Министерство образования и науки Российской Федерации
Федеральное агентство по образованию
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Отчет по лабораторной работе №8 дисциплины
«Разработка программных систем»

Использование протокола XML-RPC

Выполнил студент группы ИВТ-31_____/Крючков И. С./
Проверил_____/Чистяков Г. А./

Киров 2023

1. Цель

Целью работы является знакомство с протоколом XML-RPC, а также получение навыков применения этого протокола для организации серверного взаимодействия.

2. Задание

В соответствии с выбранной тематикой разработать клиентское приложение, делегирующее выполнение вычислительных действий серверному приложению, и серверное приложение, выполняющее вычисления клиента. Для организации взаимосвязи между клиентом и сервером использовать протокол XML-RPC.

Для выполнения лабораторной работы необходимо решить следующие задачи:

- Согласовать тематику разработки с преподавателем
- Разработать структуры клиентского и серверного приложений
- Реализовать приложения
- Продемонстрировать работу приложения.

3. Листинг программы

Листинг программной реализации приведен в приложении А.

4. Вывод

В ходе выполнения лабораторной работы были изучены основные возможности протокола XML-RPC. Разработано серверное приложение, реализующее функции для работы с простыми числами. На основе предыдущей лабораторной работы разработано клиентское приложение, делегирующее выполнение вычислительных действий серверному приложению. Взаимосвязь между клиентом и сервером организована с помощью протокола XML-RPC.

Приложение А.

Листинг программы

SERVER

main.py

```
from xmlrpc.server import SimpleXMLRPCServer
import prime

def main():
    server = SimpleXMLRPCServer(("localhost", 8000))
    print("Listening on port 8000...")

    server.register_function(prime.isPrime, "isPrime")
    server.register_function(prime.factorize, "factorize")
    server.register_function(prime.getNext, "getNext")
    server.register_function(prime.getRandomPrime, "getRandomPrime")

    server.serve_forever()

if __name__ == "__main__":
    main()
```

prime.py

```
import random
import sys
sys.setrecursionlimit(2000)

fact = lambda n, k = 2: [1, n,] if isPrime(n) else [k] + fact(n//k, k) if n % k == 0 else
fact(n, k+1) if k <= n else []

def factorize(n):
    n = int(n)
    return ' '.join(str(x) for x in fact(n))

def isPrime(x):
    x = int(x)
    return len(list(filter(lambda i: x % i == 0, range(2, int(x**0.5) + 1)))) == 0

def getNext(i):
    i = int(i)
    return str(getNext(i + 1) if not isPrime(i + 1) else i + 1)

getRandomPrime = lambda: str(getNext(random.randint(2, 10**12)))
```

CLIENT

app.py

```
from tkinter import *
from tkinter import ttk

from view.controls import Controls

from controller.controller import Controller
from model.prime import Prime


class App(Tk):
    def __init__(self):
        super().__init__()

        prime = Prime()
        connected = prime.connect()
```

```
        if not connected:
            print("Server is not connected")
            self.destroy()
            return

        self.title('Lab')
        self.geometry('250x200')
        self.resizable(False, False)
        self.protocol("WM_DELETE_WINDOW", self.on_closing)

        controls = Controls(self)

        views = {
            'controls': controls
            }

        self.controller = Controller(prime, views)

        controls.set_controller(self.controller)

        controls.create_controls()

        self.grid_columnconfigure(0, weight=1)


    def on_closing(self):
        self.destroy()


if __name__ == '__main__':
    app = App()
    app.mainloop()
```

