МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЯ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ
Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №7
СПП

Выполнила
Гаврилюк Р. И., студентка
группы ПО3

Проверил
Крощенко А. А.

Брест 2021

Вариант 6

Цель работы: освоить приемы разработки оконных клиент-серверных приложений на Java с использованием сокетов.

Задание.
Разработать клиент-серверное оконное приложение на Java с использованием сокетов и JavaFX. Можно сделать одну программу с сочетанием функций клиента и сервера либо две отдельных (клиентская часть и серверная часть). Продемонстрировать работу разработанной программы в сети либо локально (127.0.0.1).

6) Игра «Города». Классическая игра в города для двух игроков. Игра генерирует название города, после чего задача каждого игрока в свой ход – набрать название города, начинающегося на последнюю букву слова, названного в предыдущий ход. Причем сделать это нужно до истечения таймера (10 секунд). Игрок, который не смог или не успел назвать город, проигрывает.
Приложение должно хранить словарь городов мира, проверяя правильность записанного города (https://habr.com/ru/post/21949/)

Код программы:

server.py

```
import time

from flask import Flask, request, Response
from utils import filter_by_key

from cities import Cities

app = Flask(__name__)

# Список сообщений, отпрпвленных за текущую сессию
messages = []

# Города и работа с ними
cities = Cities()

# Функция типа send - отправка сообщений от клиента серверу
# Метод POST используется для запроса, при котором адресуемый сервер принимает объект
@app.route("/send", methods=['POST'])
def send():
    # Извлечение из JSON-объекта информации
    name = request.json.get('username')  # Имя пользователя
    text = request.json.get('text')  # Текст сообщения

    if cities.add_city_to_named(text):
        # Если город неверный
        messages.clear()
```

```
        cities.restart()
        return Response(status=400)  # Вернуть код ошибки 400

    message = {
        'username': name,
        'time': time.time(),
        'text': text
    }
    messages.append(message)

    return Response(status=200)  # 200 - код "OK"


def send_first_name():
    if not messages:
        text = cities.get_last_named_city()
        message = {
            'username': 'Chat-bot (⌒▽⌒)♡',
            'time': time.time(),
            'text': text
        }
        messages.append(message)


# Функция для фильтрации сообщений
@app.route("/messages")
def messages_view():
    # Дроп ошибки 400, если порог времени некорректен
    try:
        after = float(request.args['after'])
    except:
        return Response(status=400)
    send_first_name()

    # Возвращает объект с отфильтрованными сообщениями под ключом 'messages'
    return {
        'messages': filter_by_key(messages, key='time', threshold=after)
    }


send_first_name()

app.run()
```

client.py

```
from PyQt5 import QtWidgets
import registrationWindow
import mainWindow

app = QtWidgets.QApplication([])
window = registrationWindow.RegistrationWindow()
window.show()
app.exec_()
```

```
window2 = mainWindow.MainWindow('http://127.0.0.1:5000', window.getUsername())
window2.show()
app.exec_()
```

