**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

Курсовая РАБОТА

**по дисциплине «Сети ЭВМ»**

Тема: Работа с сетевым ресурсом

|  |  |  |
| --- | --- | --- |
| Студент гр. 7302 |  | Карманов Д.А. |
| Преподаватель |  | Горячев А.В. |

Санкт-Петербург

2020

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Карманов Д.А. | | |
| Группа 7302 | | |
| Тема работы: работа с сетевым ресурсом | | |
| Исходные данные:  Сервис OpenWeatherMap и программа с ним взаимодействующая;  Язык программирования C# (.NET Framework 4.7.2) | | |
| Содержание пояснительной записки:  «Содержание», «Введение», «Задание», «Теоретическая часть», «Формализация задачи», «Спецификация программы», «Руководство пользователя», «Руководство программиста» «Контрольные примеры», «Листинг», «Вывод», «Литература» | | |
| Предполагаемый объем пояснительной записки:  Не менее 20 страниц(ы). | | |
| Дата выдачи задания: | | |
| Дата сдачи реферата: | | |
| Дата защиты реферата: | | |
| Студент |  | Карманов Д.А. |
| Преподаватель |  | Горячев А.В. |

**АННОТАЦИЯ**

В данной курсовой работе будет разработана программа, взаимодействующая с сетевым ресурсом, конкретно – с сервисом погоды OpenWeatherMap. Программа будет написана на языке C# с использованием WinForms.

**SUMMARY**

In this coursework a program will be developed that interacts with a network resource, specifically with the OpenWeatherMap weather service. The program will be written in C# using WinForms.

Оглавление

[**ВВЕДЕНИЕ** 5](#_Toc42357191)

[**Цель работы** 6](#_Toc42357192)

[**Задание** 6](#_Toc42357193)

[**Теоретическая часть** 6](#_Toc42357194)

[**Формализация задачи** 7](#_Toc42357195)

[**Спецификация программы** 7](#_Toc42357196)

[**Руководство пользователя** 9](#_Toc42357197)

[**Руководство программиста** 10](#_Toc42357198)

[**Контрольные примеры** 10](#_Toc42357199)

[**Листинг** 13](#_Toc42357200)

[**Вывод** 20](#_Toc42357201)

[**Cписок литературы** 21](#_Toc42357202)

# **ВВЕДЕНИЕ**

В XXI веке мир вовсю использует информационные технологии. Всё больше аспектов переходит в сферу сети: хранение данных, общение, организационное обеспечение, развлечения, такие как видео и игры. Теперь, вместо заполнения бумажных вариантов справок и журналов, коих может быть тысячи, можно заполнять их электронные версии. Вместо хранения данных на компьютере, где, во-первых, может не хватать места, а во-вторых, не всегда такой способ безопасен, можно хранить данные на отдельном ресурсе, например, GoogleDrive или Яндекс.Диск. В современные игры, отличающиеся своими заоблачными техническими требованиями, можно играть теперь и на динозаврах 00-х годов, нужен лишь хороший интернет, а нужный хостинг сам предоставит необходимую мощность.

Таким образом, облачный ресурс – это ресурс, предоставляющий свои вычислительные ресурсы по сетевому доступу.

# **Цель работы**

Целью курсовой работы является закрепление теоретических знаний и практических навыков работы с сетью Интернет посредством разработки программного обеспечения, использующего различные сетевые технологии в своей реализации.

# **Задание**

Заданием является разработка программного обеспечения, собирающего данные с сервиса погоды, который в свою очередь собирает данные с партнерских метеостанций со всего мира. Программа будет предлагать такие данные, как:

1. Текущая погода с температурой, влажностью, атмосферным давлением, названием города и датой, а также с данными о ветре, его скорости и направлении;
2. Прогноз по часам с интервалом в 3 часа с информацией об общем состоянии погоды в виде картинки и температуры;
3. Ежедневный прогноз с общей информацией о погодном состоянии и максимальной и минимальной температурами.

Также будет возможен выбор города из предложенного списка (выпадающий список) в случае, если программа не сможет определить местоположение вашего устройства самостоятельно.

