1. **Назначение и условия применения программы**

Программа предназначена для просмотра данных о погоде пользователем.

Каждый пользователь проходит обязательную регистрацию, при которой данные его учетной записи сохраняются в базу данных.

Приложение позволяет узнать минимальную температуру за период, максимальную температуру за период, среднюю температуру за период, текущую погоду, посмотреть график изменения погоды.

|  |  |
| --- | --- |
| Процессор | Intel Celeron J4025, 2 ГГц (2.9 ГГц, в режиме Turbo); |
| Оперативная Память | 2 ГБ, DDR4, DIMM, 2133 МГц; |
| Разрешение экрана | 1280x1080 |
| Размер экрана | От 15 дюймов |
| Устройства ввода | Мышь, клавиатура |
| Дисковое пространство | 0.5Гб |
| Операционная система | Windows 8.1 10 |

Требования к программным средствам:

Для корректной работы программы необходимо:

ОС Windows 10 – операционная система, сделанная корпорацией Microsoft;

Visual Studio 2022 – лучшая интегрированная среда разработки для создания многофункциональных, привлекательных кроссплатформенных приложений для Windows;

PostgreSQL – это реляционная база данных с открытым кодом, которая поддерживается в течение 30 лет разработки и является одной из наиболее известных среди всех существующих реляционных баз данных.

1. **Характеристика программы**

Программа должна обеспечивать следующие функции:

* Отображение списка городов
* Возможность выбора города для отображения статистики
* Возможно добавлять города
* Возможно удалять города
* Вывод минимальной температуры за весь период в выбранном городе
* Вывод максимальной температуры за весь период в выбранном городе
* Вывод средней температуры за весь период в выбранном городе
* Вывод текущей температуры за весь период в выбранном городе
* Вывод графика температуры за выбранный период в выбранном городе

1. **Обращение к программе**

С# — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота как язык разработки приложений для платформы Microsoft .NET Framework. Впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270

Код главной страницы:

using System.Windows;

using System.Windows.Input;

namespace Weather

{

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

MainFrame.Navigate(new Login());

manager.MainFrame = MainFrame;

}

private void Input\_Click(object sender, RoutedEventArgs e)

{

MainFrame.Navigate(new Login());

manager.MainFrame = MainFrame;

}

private void MainFrame\_ContentRendered(object sender, System.EventArgs e)

{

}

}

}

Код функционала веб-сервера:

public class Utilities

{

bool chkDBExists(string connectionStr, string dbname)

{

using (NpgsqlConnection conn = new NpgsqlConnection(connectionStr))

{

using (NpgsqlCommand command = new NpgsqlCommand

($"SELECT DATNAME FROM pg\_catalog.pg\_database WHERE DATNAME = '{dbname}'", conn))

{

try

{

conn.Open();

var i = command.ExecuteScalar();

conn.Close();

if (i.ToString().Equals(dbname)) //always 'true' (if it exists) or 'null' (if it doesn't)

return true;

else return false;

}

catch (Exception e) { return false; }

}

}

}

public void create\_db()

{

const string connStr = "Server=localhost;Port=5432;User Id=postgres;Password=1234;";

const string connStr2 = "Server=localhost;Port=5432;User Id=postgres;Password=1234;Database=weather;";

var m\_conn = new NpgsqlConnection(connStr); // db connction

var m\_conn2 = new NpgsqlConnection(connStr2); // table connection

// creating a database in Postgresql

var m\_createdb\_cmd = new NpgsqlCommand("CREATE DATABASE weather;", m\_conn);