mainWindow.py

```
from PyQt5 import QtWidgets, QtCore
import mainWindowUI

from datetime import datetime
import requests


class MainWindow(QtWidgets.QMainWindow, mainWindowUI.Ui_MainWindow):
    def __init__(self, server_url, username):
        super().__init__()
        self.setupUi(self)
        self.userNameLabel.setText(username)
        self.__username = username
        self.__server_url = server_url
        self.sendMessageButton.clicked.connect(self.send_message)
        self.after = 0
        self.timerValue = 0
        self.timerLabel.setText(str(self.timerValue))
        self.server_url = server_url
        self.timer = QtCore.QTimer()
        self.timer.timeout.connect(self.load_messages)
        self.timer.start(1000)
        self.isClickedStartButton = False
        self.score = 0
        self.startButton.clicked.connect(self.clickOnStartButton)

    def pretty_message(self, message):
        dt = datetime.fromtimestamp(message['time'])
        dt_str = dt.strftime('%H:%M:%S')
        self.messageList.append(message['username'] + ', ' + dt_str)
        self.messageList.append(message['text'])
        self.messageList.append('')
        self.messageList.repaint()

    def load_messages(self):
        """Подгрузка сообщений за данную сессию"""
        try:
            data = requests.get(self.server_url + '/messages', params={'after':
self.after}).json()
        except:
            return

        for message in data['messages']:
            self.pretty_message(message)
            self.after = message['time']

        if self.isClickedStartButton:
            self.timerValue += 1
            self.timerLabel.setText(str(self.timerValue))
            self.timerLabel.repaint()
```

```
            if self.timerValue >= 10:
                self.timerValue = 0
                self.send_message("КОНЕЦ ИГРЫ")


    def send_message(self, text=""):
        """Обработка нажатия на кнопку Отправить сообщение"""
        text = self.messageText.toPlainText()
        self.messageText.clear()
        self.messageText.repaint()
        message = {
            'username': self.__username,
            'text': text
        }

        try:
            response = requests.post(self.server_url + '/send', json=message)
            # self.prohibitSendingMessages()
        except:
            self.messageList.append('Сервер недоступен. Повторите попытку
позже.')
            self.messageList.repaint()
            self.allowSendingMessages()
            return

        if response.status_code == 400:
            self.messageList.append('КОНЕЦ ИГРЫ!\n')
            self.isClickedStartButton = False
            self.startGame()

            return

        self.timerValue = 0
        self.timerLabel.repaint()
        self.score += 1
        self.scoreLabel.setText("SCORE: " + str(self.score))

    def clickOnStartButton(self):
        """Обработка нажатия на кнопку Начать игру"""
        self.messageText.show()
        self.sendMessageButton.show()
        self.startButton.hide()
        self.isClickedStartButton = True
        if self.messageList:
            self.allowSendingMessages()
            self.messageList.clear()
            self.timerValue = 0
            self.after = 0
            self.score = 0
            self.scoreLabel.setText("SCORE: " + str(self.score))
            self.load_messages()
            self.messageList.repaint()
```

cities.py

```
import csv
import random


class Cities:
    def __init__(self):
        self.__existing_cities = get_existing_cities("city.csv")
        self.__named_cities = []
        self.__last_named_city = self.generate_first_city()
        print(self.__last_named_city)

    def generate_first_city(self):
        """Генерирует рандомный первый город дляачала игры"""
        first_city = random.choice(self.__existing_cities)
        self.__last_named_city = first_city
        self.__named_cities.append(first_city)
        return first_city

    def city_exists(self, city_name):
        """Проверяет существует ли такой город"""
        for city in self.__existing_cities:
            if city.lower() == city_name.lower():
                return True
        return False

    def city_was_named(self, city_name):
        """Проверяет был ли упомняут ранее такой город"""
        for city in self.__named_cities:
            if city.lower() == city_name.lower():
                return True
        return False

    def check_conditions(self, city_name):
        """Проверяет удовлетворяет ли требованиям имени города"""
        if city_name[0].lower() == self.__last_named_city[-1:]:
            return True
        if self.__last_named_city[-1:] == 'ъ' or self.__last_named_city[-1:] ==
'ь':
            if city_name[0].lower() == self.__last_named_city[-2:-1]:
                return True
        return False

    def add_city_to_named(self, city_name):
        """
        Добавляет город в список названных, если он удоавлетворяет условиям
        :return
        0 - успешное добавление
        -1 - город не удовлетворяет требованиям
        """
        if self.city_exists(city_name):
            if self.check_conditions(city_name):
                if not self.city_was_named(city_name):
                    self.__last_named_city = city_name
                    self.__named_cities.append(city_name)
```