# **Теоретическая часть**

Обращение к сервису происходит с помощью API запроса по протоколу HTTP, относящемуся к прикладному уровню модели OSI. На запрос сервис отсылает JSON файл по тому же протоколу со всей информацией.

Обращение может происходить как по координатам, так и по названию или ID города, но не у всех запросов. Так, чтобы получить только текущую погоду со всей информацией, в том числе названием и координатами города, необходимо сделать *Current Weather Data* запрос. Если же делать *One Call API* запрос (так он назван самими разработчиками сервиса), то будет получен другой JSON файл, который позволяет узнать не только текущую погоду, но и прогноз по часам и по дням, но в таком запросе нет информации о городе кроме его координат.

# **Формализация задачи**

Для решения поставленной задачи было решено использовать классы:

1. *OpenWeather*, представляющий класс для десериализованного JSON файла *Current Weather Data* запроса;
2. *OpenWeatherOneCallAPI*, представляющий класс для десериализованного JSON файла *One Call API* запроса;
3. *Weather*, предоставляющий методы для получения объектов *OpenWeatherOneCallAPI* по координатам, *OpenWeather* по координатами *OpenWeather* по названию города. Все методы используют обращение к сервису по протоколу HTTP*;*
4. *UnixConverter*, предоставляющий метод для конвертации времени в unix формате в удобный для программиста формат *DateTime*;
5. Множество других небольших классов для корректной десериализации JSON файлов.

# **Спецификация программы**

Описание методов классов *Weather*, *UnixConverter* и *Form1* представлены в таблицах 1.1.*a* – 1.*a*

Класс *UnixConverter*

Таблица 1.1.*a* – описание методов *UnixConverter*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Метод* | *Назначение* | *Возвращаемый тип* | *Модификатор доступа* | *Входные параметры* |
| ConvertFromUnixTimestamp | Конвертирует указанный unix формат времени в формат DateTime и если timezone указана, DateTime показывает текущее время этой временной зоны | DateTime | public | string timezone,  double timestamp |

Класс *Weather*

Таблица 1.2.*a* – описание методов *Weather*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Метод* | *Назначение* | *Возвращаемый тип* | *Модификатор доступа* | *Входные параметры* |
| GetWeather | По входящим координатам возвращает объект OpenWeatherOneCallAPI по протоколу HTTP | OpenWeatherOneCallAPI | public | GeoCoordinate coordinate |
| GetOpenWeatherByCoord | По входящим координатам возвращает объект OpenWeather по протоколу HTTP | OpenWeather | public | GeoCoordinate coordinate |
| GetOpenWeatherByCity | По названию города возвращает объект OpenWeather по протоколу HTTP | OpenWeather | public | string sity |

Класс *Form1*

Таблица 1.3.*a* – описание методов *Form1*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Метод* | *Назначение* | *Возвращаемый тип* | *Модификатор доступа* | *Входные параметры* |
| GeoDetect | Срабатывает единожды и только при запуске программы, пытается определить местоположение устройства в течение 10 секунд по GPS, либо триангуляции Wi-Fi | void | private | - |
| ProgressBar\_Change | Меняет состояние progressBar’а на value | void | private | int value |
| SetParameters | Устанавливает параметры формы данными о погоде из классов OpenWeather и OpenWeatherOneCallAPI, также обращается к сервису за картинками погода по протоколу HTTP | void | private | - |

# **Руководство пользователя**

Назначение программы

Программа собирает данные о погоде с веб-сервиса OpenWeatherMap, который в свою очередь собирает данные с партнерских метеостанций. Затем программа выводит эти данные на интерфейс пользователя.

Условие выполнения программы

Для запуска программы необходим .NET Framework версии не ниже 4.7.2, возможно VPN, и желательно разрешение программе на использование GPS.

