Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

кафедра физики и прикладной математики

ОТЧЕТ

ПО КУРСОВОЙ РАБОТЕ

по дисциплине «Разработка кросс-платформенных приложений»

на тему

Клиент-серверное приложение

Студента Овчинниковой Вероники Александровны

Направления 02.03.02 «Фундаментальная информатика и информационные технологии»

Группы ИТ-122

*Руководитель*:

доцент кафедры ФиПМ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. Ю. Лексин

Владимир 2024

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc185619580)

[1. ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ 3](#_Toc185619581)

[2. ОПИСАНИЕ ХРАНЕНИЯ ДАННЫХ 4](#_Toc185619582)

[3. МАКЕТЫ СТРАНИЦ 4](#_Toc185619583)

[4. ДИАГРАММА КЛАССОВ 6](#_Toc185619584)

[5. UML-ДИАГРАММА 7](#_Toc185619585)

[6. ДИАГРАММА ДЕЯТЕЛЬНОСТИ 8](#_Toc185619586)

[7. РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ 9](#_Toc185619587)

[ПРИЛОЖЕНИЕ 13](#_Toc185619588)

[Клиент 13](#_Toc185619589)

[Сервер 19](#_Toc185619590)

# ВВЕДЕНИЕ

Цель работы: разработать клиент-серверное приложение. Сервер рассылает сигналы точного времени, клиент синхронизирует свое время.

# 1. ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ

1. Настройка сервера:

-Регулярная рассылка сигналов точного времени.

2. Рассылка сигналов времени:

- Сервер отправляет актуальные временные сигналы клиентам с учетом

установленного часового пояса.

3. Настройка клиента:

 - Получение временных сигналов от сервера.

- Возможность выбора часового пояса пользователем.

- Задание интервала синхронизации времени (например, пользователь

может выбрать, как часто синхронизировать время).

4. Синхронизация времени:

- Автоматическое обновление локального времени клиента на основе полученных сигналов.

-Синхронизация возможна если пользователь вошел в систему.

5. Обработка ошибок:

- Проверка наличия соединения с сервером.

- Обработка ошибок, связанных с получением временных сигналов.

6. Вход/регистрация

-Клиент отправляет данные серверу.

-Сервер проверяет существует ли пользователь в системе.

-Сервер вносит изменения в БД.

# 2. ОПИСАНИЕ ХРАНЕНИЯ ДАННЫХ

В качестве СУБД было выбрано **PostgreSQL.**

**Общая структура БД.**

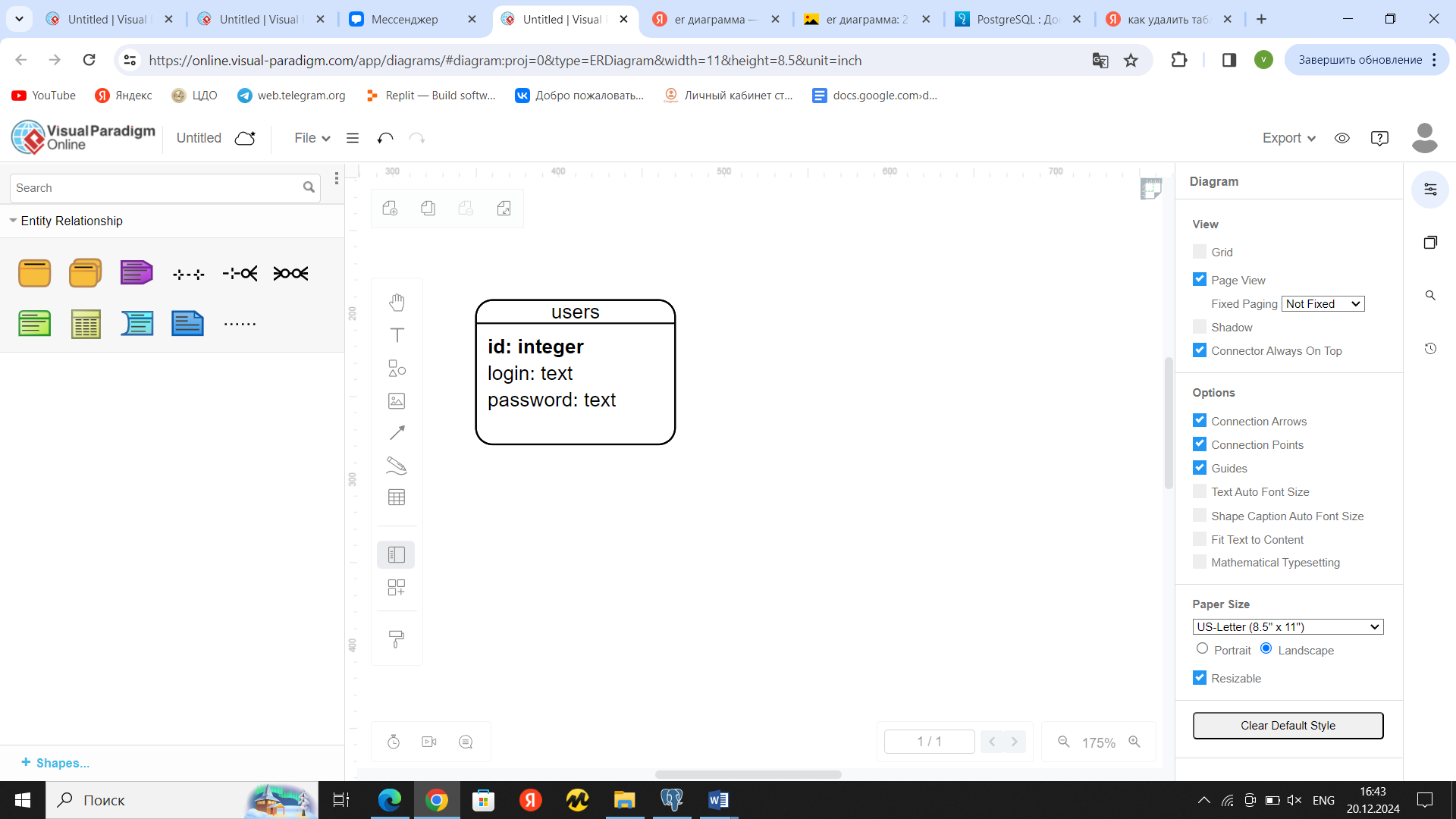
Сущность users хранит данные о зарегистрированных пользователях, которая включает атрибуты:

-id;

-login;

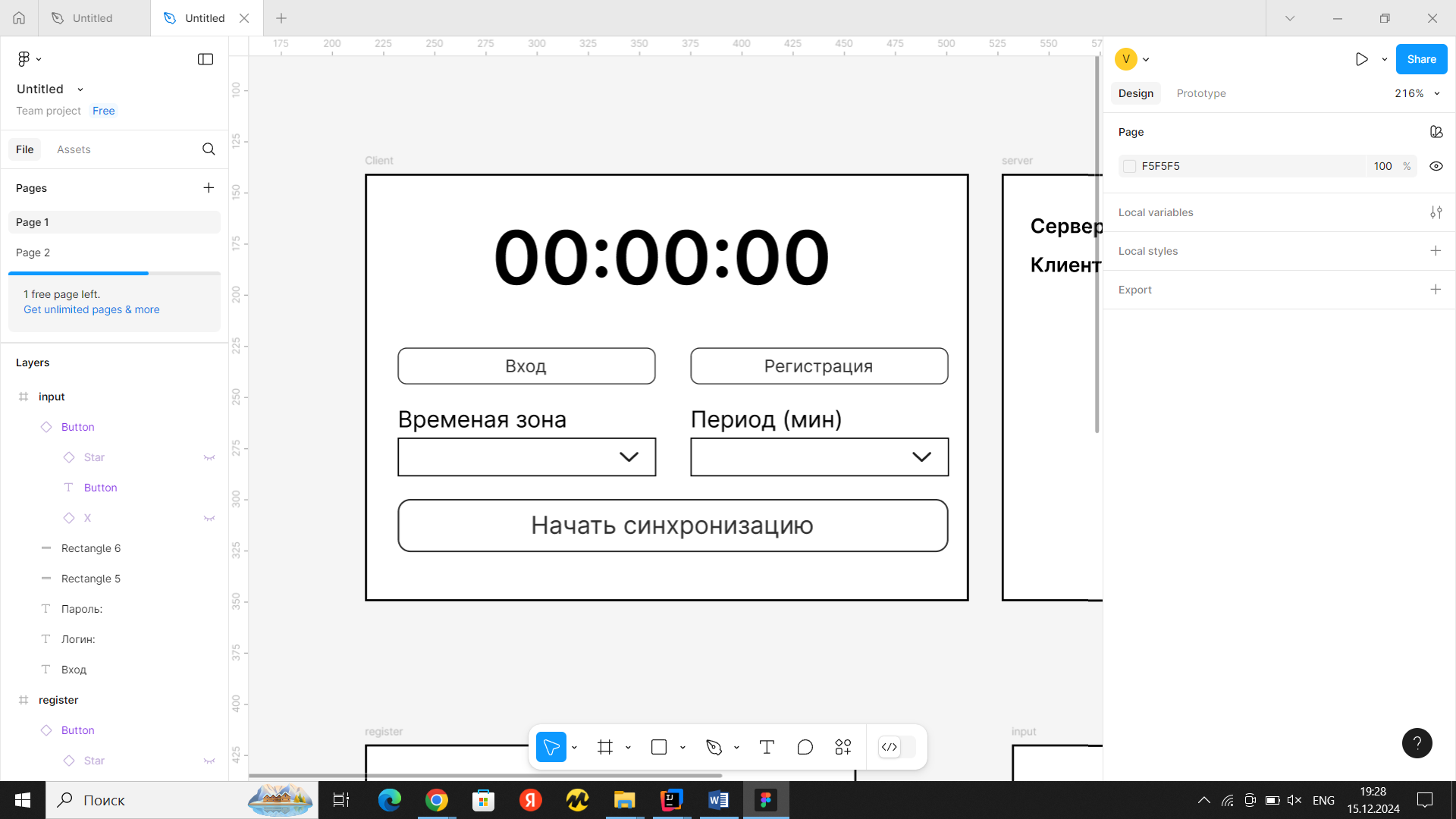
-password.