string script = "--\r\n-- PostgreSQL database dump\r\n--\r\n\r\n-- Dumped from database version 15.1\r\n-- Dumped by pg\_dump version 15.1\r\n\r\nSET statement\_timeout = 0;\r\nSET lock\_timeout = 0;\r\nSET idle\_in\_transaction\_session\_timeout = 0;\r\nSET client\_encoding = 'UTF8';\r\nSET standard\_conforming\_strings = on;\r\nSELECT pg\_catalog.set\_config('search\_path', '', false);\r\nSET check\_function\_bodies = false;\r\nSET xmloption = content;\r\nSET client\_min\_messages = warning;\r\nSET row\_security = off;\r\n\r\nSET default\_tablespace = '';\r\n\r\nSET default\_table\_access\_method = heap;\r\n\r\n--\r\n-- Name: cities; Type: TABLE; Schema: public; Owner: postgres\r\n--\r\n\r\nCREATE TABLE public.cities (\r\n id bigint NOT NULL,\r\n city character varying(89) NOT NULL,\r\n longitude double precision,\r\n latitude double precision\r\n);\r\n\r\n\r\nALTER TABLE public.cities OWNER TO postgres;\r\n\r\n--\r\n-- Name: cities\_id\_seq; Type: SEQUENCE; Schema: public; Owner: postgres\r\n--\r\n\r\nCREATE SEQUENCE public.cities\_id\_seq\r\n START WITH 1\r\n INCREMENT BY 1\r\n NO MINVALUE\r\n NO MAXVALUE\r\n CACHE 1;\r\n\r\n\r\nALTER TABLE public.cities\_id\_seq OWNER TO postgres;\r\n\r\n--\r\n-- Name: cities\_id\_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: postgres\r\n--\r\n\r\nALTER SEQUENCE public.cities\_id\_seq OWNED BY public.cities.id;\r\n\r\n\r\n--\r\n-- Name: city\_info; Type: TABLE; Schema: public; Owner: postgres\r\n--\r\n\r\nCREATE TABLE public.city\_info (\r\n id bigint NOT NULL,\r\n max\_weather double precision[] NOT NULL,\r\n min\_weather double precision[] NOT NULL,\r\n period\_weather double precision[] NOT NULL,\r\n current\_weather double precision NOT NULL,\r\n city\_id bigint NOT NULL\r\n);\r\n\r\n\r\nALTER TABLE public.city\_info OWNER TO postgres;\r\n\r\n--\r\n-- Name: city\_info\_id\_seq; Type: SEQUENCE; Schema: public; Owner: postgres\r\n--\r\n\r\nCREATE SEQUENCE public.city\_info\_id\_seq\r\n START WITH 1\r\n INCREMENT BY 1\r\n NO MINVALUE\r\n NO MAXVALUE\r\n CACHE 1;\r\n\r\n\r\nALTER TABLE public.city\_info\_id\_seq OWNER TO postgres;\r\n\r\n--\r\n-- Name: city\_info\_id\_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: postgres\r\n--\r\n\r\nALTER SEQUENCE public.city\_info\_id\_seq OWNED BY public.city\_info.id;\r\n\r\n\r\n--\r\n-- Name: user\_info; Type: TABLE; Schema: public; Owner: postgres\r\n--\r\n\r\nCREATE TABLE public.user\_info (\r\n id bigint NOT NULL,\r\n user\_id bigint NOT NULL,\r\n city\_id bigint NOT NULL\r\n);\r\n\r\n\r\nALTER TABLE public.user\_info OWNER TO postgres;\r\n\r\n--\r\n-- Name: user\_info\_id\_seq; Type: SEQUENCE; Schema: public; Owner: postgres\r\n--\r\n\r\nCREATE SEQUENCE public.user\_info\_id\_seq\r\n START WITH 1\r\n INCREMENT BY 1\r\n NO MINVALUE\r\n NO MAXVALUE\r\n CACHE 1;\r\n\r\n\r\nALTER TABLE public.user\_info\_id\_seq OWNER TO postgres;\r\n\r\n--\r\n-- Name: user\_info\_id\_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: postgres\r\n--\r\n\r\nALTER SEQUENCE public.user\_info\_id\_seq OWNED BY public.user\_info.id;\r\n\r\n\r\n--\r\n-- Name: users; Type: TABLE; Schema: public; Owner: postgres\r\n--\r\n\r\nCREATE TABLE public.users (\r\n id bigint NOT NULL,\r\n username character varying(50) NOT NULL,\r\n password character varying(50) NOT NULL\r\n);\r\n\r\n\r\nALTER TABLE public.users OWNER TO postgres;\r\n\r\n--\r\n-- Name: users\_id\_seq; Type: SEQUENCE; Schema: public; Owner: postgres\r\n--\r\n\r\nCREATE SEQUENCE public.users\_id\_seq\r\n START WITH 1\r\n INCREMENT BY 1\r\n NO MINVALUE\r\n NO MAXVALUE\r\n CACHE 1;\r\n\r\n\r\nALTER TABLE public.users\_id\_seq OWNER TO postgres;\r\n\r\n--\r\n-- Name: users\_id\_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: postgres\r\n--\r\n\r\nALTER SEQUENCE public.users\_id\_seq OWNED BY