Выполнение программы

После запуска программы идет 10-секундный отсчет на определение геокоординат устройства, на котором запущена программа. На интерфейсе можно увидеть статус и прогресс определения. Если прогресс не меняется, то это либо проблема связи с сервисом, либо данные получены и записываются в форму. В обоих случаях необходимо просто подождать, но в первом случае можно ускорить получение данных, используя VPN при работе программы. Если геоданные не определены автоматически, то становится доступен выпадающий список, в котором можно выбрать интересующий вас город.

Для работы приложения ОБЯЗАТЕЛЬНО требуется наличие интернет соединения!

# **Руководство программиста**

Назначение и условие выполнения программы

См. пункт «Руководство пользователя»

Характеристики программы

Программа разработана в среде Microsoft Visual Studio 2019, использовалась версия платформы .NET 4.7.2 Язык разработки – C#. Программа старается отвечать понятию сетевых технологий (а именно использованию протокола HTTP), и имеет простой интерфейс, реализованный с помощью платформы WindowsForms.

Обращение к программе

См. «Руководство пользователя»

Входные и выходные данные

Входными данными выступают либо геокоординаты устройства, определяемые автоматически при первом запуске программы, либо же название города, выбираемое самостоятельно в выпадающем списке. Выходными параметрами выступают данные о погоде в определенном/выбранном регионе.

# **Контрольные примеры**

Приведенные примеры описывают все случаи пути выполнения программы. Случаи:

1. При запуске программа смогла определить местоположение устройства;
2. При запуске программа не смогла определить местоположение устройства. Это возможно в случаях, когда GPS устройства не разрешает приложению определять местоположение;
3. При выборе города самостоятельно программа смогла получить данные от сервиса;
4. При выборе города самостоятельно программа не смогла получить данные от сервиса. Это возможно в случаях отсутствия интернет соединения или неполадок со стороны сервиса. Также, возможно, необходим VPN.

На рисунках 1-4 показаны случаи 1-4 соответственно.

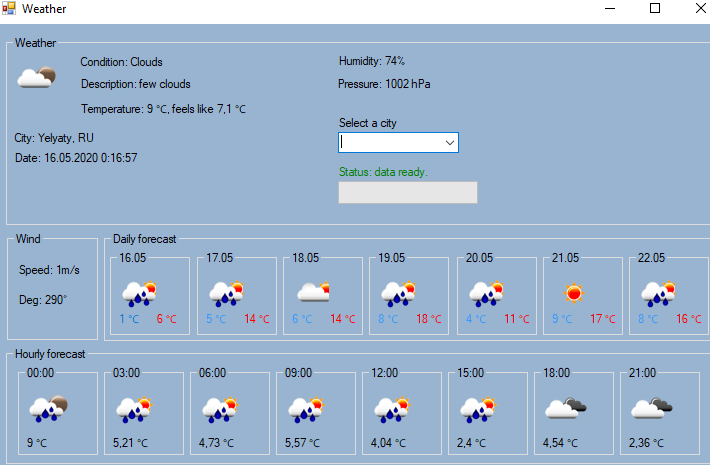


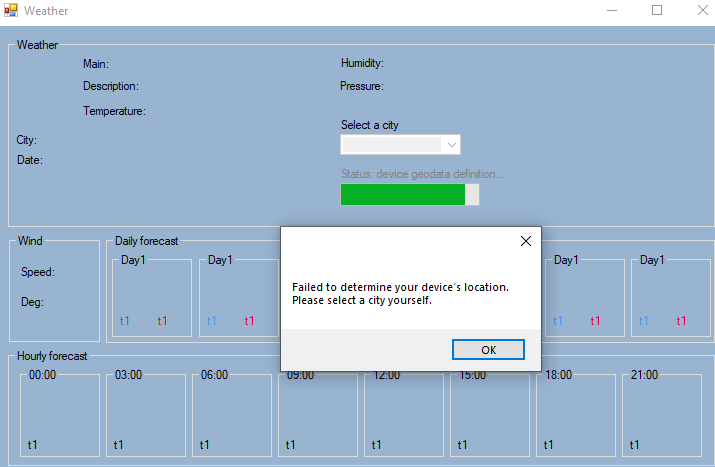
Рисунок 1. Программа успешно определила местоположение устройства