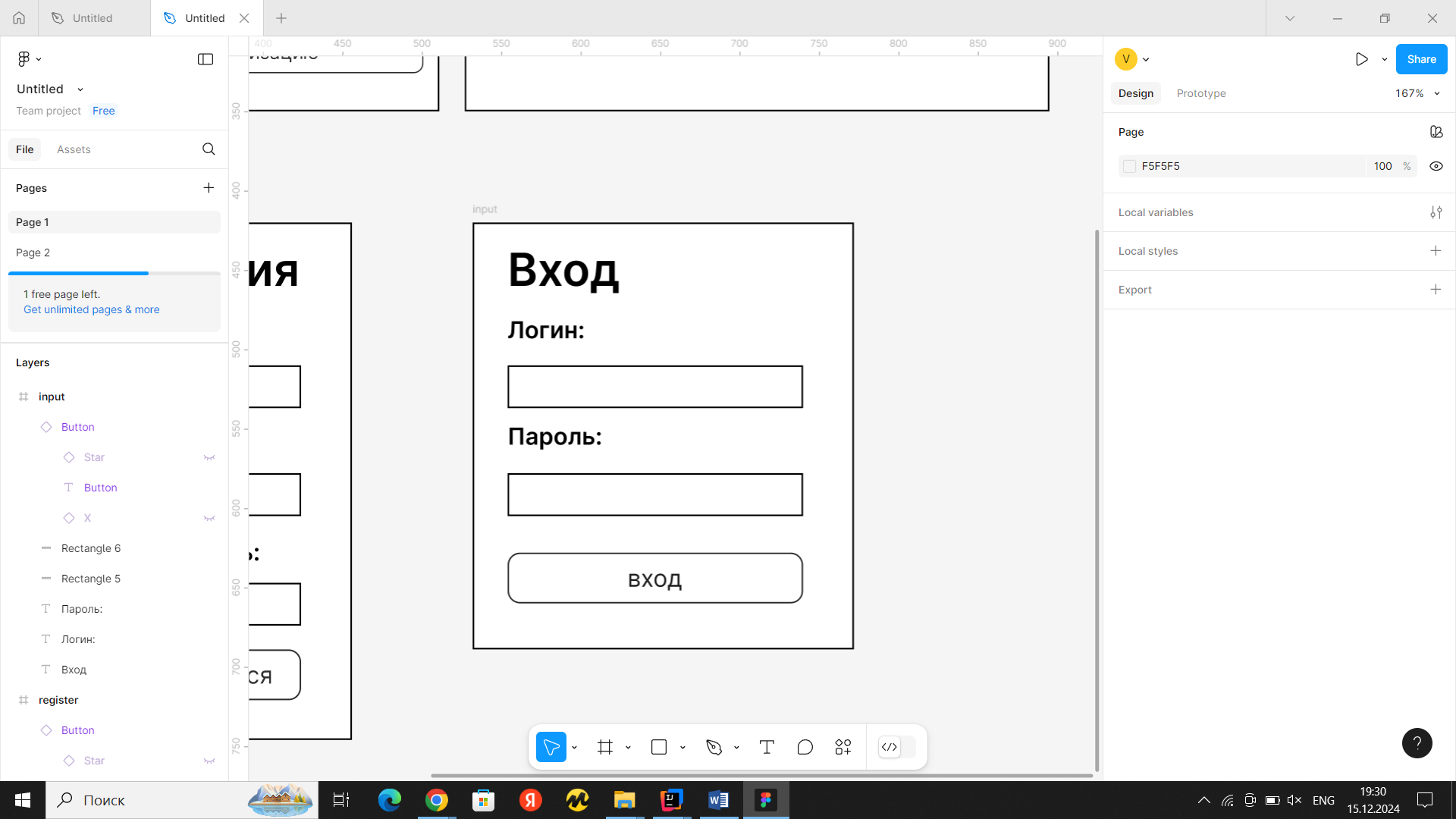
ER-диаграмма.