public.users.id;\r\n\r\n\r\n--\r\n-- Name: cities id; Type: DEFAULT; Schema: public; Owner: postgres\r\n--\r\n\r\nALTER TABLE ONLY public.cities ALTER COLUMN id SET DEFAULT nextval('public.cities\_id\_seq'::regclass);\r\n\r\n\r\n--\r\n-- Name: city\_info id; Type: DEFAULT; Schema: public; Owner: postgres\r\n--\r\n\r\nALTER TABLE ONLY public.city\_info ALTER COLUMN id SET DEFAULT nextval('public.city\_info\_id\_seq'::regclass);\r\n\r\n\r\n--\r\n-- Name: user\_info id; Type: DEFAULT; Schema: public; Owner: postgres\r\n--\r\n\r\nALTER TABLE ONLY public.user\_info ALTER COLUMN id SET DEFAULT nextval('public.user\_info\_id\_seq'::regclass);\r\n\r\n\r\n--\r\n-- Name: users id; Type: DEFAULT; Schema: public; Owner: postgres\r\n--\r\n\r\nALTER TABLE ONLY public.users ALTER COLUMN id SET DEFAULT nextval('public.users\_id\_seq'::regclass);\r\n\r\n\r\n--\r\n-- Data for Name: cities; Type: TABLE DATA; Schema: public; Owner: postgres\r\n--\r\n\r\n\r\n\r\n--\r\n-- Data for Name: city\_info; Type: TABLE DATA; Schema: public; Owner: postgres\r\n--\r\n\r\n\r\n\r\n--\r\n-- Data for Name: user\_info; Type: TABLE DATA; Schema: public; Owner: postgres\r\n--\r\n\r\n\r\n\r\n--\r\n-- Data for Name: users; Type: TABLE DATA; Schema: public; Owner: postgres\r\n--\r\n\r\n\r\n\r\n--\r\n-- Name: cities\_id\_seq; Type: SEQUENCE SET; Schema: public; Owner: postgres\r\n--\r\n\r\nSELECT pg\_catalog.setval('public.cities\_id\_seq', 14, true);\r\n\r\n\r\n--\r\n-- Name: city\_info\_id\_seq; Type: SEQUENCE SET; Schema: public; Owner: postgres\r\n--\r\n\r\nSELECT pg\_catalog.setval('public.city\_info\_id\_seq', 7, true);\r\n\r\n\r\n--\r\n-- Name: user\_info\_id\_seq; Type: SEQUENCE SET; Schema: public; Owner: postgres\r\n--\r\n\r\nSELECT pg\_catalog.setval('public.user\_info\_id\_seq', 19, true);\r\n\r\n\r\n--\r\n-- Name: users\_id\_seq; Type: SEQUENCE SET; Schema: public; Owner: postgres\r\n--\r\n\r\nSELECT pg\_catalog.setval('public.users\_id\_seq', 8, true);\r\n\r\n\r\n--\r\n-- Name: cities cities\_pkey; Type: CONSTRAINT; Schema: public; Owner: postgres\r\n--\r\n\r\nALTER TABLE ONLY public.cities\r\n ADD CONSTRAINT cities\_pkey PRIMARY KEY (id);\r\n\r\n\r\n--\r\n-- Name: city\_info city\_info\_pkey; Type: CONSTRAINT; Schema: public; Owner: postgres\r\n--\r\n\r\nALTER TABLE ONLY public.city\_info\r\n ADD CONSTRAINT city\_info\_pkey PRIMARY KEY (id);\r\n\r\n\r\n--\r\n-- Name: user\_info user\_info\_pkey; Type: CONSTRAINT; Schema: public; Owner: postgres\r\n--\r\n\r\nALTER TABLE ONLY public.user\_info\r\n ADD CONSTRAINT user\_info\_pkey PRIMARY KEY (id);\r\n\r\n\r\n--\r\n-- Name: users users\_pkey; Type: CONSTRAINT; Schema: public; Owner: postgres\r\n--\r\n\r\nALTER TABLE ONLY public.users\r\n ADD CONSTRAINT users\_pkey PRIMARY KEY (id);\r\n\r\n\r\n--\r\n-- Name: users users\_username\_key; Type: CONSTRAINT; Schema: public; Owner: postgres\r\n--\r\n\r\nALTER TABLE ONLY public.users\r\n ADD CONSTRAINT users\_username\_key UNIQUE (username);\r\n\r\n\r\n--\r\n-- Name: city\_info city\_info\_city\_id\_fkey; Type: FK CONSTRAINT; Schema: public; Owner: postgres\r\n--\r\n\r\nALTER TABLE ONLY public.city\_info\r\n ADD CONSTRAINT city\_info\_city\_id\_fkey FOREIGN KEY (city\_id) REFERENCES public.cities(id);\r\n\r\n\r\n--\r\n-- Name: user\_info user\_info\_city\_id\_fkey; Type: FK CONSTRAINT; Schema: public; Owner: postgres\r\n--\r\n\r\nALTER TABLE ONLY public.user\_info\r\n ADD CONSTRAINT user\_info\_city\_id\_fkey FOREIGN KEY (city\_id) REFERENCES public.cities(id);\r\n\r\n\r\n--\r\n-- Name: user\_info user\_info\_user\_id\_fkey; Type: FK CONSTRAINT; Schema: public; Owner: postgres\r\n--\r\n\r\nALTER TABLE ONLY public.user\_info\r\n ADD CONSTRAINT user\_info\_user\_id\_fkey FOREIGN KEY (user\_id) REFERENCES public.users(id);\r\n\r\n\r\n--\r\n-- PostgreSQL database dump complete\r\n--\r\n\r\n";