Рисунок 2. Программа не смогла определить местоположение устройства

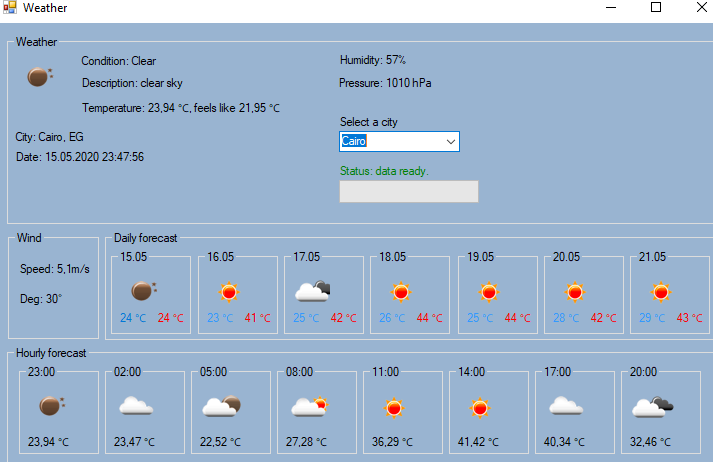


Рисунок 3. Программа успешно получила данные при самостоятельном выборе города. Ну и как там в Египте?

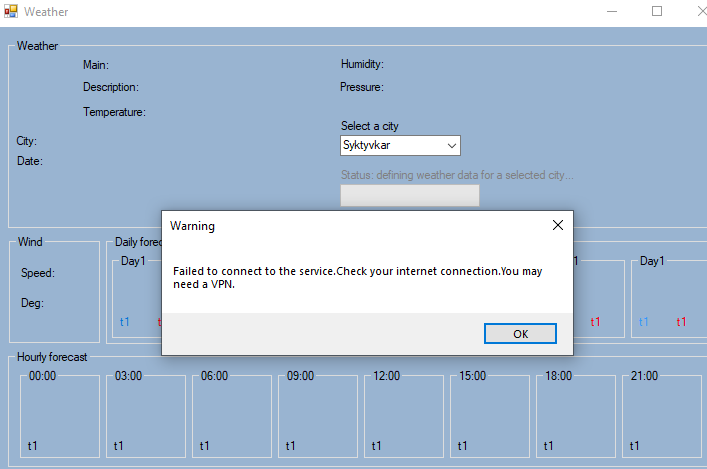


Рисунок 4. Программе не удалось получить данные от сервиса

# **Листинг**

*Form1.cs*

using System;

using System.Drawing;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Device.Location;

using System.Threading;

namespace WeatherApp

{

public partial class Form1 : Form

{

Weather weather;

OpenWeatherOneCallAPI.OpenWeatherOneCallAPI openWeatherOneAPI;

GeoCoordinateWatcher watcher;

GeoCoordinate coordinate;

bool geoDetected = false;

public Form1()

{

InitializeComponent();

comboBox1.Enabled = false;

weather = new Weather();

watcher = new GeoCoordinateWatcher();

watcher.Start();

Task.Run(() => GeoDetect());

}

private void GeoDetect()

{

if (coordinate == null)

{

label16.Text = "Status: device geodata definition...";

label16.ForeColor = Color.Gray;

}

int seconds = 10;

int step = progressBar1.Maximum / seconds;

for (int i = 0; i < seconds; i++)

{

if (geoDetected == false)

{

if (watcher.Status == GeoPositionStatus.Ready && !watcher.Position.Location.IsUnknown)

{

coordinate = watcher.Position.Location;

openWeatherOneAPI = weather.GetWeather(coordinate);

if (InvokeRequired)

Invoke(new Action(() => SetParameters()));

else SetParameters();

geoDetected = true;

break;

}

}

if (InvokeRequired)

Invoke(new Action(() => ProgressBar\_Change(i \* step)));

else ProgressBar\_Change(i \* step);

Thread.Sleep(1000);

}

if (geoDetected == false)

