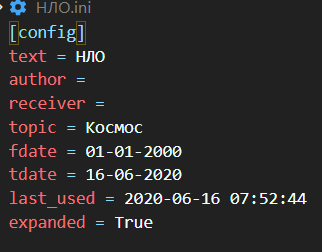
Вкладка настроек:



Пример запроса:



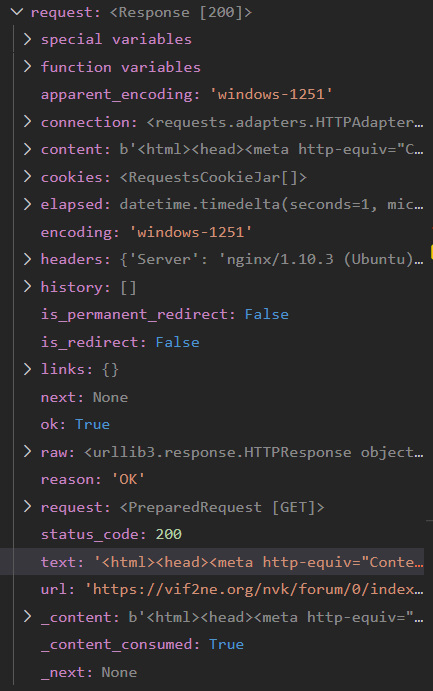
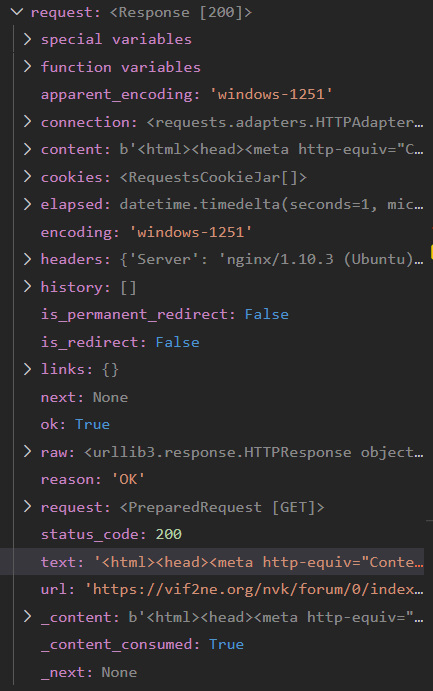
Сама конфигурация сохраняется в .ini файл в папке config и имеет вид:



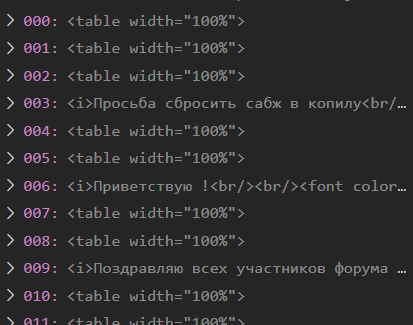
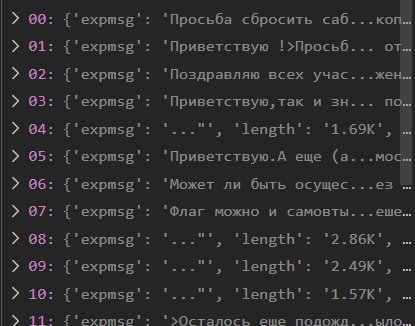
Из данной конфигурации формируется ссылка для запроса:

*https://vif2ne.org/nvk/forum/0/index/list?text=&topic=CAEEF1ECEEF1&fday=01&fmonth=01&fyear=2000&tday=16&tmonth=06&tyear=2020&wide=on&author=&toauthor=&from=1&to=100*

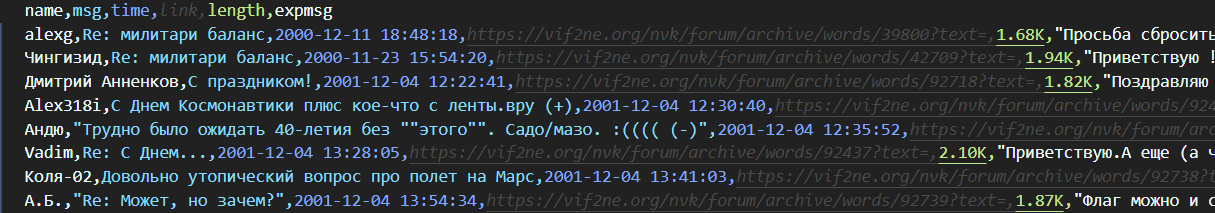
Далее при помощи request *= session.get(url),* получаем html-копию страницы через *request.text* из которой при помощи BeautifulSoup ищем и сохраняем в словарь нужные нам поля. Сам запрос в отладчике имеет вид:

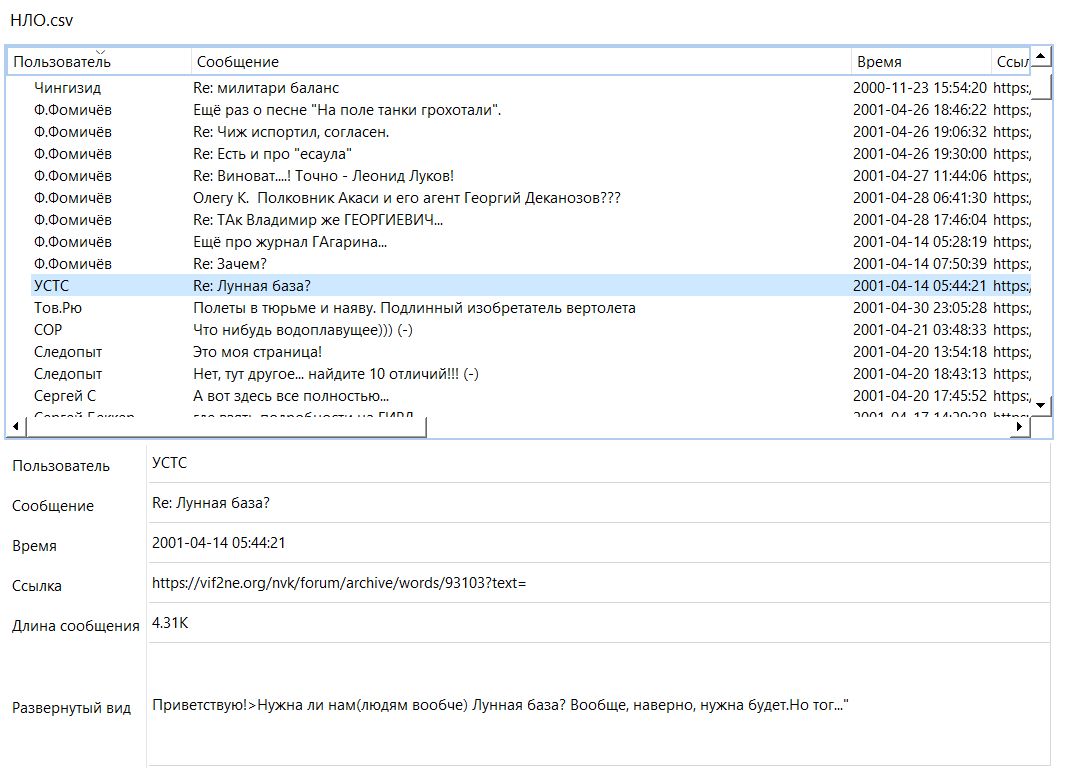
 

Так как все данные нужные нам находятся внутри тегов <table></table>, мы вытаскиваем все таблицы в отдельный массив, чтоб работать с ними, а не со всей html-страницей. После парсинга каждой таблицы из массива отдельно мы получаем массив словарей:

Результаты парсинга сохраняются в .csv файл в файле export, а затем их можно посмотреть прямо в программе. Удобство для пользователя в том, что пользователю не нужно заходить в папку и искать нужный файл. Помимо того, в программе реализовано копирование в буфер обмена: если посмотреть на вкладку расширенного предпросмотра, внизу будет табличка. Она заполняется при клике на какое-либо из полей таблицы, таким образом просто копируя текст, делая его более читаемым и структурированным, нежели в таблице. При двойном клике на любую ячейку таблицы (например, на ссылку), текст из нее будет копироваться в буфер обмена и выведется уведомление об этом пользователю.





В программе предусмотрена проверка файлов, таким образом, если файл поврежден, программа уведомит об этом пользователя.

# 3. Листинг

##### main.py

Основной модуль. Инициализирует класс из модуля MainWindow.py.

*import* sys

*import* PyQt5

*from* PyQt5 *import* QtWidgets

*from* PyQt5 *import* Qt

*from* MainWin *import* MainWindow

*import* configparser

*import* datetime

*import* glob

*import* os

*def* main():

*# ЗАПУСК ОСНОВНОГО ОКНА*

    app = QtWidgets.QApplication(sys.argv)

    window = MainWindow()

    window.show()

    app.exec\_()

*if* \_\_name\_\_ == '\_\_main\_\_':

    main()

##### MainWindow.py

Модуль класса основного окна.

*import* sys

*import* PyQt5

*from* PyQt5 *import* QtWidgets, Qt, QtCore

*from* designer *import* MainDesigner

*from* RenameWin *import* RenameWin

*from* parsing *import* parserMain

*import* configparser

*import* requests

*import* datetime

*import* glob

*import* math

*import* csv

*import* os

*def* relativePath(folder, name="", ftype=""):

*# ФУНКЦИЯ ВОЗВРАЩАЕТ ОТНОСИТЕЛЬНЫЙ ПУТЬ К ФАЙЛУ ИЛИ ПАПКЕ*

    path = os.path.abspath(os.path.join(os.path.dirname(

        "\_\_file\_\_"),  folder) + "\\" + name + ftype)

*return* path

*def* deltatime(time):

*# ФУНКЦИЯ ВОЗВРАЩАЕТ ОТНОСИТЕЛЬНОЕ ВРЕМЯ*

    HOUR, WEEK, MONTH, YEAR = 3600, 7, 31, 365

*# СРАВНИВАЕМ ТОЛЬКО ДАТЫ*

    zerotimeNow = datetime.datetime.now().replace(

        hour=0, minute=0, second=0, microsecond=0)

    zerotimeThen = datetime.datetime.strptime(

        time, '%Y-%m-%d %H:%M:%S').replace(hour=0, minute=0, second=0)

    delta = zerotimeNow - zerotimeThen

*if* delta.days < 1:

        curtime = datetime.datetime.now()

        deltaToday = curtime - \

            datetime.datetime.strptime(time, '%Y-%m-%d %H:%M:%S')

*if* deltaToday.seconds < HOUR:

            deltaToday = "Меньше часа назад"

*elif* deltaToday.seconds > HOUR and deltaToday.seconds < 2 \* HOUR:

            deltaToday = "Час назад"

*else*:

            deltaToday = "Сегодня"

*elif* delta.days == 1:

        deltaToday = "Вчера"

*elif* delta.days > 1 and delta.days <= WEEK:

        deltaToday = "На этой неделе"

*elif* delta.days > WEEK and delta.days < MONTH:

        deltaToday = "В этом месяце"

*elif* delta.days >= MONTH and delta.days <= YEAR:

        deltaToday = "В этом году"

*else*:

        deltaToday = "Больше года назад"

*return* deltaToday

*class* MainWindow(QtWidgets.QMainWindow, MainDesigner.Ui\_MainWindow):

*def* \_\_init\_\_(self):

        super().\_\_init\_\_()

*self*.setupUi(*self*)

*self*.loadSettings()

*# ФУНКЦИИ ГЛАВНОГО ОКНА*

*self*.tabWidget.setCurrentIndex(0)

*self*.startBtn.clicked.connect(*self*.startParsing)

*self*.openManagerBtn.clicked.connect(*self*.openmanager)

*self*.deleteBtn.clicked.connect(*self*.deleteExport)