var m\_createtbl\_cmd = new NpgsqlCommand(script);

m\_createtbl\_cmd.Connection = m\_conn2;

// 3.. Make connection and create

// open connection to create DB

m\_conn.Open();

m\_createdb\_cmd.ExecuteNonQuery();

m\_conn.Close();

// open connection to create table

m\_conn2.Open();

m\_createtbl\_cmd.ExecuteNonQuery();

m\_conn2.Close();

}

static async Task<string> Main()

{

HttpClient httpClient = new HttpClient();

var dt = DateTime.Now;

DateTime month = dt.AddMonths(-1);

// получаем ответ

using HttpResponseMessage response = await httpClient.GetAsync("https://api.open-meteo.com/v1/forecast?latitude=55.71&longitude=38.95&hourly=temperature\_2m,weathercode&daily=temperature\_2m\_max,temperature\_2m\_min&current\_weather=true&timezone=auto&start\_date=" + month.ToString("yyyy-MM-dd") + "&end\_date=" + DateTime.Now.ToString("yyyy-MM-dd"));

// получаем ответ

return await response.Content.ReadAsStringAsync();

}

static async Task<string> get\_data(string latitude, string longtitude)

{

HttpClient httpClient = new HttpClient();

var dt = DateTime.Now;

DateTime month = dt.AddMonths(-1);

// получаем ответ

using HttpResponseMessage response = await httpClient.GetAsync("https://api.open-meteo.com/v1/forecast?latitude=" + latitude + "&longitude=" + longtitude + "&hourly=temperature\_2m,weathercode&daily=temperature\_2m\_max,temperature\_2m\_min&current\_weather=true&timezone=auto&start\_date=" + month.ToString("yyyy-MM-dd") + "&end\_date=" + DateTime.Now.ToString("yyyy-MM-dd"));

// получаем ответ

return await response.Content.ReadAsStringAsync();

}

static async Task<string> GetGEO(string city)

{

HttpClient httpClient = new HttpClient();

// получаем ответ

using HttpResponseMessage response = await httpClient.GetAsync("https://geocoding-api.open-meteo.com/v1/search?name=" + city + "&language=ru");

// получаем ответ

return await response.Content.ReadAsStringAsync();

}

public async Task<string> Mine()

{

const string connStr = "Server=localhost;Port=5432;User Id=postgres;Password=1234;";

if (chkDBExists(connStr, "weather") != true)