{

MessageBox.Show("Failed to determine your device’s location.\n" +

"Please select a city yourself.");

if (label16.InvokeRequired)

label16.Invoke(new Action(() =>

{

label16.Text = "Status: device geodata not defined.";

label16.ForeColor = Color.Red;

}));

else

{

label16.Text = "Status: device geodata not defined.";

label16.ForeColor = Color.Red;

}

}

if (comboBox1.InvokeRequired)

comboBox1.Invoke(new Action(() => comboBox1.Enabled = true));

else

comboBox1.Enabled = true;

if (progressBar1.InvokeRequired)

progressBar1.Invoke(new Action(() => progressBar1.Value = 0));

else

progressBar1.Value = 0;

}

private void ProgressBar\_Change(int value)

{

progressBar1.Value = value;

}

private void SetParameters()

{

if (openWeatherOneAPI != null)

{

// Get OpenWeather to get info about the city (OpenWeatherOneCallAPI doesn't have it).

GeoCoordinate coordinate = new GeoCoordinate(openWeatherOneAPI.Lat, openWeatherOneAPI.Lon);

OpenWeather.OpenWeather openWeather = weather.GetOpenWeatherByCoord(coordinate);

// Set current forecast.

MainLabel.Text = "Condition: " + openWeatherOneAPI.Current.Weather[0].Main;

DescriptionLabel.Text = "Description: " + openWeatherOneAPI.Current.Weather[0].Description;

TemperatureLabel.Text = "Temperature: " + openWeatherOneAPI.Current.Temp + " ℃, feels like " + openWeatherOneAPI.Current.Feels\_Like + " ℃";

CityLabel.Text = "City: " + openWeather.Name + ", " + openWeather.Sys.Country;

WeatherPicture.Load($"http://openweathermap.org/img/w/{openWeatherOneAPI.Current.Weather[0].Icon}.png");

DateLabel.Text = "Date: " + UnixConverter.ConvertFromUnixTimestamp(openWeatherOneAPI.Current.Dt, openWeatherOneAPI.Timezone);

HumidityLabel.Text = "Humidity: " + Convert.ToString(openWeatherOneAPI.Current.Humidity) + "%";

PressureLabel.Text = "Pressure: " + Convert.ToString(openWeatherOneAPI.Current.Pressure) + " hPa";

SpeedLabel.Text = "Speed: " + Convert.ToString(openWeatherOneAPI.Current.Wind\_Speed) + "m/s";

DegLabel.Text = "Deg: " + Convert.ToString(openWeatherOneAPI.Current.Wind\_Deg) + "°";

// Set hourly forecast.

pictureBox1.Load($"http://openweathermap.org/img/w/{openWeatherOneAPI.Hourly[0].Weather[0].Icon}.png");

HourlyTempLabel1.Text = Convert.ToString(openWeatherOneAPI.Hourly[0].Temp) + " ℃";

groupBox5.Text = UnixConverter.ConvertFromUnixTimestamp(openWeatherOneAPI.Hourly[0].Dt, openWeatherOneAPI.Timezone).ToString("HH:mm");

pictureBox2.Load($"http://openweathermap.org/img/w/{openWeatherOneAPI.Hourly[3].Weather[0].Icon}.png");

HourlyTempLabel2.Text = Convert.ToString(openWeatherOneAPI.Hourly[3].Temp) + " ℃";

groupBox6.Text = UnixConverter.ConvertFromUnixTimestamp(openWeatherOneAPI.Hourly[3].Dt, openWeatherOneAPI.Timezone).ToString("HH:mm");

pictureBox3.Load($"http://openweathermap.org/img/w/{openWeatherOneAPI.Hourly[6].Weather[0].Icon}.png");

HourlyTempLabel3.Text = Convert.ToString(openWeatherOneAPI.Hourly[6].Temp) + " ℃";

groupBox7.Text = UnixConverter.ConvertFromUnixTimestamp(openWeatherOneAPI.Hourly[6].Dt, openWeatherOneAPI.Timezone).ToString("HH:mm");

pictureBox4.Load($"http://openweathermap.org/img/w/{openWeatherOneAPI.Hourly[9].Weather[0].Icon}.png");