*self*.renameBtn.clicked.connect(*self*.renameExport)

*self*.expandedBtn.clicked.connect(*self*.openPreview)

*self*.exportbox.itemClicked.connect(*self*.preview)

*self*.exportbox.itemDoubleClicked.connect(*self*.openPreview)

*self*.openSettings.clicked.connect(*self*.openPreferences)

*# ФУНКЦИИ ОКНА КОНФИГУРАЦИИ*

*self*.configbox.itemClicked.connect(*self*.loadCfgFromList)

*self*.configbox.itemDoubleClicked.connect(*self*.useConfig)

*self*.newConfigBtn.clicked.connect(*self*.newConfig)

*self*.renameConfigBtn.clicked.connect(*self*.renameConfig)

*self*.delConfigBtn.clicked.connect(*self*.delConfig)

*self*.useConfigBtn.clicked.connect(*self*.useConfig)

*self*.nameEntry.returnPressed.connect(*self*.useConfig)

*self*.textEntry.returnPressed.connect(*self*.useConfig)

*self*.authorEntry.returnPressed.connect(*self*.useConfig)

*self*.receiverEntry.returnPressed.connect(*self*.useConfig)

*self*.lineEdit.returnPressed.connect(*self*.startParsing)

*self*.cfgCheck.toggled.connect(*self*.setName)

*self*.dateCheck.toggled.connect(*self*.setName)

*self*.relativetime.toggled.connect(*self*.setTime)

*self*.exactTime.toggled.connect(*self*.setTime)

*# ФУНКЦИИ ОКНА ПРЕДПРОСМОТРА*

*self*.treeWidget.itemClicked.connect(*self*.onItemClicked)

*self*.tableWidget.itemDoubleClicked.connect(*self*.copy)

*self*.updateMain()

*self*.newConfig()

*def* loadSettings(self):

*# ЗАГРУЖАЕТ НАСТРОЙКИ*

*try*:

            settings = configparser.ConfigParser()

            settings.read('settings.ini', encoding='utf-8-sig')

            configDict = settings.\_sections['settings']

*self*.RELATIVETIME = True

*self*.EXPORTCFGNAME = True

*if* configDict['loadstg'] == "True":

*if* configDict['savestg'] != "True":

*self*.saveStgCheck.setChecked(False)

*if* configDict['exitapp'] != "True":

*self*.exitCheck.setChecked(False)

*if* configDict['relativetime'] != "True":

*self*.exactTime.setChecked(True)

*self*.RELATIVETIME = False

*if* configDict['cfgname'] != "True":

*self*.dateCheck.setChecked(True)

*self*.EXPORTCFGNAME = False

*else*:

*self*.loadStgCheck.setChecked(False)

*except*:

*return*

*# ФУНКЦИИ ГЛАВНОГО ОКНА*

*def* updateMain(self):

*# ОБНОВЛЯЕТ ФАЙЛЫ ОСНОВНОГО ОКНА*

*def* convert\_size(size\_bytes):

*# РАССЧИТЫВАЕТ РАЗМЕР ФАЙЛА*

*if* size\_bytes == 0:

*return* "0B"

            size\_name = ("B", "KB", "MB", "GB", "TB", "PB", "EB", "ZB", "YB")

            i = int(math.floor(math.log(size\_bytes, 1024)))

            p = math.pow(1024, i)

            s = round(size\_bytes / p, 2)

*return* "%s %s" % (s, size\_name[i])

*# ЗАПОЛНЕНИЕ ДЕРЕВА ЭКСПОРТИРОВАННЫХ ДАННЫХ*

*self*.exportbox.clear()

        path = relativePath("export")

        files = [file *for* file *in* glob.glob(path + "\*\*/\*.csv")]

*for* file *in* files:

            filename = os.path.basename(file).split(".")[0]

            filesize = convert\_size(os.path.getsize(file))

*with* open(file,  encoding='utf-8-sig') *as* f:

                length = sum(1 *for* line *in* f) - 1

*if* length == 0:

                length = "<Ничего не найдено>"

            f.close()

            item = QtWidgets.QTreeWidgetItem(*self*.exportbox)

            values = (filename, length, filesize)

*for* i *in* range(len(values)):

                item.setText(i, str(values[i]))

                item.setTextAlignment(i, 4)

*for* i *in* range(*self*.exportbox.columnCount()):

*self*.exportbox.resizeColumnToContents(i)

*# ЗАПОЛНЕНИЕ ДЕРЕВА КОНФИГУРАЦИЙ*

        path = relativePath("config")

*self*.configbox.clear()

        files = [file *for* file *in* glob.glob(path + "\*\*/\*.ini")]

*for* file *in* files:

            config = configparser.ConfigParser()

            config.read(file, encoding='utf-8-sig')

            configDict = config.\_sections['config']

*if* *self*.RELATIVETIME:

                date = deltatime(configDict['last\_used'])

*else*:

                date = configDict['last\_used']

            filename = os.path.basename(file).split(".")[0]

            item = QtWidgets.QTreeWidgetItem(*self*.configbox)

            values = (filename, configDict['text'], configDict['topic'], configDict['fdate'],

                      configDict['tdate'],  configDict['author'],  configDict['receiver'], configDict['expanded'], date)

*for* i *in* range(len(values)):

                item.setText(i, values[i])

*for* i *in* range(*self*.configbox.columnCount()):

*self*.configbox.resizeColumnToContents(i)

*def* startParsing(self):

*# ФУНКЦИЯ НАЧАЛА ПАРСИНГА*

*# ПРОВЕРКА ВВОДА КОЛИЧЕСТВА СТРАНИЦ*

*self*.maxPage = *self*.lineEdit.text()

*if* *self*.maxPage == "":

            QtWidgets.QMessageBox.about(

*self*, "Внимание!", "Укажите количество страниц!")

*return*

*try*:

            int(*self*.maxPage)

*except*:

            QtWidgets.QMessageBox.about(

*self*, "Внимание!", "Неверное количество!")