{

create\_db();

}

return await Main();

}

public double get\_current(string city\_id)

{

var res = WeatherContext.GetContext().CityInfos.Where(x => x.CityId == int.Parse(city\_id)).ToList();

return res[0].CurrentWeather;

}

public async Task<Dictionary<int, string[]>> get\_geo(string geo)

{

string data = await GetGEO(geo);

Root2 datas = JsonConvert.DeserializeObject<Root2>(data);

string[] cities = new string[datas.results.Count()];

if (datas.results.Count() == 1)

{

var results = new Dictionary<int, string[]>();

string[] coordinates = { datas.results[0].name, datas.results[0].longitude.ToString(), datas.results[0].latitude.ToString() };

results.Add(1, coordinates);

return results;

}

for (int i = 0; i < cities.Length; i++)

{

cities[i] = datas.results[i].name;

}

string[] longtitude = new string[datas.results.Count()];

for (int i = 0; i < longtitude.Length; i++)

{

longtitude[i] = datas.results[i].longitude.ToString();

}

string[] latitude = new string[datas.results.Count()];

for (int i = 0; i < latitude.Length; i++)

{

latitude[i] = datas.results[i].latitude.ToString();

}

var result = new Dictionary<int, string[]>();

for (int i = 1; i < cities.Length; i++)

{

string[] coordinates = { cities[i], longtitude[i], latitude[i] };

result.Add(i, coordinates);

}

return result;

}

public string Login(string username, string password)

{

var res = WeatherContext.GetContext().Users.Where(x => x.Password == password && x.Username == username).ToList();

if (res.Count == 1) return "Success";

else return "Failed";

}

public List<City> get\_cities(string user\_id)

{

var ss = WeatherContext.GetContext().UserInfos.Where(x => x.UserId == int.Parse(user\_id)).ToList();

long[] cc = new long[ss.Count()];

for (int i = 0; i < ss.Count(); i++)

{

cc[i] = ss[i].CityId;

}

var stupid = WeatherContext.GetContext().Cities.Where(x => cc.Contains(x.Id)).ToList();

List<City> cits = new List<City>();

for (int i = 0; i < stupid.Count(); i++)

{

City cur = new City();

cur.City1 = stupid[i].City1;

cur.Longitude = stupid[i].Longitude;

cur.Latitude = stupid[i].Latitude;

cur.Id = stupid[i].Id;

cits.Add(cur);

} return cits;

}

public string get\_user\_id(string username, string password)

{

var res = WeatherContext.GetContext().Users.Where(x => x.Password == password && x.Username == username).ToList();

if (res.Count == 1) return res[0].Id.ToString();

else return "Failed";

}

public async Task<string> add\_city(string city, string longtitude, string latitude, string user\_id)

{

var res = WeatherContext.GetContext().Cities.Where(x => x.City1 == city && x.Longitude == double.Parse(longtitude.Replace(".", ",")) && x.Latitude == double.Parse(latitude.Replace(".", ","))).ToList();

if (res.Count == 1)

{

UserInfo info = new UserInfo();

info.UserId = int.Parse(user\_id);

info.CityId = res[0].Id;

WeatherContext.GetContext().UserInfos.Add(info);

WeatherContext.GetContext().SaveChanges();

return "City already exist";

}

else {

City area = new City();

area.City1 = city;

area.Longitude = double.Parse(longtitude.Replace(".", ","));

area.Latitude = double.Parse(latitude.Replace(".", ","));

WeatherContext.GetContext().Cities.Add(area);

WeatherContext.GetContext().SaveChanges();

var current\_area = WeatherContext.GetContext().Cities.Where(x => x.City1 == city && x.Longitude == double.Parse(longtitude.Replace(".", ",")) && x.Latitude == double.Parse(latitude.Replace(".", ","))).ToList();

CityInfo city\_infos = new CityInfo();

var data = await get\_data(current\_area[0].Latitude.ToString().Replace(",", "."), current\_area[0].Longitude.ToString().Replace(",", "."));

UserInfo info = new UserInfo();

Root datas = JsonConvert.DeserializeObject<Root>(data);

List<double> max = datas.daily.temperature\_2m\_max; /\*data.Split("\"temperature\_2m\_max\":[")[1].Split("]")[0].Replace(",", ";").Replace(".", ",").Split(";");\*/