HourlyTempLabel4.Text = Convert.ToString(openWeatherOneAPI.Hourly[9].Temp) + " ℃";

groupBox8.Text = UnixConverter.ConvertFromUnixTimestamp(openWeatherOneAPI.Hourly[9].Dt, openWeatherOneAPI.Timezone).ToString("HH:mm");

pictureBox5.Load($"http://openweathermap.org/img/w/{openWeatherOneAPI.Hourly[12].Weather[0].Icon}.png");

HourlyTempLabel5.Text = Convert.ToString(openWeatherOneAPI.Hourly[12].Temp) + " ℃";

groupBox9.Text = UnixConverter.ConvertFromUnixTimestamp(openWeatherOneAPI.Hourly[12].Dt, openWeatherOneAPI.Timezone).ToString("HH:mm");

pictureBox6.Load($"http://openweathermap.org/img/w/{openWeatherOneAPI.Hourly[15].Weather[0].Icon}.png");

HourlyTempLabel6.Text = Convert.ToString(openWeatherOneAPI.Hourly[15].Temp) + " ℃";

groupBox10.Text = UnixConverter.ConvertFromUnixTimestamp(openWeatherOneAPI.Hourly[15].Dt, openWeatherOneAPI.Timezone).ToString("HH:mm");

pictureBox7.Load($"http://openweathermap.org/img/w/{openWeatherOneAPI.Hourly[18].Weather[0].Icon}.png");

HourlyTempLabel7.Text = Convert.ToString(openWeatherOneAPI.Hourly[18].Temp) + " ℃";

groupBox11.Text = UnixConverter.ConvertFromUnixTimestamp(openWeatherOneAPI.Hourly[18].Dt, openWeatherOneAPI.Timezone).ToString("HH:mm");

pictureBox8.Load($"http://openweathermap.org/img/w/{openWeatherOneAPI.Hourly[21].Weather[0].Icon}.png");

HourlyTempLabel8.Text = Convert.ToString(openWeatherOneAPI.Hourly[21].Temp) + " ℃";

groupBox12.Text = UnixConverter.ConvertFromUnixTimestamp(openWeatherOneAPI.Hourly[21].Dt, openWeatherOneAPI.Timezone).ToString("HH:mm");

// Set daily forecast.

pictureBox9.Load($"http://openweathermap.org/img/w/{openWeatherOneAPI.Daily[0].Weather[0].Icon}.png");

groupBox13.Text = UnixConverter.ConvertFromUnixTimestamp(openWeatherOneAPI.Daily[0].Dt, openWeatherOneAPI.Timezone).ToString("dd.MM");

label2.Text = Convert.ToString(Convert.ToInt32(openWeatherOneAPI.Daily[0].Temp.Min)) + " ℃";

label3.Text = Convert.ToString(Convert.ToInt32(openWeatherOneAPI.Daily[0].Temp.Max)) + " ℃";

pictureBox10.Load($"http://openweathermap.org/img/w/{openWeatherOneAPI.Daily[1].Weather[0].Icon}.png");

groupBox14.Text = UnixConverter.ConvertFromUnixTimestamp(openWeatherOneAPI.Daily[1].Dt, openWeatherOneAPI.Timezone).ToString("dd.MM");

label4.Text = Convert.ToString(Convert.ToInt32(openWeatherOneAPI.Daily[1].Temp.Min)) + " ℃";

label5.Text = Convert.ToString(Convert.ToInt32(openWeatherOneAPI.Daily[1].Temp.Max)) + " ℃";

pictureBox11.Load($"http://openweathermap.org/img/w/{openWeatherOneAPI.Daily[2].Weather[0].Icon}.png");

groupBox15.Text = UnixConverter.ConvertFromUnixTimestamp(openWeatherOneAPI.Daily[2].Dt, openWeatherOneAPI.Timezone).ToString("dd.MM");

