МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»

(САМАРСКИЙ УНИВЕРСИТЕТ)

Отчёт по лабораторной работе №2

«Исследование web-сайта»  
по курсу «Сети ЭВМ и телекоммуникации»

Выполнил:  
Грибанов Д. Н.

гр.6101

Проверил:  
Еленев Д. В.

Самара 2020

**Задание:**

Создать сетевое клиентское приложение, реализующее обращения к www-серверу по протоколу HTTP и производящее обработку полученных данных. Приложение должно выполнять функции, изложенные в задании, и обладать дружественным к пользователю интерфейсом.

Вариант задания: 2.

Составить и вывести список всех рисунков, используемых страницами www-сервера. Вывод разделить на две части: рисунки, расположенные на сервере, и рисунки, расположенные на других серверах

**Код программы:**

import ssl  
from urllib.parse import urljoin, urlparse  
import traceback, sys  
from bs4 import BeautifulSoup  
import socket  
  
from .signal\_sender import SignalSender  
from main\_window.main\_widget.signals.prograss\_bar\_signal import PrograssBarSignalSender  
from main\_window.main\_widget.signals.label\_process\_signal import LabelProcessSignalSender  
from PySide2.QtCore import Slot, QRunnable  
  
  
class LinkWorker(QRunnable):  
 *"""  
 Thread (Worker) to find information from certain urls  
 Using socket (in most cases ssl - which work with HTTPS)  
  
 """* REQUEST\_SIZE\_TO\_READ = 4096  
  
 def \_\_init\_\_(  
 *self*,  
 address: str,  
 port: int,  
 depth: int,  
 max\_number\_addresses: int,  
 progress\_bar\_sender: PrograssBarSignalSender,  
 label\_process\_sender: LabelProcessSignalSender  
 ):  
 super(LinkWorker, *self*).\_\_init\_\_()  
 # Store images from main page  
 *self*.main\_images\_set = set()  
 # Store images from secondary sites  
 *self*.second\_images\_set = set()  
  
 *self*.address = address  
 *self*.port = port  
 *self*.depth = depth  
 *self*.max\_number\_addresses = max\_number\_addresses  
 *self*.all\_urls = set()  
 # Keep signal sender, to update progress bar  
 *self*.progress\_bar\_sender = progress\_bar\_sender  
 *self*.label\_process\_sender = label\_process\_sender  
 # Sender for emit final result  
 *self*.signals = SignalSender()  
  
 @Slot()  
 def run(*self*):  
 try:  
 # Extract images which are located on other sites  
 urls = *self*.\_\_get\_urls\_from\_address\_socket(*self*.address)  
  
 *self*.main\_images\_set = set(*self*.\_\_extract\_img\_from\_other\_sites\_socket(*self*.address))  
 *self*.second\_images\_set = *self*.\_\_extract\_img\_from\_other\_sites\_socket(urls)  
  
 except ValueError as ex:  
 *self*.signals.error.emit(**f"Exception:** {ex}\n**Bad url!** {*self*.address}**"**)  
  
 except Exception as ex:  
 traceback.print\_exc()  
 exctype, value = sys.exc\_info()[:2]  
 *self*.signals.error.emit((exctype, value, traceback.format\_exc()))  
 else:  
 *self*.signals.result.emit(*self*) # Return the result of the processing  
 finally:  
 *self*.signals.finished.emit() # Done  
  
 def \_\_separe\_path\_and\_address(*self*, url):  
 *"""  
 Return two indexes:  
 first - index at which main url starts (for example ssau.ru)  
 second - index at which path starts (for example /news), if there is no path, return -1  
  
 """* first\_ind = -1  
 second\_ind = -1  
  
 for i in range(len(url)):  
 if url[i] == **'/'**:  
 # Scip two // slesh  
 first\_ind = i + 2  
 break  
  
 for i in range(first\_ind, len(url)):  
 if url[i] == **'/'**:  
 second\_ind = i  
 break  
  
 return first\_ind, second\_ind  
  
 def \_\_get\_urls\_from\_address\_socket(*self*, address, current\_depth=0) -> list:  
 *"""  
 Return list of lists urls found at certain address  
  
 """* try:  
 soup = BeautifulSoup(*self*.\_\_extract\_html(address), **"html.parser"**)  
  
 if soup is None:  
 raise ValueError(**f"Wrong url** {address}**"**)  
 *self*.all\_urls.add(address)  
 # Store all found urls  
 urls = [[]]  
 # Search every urls and save it  
 for a\_tag in soup.findAll(**"a"**):  
 href = a\_tag.attrs.get(**"href"**)  
 if href == **""** or href is None:  
 # href empty tag  
 continue  
 # Join urls  
 # for example self.adress = https://ssau.ru/, href = news, final output = https://ssau.ru/news  
 if *self*.address not in href:  
 href = urljoin(*self*.address, href)  
 parsed\_href = urlparse(href)  
 # remove URL GET parameters, URL fragments, etc.  
 href = parsed\_href.scheme + **"://"** + parsed\_href.netloc + parsed\_href.path  
 # Skip href if its already in `urls` or equal to input address  
 if href in urls[0] or href in *self*.all\_urls:  
 continue  
 urls[0].append(href)  
 *self*.all\_urls.add(href)  
 if len(urls[0]) >= *self*.max\_number\_addresses:  
 break  
  
 if current\_depth < *self*.depth:  
 for single\_url in urls[0]:  
 *self*.label\_process\_sender.whatProcess.emit(single\_url)  
 urls.append(*self*.\_\_get\_urls\_from\_address\_socket(single\_url, current\_depth=current\_depth+1))  
  
 return urls  
  
 except Exception as ex:  
 *self*.signals.error.emit(**f"Exception:** {ex}\n**Bad url!** {address}**"**)  
 return []  
  
 def \_\_extract\_img\_from\_other\_sites\_socket(*self*, addresses) -> set:  
 *"""  
 Extract images which are located on other sites  
 And return list of these images  
  