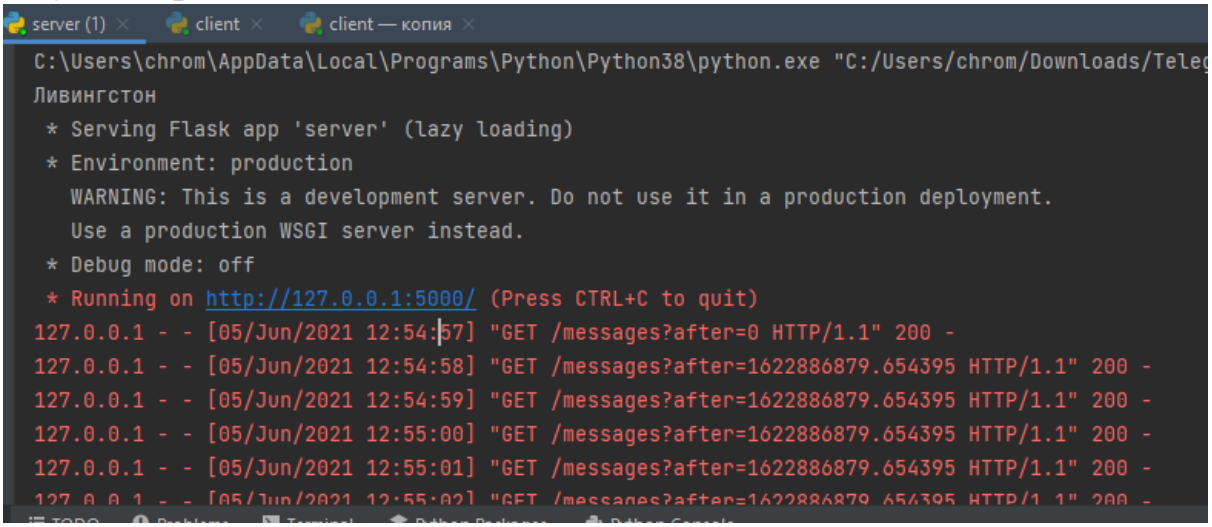
```
                    return 0
        return -1

    def get_last_named_city(self):
        return self.__last_named_city

    def restart(self):
        self.__named_cities = []
        self.__last_named_city = self.generate_first_city()


def get_existing_cities(csv_path):
    """Возвращает список существующих городов России"""
    file_obj = open(csv_path, "r")
    cities = []
    reader = csv.DictReader(file_obj, delimiter=';')
    for line in reader:
        cities.append(line['name'])
    return cities
```

