**Министерство образования Московской области**

**ГОУ ВО МО «Государственный гуманитарно-технологический университет» (ГГТУ)**

**Ликино-Дулевский политехнический колледж – филиал ГГТУ**

**О Т Ч Ё Т**

**ПО УЧЕБНОЙ ПРАКТИКЕ**

ПМ.01 Разработка модулей программного обеспечения для компьютерных систем

ПМ.11 Разработка, администрирование и защита баз данных

**ОБУЧАЮЩЕГОСЯ \_\_\_\_\_\_\_\_Ульянов Андрей Дмитриевич \_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**фамилия имя отчество**

**СПЕЦИАЛЬНОСТЬ**

**09.02.07 Информационные системы и программирование\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**КУРС 3 ГРУППА ИСП.20А \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**ВИД ПРАКТИКИ \_\_\_\_\_\_Учебная практика \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**ПЕРИОД ПРАКТИКИ 24.11.2022 г.-07.12.2022г., 06.04.2023 г.-19.04.2023 г.**

**МЕСТО ПРОХОЖДЕНИЯ ПРАКТИКИ**

**\_\_Ликино-Дулевский политехнический колледж – филиал ГГТУ\_\_\_\_\_\_\_\_\_\_\_**

**наименование организации**

**г. Ликино-Дулево**

**2023 г.**

Оглавление

[Введение 3](#_Toc133996480)

[1. Постановка задачи 3](#_Toc133996481)

[2. Проектирование интерфейса приложения 3](#_Toc133996482)

[2.1. Правила и принципы разработки интерфейса 3](#_Toc133996483)

[2.2. Разработка макета приложения 4](#_Toc133996484)

[3. Структура хранения данных 4](#_Toc133996485)

[3.1. Проектирование структуры хранения данных 4](#_Toc133996486)

[3.2. Наполнение структуры данными 6](#_Toc133996487)

[4 Руководство программиста 6](#_Toc133996488)

[4.1 Выбор средств разработки 6](#_Toc133996489)

[4.2 Технологии доступа к данным 7](#_Toc133996490)

[4.3 Программные методы разработки интерфейса 7](#_Toc133996491)

[4.4 Реализация функционала приложения 8](#_Toc133996492)

[4.5 Обработки исключений 10](#_Toc133996493)

[5. Тестирование программных модулей 10](#_Toc133996494)

[5.1. План тестирования 10](#_Toc133996495)

[5.2. Сценарии тестирования с результатами 10](#_Toc133996496)

[5.3. Предложения по улучшению функциональности 11](#_Toc133996497)

[Заключение 11](#_Toc133996498)

[Список литературы 12](#_Toc133996499)

[Приложения 12](#_Toc133996500)

[Приложение 1. Техническое задание 12](#_Toc133996501)

[Приложение 2. Руководство программиста 12](#_Toc133996502)

[Приложение 3. Руководство пользователя 12](#_Toc133996503)

**Введение**

Метеорология – наука об атмосфере, ее составе, строении, свойствах, физических и химических процессах, в ней происходящих. Теоретической основой метеорологии служат фундаментальные законы физики и химии.

Температура воздуха – один из термодинамических параметров состояния атмосферы.

Прогноз погоды – научно обоснованное предположение о будущем состоянии погоды в определённом пункте или регионе на определённый период

1. **Постановка задачи**

Разработка проводятся на основании Договора Между ЛДПК и ООО «Evion» № 17 от 17 апреля 2023 г.

ООО «Evion»

Наименование работы: Настольно приложение “WeatherApp”.

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.

1. **Проектирование интерфейса приложения**

## 2.1. Правила и принципы разработки интерфейса

Интерфейс – это совокупность информационной модели проблемной области, средств и способов взаимодействия пользователя с информационной моделью, а также компонентов, обеспечивающих формирование ин­формационной модели в процессе работы программной системы.

Качество пользователь­ского интерфейса является самостоятельной характеристикой программного про­дукта, сопоставимой по значимости с такими его показателями, как надежность и эффективность использования вычислительных ресурсов.