label6.Text = Convert.ToString(Convert.ToInt32(openWeatherOneAPI.Daily[2].Temp.Min)) + " ℃";

label7.Text = Convert.ToString(Convert.ToInt32(openWeatherOneAPI.Daily[2].Temp.Max)) + " ℃";

pictureBox12.Load($"http://openweathermap.org/img/w/{openWeatherOneAPI.Daily[3].Weather[0].Icon}.png");

groupBox16.Text = UnixConverter.ConvertFromUnixTimestamp(openWeatherOneAPI.Daily[3].Dt, openWeatherOneAPI.Timezone).ToString("dd.MM");

label8.Text = Convert.ToString(Convert.ToInt32(openWeatherOneAPI.Daily[3].Temp.Min)) + " ℃";

label9.Text = Convert.ToString(Convert.ToInt32(openWeatherOneAPI.Daily[3].Temp.Max)) + " ℃";

pictureBox13.Load($"http://openweathermap.org/img/w/{openWeatherOneAPI.Daily[4].Weather[0].Icon}.png");

groupBox17.Text = UnixConverter.ConvertFromUnixTimestamp(openWeatherOneAPI.Daily[4].Dt, openWeatherOneAPI.Timezone).ToString("dd.MM"); ;

label10.Text = Convert.ToString(Convert.ToInt32(openWeatherOneAPI.Daily[4].Temp.Min)) + " ℃";

label11.Text = Convert.ToString(Convert.ToInt32(openWeatherOneAPI.Daily[4].Temp.Max)) + " ℃";

pictureBox14.Load($"http://openweathermap.org/img/w/{openWeatherOneAPI.Daily[5].Weather[0].Icon}.png");

groupBox18.Text = UnixConverter.ConvertFromUnixTimestamp(openWeatherOneAPI.Daily[5].Dt, openWeatherOneAPI.Timezone).ToString("dd.MM");

label12.Text = Convert.ToString(Convert.ToInt32(openWeatherOneAPI.Daily[5].Temp.Min)) + " ℃";

label13.Text = Convert.ToString(Convert.ToInt32(openWeatherOneAPI.Daily[5].Temp.Max)) + " ℃";

pictureBox15.Load($"http://openweathermap.org/img/w/{openWeatherOneAPI.Daily[6].Weather[0].Icon}.png");

groupBox19.Text = UnixConverter.ConvertFromUnixTimestamp(openWeatherOneAPI.Daily[6].Dt, openWeatherOneAPI.Timezone).ToString("dd.MM");

label14.Text = Convert.ToString(Convert.ToInt32(openWeatherOneAPI.Daily[6].Temp.Min)) + " ℃";

label15.Text = Convert.ToString(Convert.ToInt32(openWeatherOneAPI.Daily[6].Temp.Max)) + " ℃";

label16.Text = "Status: data ready.";

label16.ForeColor = Color.Green;

comboBox1.Enabled = true;

progressBar1.Value = 0;

}

}

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

label16.Text = "Status: defining weather data for a selected city...";

label16.ForeColor = Color.Gray;

string city = Convert.ToString(comboBox1.SelectedItem);

Task.Run(() =>

{

OpenWeather.OpenWeather openWeather = weather.GetOpenWeatherByCity(city);

if (openWeather != null)

{

coordinate = new GeoCoordinate(openWeather.Coord.Lat, openWeather.Coord.Lon);

openWeatherOneAPI = weather.GetWeather(coordinate);

if (InvokeRequired)

Invoke(new Action(() => SetParameters()));

else SetParameters();

}

else

{

label16.Text = "Status: city geodata not defined.";

label16.ForeColor = Color.Red;

}

});

}

}

}

*Weather.cs*

using Newtonsoft.Json;

using System;

using System.Device.Location;

using System.IO;

using System.Net;

using System.Windows.Forms;

namespace WeatherApp

{

public class Weather

{

public OpenWeatherOneCallAPI.OpenWeatherOneCallAPI GetWeather(GeoCoordinate coordinate)