/\* double[] max = new double[max\_weath.Length];\*/

List<double> min = datas.daily.temperature\_2m\_min; /\*data.Split("\"temperature\_2m\_min\":[")[1].Split("]")[0].Replace(",", ";").Replace(".", ",").Split(";");\*/

/\* double[] min = new double[min\_weath.Length];\*/

double current\_weath = datas.current\_weather.temperature;

/\*double current\_weath = double.Parse(data.Split("\"current\_weather\":{\"temperature\":")[1].Split(",")[0].Replace(".", ","));\*/

List<double> period = datas.hourly.temperature\_2m; /\*data.Split("\"temperature\_2m\":[")[1].Split("]")[0].Replace(",", ";").Replace(".", ",").Split(";");\*/

/\*double[] period = new double[per.Length];\*/

city\_infos.CurrentWeather = current\_weath;

city\_infos.MaxWeather = max.ToArray();

city\_infos.MinWeather = min.ToArray();

city\_infos.PeriodWeather = period.ToArray();

city\_infos.CityId = current\_area[0].Id;

info.UserId = int.Parse(user\_id);

info.CityId = current\_area[0].Id;

WeatherContext.GetContext().CityInfos.Add(city\_infos);

WeatherContext.GetContext().SaveChanges();

WeatherContext.GetContext().UserInfos.Add(info);

WeatherContext.GetContext().SaveChanges();

return "Success"; }

}

public string register(string username, string password)

{

var res = WeatherContext.GetContext().Users.Where(x => x.Username == username).ToList();

try

{

if (res.Count == 0)

{ User user = new User();

user.Username = username;

user.Password = password;

WeatherContext.GetContext().Users.Add(user);

WeatherContext.GetContext().SaveChanges();

return "Success";

}

else return "Username is already exist";

}

catch

{

return "Error";

}

}

public string delete(string user\_id, string city\_id)

{

var res = WeatherContext.GetContext().UserInfos.Where(x => x.UserId == int.Parse(user\_id) && x.CityId == int.Parse(city\_id)).ToList();

WeatherContext.GetContext().UserInfos.Remove(res[0]);

WeatherContext.GetContext().SaveChanges();

return "Success";

}

public double[] get\_maximum(string city\_id)

{

var res = WeatherContext.GetContext().CityInfos.Where(x => x.CityId == int.Parse(city\_id)).ToList();

return res[0].MaxWeather;

}

public double[] get\_minimum(string city\_id)

{

var res = WeatherContext.GetContext().CityInfos.Where(x => x.CityId == int.Parse(city\_id)).ToList();

return res[0].MinWeather;

}

public double[] get\_graph(string city\_id)

{

var res = WeatherContext.GetContext().CityInfos.Where(x => x.CityId == int.Parse(city\_id)).ToList();

return res[0].PeriodWeather;

//WeatherContext.GetContext().User.Add(\_currentUser);

}

public List<UserInfo> get\_info()

{

var res = WeatherContext.GetContext().UserInfos.ToList();

return res;

//WeatherContext.GetContext().User.Add(\_currentUser);

}

public List<int> get\_city\_info()

{

var ss = WeatherContext.GetContext().CityInfos.ToList();

List<int> cits = new List<int>();

for (int i = 0; i<ss.Count(); i++)

{

cits.Add(Convert.ToInt32(ss[i].CityId));

}

return cits;

/\* var res = WeatherContext.GetContext().CityInfos.ToList();

return res;\*/

//WeatherContext.GetContext().User.Add(\_currentUser);

}

public List<City> get\_city(string city\_id)

{

var res = WeatherContext.GetContext().Cities.Where(x => x.Id == int.Parse(city\_id)).ToList();

return res;

//WeatherContext.GetContext().User.Add(\_currentUser);

}

public async Task<string> update\_city\_info(string city\_id)

{

List<CityInfo> cc = WeatherContext.GetContext().CityInfos.Where(x => x.CityId == int.Parse(city\_id)).ToList();

List<City> current\_area = WeatherContext.GetContext().Cities.Where(x => x.Id == int.Parse(city\_id)).ToList();

var data = await get\_data(current\_area[0].Latitude.ToString().Replace(",", "."), current\_area[0].Longitude.ToString().Replace(",", "."));