*return*

*# ПРОВЕРКА НАЛИЧИЯ КОНФИГУРАЦИИ*

*if* *self*.cfgPATH == "":

            QtWidgets.QMessageBox.about(

*self*, "Внимание!", "Вы не выбрали конфигурацию!")

*return*

*# ПРОВЕРКА ДОСТУПНОСТИ СЕРВЕРА*

*self*.progressBar.setValue(0)

*self*.progressBar.setMaximum(int(*self*.maxPage)+3)

*self*.progresslbl.setText("Устанавливаю соединение...")

*try*:

            requests.head("https://vif2ne.org/")

*pass*

*except* requests.ConnectionError:

            QtWidgets.QMessageBox.about(

*self*, "Внимание!", "Проверьте ваше подключение к интернету и доступность сайта")

*self*.progresslbl.setText("Ошибка при установке соединения!")

*return*

*self*.progresslbl.setText("Соединение установлено!")

*self*.progressBar.setValue(*self*.progressBar.value()+1)

*self*.saveConfig(started = True)

*# СЧИТЫВАНИЕ ФАЙЛА*

*try*:

            config = configparser.ConfigParser()

            config.read(*self*.cfgPATH, encoding='utf-8-sig')

            filename = os.path.basename(*self*.cfgPATH).split(".")[0]

            configDict = config.\_sections['config']

*self*.ctext, *self*.ctopic, *self*.cstrtDate, *self*.cendDate, *self*.cauthor, *self*.cadressed, *self*.cexpanded = configDict['text'], configDict[

                'topic'], configDict['fdate'], configDict['tdate'],  configDict['author'],  configDict['receiver'], configDict['expanded']

*self*.progresslbl.setText("Конфигурация загружена!")

*self*.progressBar.setValue(*self*.progressBar.value()+1)

*except*:

            QtWidgets.QMessageBox.about(

*self*, "Файл конфигурации поврежден!")

*return*

*# ЗАПУСК ТРЕДА ПАРСИНГА*

*self*.parser = parserMain.Parser(filename,

                                        int(*self*.maxPage), *self*.ctopic, *self*.ctext, *self*.cstrtDate, *self*.cendDate, *self*.cauthor, *self*.cadressed, str(*self*.cexpanded), *self*.EXPORTCFGNAME)

*self*.parser.progress.connect(*self*.moveprogress)

*self*.parser.done.connect(*self*.complete)

*self*.parser.start()

*self*.progresslbl.setText("Начинаю парсинг!")

*def* moveprogress(self, value):

*# ОТВЕЧАЕТ ЗА ВИЗУАЛЬЗАЦИЮ ПРОГРЕССА ПАРСИНГА*

*if* value <= int(*self*.maxPage):

*self*.progresslbl.setText(

*f*"Парсинг: страница: {value}/{*self*.maxPage}")

*self*.progressBar.setValue(*self*.progressBar.value()+1)

*else*:

*self*.progresslbl.setText(*f*"Создания файла с данными!")

*self*.progressBar.setValue(*self*.progressBar.value()+1)

*def* complete(self, done):

*# ОКОНЧАНИЕ ПАРСИНГА*

*# ЗАПИСЫВАЕТ В КОНФИГУРАЦИЮ ВРЕМЯ ПОСЛЕДНЕГО ПАРСИНГА (ТЕКУЩЕЕ ВРЕМЯ)*

        config = configparser.ConfigParser()

        config.read(*self*.cfgPATH, encoding='utf-8-sig')

        last\_used = *f*"{datetime.datetime.now():%Y-%m-%d %H:%M:%S}"

        config['config'] = {'text': *self*.ctext, 'author': *self*.cauthor,

                            'receiver': *self*.cadressed, 'topic': *self*.ctopic, 'fdate': *self*.cstrtDate, 'tdate': *self*.cendDate, 'last\_used': last\_used, 'expanded': *self*.cexpanded}

*with* open(*self*.cfgPATH, "w", encoding='utf-8-sig') *as* configfile:

            config.write(configfile)

            configfile.close()

        QtWidgets.QMessageBox.about(

*self*, "Успешно!", "Парсинг окончен!")

*self*.updateMain()

*# ВОЗВРАЩАЕТ В ИСХОДНОЕ ПОЛОЖЕНИЕ ЛЕЙБЛ И ПРОГРЕССБАР*

*self*.progresslbl.setText(*f*"Ожидание начала парсинга...")

*self*.progressBar.setValue(0)

*self*.parser.stop()

*def* deleteExport(self):

*# ОТВЕЧАЕТ ЗА УДАЛЕНИЕ ФАЙЛА ЭКСПОРТА*

*try*:

*# ДЛЯ ВСЕХ ВЫБРАННЫХ ФАЙЛОВ*

*for* item *in* *self*.exportbox.selectedItems():

*# СОЗДАЕМ ИХ ПУТЬ И УДАЛЯЕМ*

                filepath = relativePath('export',  item.text(0), '.csv')

*if* os.path.exists(filepath):

                    os.remove(filepath)

*else*:

                    QtWidgets.QMessageBox.about(

*self*, "Внимание!", "Не удалось удалить файл")

*# ОБНОВЛЯЕМ ГЛАВНОЕ ОКНО И ОЧИЩАЕМ ОКНА ПРЕДПРОСМОТРА*

*self*.updateMain()

*self*.previewShowbox.clear()

*self*.treeWidget.clear()

*self*.label\_16.setText("Выберите файл!")

*except*:

            QtWidgets.QMessageBox.about(

*self*, "Внимание!", "Выберите файл!")

*def* renameExport(self):

*# ОТВЕЧАЕТ ЗА ПЕРЕИМЕНОВАНИЕ ФАЙЛА ЭКСПОРТА*

*try*:

*# СОЗДАЕТ ОКНО ДЛЯ ПЕРЕИМЕНОВАНИЯ ФАЙЛА*

            renameWin = RenameWin(*self*.exportbox.selectedItems()[0].text(0))

*if* renameWin.exec\_():

*try*:

*self*.newName = relativePath(

                        'export',  renameWin.text, '.csv')

                    current = relativePath(

                        'export',  *self*.exportbox.selectedItems()[0].text(0), '.csv')

                    os.rename(current, *self*.newName)

*self*.updateMain()

*self*.previewShowbox.clear()

*except*:

                    QtWidgets.QMessageBox.about(

*self*, "Внимание!", "Некорректное имя!")