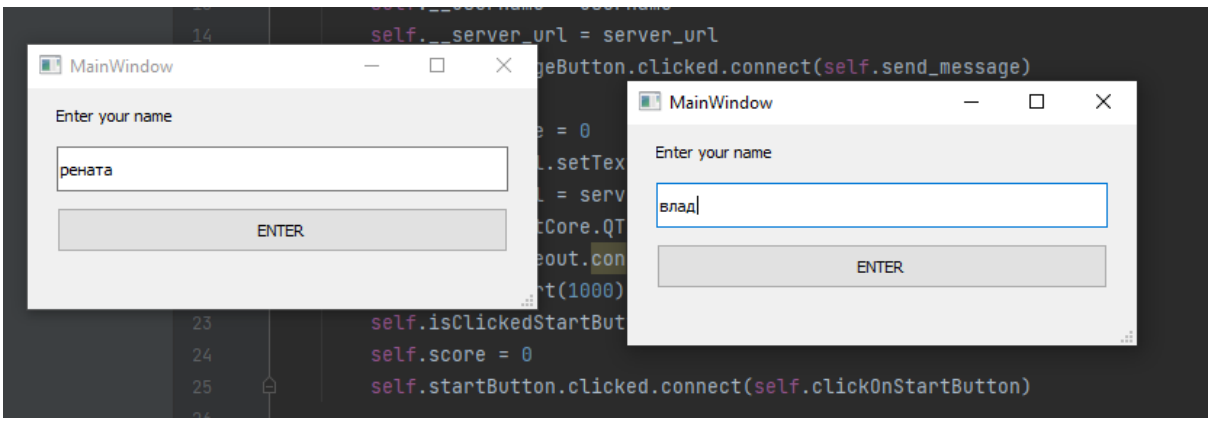
registrationWindow.py

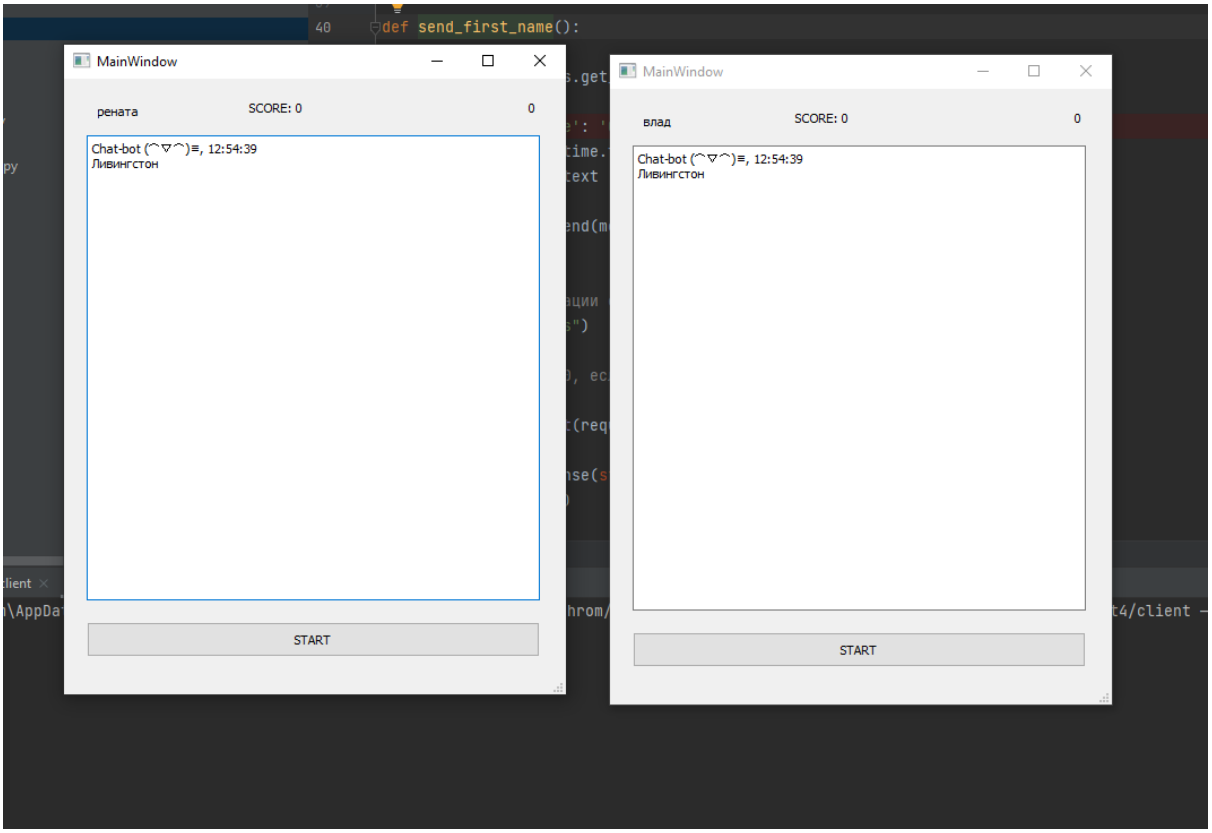
```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'registrationWindow.ui'
#
# Created by: PyQt5 UI code generator 5.15.4
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtWidgets

class Ui_RegistrationWindow(object):
    def setupUi(self, RegistrationWindow):
        self.RegistrationWindow = RegistrationWindow
        self.RegistrationWindow.setObjectName("RegistrationWindow")
        self.RegistrationWindow.resize(351, 152)
        self.centralwidget = QtWidgets.QWidget(self.RegistrationWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.enterNameButton = QtWidgets.QPushButton(self.centralwidget)
        self.enterNameButton.setGeometry(QtCore.QRect(20, 82, 311, 31))
        self.enterNameButton.setObjectName("enterNameButton")
        self.nameText = QtWidgets.QLineEdit(self.centralwidget)
        self.nameText.setGeometry(QtCore.QRect(20, 40, 311, 31))
        self.nameText.setObjectName("nameText")
        self.nameText.setText('')
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(20, 10, 311, 16))
        self.label.setObjectName("label")
        self.RegistrationWindow.setCentralWidget(self.centralwidget)
        self.statusbar = QtWidgets.QStatusBar(self.RegistrationWindow)
        self.statusbar.setObjectName("statusbar")
        self.RegistrationWindow.setStatusBar(self.statusbar)
```