controller.py

```
from decimal import Decimal

class Controller:
    def __init__(self, model, views):
        self.prime = model
        self.views = views

    def _checkInput(self, value):
        try:
            v = int(value)

            if v < 0:
                return None

            return v
        except ValueError:
            return None

    def check(self, value):

        v = self._checkInput(value)

        if v is None:
            self.views['controls'].showModal(1, "Введите неотрицательное целое число")
            return

        if v == 1 or v == 0:
            self.views['controls'].showModal(0, "Число не является ни простым ни составным")
        else:
            self.views['controls'].showModal(0, "Число простое" if self.prime.isPrime(v) else
"Число составное")

    def factorize(self, value):
```

```
        v = self._checkInput(value)

        if v is None:
            self.views['controls'].showModal(1, "Введите неотрицательное целое число")
            return

        if v == 1 or v == 0:
            self.views['controls'].showModal(0, "Число не является ни простым ни составным")
        else:
            q = self.prime.factorize(v)
            r = list(int(x) for x in q.split())
            self.views['controls'].showModal(0, f"Простые множители: {*r,}" if len(r) > 0 else
"Это простое число")

    def getRandomPrime(self):
        self.views['controls'].setInputValue(self.prime.getRandomPrime())

    def next_prime(self, value):
        v = self._checkInput(value)

        if v is None:
            self.views['controls'].showModal(1, "Введите неотрицательное целое число")
            return

        self.views['controls'].setInputValue(self.prime.getNext(v))
```

prime.py

```
import xmlrpc.client
import socket

class Prime:
    def __init__(self):
        self.proxy = None

    def connect(self):
        connected, self.proxy = self._get_rpc()
        return connected

    def _get_rpc(self):
        a = xmlrpc.client.ServerProxy("http://localhost:8000/")
        try:
            a._()   # Call a fictive method.
        except xmlrpc.client.Fault:
            # connected to the server and the method doesn't exist which is expected.
            pass
        except ConnectionRefusedError:
            return False, None
        else:
            return False, None
        return True, a


    isPrime = lambda self, x: self.proxy.isPrime(str(x))
    factorize = lambda self, n: self.proxy.factorize(str(n))
    getNext = lambda self, i: self.proxy.getNext(str(i))
    getRandomPrime = lambda self: self.proxy.getRandomPrime()
```

view.py

```
from tkinter import *
from tkinter import ttk

class View(ttk.Frame):

    def set_controller():
        raise NotImplementedError
```

controls.py

```
from tkinter import *
from tkinter import ttk
from view.view import View
import tkinter.messagebox as mb

class Controls(View):
    def __init__(self, parent):
        super().__init__(parent)

        self.controller = None

        self.number_input = None

        self.grid_columnconfigure(0, weight=1)

    def set_controller(self, c):
        self.controller = c

    def create_controls(self):

        l_input = Label(self, text="Число")
        l_input.grid(row=0, column=0, padx=10, sticky="w")

        self.number_input = Entry(self)
        self.number_input.grid(row=1, column=0, padx=10, pady=(0, 10), sticky="we")

        check_btn = ttk.Button(self, text='Проверить на простоту', command=self.check)
        check_btn.grid(row=2, column=0, padx=10, pady=2, sticky="we")


        factorize_btn = ttk.Button(self, text='Факторизация', command=self.factorize)
        factorize_btn.grid(row=3, column=0, padx=10, pady=2, sticky="we")

        rnd_btn = ttk.Button(self, text='Случайное простое число', command=self.rnd)
        rnd_btn.grid(row=4, column=0, padx=10, pady=2, sticky="we")

        next_prime_btn = ttk.Button(self, text='Следующее простое число',
command=self.next_prime)
        next_prime_btn.grid(row=5, column=0, padx=10, pady=2, sticky="we")

        self.grid(row=0, column=0, pady=10, padx=10,  sticky="we")

    def check(self):
        if self.controller:
            if self.number_input.get():
                self.controller.check(self.number_input.get())

    def factorize(self):
        if self.controller:
            if self.number_input.get():
                self.controller.factorize(self.number_input.get())

    def rnd(self):
        if self.controller:
            self.controller.getRandomPrime()

    def next_prime(self):
        if self.controller:
            if self.number_input.get():
                self.controller.next_prime(self.number_input.get())

    def showModal(self, mtype, msg):
        if mtype == 0:
            mb.showinfo("Результат", msg)
        elif mtype == 1:
            mb.showerror("Ошибка", msg)
```

```
def setInputValue(self, v):
    self.number_input.delete(0, END)
    self.number_input.insert(0, v)
```