*except*:

            QtWidgets.QMessageBox.about(

*self*, "Внимание!", "Выберите файл!")

*def* openmanager(self):

*# ОТКРЫВАЕТ МЕНЕДЖЕР НА ВТОРОЙ ВКЛАДКЕ*

*self*.tabWidget.setCurrentIndex(1)

*def* openPreview(self):

*# ПЕРЕКЛЮЧАЕТ ВКЛАДКУ НА ВКЛАДКУ ПРЕДПРОСМОТРА*

*self*.tabWidget.setCurrentIndex(2)

*def* openPreferences(self):

*# ОТКРЫВАЕТ НАСТРОЙКИ НА ЧЕТВЕРТОЙ ВКЛАДКЕ*

*self*.tabWidget.setCurrentIndex(3)

*def* preview(self, it, colf):

*# ЗАГРУЖАЕТ ФАЙЛ В ОКНА ПРЕДПРОСМОТРА*

*self*.previewShowbox.clear()

*try*:

            filepath = relativePath('export',  it.text(0), '.csv')

*with* open(filepath, encoding='utf-8-sig') *as* f:

                reader = csv.DictReader(f, delimiter=',')

*for* row *in* reader:

                    item = QtWidgets.QTreeWidgetItem(*self*.previewShowbox)

                    values = (row['name'], row['msg'], row['time'])

*for* i *in* range(len(values)):

                        item.setText(i, values[i])

*for* i *in* range(*self*.previewShowbox.columnCount()):

*self*.previewShowbox.resizeColumnToContents(i)

*self*.populateExpandedExport(

                relativePath('export',  it.text(0), '.csv'))

*self*.label\_16.setText(it.text(0) + ".csv")

*except*:

            QtWidgets.QMessageBox.about(

*self*, "Внимание!", "Не удалось открыть файл. Возможно он поврежден")

*def* loadfromHst(self, it, col):

*# ЗАГРУЖАЕТ КОНФИГУРАЦИЮ ИЗ ОКНА КОНФИГУРАЦИЙ*

*self*.cfgPATH = relativePath('config',  it.text(0), '.ini')

*try*:

*self*.configMain.clearContents()

*self*.configMain.setItem(

                            0, 0, QtWidgets.QTableWidgetItem(it.text(0)))

*self*.configMain.setItem(

                            1, 0, QtWidgets.QTableWidgetItem(it.text(1)))

*self*.configMain.setItem(

                            2, 0, QtWidgets.QTableWidgetItem(it.text(5)))

*self*.configMain.setItem(

                            3, 0, QtWidgets.QTableWidgetItem(it.text(6)))

*self*.guidelbl.setText("Выбрано успешно!")

*except*:

            QtWidgets.QMessageBox.about(

*self*, "Внимание!", "Конфигурация не выбрана!")

*self*.updateMain()

*# ФУНКЦИИ ОКНА КОНФИГУРАЦИИ*

*def* loadCfgFromList(self, it, col):

*# ЗАПОЛНЯЕТ ВСЕ ПОЗИЦИИ ОКНА КОНФИГУРАЦИИ ИЗ ФАЙЛА*

*self*.selectedCfg = str(it.text(0))

*self*.nameEntry.setText(it.text(0))

*self*.textEntry.setText(it.text(1))

        items = *self*.topicbox.findItems(it.text(2), Qt.Qt.MatchExactly)[0]

*self*.topicbox.setCurrentRow(*self*.topicbox.row(items))

*self*.fdate.setDate(datetime.datetime.strptime(it.text(3), '%d-%m-%Y'))

*self*.tdate.setDate(datetime.datetime.strptime(it.text(4), '%d-%m-%Y'))

*self*.authorEntry.setText(it.text(5))

*self*.receiverEntry.setText(it.text(6))

*if* (it.text(7)) == "True" and not *self*.checkBox.isChecked():

*self*.checkBox.toggle()

*elif* (it.text(7)) == "False" and *self*.checkBox.isChecked():

*self*.checkBox.toggle()

*self*.loadfromHst(it, col)

*def* newConfig(self):

*# ОБНУЛЯЕТ ВСЕ ПОЛЯ КОНФИГУРАЦИИ*

*self*.cfgPATH = ""

*self*.nameEntry.setText("Новая конфигурация")

*self*.textEntry.setText("")

        items = *self*.topicbox.findItems("Все", Qt.Qt.MatchExactly)[0]

*self*.topicbox.setCurrentRow(*self*.topicbox.row(items))

*self*.fdate.setDate(datetime.datetime.strptime('01-01-2000', '%d-%m-%Y'))

*self*.tdate.setDate(datetime.datetime.today())

*self*.authorEntry.setText("")

*self*.receiverEntry.setText("")

*if* *self*.checkBox.isChecked():

*self*.checkBox.toggle()

*def* saveConfig(self, started = False):

*# СОХРАНЯЕТ КОНФИГУРАЦИЮ*

*self*.savedflag = 0

        path = relativePath("config")

*if* not os.path.exists(path):

            os.makedirs(path)

            print("DEBUG: CFG FOLDER CREATED")

*if* not started:

*self*.name = "".join(x *for* x *in* *self*.nameEntry.text()

*if* x.isalnum() *or* x == " ")

            text = *self*.textEntry.text()

            author = *self*.authorEntry.text()

            receiver = *self*.receiverEntry.text()

*else*:

*self*.name = "".join(x *for* x *in* *self*.configMain.item(0,0).text()

*if* x.isalnum() *or* x == " ")

*try*:

                text = *self*.configMain.item(0,1).text()

*except*:

                text = ""

*try*:

                author = *self*.configMain.item(0,2).text()

*except*:

                author = ""

*try*:

                receiver = *self*.configMain.item(0,3).text()

*except*:

                receiver = ""

*if* len(*self*.name) == 0:

            QtWidgets.QMessageBox.about(

*self*, "Ошибка!", "Некорректное имя!")