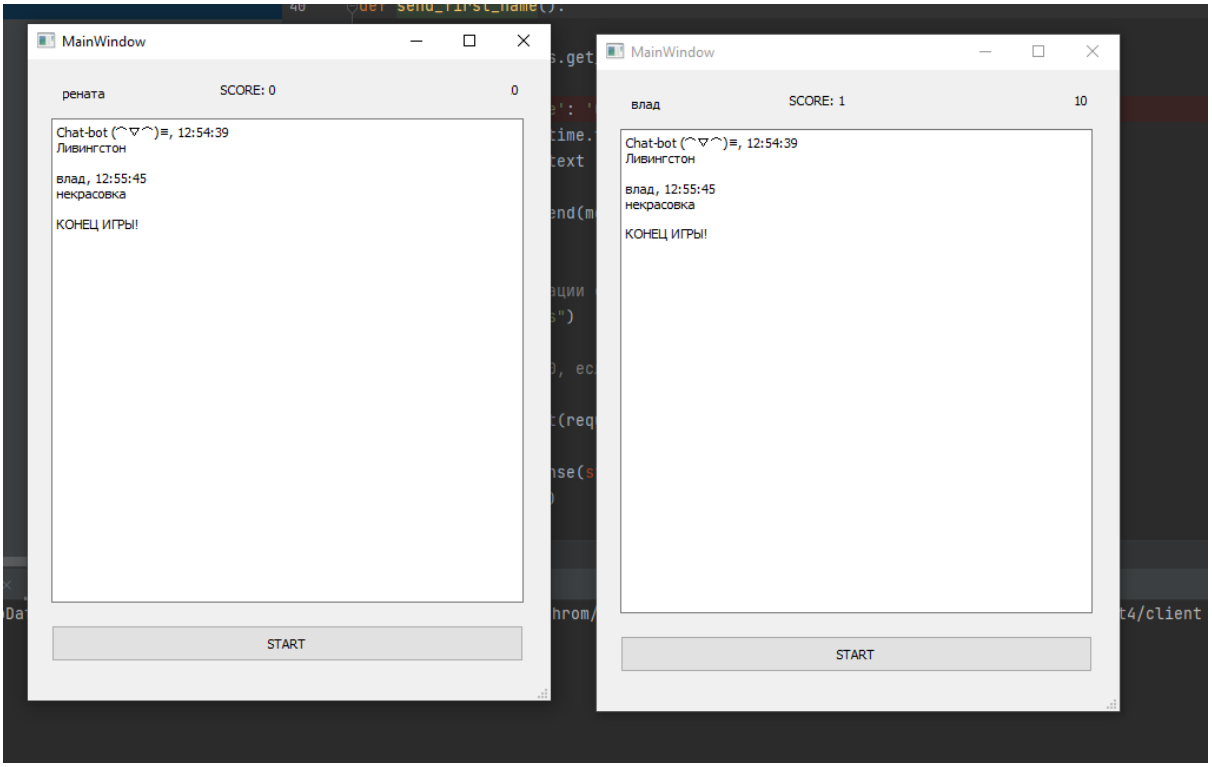
```
        self.retranslateUi()
        self.enterNameButton.clicked.connect(self.registerUser)
        QtCore.QMetaObject.connectSlotsByName(self.RegistrationWindow)

    def retranslateUi(self):
        _translate = QtCore.QCoreApplication.translate
        self.RegistrationWindow.setWindowTitle(_translate("RegistrationWindow",
"MainWindow"))
        self.enterNameButton.setText(_translate("RegistrationWindow", "ENTER"))
        self.label.setText(_translate("RegistrationWindow", "Enter your name"))

    def registerUser(self):
        self.__username = self.nameText.text()
        if self.__username != "":
            self.RegistrationWindow.close()

    def getUsername(self):
        return self.__username
```