Для создания у пользователя такого ощущения «внутренней свободы» интер­фейс должен обладать целым рядом свойств:

* Естественность интерфейса.
* Согласованность интерфейса.
* Дружественность интерфейса (Принцип «прощения пользователя»)
* Принцип «обратной связи».
* Простота интерфейса.
* Гибкость интерфейса.
* Эстетическая привлекательность.
  1. **Разработка макета приложения**

Разработка макетов реализована в draw.io

Интерфейс вывода данных:

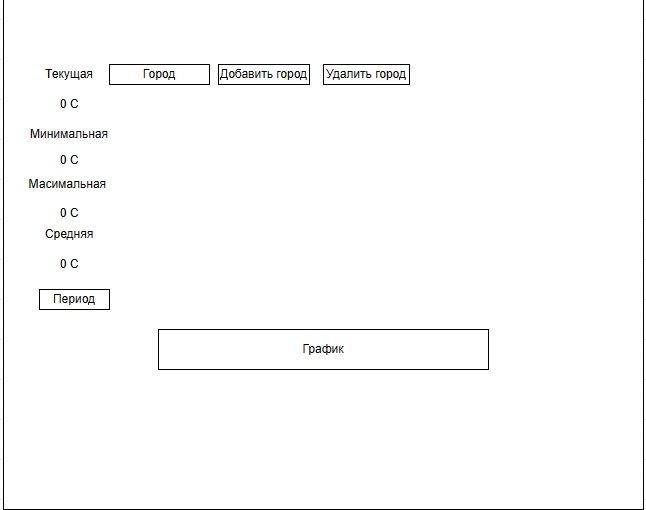


Рис.1 «Пример интерфейса №1»

Интерфейс добавления города:

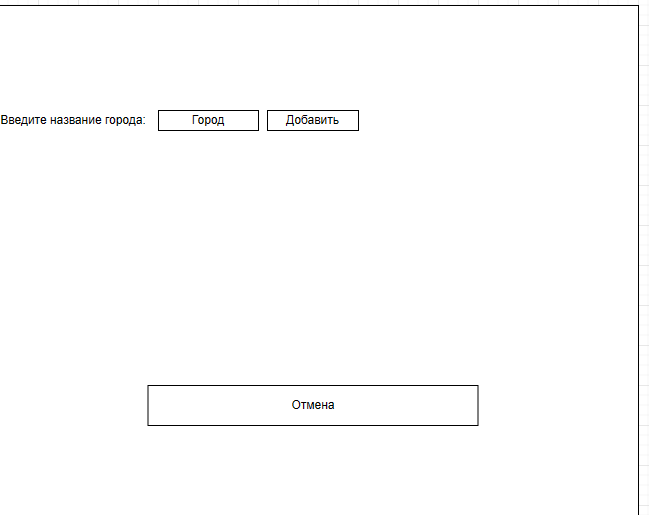
****

Рис.2 «Пример интерфейса №2»

1. **Структура хранения данных**

## 3.1. Проектирование структуры хранения данных

Базы данных (БД) – хранилище структурированных данных и методы доступа к ним, при этом данные должны быть непротиворечивы, целостны и минимально избыточны. Это особым образом организованный набор значений данных, а схема БД определяет, как именно организованы данные в БД. Можно сказать, что БД создаются для хранения и доступа к данным, содержащим сведения о некоторой предметной области, т.е. всякая БД представляет собой систему данных о предметной области.

На этом этапе анализируется имеющаяся информация с целью определить набор таблиц и их полей. Проектирование базы данных заключается в определении состава полей ее таблицы и связей между таблицами. От того, насколько тщательно проведен анализ и насколько грамотно спроектирована БД, в существенной мере зависит эффективность будущей программы и ее полезность для пользователя. Для того, чтобы создать БД, ее нужно нормализовать.

Предметная область, которая исследуется это информация о городах и температуре воздуха в них. В результате выявлены следующие сущности: координаты города (широта и долгота), наименование городов и температура воздуха в городах за периоды времени.

В процессе разработки структуры базы данных может возникнуть избыточность информации – это повторение (дублирование) данных, содержащихся в базе данных. В таблице с избыточными данными содержится много повторяющихся данных в различных записях. Для избавления от избыточности можно разбить одну большую таблицу на несколько маленьких. Такой процесс разбиения таблицы называется нормализацией базы данных. Т.е. нормализация – это процесс устранения избыточности данных.

Существует несколько так называемых нормальных форм (НФ) баз данных. Каждая из нормальных форм получается последовательно из предыдущей нормальной формы. На практике применяются в основном первые три из них.

Первая нормальная форма требует, чтобы каждое поле таблицы БД было не делимым, не содержало повторяющихся групп. Не делимость поля означает, что каждое поле не должно делится на несколько полей. А повторяющиеся группы указывают на то, чтобы поля не содержали одинаковые по смыслу значения.