*return*

        MsgBox = QtWidgets.QMessageBox.question(*self*,

                                                'Выбрана конфигурация', *f*'Вы уверены, что хотите сохранить конфигурацию "{*self*.name}"', QtWidgets.QMessageBox.Yes | QtWidgets.QMessageBox.No)

*if* MsgBox == QtWidgets.QMessageBox.Yes:

*pass*

*else*:

*return*

*try*:

            topic = [item.text() *for* item *in* *self*.topicbox.selectedItems()][0]

*except*:

            topic = 'Все'

*finally*:

*pass*

        fdate = str(*self*.fdate.date().toString('dd-MM-yyyy'))

        tdate = str(*self*.tdate.date().toString('dd-MM-yyyy'))

*if* *self*.checkBox.isChecked():

            expanded = "True"

*else*:

            expanded = "False"

        last\_used = *f*"{datetime.datetime.now():%Y-%m-%d %H:%M:%S}"

        config = configparser.ConfigParser()

        config['config'] = {'text': text, 'author': author,

                            'receiver': receiver, 'topic': topic, 'fdate': fdate, 'tdate': tdate, 'last\_used': last\_used, 'expanded': expanded}

        path = relativePath("config")

*with* open(path + '\\' + *self*.name + '.ini', 'w', encoding="utf-8-sig") *as* configfile:

            config.write(configfile)

*try*:

            f = open(path + "\\" + *self*.name + '.ini')

*except* IOError:

            QtWidgets.QMessageBox.about(

*self*, "Ошибка!", "Некорректное имя!")

*return*

*else*:

            f.close()

            QtWidgets.QMessageBox.about(

*self*, "Вы выбрали конфигурацию", *f*'Выбрана конфигурация "{*self*.name}"!')

*self*.savedflag = 1

            print("DEBUG: CFG SAVED")

*self*.updateMain()

*def* useConfig(self):

*# ИСПОЛЬЗУЕТ КОНФИГУРАЦИЮ*

*self*.saveConfig()

*if* *self*.savedflag == 1:

*try*:

*self*.cfgPATH = relativePath(

                    'config',  *self*.name, '.ini')

*self*.tabWidget.setCurrentIndex(0)

*self*.updateMain()

*try*:

*with* open(*self*.cfgPATH, encoding='utf-8-sig') *as* f:

                        lines = f.readlines()

*self*.configMain.clearContents()

*self*.configMain.setItem(

                            0, 0, QtWidgets.QTableWidgetItem(os.path.basename(*self*.cfgPATH).split(".")[0]))

*self*.configMain.setItem(

                                    1, 0, QtWidgets.QTableWidgetItem(lines[1].split("= ")[1].replace("\n", "")))

*self*.configMain.setItem(

                                    2, 0, QtWidgets.QTableWidgetItem(lines[2].split("= ")[1].replace("\n", "")))

*self*.configMain.setItem(

                                    3, 0, QtWidgets.QTableWidgetItem(lines[3].split("= ")[1].replace("\n", "")) )

*self*.guidelbl.setText("Выбрано успешно!")

                    f.close()

*except*:

                    QtWidgets.QMessageBox.about(

*self*, "Внимание!", "Конфигурация не выбрана!")

*except*:

                QtWidgets.QMessageBox.about(

*self*, "Error!", "Непредвиденная ошибка!")

*return*

*else*:

*pass*

*def* delConfig(self):

*#  УДАЛЯЕТ КОНФИГУРАЦИЮ*

        name = *self*.nameEntry.text()

        MsgBox = QtWidgets.QMessageBox.question(*self*,

                                                'Удаление', *f*'Вы уверены, что хотите удалить конфигурацию {name}?',  QtWidgets.QMessageBox.Yes | QtWidgets.QMessageBox.No)

*if* MsgBox == QtWidgets.QMessageBox.Yes:

*pass*

*else*:

*return*

        filepath = relativePath('config',  *self*.selectedCfg, '.ini')

*if* os.path.exists(filepath):

            os.remove(filepath)

*self*.newConfig()

*else*:

            print("The file does not exist")

*self*.updateMain()

*def* renameConfig(self):

*# ПЕРЕИМЕНОВЫВАЕТ КОНФИГУРАЦИЮ*

        rename = RenameWin(initName=*self*.configbox.selectedItems()[0].text(0))

*if* rename.exec\_():

*try*:

*self*.renameWintxt = "".join(

                    x *for* x *in* rename.text *if* x.isalnum() *or* x == " ")

*self*.newName = relativePath(

                    'config', *self*.renameWintxt, '.ini')

                current = relativePath(

                    'config',  *self*.configbox.selectedItems()[0].text(0), '.ini')

                os.rename(current, *self*.newName)

*self*.updateMain()

*self*.configbox.setCurrentItem(*self*.configbox.findItems(*self*.renameWintxt, Qt.Qt.MatchExactly)[

                    0])

*self*.configbox.itemClicked.emit(

*self*.configbox.selectedItems()[0], 0)

*except*:

                QtWidgets.QMessageBox.about(

*self*, "Внимание!", "Файл с таким именем уже существует!")