Root datas = JsonConvert.DeserializeObject<Root>(data);

List<double> max = datas.daily.temperature\_2m\_max; /\*data.Split("\"temperature\_2m\_max\":[")[1].Split("]")[0].Replace(",", ";").Replace(".", ",").Split(";");\*/

/\* double[] max = new double[max\_weath.Length];\*/

List<double> min = datas.daily.temperature\_2m\_min; /\*data.Split("\"temperature\_2m\_min\":[")[1].Split("]")[0].Replace(",", ";").Replace(".", ",").Split(";");\*/

/\* double[] min = new double[min\_weath.Length];\*/

double current\_weath = datas.current\_weather.temperature;

/\*double current\_weath = double.Parse(data.Split("\"current\_weather\":{\"temperature\":")[1].Split(",")[0].Replace(".", ","));\*/

List<double> period = datas.hourly.temperature\_2m; /\*data.Split("\"temperature\_2m\":[")[1].Split("]")[0].Replace(",", ";").Replace(".", ",").Split(";");\*/

/\*double[] period = new double[per.Length];\*/

cc[0].CurrentWeather = current\_weath;

cc[0].MaxWeather = max.ToArray();

cc[0].MinWeather = min.ToArray();

cc[0].PeriodWeather = period.ToArray();

WeatherContext.GetContext().SaveChanges();

return "Success";

//WeatherContext.GetContext().User.Add(\_currentUser);

}

}

}

# Входные и выходные данные

Входные данные:

|  |  |
| --- | --- |
| **Поля** | **Тип данных** |
| Graph | Массив чисел |
| Min | Массив чисел |
| Max | Массив чисел |
| Avg | Массив чисел |
| Longtitude | Числовой |
| Latitude | Числовой |
| Название города | Текстовой |

**Выходная информация:**

Выходной информацией данные о погоде для пользователя.

# Сообщения

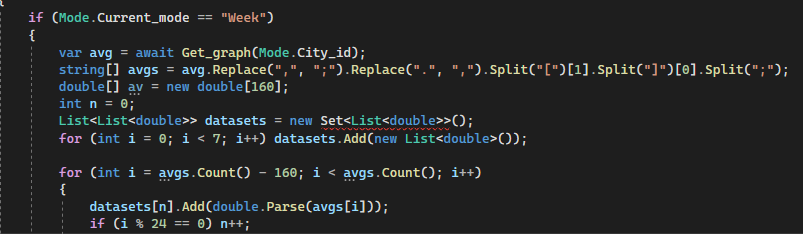
Участок кода с получением данных о температуре за неделю, работающий некорректно:

Рис.1 «Неявное присваивание»

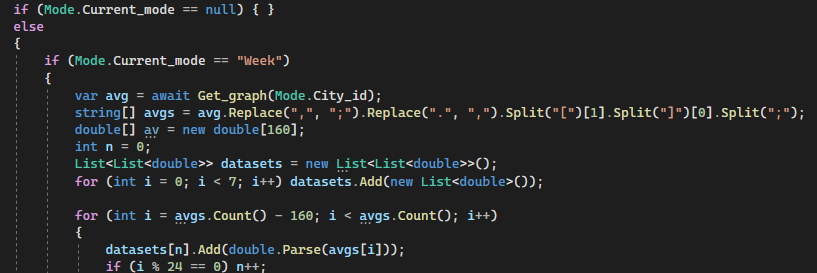
Исправленный участок кода с получением данных о температуре за неделю, работающий корректно: 

Рис.2 «Присваивание списка списков чисел»

Участок кода с переходом на страницу добавления города, работающий некорректно:

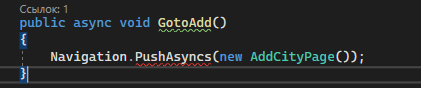


Рис.3 «Неправильно указано имя модуля»

Участок кода с переходом на страницу добавления города, работающий корректно:

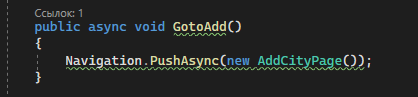


Рис.4 «Указания правильного имени модуля»