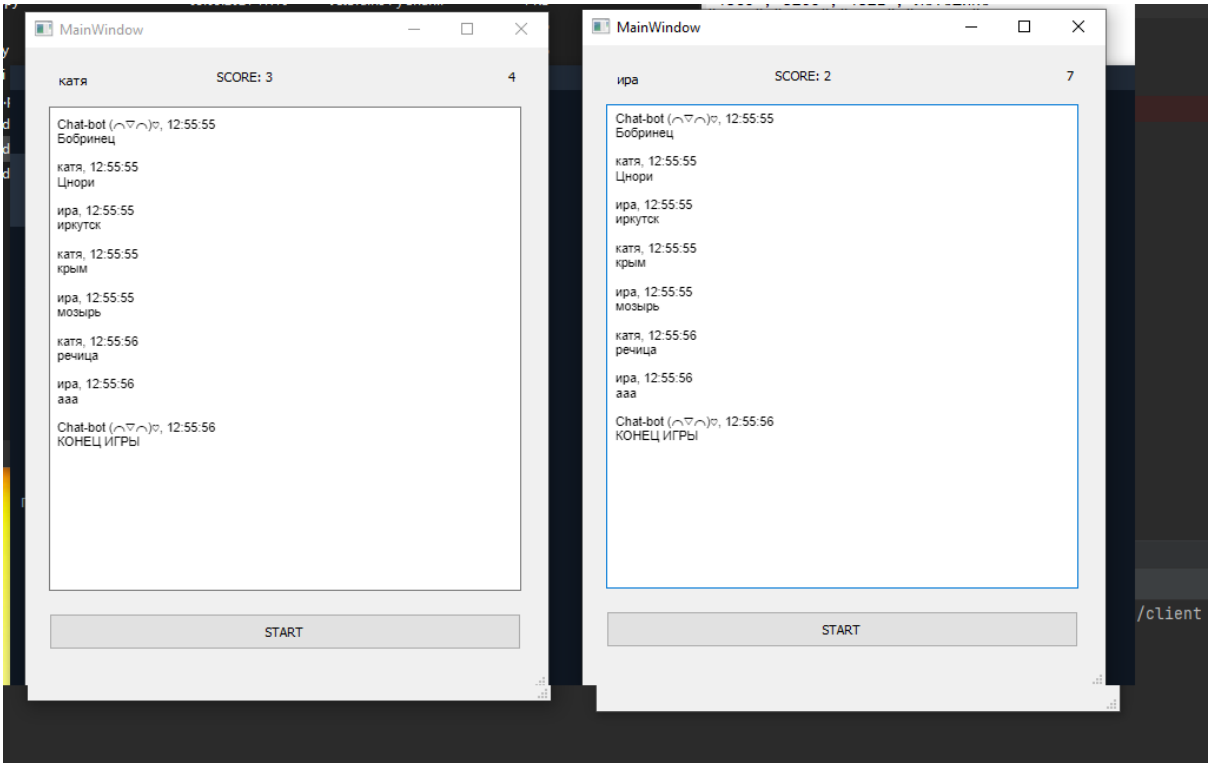
Результат работы:






MainWindow
рената
SCORE: 0
0
Chat-bot (⌒▽⌒)≡, 12:54:39
Ливингстон
START
MainWindow
влад
SCORE: 0
0
Chat-bot (⌒▽⌒)≡, 12:54:39
Ливингстон
START


MainWindow
рената
SCORE: 0
0
Chat-bot (⌒▽⌒)≡, 12:54:39
Ливингстон
влад, 12:55:45
некрасовка
КОНЕЦ ИГРЫ!
START
MainWindow
влад
SCORE: 1
10
Chat-bot (⌒▽⌒)≡, 12:54:39
Ливингстон
влад, 12:55:45
некрасовка
КОНЕЦ ИГРЫ!
START



Вывод: освоила приемы разработки оконных клиент-серверных приложений на Java с использованием сокетов.