*# ФУНКЦИИ ОКНА ПРЕДПРОСМОТРА*

*def* populateExpandedExport(self, file=""):

*with* open(file, encoding='utf-8-sig') *as* f:

            reader = csv.DictReader(f, delimiter=',')

*try*:

*for* row *in* reader:

                    item = QtWidgets.QTreeWidgetItem(*self*.treeWidget)

                    values = (row['name'], row['msg'], row['time'],

                              row['link'], row['length'], row['expmsg'])

*for* i *in* range(len(values)):

                        item.setText(i, values[i])

*except*:

*for* row *in* reader:

                    item = QtWidgets.QTreeWidgetItem(*self*.treeWidget)

                    values = (row['name'], row['msg'], row['time'],

                              row['link'], row['length'], "")

*for* i *in* range(len(values)):

                        item.setText(i, values[i])

*for* i *in* range(*self*.treeWidget.columnCount()):

*self*.treeWidget.resizeColumnToContents(i)

        f.close()

*def* onItemClicked(self, it, col):

*self*.tableWidget.clearContents()

*for* i *in* range(*self*.tableWidget.rowCount() + 1):

*self*.tableWidget.setItem(

                i, 0, QtWidgets.QTableWidgetItem(it.text(i)))

*self*.tableWidget.setEditTriggers(QtWidgets.QTableWidget.NoEditTriggers)

*def* copy(self, it):

        cb = QtWidgets.QApplication.clipboard()

        cb.clear(mode=cb.Clipboard)

        cb.setText(it.text(), mode=cb.Clipboard)

        QtWidgets.QMessageBox.about(

*self*, "Скопировано!", "Скопировано в буфер обмена!")

*def* setName(self):

*if* *self*.cfgCheck.isChecked():

*self*.EXPORTCFGNAME = True

*else*:

*self*.EXPORTCFGNAME = False

*def* setTime(self):

*if* *self*.relativetime.isChecked():

*self*.RELATIVETIME = True

*else*:

*self*.RELATIVETIME = False

*self*.updateMain()

*def* closeEvent(self, event):

*if* *self*.exitCheck.isChecked():

            MsgBox = QtWidgets.QMessageBox.question(*self*,

                                                    'Завершение работы', 'Вы уверены, что хотите выйти?', QtWidgets.QMessageBox.Yes | QtWidgets.QMessageBox.No)

*if* MsgBox == QtWidgets.QMessageBox.Yes:

*pass*

*else*:

                event.ignore()

*return*

*try*:

*if* *self*.saveStgCheck.isChecked():

*if* *self*.loadStgCheck.isChecked():

                    loadStg = 'True'

*else*:

                    loadStg = 'False'

*if* *self*.exitCheck.isChecked():

                    exitApp = 'True'

*else*:

                    exitApp = 'False'

*if* *self*.cfgCheck.isChecked():

                    cfgName = 'True'

*else*:

                    cfgName = 'False'

*if* *self*.relativetime.isChecked():

                    relativetime = 'True'

*else*:

                    relativetime = 'False'

                settings = configparser.ConfigParser()

                settings['settings'] = {'saveStg': 'True', 'loadStg': loadStg,

                                        'exitApp': exitApp, 'cfgName': cfgName, 'relativeTime': relativetime}

*with* open('settings.ini', 'w', encoding="utf-8-sig") *as* configfile:

                    settings.write(configfile)

*else*:

                settings = configparser.ConfigParser()

                settings.read('settings.ini', encoding='utf-8-sig')

                configDict = settings.\_sections['settings']

                loadStg, exitApp, relativetime, cfgName = configDict['loadStg'], \

                    configDict['exitApp'],  configDict['relativetime'], configDict['cfgName']

                settings['settings'] = {'saveStg': 'False', 'loadStg': loadStg,

                                        'exitApp': exitApp, 'cfgName': cfgName, 'relativeTime': relativetime}

*with* open('settings.ini', 'w', encoding="utf-8-sig") *as* configfile:

                    settings.write(configfile)

*except*:

            QtWidgets.QMessageBox.about(

*self*, "Ошибка!", "Ошибка при сохранении настроек!")

        event.accept()

##### parserMain.py

Основной модуль треда парсера

*import* requests

*from* bs4 *import* BeautifulSoup

*import* datetime

*from* parsing *import* parserFunc *as* p

*from* PyQt5 *import* QtCore

*import* os

*import* itertools

*import* pandas *as* pd

*def* relativePath(folder, name="", ftype=""):

*# ФУНКЦИЯ ВОЗВРАЩАЕТ ОТНОСИТЕЛЬНЫЙ ПУТЬ К ФАЙЛУ ИЛИ ПАПКЕ*

    path = os.path.abspath(os.path.join(os.path.dirname(

        "\_\_file\_\_"),  folder) + "\\" + name + ftype)

*return* path

*class* Parser(QtCore.QThread):

    progress = QtCore.pyqtSignal(int)

    done = QtCore.pyqtSignal(int)

*def* \_\_init\_\_(self, filename, maxPage, topic, text, strtDate, endDate, author, adressed, expanded, EXPORTCFGNAME):

        QtCore.QThread.\_\_init\_\_(*self*)

*self*.filename = filename

*self*.maxPage = maxPage

*self*.topic = topic

*self*.text = text

*self*.strtDate = strtDate

*self*.endDate = endDate

*self*.author = author

*self*.adressed = adressed

*self*.expanded = expanded

*self*.EXPORTCFGNAME = EXPORTCFGNAME

*def* stop(self):

*self*.terminate()

*def* run(self):

        headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)'

                   'AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.113 Safari/537.36'}

        s = requests.session()

        s.headers.update(headers)

        page = 0

        records = []

*while* page < *self*.maxPage:

            data = p.load\_url(page, *self*.text, *self*.topic, *self*.strtDate, *self*.endDate,

*self*.author, *self*.adressed, *self*.expanded, s)

*if* p.contain\_forum\_data(data):

                soup = BeautifulSoup(data, 'html.parser')

*if* *self*.expanded == "False":

                    tables = soup.find\_all('table')

                    records.append(p.getData(tables))

*else*:

                    expMsg = soup.find\_all(['table', 'i'])

                    records.append(p.getMsg(expMsg))

                page += 1

*self*.sleep(1)

*self*.progress.emit(page)

*else*:

*break*

        records = list(itertools.chain.from\_iterable(records))

*self*.listToCSV(records, *self*.expanded)

*self*.progress.emit(page + 1)

*self*.done.emit(1)

*def* listToCSV(self, records, expanded):

*if* *self*.EXPORTCFGNAME == False:

            csv\_name = datetime.datetime.now().strftime('%Y-%m-%d\_%H-%M-%S')