Вторая нормальная форма, требует: все поля таблицы зависели от первичного ключа, то есть первичный ключ однозначно определен и является не избыточным; те поля, которые зависят от части первичного ключа, должны быть выделены в отдельные таблицы.

Третья нормальная форма требует, чтобы значение любого поля таблицы, не входящего в первичный ключ, не зависело от значения другого поля, не входящего в первичный ключ.

Нормализованной БД называется база, в которой выполняется как минимум три условия. В результате нормализации получаться следующие таблицы:

Таблица «cities» содержит данные о городах: наименование, координаты.

Таблица №1 «cities»

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | city | longtitude | latitude |
| 15 | Ликино-Дулево | 38.9542 | 55.7083 |
| 16 | Орехово-Зуево | 38.96178 | 55.80672 |

Таблица «city\_info» содержит данные о погоде для каждого города: максимальная, минимальная, средняя и текущая температура воздуха.

Таблица №2 «city\_info»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **id** | min\_weather | max\_weather | period\_weather | current\_weather | city\_id |
| 15 | Ликино-Дулево | 5,4,5,1,4,5,1 | 3,4,3,5,3,1 | 4,5,6,7,8,9 | 15 |
| 16 | Орехово-Зуево | 6,5,2,5,6,2 | 3,4,3,3,5,1 | 4,5,7,7,8,9 | 16 |

Таблица «users» содержит данные о пользователях, их логины и пароли.

Таблица №3 «users»

|  |  |  |
| --- | --- | --- |
| **id** | username | password |
| 1 | admin | admin |
| 2 | user | user |

Таблица «user\_info» содержит данные городах, который добавил для себя каждый юзер.

Таблица №4 «user\_info»

|  |  |  |
| --- | --- | --- |
| **id** | user\_id | city\_id |
| 1 | 1 | 15 |
| 2 | 2 | 16 |

## 3.2. Наполнение структуры данными

Данные хранятся в таблицах. Наполнение структуры данными происходит автоматически, извлекая информацию по API.

1. **Руководство программиста**
   1. **Выбор средств разработки**

Для разработки программы выбраны следующие средства: ОС Windows 10, Visual Studio, C#:

Windows 10 — операционная система для персональных компьютеров и рабочих станций, разработанная корпорацией Microsoft в рамках семейства Windows NT.

Интегрированная среда разработки Visual Studio — это стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений. Помимо стандартного редактора и отладчика, которые есть в большинстве сред IDE, Visual Studio включает в себя компиляторы, средства автозавершения кода, графические конструкторы и многие другие функции для улучшения процесса разработки.

C# (Си-шарп) — объектно-ориентированный язык программирования для платформы .NET. Разработан в 2000 году Андерсом Хейлсбергом, Скоттом Вилтамутом и Питером Гольде под эгидой Microsoft Research.

* 1. **Технологии доступа к данным**

Технологии доступа к данным являются прослойкой между API конкретного сервера и приложением пользователя, предоставляя программисту простой унифицированный механизм работы с данными.

На сегодняшний день существует множество технологий доступа к данным, таких как BDE, OLE, ODBC, DАО, АDО, и до сих пор разрабатываются новые, более надежные, удобные в работе и более быстродействующие технологии.

Механизмы доступа к базам данных снижают сложность обмена информацией с базами, однако интерпретация результатов их работы также достаточно трудоемка. Поэтому реализованы наборы компонентов, предназначенные для взаимодействия с механизмами обмена.

Таким образом, можно выделить несколько субъектов, участвующих в движении информации между базой данных и приложением (например, пользовательским интерфейсом):

1) интерфейсная часть приложения или его программная часть, манипулирующая информацией, хранимой в базе данных.;

2) компоненты, обеспечивающие связь приложения с механизмом доступа к базе данных;

3) механизм доступа к базе данных;

4) база данных.

* 1. **Программные методы разработки интерфейса**

Пользовательский интерфейс представляет собой совокупность программных и аппаратных средств, обеспечивающих взаимодействие пользователя с компьютером. Основу такого взаимодействия составляют диалоги. Под диалогом в данном случае понимают регламентированный обмен информацией между человеком и компьютером, осуществляемый в реальном масштабе времени и направленный на совместное решение конкретной задачи: обмен информацией и координация действий. Каждый диалог состоит из отдельных процессов ввода-вывода, которые физически обеспечивают связь пользователя и компьютера.