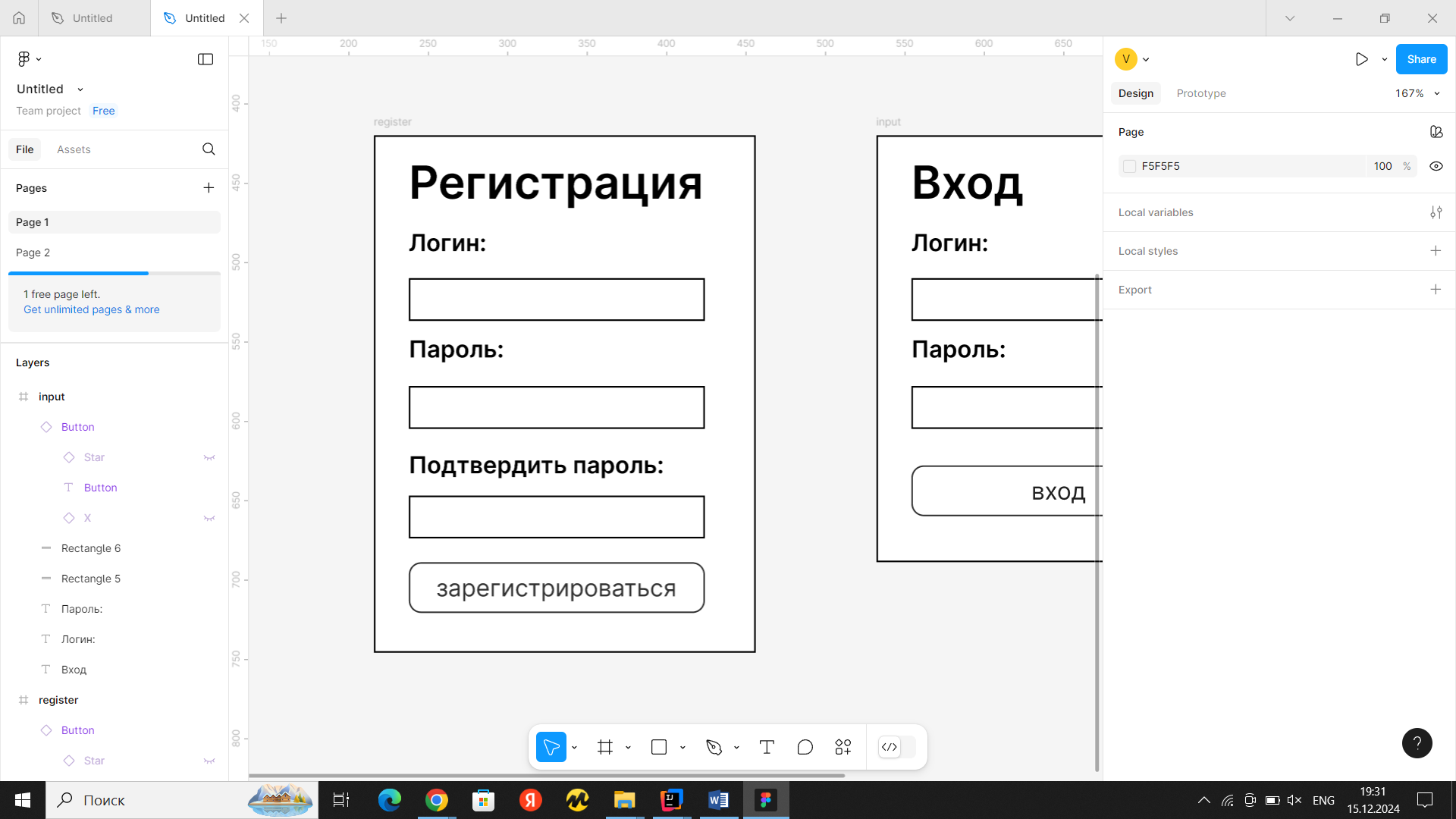


# 3. МАКЕТЫ СТРАНИЦ

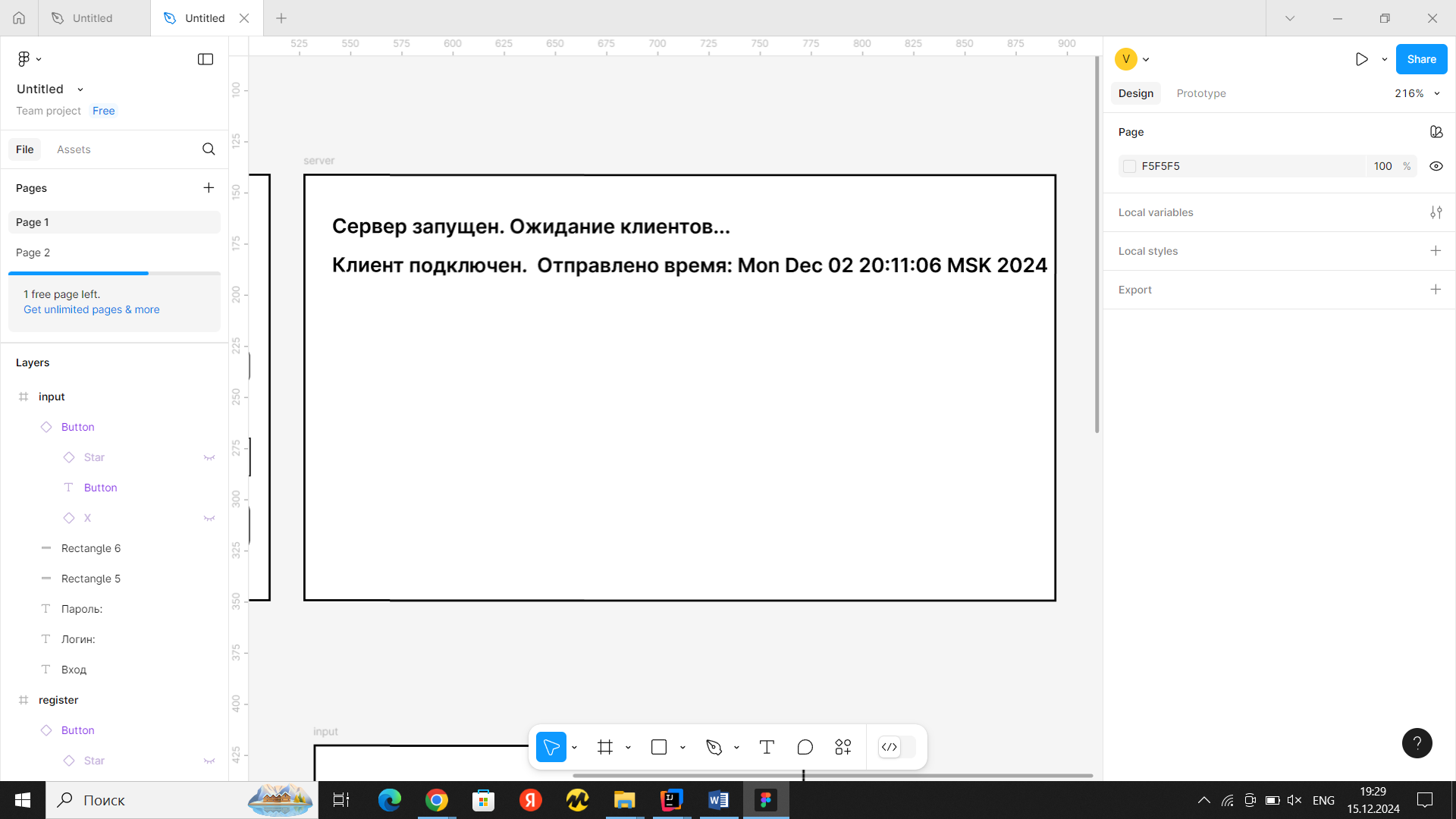
Клиент



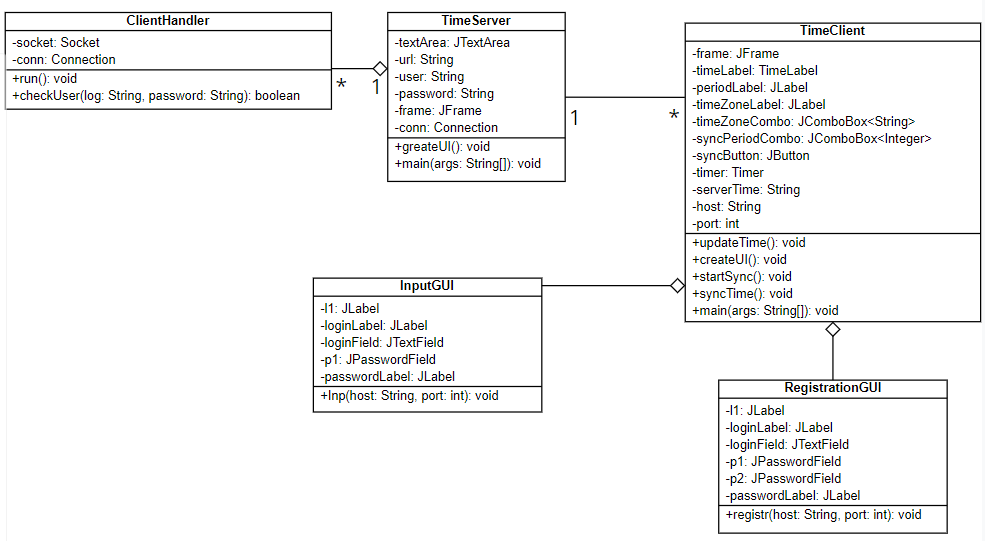




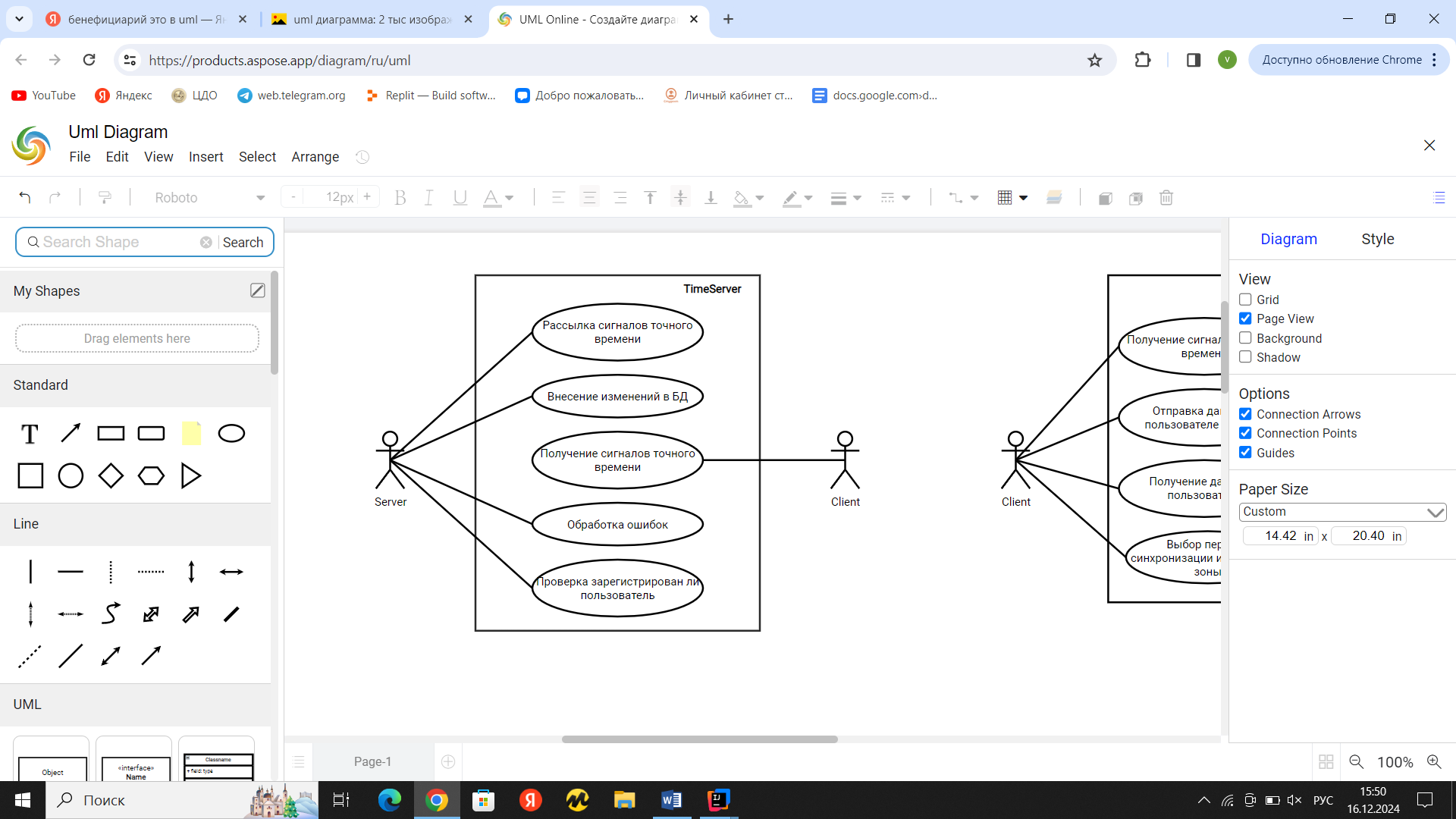
Сервер

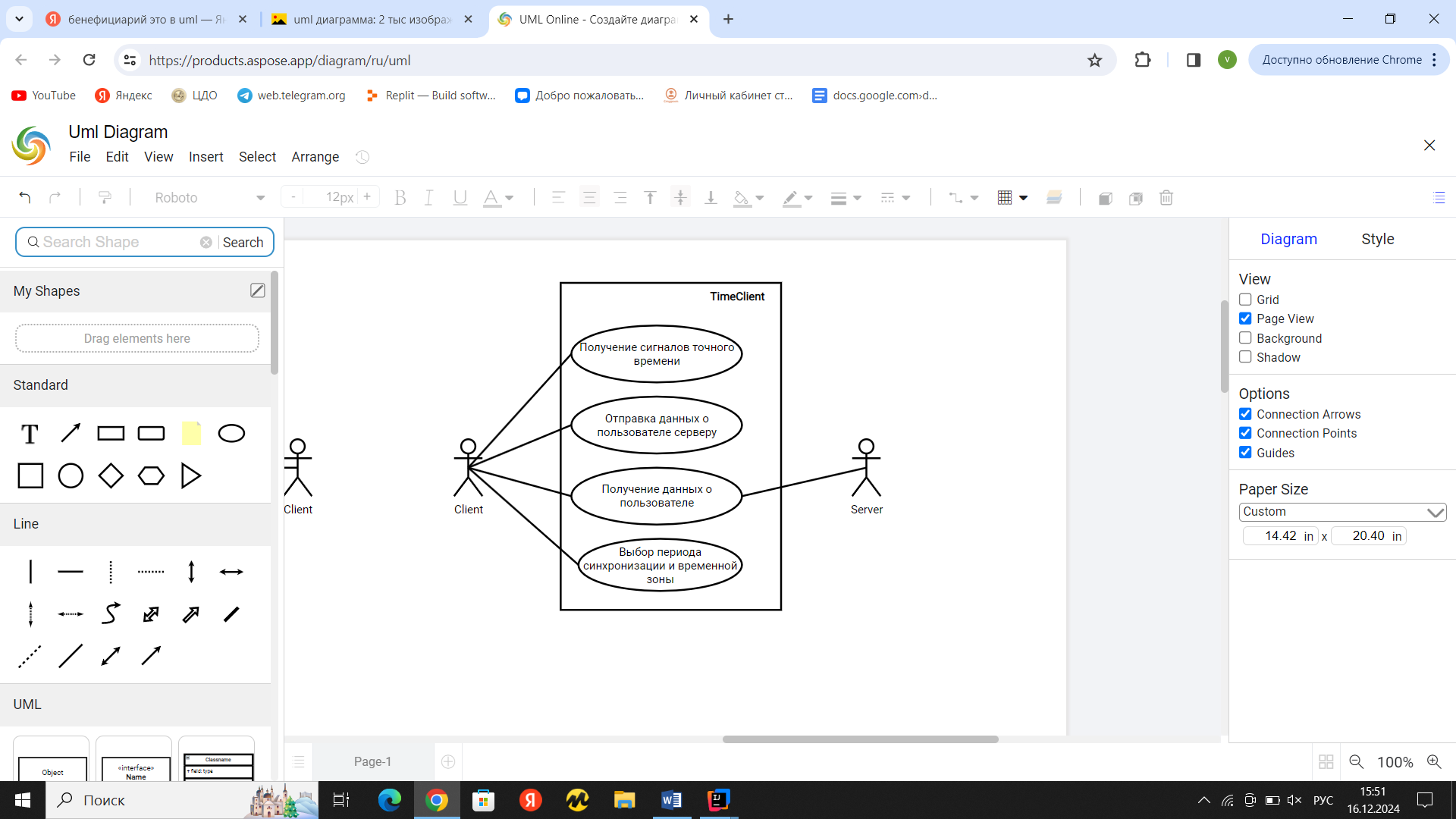


# 4. ДИАГРАММА КЛАССОВ

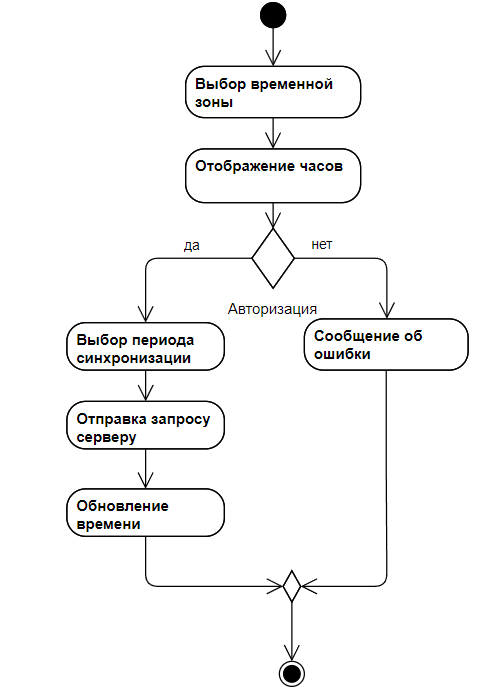


# 5. UML-ДИАГРАММА

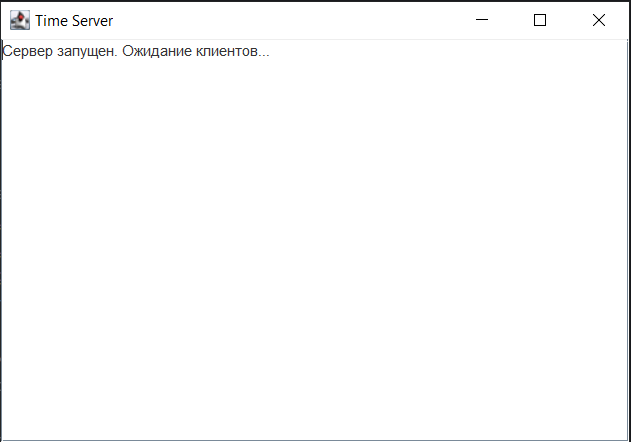


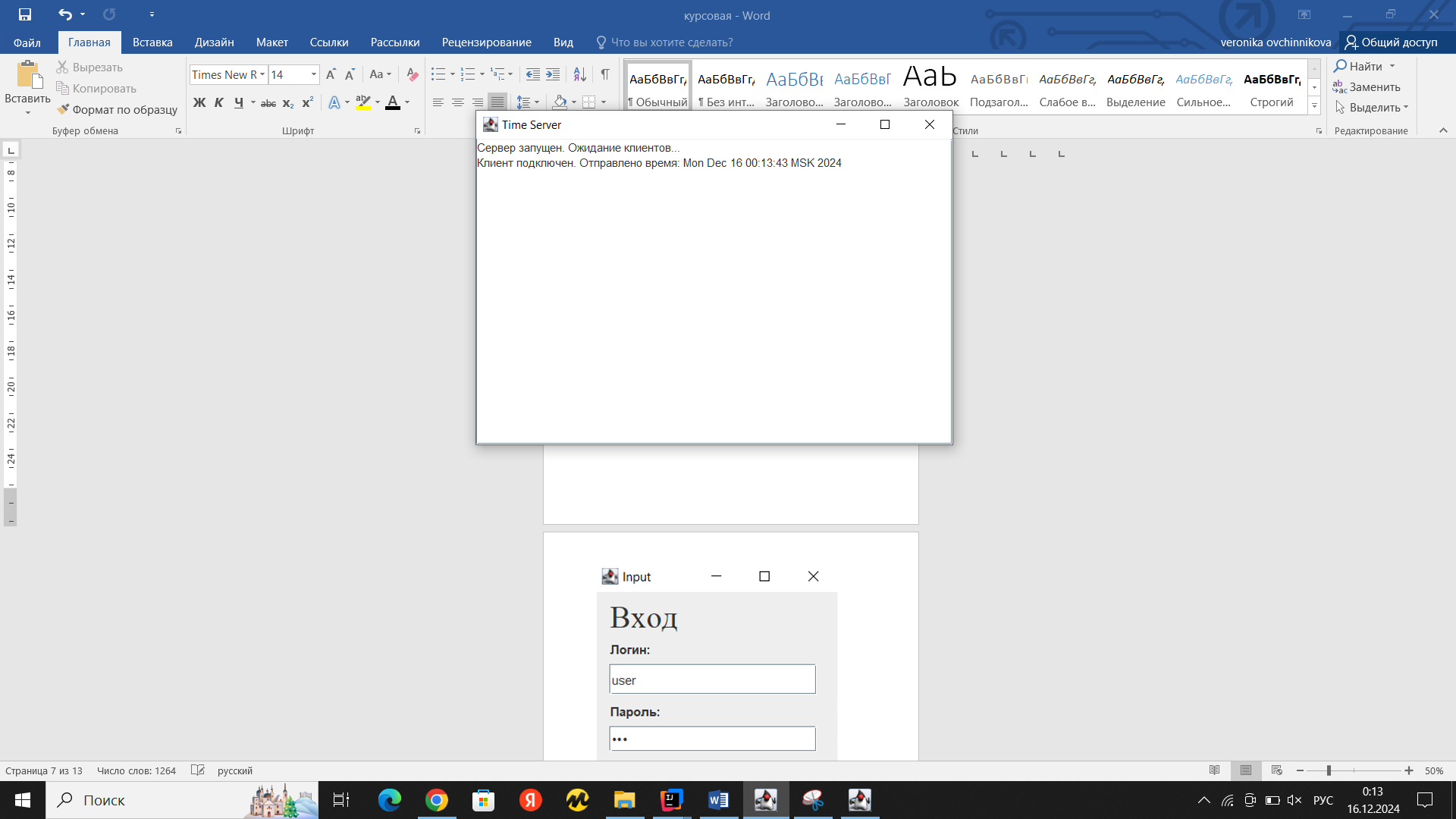


# 6. ДИАГРАММА ДЕЯТЕЛЬНОСТИ

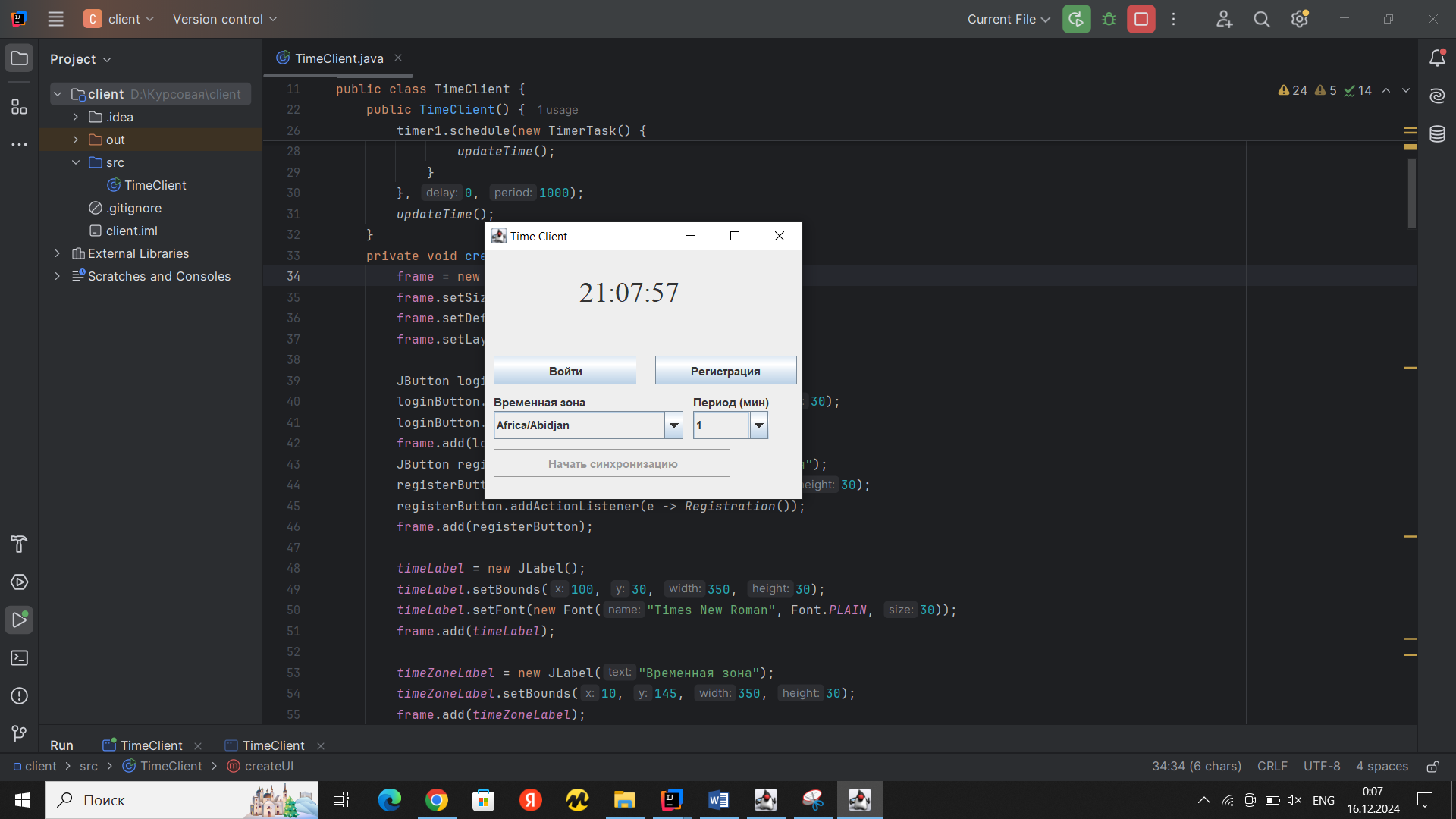


# 7. РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

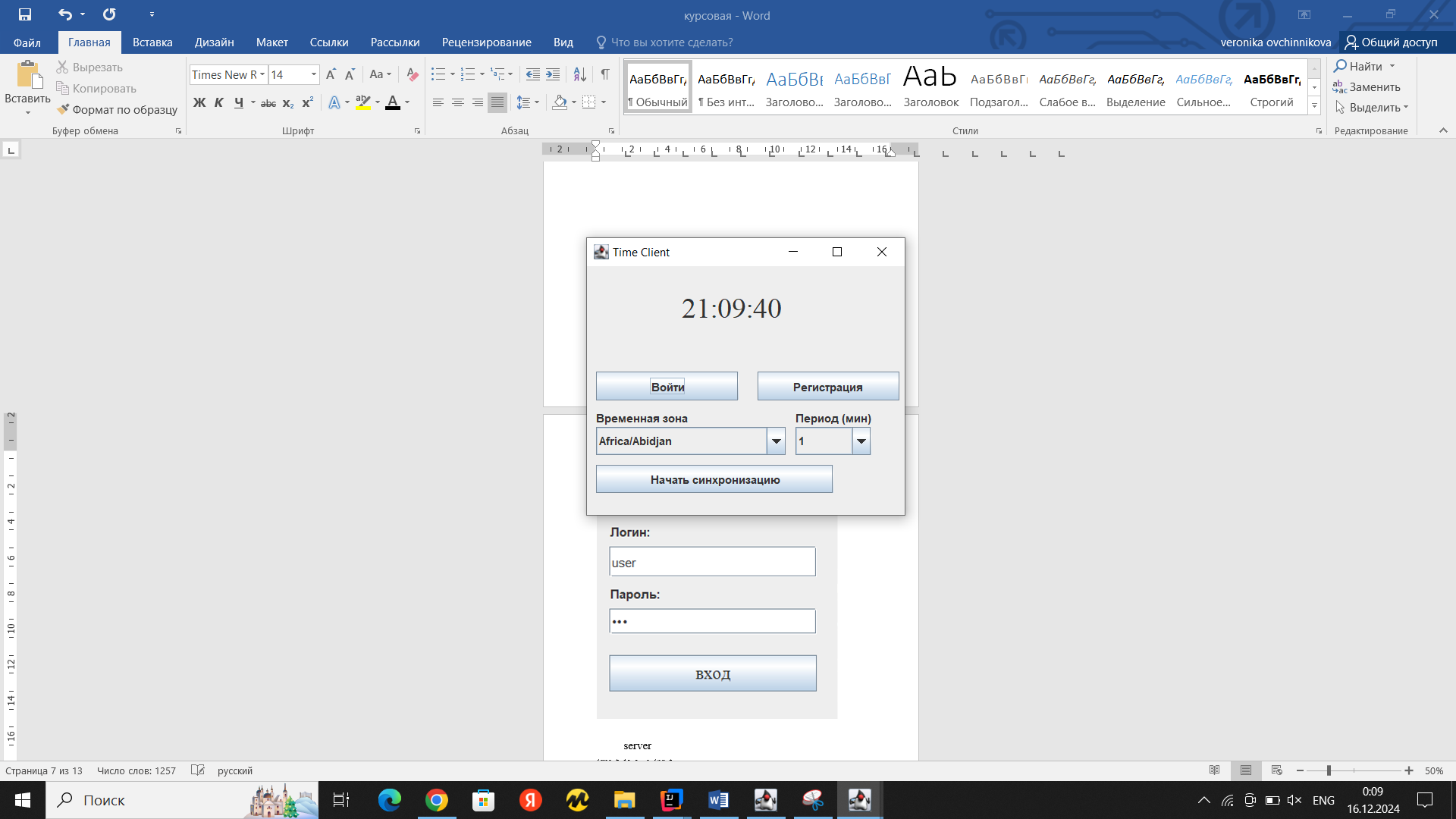


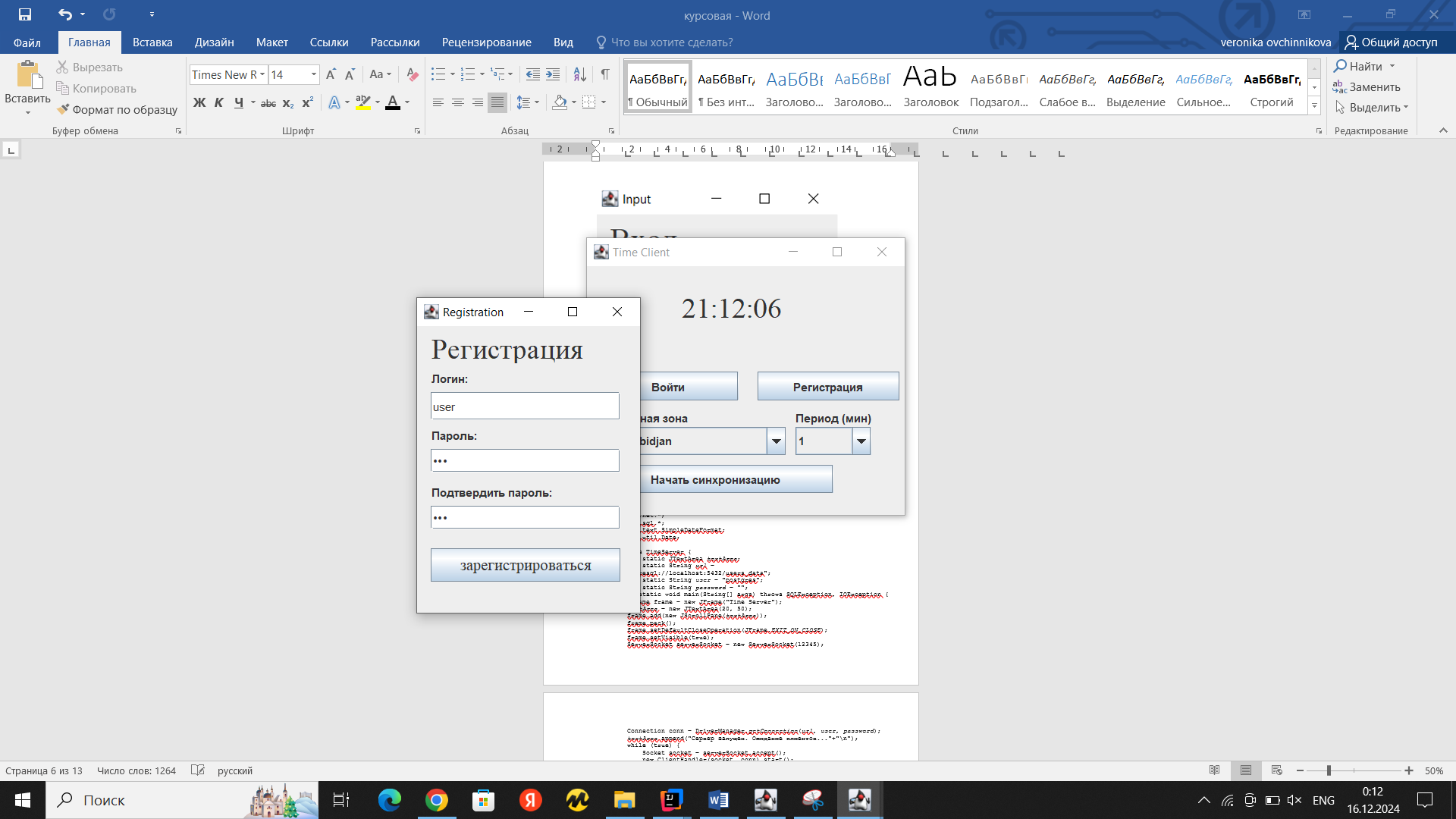


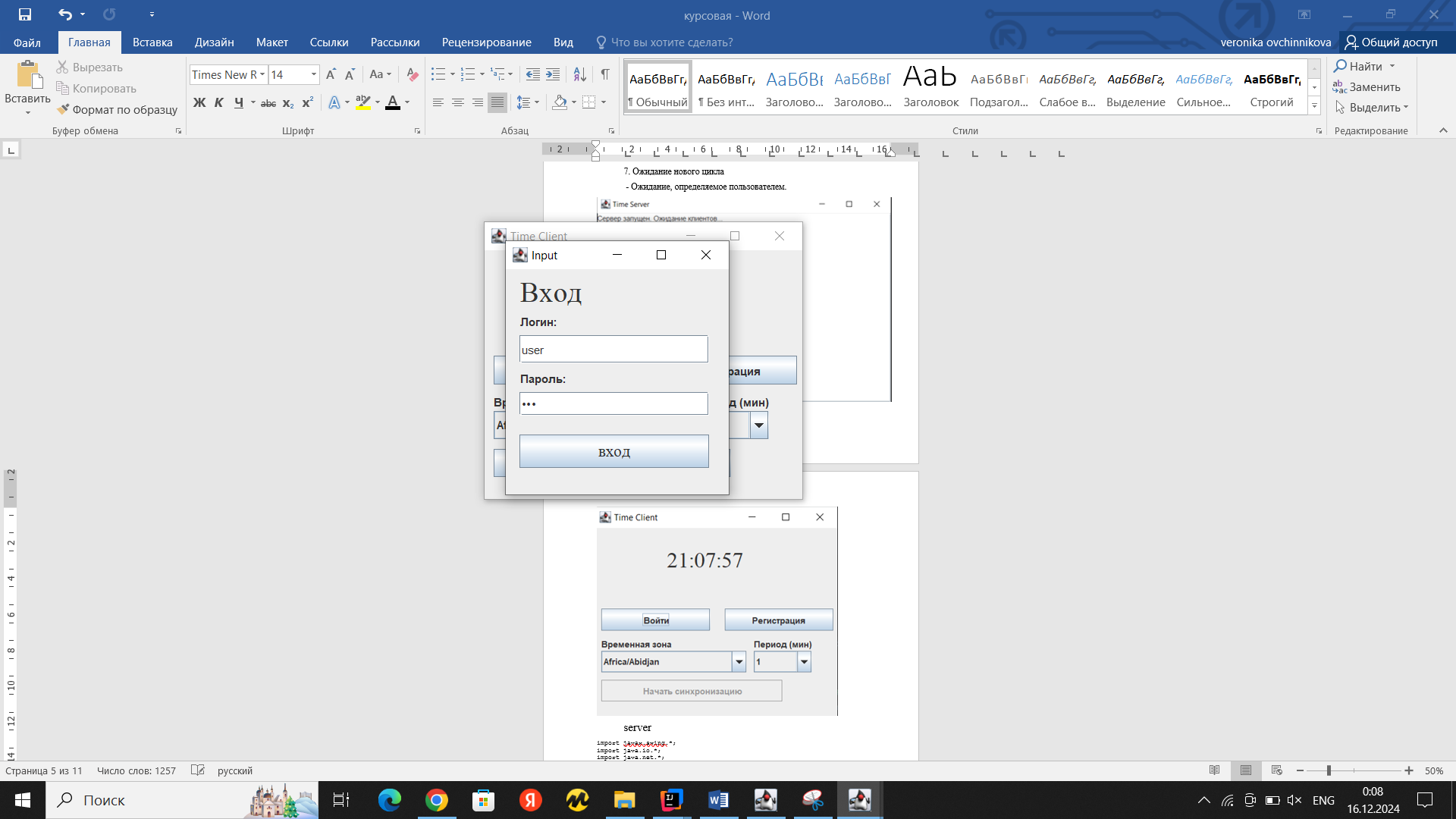
Пользователь не зашел в систему



Авторизированный пользователь







# ПРИЛОЖЕНИЕ

## Клиент

import javax.swing.\*;  
import java.awt.\*;  
import java.util.\*;  
import java.net.\*;  
import java.io.\*;  
import java.util.Timer;  
import java.util.Calendar;  
import java.util.TimeZone;  
import static javax.swing.JOptionPane.*showMessageDialog*;  
public class Registration {  