*else*:

            i = 0

*if* os.path.exists(relativePath('export', *self*.filename, ".csv")):

                i += 1

*while* os.path.exists(relativePath('export', *f*"{*self*.filename} ({i})", ".csv")):

                i += 1

*if* i != 0:

                csv\_name = *f*"{*self*.filename} ({i})"

*else*:

                csv\_name = *self*.filename

*if* expanded == "False":

            df = pd.DataFrame(records, columns=[

                'name', 'msg', 'time', 'link', 'length'])

            df['time'] = pd.to\_datetime(df['time'])

            df.to\_csv(relativePath('export', csv\_name, ".csv"),

                      index=False, encoding='utf-8-sig')

*else*:

            df = pd.DataFrame(records, columns=[

                'name', 'msg', 'time', 'link', 'length', 'expmsg'])

            df['time'] = pd.to\_datetime(df['time'])

            df.to\_csv(relativePath('export', csv\_name, ".csv"),

                      index=False, encoding='utf-8-sig')

##### parserFunc.py

Модуль с дополнительными функциями парсинга

*from* bs4 *import* BeautifulSoup

*import* urllib.parse

*def* load\_url(page, text, topic, strtDate, endDate, author, adressed, expanded, session):

    [fday, fmonth, fyear] = strtDate.split("-")

    [tday, tmonth, tyear] = endDate.split("-")

    strtdoc = 1 + int(page) \* 100

    enddoc = (int(page) + 1) \* 100

    text = encodestr(text)

*if* topic == 'Все':

        topic = ""

*else*:

        topic = encodestr(topic)

    topic = topic.replace("%", "")

    author = encodestr(author)

    adressed = encodestr(adressed)

*if* expanded != "False":

        expanded = "on"

        url = *f*"https://vif2ne.org/nvk/forum/0/index/list?text={text}&topic={topic}&fday={fday}&fmonth={fmonth}&fyear={fyear}&tday={tday}&tmonth={tmonth}&tyear={tyear}&wide={expanded}&author={author}&toauthor={adressed}&from={str(strtdoc)}&to={str(enddoc)}"

*else*:

        url = *f*"https://vif2ne.org/nvk/forum/0/index/list?text={text}&topic={topic}&fday={fday}&fmonth={fmonth}&fyear={fyear}&tday={tday}&tmonth={tmonth}&tyear={tyear}&author={author}&toauthor={adressed}&from={str(strtdoc)}&to={str(enddoc)}"

    request = session.get(url)

*return* request.text

*def* contain\_forum\_data(text):

*try*:

        soup = BeautifulSoup(text, 'html.parser')

        line = soup.find('div', id='wrapper').find\_all('table')

*return* line is not None

*except*:

*return*

*def* getData(tables):

    parsed = []

    i = 0

*for* table\_i *in* tables:

*try*:

            i += 1

*if* i % 2 != 0:

                dateTime = table\_i.find('td').text

                textlink = "https://vif2ne.org/nvk/forum/" + \

                    table\_i.find('a')['href'][3:]

                textlen = table\_i.find\_all('td')[1].text[6:]

                tmpdict2 = {'time': dateTime,

                            'link': textlink, 'length': textlen}

                tmpdict = {\*\*tmpdict1, \*\*tmpdict2}

                parsed.append(tmpdict)

                tmpdict = {}

*else*:

                nickname = table\_i.find('b').text

                message = table\_i.find('a').text

                tmpdict1 = {'name': nickname, 'msg': message}

*except*:

*pass*

*return* parsed

*def* getMsg(expMsg):

    parsed = []

    i = 0

*for* table\_i *in* expMsg:

*try*:

*try*:

                nickname = table\_i.find('b').text

                message = table\_i.find('a').text

                tmpdict1 = {'name': nickname, 'msg': message}

*except*:

*pass*

*else*:

                i = 0

            i += 1

*if* i % 3 == 0:

                expanded = table\_i.text

                tmpdict3 = {'expmsg': expanded}

                tmpdict = {\*\*tmpdict1, \*\*tmpdict2, \*\*tmpdict3}

                parsed.append(tmpdict)

                tmpdict = {}

*if* i % 2 == 0:

                dateTime = table\_i.find('td').text

                textlink = "https://vif2ne.org/nvk/forum/" + \

                    table\_i.find('a')['href'][3:]

                textlen = table\_i.find\_all('td')[1].text[6:]

                tmpdict2 = {'time': dateTime,

                            'link': textlink, 'length': textlen}

*else*:

                nickname = table\_i.find('b').text

                message = table\_i.find('a').text

                tmpdict1 = {'name': nickname, 'msg': message}

*except*:

*pass*

*return* parsed

*def* encodestr(text):

    text = urllib.parse.quote\_plus(text.encode('cp1251'))

*return* text

##### RenameWin.py

Окно переименования

*import* sys

*import* PyQt5

*from* PyQt5 *import* QtWidgets, QtCore

*from* PyQt5 *import* Qt

*from* designer *import* RenameDesigner

*class* RenameWin(QtWidgets.QDialog, RenameDesigner.Ui\_Rename):

*def* \_\_init\_\_(self,  initName="", parent=None):

        super(RenameWin, *self*).\_\_init\_\_()

*self*.setupUi(*self*)

*self*.lineEdit.returnPressed.connect(*self*.on\_pushButtonOK\_clicked)

*self*.lineEdit.setText(initName)

    @QtCore.pyqtSlot()

*def* on\_pushButtonOK\_clicked(self):

*self*.text = *self*.lineEdit.text()

*if* *self*.text != "":

            name = "".join(x *for* x *in* *self*.text *if* x.isalnum())

*if* len(name) == 0 or "." in *self*.text:

                QtWidgets.QMessageBox.about(

*self*, "Ошибка!", "Некорректное имя!")

*self*.close()

*else*:

*self*.accept()

*self*.close()

*else*:

            QtWidgets.QMessageBox.about(

*self*, "Внимание!", "Имя не может быть пустым!")

Помимо этого, модули MainDesigner.py и RenameDesigner.py созданные автоматически через pyuic5 из файлов .ui