{

string appid = "e67f327620e3cc056d936f8d1fbf20fd";

string url;

if (coordinate != null)

url = $"https://api.openweathermap.org/data/2.5/onecall?units=metric&lat={coordinate.Latitude}&lon={coordinate.Longitude}&appid={appid}";

else

throw new Exception();

try

{

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);

HttpWebResponse response = (HttpWebResponse)request.GetResponse();

int l = 0;

using (StreamReader streamReader = new StreamReader(response.GetResponseStream()))

{

string line;

if ((line = streamReader.ReadToEnd()) != null)

{

OpenWeatherOneCallAPI.OpenWeatherOneCallAPI weather =

JsonConvert.DeserializeObject<OpenWeatherOneCallAPI.OpenWeatherOneCallAPI>(line);

return weather;

}

}

}

catch

{

MessageBox.Show("Failed to connect to the service. Check your internet connection. You may need a VPN.");

}

return null;

}

public OpenWeather.OpenWeather GetOpenWeatherByCoord(GeoCoordinate coord)

{

try

{

string appid = "e67f327620e3cc056d936f8d1fbf20fd";

string url = $"http://api.openweathermap.org/data/2.5/weather?units=metric&lat={coord.Latitude}&lon={coord.Longitude}&appid={appid}";

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);

HttpWebResponse response = (HttpWebResponse)request.GetResponse();

using (StreamReader streamReader = new StreamReader(response.GetResponseStream()))

{

string line;

if ((line = streamReader.ReadToEnd()) != null)

{

OpenWeather.OpenWeather weather =

JsonConvert.DeserializeObject<OpenWeather.OpenWeather>(line);

return weather;

}

}

}

catch

{

MessageBox.Show("Failed to connect to the service.Check your internet connection.You may need a VPN.", "Warning");

}

return null;

}

public OpenWeather.OpenWeather GetOpenWeatherByCity(string city)

{

try

{

string appid = "e67f327620e3cc056d936f8d1fbf20fd";

string url = $"http://api.openweathermap.org/data/2.5/weather?units=metric&q={city}&appid={appid}";

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);

HttpWebResponse response = (HttpWebResponse)request.GetResponse();

using (StreamReader streamReader = new StreamReader(response.GetResponseStream()))

{

string line;

if ((line = streamReader.ReadToEnd()) != null)

{

OpenWeather.OpenWeather weather =

JsonConvert.DeserializeObject<OpenWeather.OpenWeather>(line);

return weather;

}

}

}

catch

{

MessageBox.Show("Failed to connect to the service.Check your internet connection.You may need a VPN.", "Warning");

}

return null;

}

}

}

*UnixConverter.cs*

using System;

namespace WeatherApp

{

static class UnixConverter

{

static public DateTime ConvertFromUnixTimestamp(double timestamp, string timezone)

{

DateTime origin = new DateTime(1970, 1, 1, 0, 0, 0, 0);

if (timezone == "Europe/Moscow")

return origin.AddSeconds(10800 + timestamp);

else if (timezone == "Europe/Rome")

// If not summer, then +3600.

return origin.AddSeconds(7200 + timestamp);

else if (timezone == "Africa/Cairo")

return origin.AddSeconds(7200 + timestamp);

else if (timezone == "Pacific/Auckland")

return origin.AddSeconds(43200 + timestamp);

else if (timezone == "America/Los\_Angeles")

return origin.AddSeconds(timestamp - 25200);

else if (timezone == "America/Sao\_Paulo")

return origin.AddSeconds(timestamp - 10800);

else

return origin.AddSeconds(timestamp);

}

}

}

# **Вывод**

В ходе курсовой работы было разработано приложение, предоставляющее сведение о погоде. Для этого были изучены понятия «API» и «облачный ресурс». Также был разработан ряд мер для более отзывчивой работы приложения, такие как внедрение многопоточности.

# **Cписок литературы**

1. REST API: <https://lectureswww.readthedocs.io/8.www.gui/csharp.html>
2. Облачные вычисления: <https://www.moysklad.ru/poleznoe/statyi/chto-takoe-oblachnye-servisy/>