 static JLabel *l1*,*loginLable*, *passwordLable*;  
 static JTextField *loginField*;  
 static JPasswordField *p1*, *p2*;  
 public static void Registr(String host, int port) {  
  
 JFrame frame1 = new JFrame("Registration");  
 *l1* = new JLabel("Регистрация");  
 *l1*.setFont(new Font("Times New Roman", Font.*PLAIN*, 30));  
 *l1*.setBounds(15, 10, 200, 30);  
 frame1.add(*l1*);  
  
 *loginLable* = new JLabel("Логин:");  
 *loginLable*.setBounds(15, 40, 200, 30);  
 frame1.add(*loginLable*);  
 *loginField* = new JTextField();  
 *loginField*.setBounds(15, 70, 200, 30);  
 frame1.add(*loginField*);  
  
  
 *passwordLable*= new JLabel("Пароль:");  
 *passwordLable*.setBounds(15, 100, 200, 30);  
 frame1.add(*passwordLable*);  
 *p1* = new JPasswordField();  
 *p1*.setBounds(15, 130, 200, 25);  
 frame1.add(*p1*);  
  
 *passwordLable*= new JLabel("Подтвердить пароль:");  
 *passwordLable*.setBounds(15, 160, 200, 30);  
 frame1.add(*passwordLable*);  
 *p2* = new JPasswordField();  
 *p2*.setBounds(15, 190, 200, 25);  
 frame1.add(*p2*);  
  