 """* all\_img\_set = set()  
 if isinstance(addresses, list):  
 *self*.progress\_bar\_sender.maximum.emit(min(*self*.max\_number\_addresses, len(addresses)))  
 for i, single\_url\_list in enumerate(addresses):  
 # Get list of all urls  
 all\_urls\_from\_single\_url = *self*.\_\_get\_all\_urls\_from\_list(single\_url\_list)  
 # On every url find images and save into list  
 for m, single\_url\_depth in enumerate(all\_urls\_from\_single\_url):  
 img\_list = *self*.\_\_extract\_img\_socket(single\_url\_depth)  
 all\_img\_set.update(img\_list)  
  
 *self*.progress\_bar\_sender.progress.emit(i)  
 if i == *self*.max\_number\_addresses:  
 break  
 else:  
 all\_img\_set = set(*self*.\_\_extract\_img\_socket(addresses))  
  
 return all\_img\_set  
  
 def \_\_get\_all\_urls\_from\_list(*self*, url\_list: list):  
 *"""  
 List of lists mapping into single big list  
  
 """* urls = []  
 for single\_url in url\_list:  
 if isinstance(single\_url, list):  
 urls += *self*.\_\_get\_all\_urls\_from\_list(single\_url)  
 else:  
 urls += [single\_url]  
  
 return urls  
  
 def \_\_extract\_img\_socket(*self*, address) -> list:  
 *"""  
 Extract image names from certain `address`  
  
 """* try:  
 html = *self*.\_\_extract\_html(address)  
  
 # Create parser to find all images  
 soup = BeautifulSoup(html, **"html.parser"**)  
 # Search images  
 imgs\_list = []  
 for tag in soup.findAll(**"img"**):  
 src = tag.get(**'src'**)  
 if src is not None:  
 imgs\_list.append(src)  
  
 return imgs\_list  
 except Exception as ex:  
 print(**f'Error:** {ex}**'**)  
 return []  
  
 def \_\_extract\_html(*self*, address):  
 *"""  
 Get html from certain address  
   
 """* # Separate path and link for main address  
 f\_ind, s\_ind = *self*.\_\_separe\_path\_and\_address(address)  
 if s\_ind == -1:  
 # There is only main address, for example: ssau.ru  
 address = address[f\_ind:]  
 path = **'/'** else:  
 # There is path and main address, for example: ssau.ru and ssau.ru/news  
 address, path = (address[f\_ind:s\_ind], address[s\_ind:])  
  
 html = **""** # Create and connect socket with server  
 with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as sock:  
 sock.connect((address, *self*.port))  
 context = ssl.create\_default\_context()  
 with context.wrap\_socket(sock, server\_hostname=address) as ssock:  
 # Create message for server  
 send\_mess = **f"GET** {path} **HTTP/1.0**\r\n**"** \  
 **f"Host:** {address}\r\n**"** \  
 **"Connection: close**\r\n\r\n**"** # Send message  
 ssock.send(send\_mess.encode())  
  
 # Grab all html text  
 while True:  
 data = ssock.recv(*self*.REQUEST\_SIZE\_TO\_READ)  
 if not data:  
 break  
 # Set ignore, if there is string with size lower than 4096  
 html += data.decode(**'utf-8'**, **'ignore'**)  
  
 return html

**Результат выполнения программы на примере:**

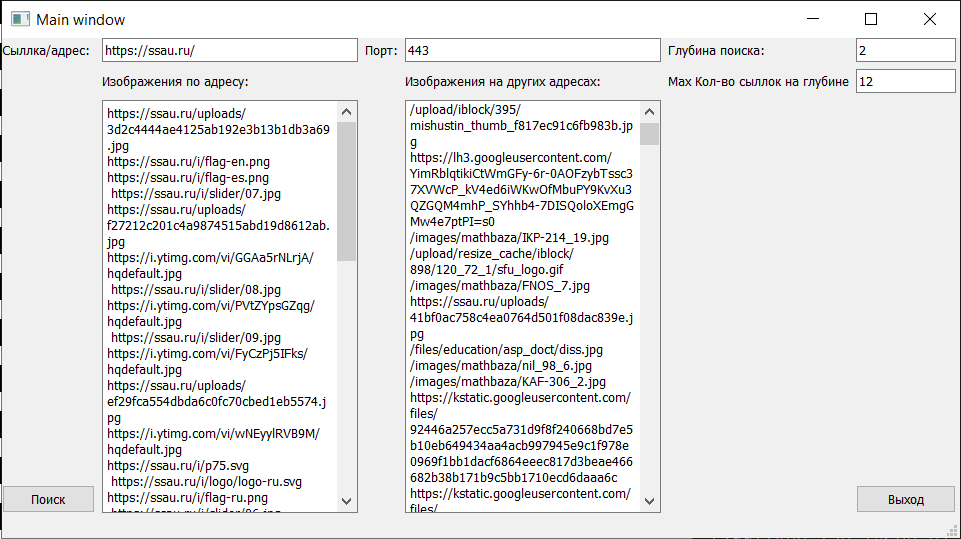


Рис. 1 – Пример работы программы