 JButton buttonRegistration = new JButton("зарегистрироваться");  
 buttonRegistration.setBounds(15, 235, 200, 35);  
 buttonRegistration.addActionListener(e -> {  
 String login = *loginField*.getText().replaceAll("\\s", "");;  
 char[] s3 = *p1*.getPassword();  
 char[] s4 = *p2*.getPassword();  
 String s8 = new String(s3);  
 String s9 = new String(s4);  
 if(s8.equals("") | s9.equals("") | Objects.*equals*(login, "")){  
 *showMessageDialog*(null, "Все поля должны быть заполнены");  
 }else{  
 if(s8.equals(s9)){  
 try (Socket socket = new Socket(host, port)){  
  
 BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 PrintWriter out = new PrintWriter(new OutputStreamWriter(socket.getOutputStream()));  
 out.println("true");  
 out.println("true");  
 out.println(login);  
 out.println(s8);  
 out.flush();  
  
 if(Objects.*equals*(in.readLine(), "Registration successful")){  
 *showMessageDialog*(frame1, "Registration successful");  
 frame1.setVisible(false);  
 frame1.getContentPane().removeAll();  
 TimeClient.*syncButton*.setEnabled(true);  
  
 }else{  
 *showMessageDialog*(frame1,"Registration faild");  
  
 }  
 out.close();  
 in.close();  
 } catch (IOException e1) {  
 *showMessageDialog*(null, e1.getMessage());  
 }  
 }else{  
 *showMessageDialog*(null, "Пароли должны совпадать");  
 }  
 }  
 });  
 buttonRegistration.setFont(new Font("Times New Roman", Font.*PLAIN*, 17));  
 frame1.add(buttonRegistration);  
 frame1.setSize(250, 340);  
 frame1.setLayout(null);  
 frame1.setVisible(true);  
 }  
}

public class Input {  
 static JLabel *l1*,*loginLable*, *passwordLable*;  
 static JTextField *loginField*;  
 static JPasswordField *p1*;  
 public static void Inp(String host, int port) {  
  
 JFrame frame1 = new JFrame("Input");  
 *l1* = new JLabel("Вход");  
 *l1*.setFont(new Font("Times New Roman", Font.*PLAIN*, 30));  
 *l1*.setBounds(15, 10, 200, 30);  
 frame1.add(*l1*);  
  
 *loginLable* = new JLabel("Логин:");  
 *loginLable*.setBounds(15, 40, 200, 30);  
 frame1.add(*loginLable*);  
 *loginField* = new JTextField();  
 *loginField*.setBounds(15, 70, 200, 30);  
 frame1.add(*loginField*);  
  
  
 *passwordLable*= new JLabel("Пароль:");  
 *passwordLable*.setBounds(15, 100, 200, 30);  
 frame1.add(*passwordLable*);  
 *p1* = new JPasswordField();  
 *p1*.setBounds(15, 130, 200, 25);  
 frame1.add(*p1*);  
 JButton buttonRegistration = new JButton("вход");  
 buttonRegistration.setBounds(15, 175, 200, 35);  
 buttonRegistration.addActionListener(e -> {  
 String login = *loginField*.getText().replaceAll("\\s", "");;  
 char[] s3 = *p1*.getPassword();  
 String password = new String(s3);  
 if(password.equals("") | Objects.*equals*(login, "")){  
 *showMessageDialog*(null, "Все поля должны быть заполнены");  
 }else{  
 try (Socket socket = new Socket(host, port)){  
 BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 PrintWriter out = new PrintWriter(new OutputStreamWriter(socket.getOutputStream()));  
 out.println("false");  
 out.println("false");  
 out.println(login);  
 out.println(password);  
 out.flush();  
 if(Objects.*equals*(in.readLine(), "false")){  
 *showMessageDialog*(frame1, "Not exsist");  
  
 }else{  
 *showMessageDialog*(frame1, "exsist");  
  
 TimeClient.*syncButton*.setEnabled(true);  
 frame1.setVisible(false);  
 frame1.getContentPane().removeAll();  
 out.close();  
 in.close();  
 }  
  
 } catch (IOException e1) {  
 *showMessageDialog*(null, e1.getMessage());  
 }  
 }  
 });  
 buttonRegistration.setFont(new Font("Times New Roman", Font.*PLAIN*, 17));  
 frame1.add(buttonRegistration);  
 frame1.setSize(250, 275);  
 frame1.setLayout(null);  
 frame1.setVisible(true);  
 }  
}

public class TimeClient {  
 private JFrame frame;  
 private static JLabel *timeLabel*;  
 private static JLabel *periodLabel*, *timeZoneLabel*;  
 private static JComboBox<String> *timeZoneCombo*;  
 private JComboBox<Integer> syncPeriodCombo;  
 public static JButton *syncButton*;  
 private Timer timer;  
 private static String *serverTime*;  
 private static String *host* = "localhost";  
 private static int *port* = 12345;  
 public TimeClient() {  
 createUI();  
 Timer timer1;  
 timer1 = new Timer(true);  
 timer1.schedule(new TimerTask() {  
 public void run() {  
 *updateTime*();  
 }  
 }, 0, 1000);  
 *updateTime*();  
 }  
 public static void updateTime() {  
 String selectedTimeZone = (String) *timeZoneCombo*.getSelectedItem();  
 Calendar calendar = Calendar.*getInstance*(TimeZone.*getTimeZone*(selectedTimeZone));  
 String time = String.*format*("%02d:%02d:%02d",  
 calendar.get(Calendar.*HOUR\_OF\_DAY*),  
 calendar.get(Calendar.*MINUTE*),  
 calendar.get(Calendar.*SECOND*));  
 *timeLabel*.setText(time);  
 }  
 private void createUI() {  
 frame = new JFrame("Time Client");  
 frame.setSize(350, 300);  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 frame.setLayout(null);  
  
 JButton loginButton = new JButton("Войти");  
 loginButton.setBounds(10, 112, 150, 30);  
 loginButton.addActionListener(e -> Input.*Inp*(*host*, *port*));  
 frame.add(loginButton);  
 JButton registerButton = new JButton("Регистрация");  
 registerButton.setBounds(180, 112, 150, 30);  
 registerButton.addActionListener(e -> Registration.*Registr*(*host*, *port*));  
 frame.add(registerButton);  
  
 *timeLabel* = new JLabel();  
 *timeLabel*.setBounds(100, 30, 350, 30);  
 *timeLabel*.setFont(new Font("Times New Roman", Font.*PLAIN*, 30));  
 frame.add(*timeLabel*);  
  
 *timeZoneLabel* = new JLabel("Временная зона");  
 *timeZoneLabel*.setBounds(10, 145, 350, 30);  
 frame.add(*timeZoneLabel*);  
 *timeZoneCombo* = new JComboBox<>(TimeZone.*getAvailableIDs*());  
 *timeZoneCombo*.setBounds(10, 170, 200, 30);  
 frame.add(*timeZoneCombo*);  
  
 *periodLabel* = new JLabel("Период (мин)");  
 *periodLabel*.setBounds(220, 145, 200, 30);  
 frame.add(*periodLabel*);  
 Integer[] periods = {1, 5, 10, 30, 60};  
 syncPeriodCombo = new JComboBox<>(periods);  
 syncPeriodCombo.setBounds(220, 170, 80, 30);  
 frame.add(syncPeriodCombo);  
  
 *syncButton* = new JButton("Начать синхронизацию");  
 *syncButton*.setBounds(10, 210, 250, 30);  
 *syncButton*.addActionListener(e -> startSync());  
 *syncButton*.setEnabled(false);  
 frame.add(*syncButton*);  
 frame.setVisible(true);  
 }  
 private void startSync() {  
 if (timer != null) {  
 timer.cancel();  
 }  
  
 int period = (Integer) syncPeriodCombo.getSelectedItem();  
 timer = new Timer(true);  
 timer.schedule(new TimerTask() {  
 @Override  
 public void run() {  
 syncTime();  
 }  
 }, 0, period \* 60 \* 1000);  
 }  
 private void syncTime() {  
 try (Socket socket = new Socket(*host*, *port*);  
 PrintWriter out = new PrintWriter(new OutputStreamWriter(socket.getOutputStream()));  
 BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()))) {  
 out.println("TIME");  
 out.flush();  
 *serverTime* = in.readLine();  
 String selectedTimeZone = (String) *timeZoneCombo*.getSelectedItem();  
 Calendar calendar = Calendar.*getInstance*(TimeZone.*getTimeZone*(selectedTimeZone));  
 *serverTime* = String.*format*("%02d:%02d:%02d",  
 calendar.get(Calendar.*HOUR\_OF\_DAY*),  
 calendar.get(Calendar.*MINUTE*),  
 calendar.get(Calendar.*SECOND*));  
 *timeLabel*.setText(*serverTime*);  
 out.close();  
 in.close();  
  
 } catch (IOException e) {  
 *showMessageDialog*(null, e.getMessage());  
 }  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(TimeClient::new);  
 }  
}

## Сервер

import javax.swing.\*;  
import java.io.\*;  
import java.net.\*;  
import java.sql.\*;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
public class TimeServer {  
 private static JTextArea *textArea*;  
 private static String *url* = "jdbc:postgresql://localhost:5432/users\_data";  
 private static String *user* = "postgres";  
 private static String *password* = "";  
 private static JFrame *frame*;  
 private static Connection *conn*;  
  
 public static void greateUI(){  
 *frame* = new JFrame("Time Server");  
 *textArea* = new JTextArea(20, 50);  
 *frame*.add(new JScrollPane(*textArea*));  
 *frame*.pack();  
 *frame*.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 *frame*.setVisible(true);  
 *textArea*.append("Сервер запущен. Ожидание клиентов..."+"\n");  
 }  
 public static void main(String[] args) throws SQLException, IOException {  
 *greateUI*();  
 ServerSocket serverSocket = new ServerSocket(12345);  
 *conn* = DriverManager.*getConnection*(*url*, *user*, *password*);  
 while (true) {  
 Socket socket = serverSocket.accept();  
 new ClientHandler(socket, *conn*).start();  
 }  
 }  
 static class ClientHandler extends Thread {  
 private Socket socket;  
 private Connection conn;  
 public ClientHandler(Socket socket, Connection conn) {  
 this.socket = socket;  
 this.conn = conn;  
 }  
 public void run() {  
 try (BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 PrintWriter out = new PrintWriter(socket.getOutputStream(), true)) {  
 if(in.readLine().equals("TIME")){  
 out.println(new SimpleDateFormat("HH:mm:ss"));  
 *textArea*.append("Клиент подключен. Отправлено время: " + new Date() + "\n");  
 }  
 else{  
 String tmp = in.readLine();  
 String login = in.readLine();  
 String password = in.readLine();  
 if(tmp.equals("true")){  
 if (!*checkUser*(login, password)) {  
 String sql = "INSERT INTO users(login, password) VALUES(?, ?)";  
 PreparedStatement stmt = conn.prepareStatement(sql);  
 stmt.setString(1, login);  
 stmt.setString(2, password);  
 stmt.executeUpdate();  
 out.println("Registration successful");  
 out.println(new Date().toString());  
 *textArea*.append("Клиент подключен. Отправлено время: " + new Date() + "\n");  
 } else {  
 out.println("Registration failed");}  
  
 }else{  
 if (!*checkUser*(login, password)) {out.println("false");}else{out.println("true");  
 }  
 }  
 }  
 } catch (IOException | SQLException e) {  
 *textArea*.append(e.getMessage());  
 }  
 }  
 }  
 public static boolean checkUser(String log, String password) {  
 boolean result = false;  
 String Select = "select \* from users where login = ? and password = ?";  
 try (Connection con = DriverManager.*getConnection*(*url*, *user*, password);  
 PreparedStatement pst = con.prepareStatement(Select)) {  
 pst.setString(1, log);  
 pst.setString(2, password);  
 ResultSet rs = pst.executeQuery();  
 if(rs.next()) result = true;  
 }catch (SQLException ex) {  
 *textArea*.append(ex.getMessage());  
 }  
 return result;  
 }  
}