Обмен информацией осуществляется передачей сообщений и управляющих сигналов. Сообщение - порция информации, участвующая в диалоговом обмене. Различают:

• входные сообщения, которые генерируются человеком с помощью средств ввода: клавиатуры, манипуляторов, например мыши и т. п.

• выходные сообщения, которые генерируются компьютером в виде текстов, звуковых сигналов и/или изображений и выводятся пользователю на экран монитора или другие устройства вывода информации.

В основном пользователь генерирует сообщения следующих типов: запрос информации, запрос помощи, запрос операции или функции, ввод или изменение информации, выбор поля кадра и т. д. В ответ он получает: подсказки или справки, информационные сообщения, не требующие ответа, приказы, требующие действий, сообщения об ошибках, нуждающиеся в ответных действиях, изменение формата кадра и т. д.

Ниже перечислены основные устройства, обеспечивающие выполнение операций ввода-вывода.

Для вывода сообщений:

• монохромные и цветные мониторы - вывод оперативной текстовой и графической информации;

• принтеры - получение «твердой копии» текстовой и графической информации;

• графопостроители - получение твердой копии графической информации;

• синтезаторы речи - речевой вывод;

• звукогенераторы - вывод музыки и т. п.

Для ввода сообщений:

• клавиатура - текстовый ввод;

• планшеты - графический ввод;

• сканеры - графический ввод;

• манипуляторы, световое перо, сенсорный экран - позиционирование и выбор информации на экране и т. п.

* 1. **Реализация функционала приложения**

Функции серверной части:

Метод GetGEO(string city) получает координаты города, который добавляет пользователь:

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();

}

Метод get\_current(string city\_id) возвращает данные о выбранном городе:

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;

}

Метод Login(string username, string password) проверяет введеные пользователем логин и пароль, и в случае соответствия дает пользователю доступ к программе:

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";

}

Функции приложения:

Метод BtnAuth\_Click(object sender, RoutedEventArgs e) производит отправку логина и пароля на сервер для авторизации пользователя:

private async void BtnAuth\_Click(object sender, RoutedEventArgs e)

{

string username = TxbLogin.Text;

string password = PsbPassword.Password.ToString();

var response = await Auth(username, password);

if (response.ToString() == "Success")

{

User.Password = password;

User.Username = username;

manager.MainFrame.Navigate(new Weath());

}

else { MessageBox.Show("Неверные данные!"); };

}

static async Task<string> Auth(string username, string password)

{

HttpClient httpClient = new HttpClient();

var dt = DateTime.Now;

DateTime month = dt.AddMonths(-1);

// получаем ответ

using HttpResponseMessage response = await httpClient.GetAsync("https://localhost:7205/login/"+username+"/"+password);

return await response.Content.ReadAsStringAsync();

}

Аналогичный метод для регистрации:

private async void BtnAuth\_Click(object sender, RoutedEventArgs e)

{

string username = TxbLogin.Text;

string password = PsbPassword.Password.ToString();

var response = await Auth(username, password);

if (response.ToString() == "Success")

{

User.Password = password;

User.Username = username;

manager.MainFrame.Navigate(new Weath());

}

else { MessageBox.Show("Данный логин уже занят!"); };

}

static async Task<string> Auth(string username, string password)

{

HttpClient httpClient = new HttpClient();

var dt = DateTime.Now;

DateTime month = dt.AddMonths(-1);

// получаем ответ

using HttpResponseMessage response = await httpClient.GetAsync("https://localhost:7205/register/" + username + "/" + password);

return await response.Content.ReadAsStringAsync();

}

* 1. **Обработки исключений**

Иногда при выполнении программы возникают ошибки, которые трудно предусмотреть или предвидеть, а иногда и вовсе невозможно. Например, при передачи файла по сети может неожиданно оборваться сетевое подключение. такие ситуации называются исключениями. Язык C# предоставляет разработчикам возможности для обработки таких ситуаций. Для этого в C# предназначена конструкция try...catch...finally.

При использовании блока try...catch..finally вначале выполняются все инструкции в блоке try. Если в этом блоке не возникло исключений, то после его выполнения начинает выполняться блок finally. И затем конструкция try..catch..finally завершает свою работу.

Если же в блоке try вдруг возникает исключение, то обычный порядок выполнения останавливается, и среда CLR начинает искать блок catch, который может обработать данное исключение. Если нужный блок catch найден, то он выполняется, и после его завершения выполняется блок finally.

Если нужный блок catch не найден, то при возникновении исключения программа аварийно завершает свое выполнение.

# **5. Тестирование программных модулей**

## 5.1. План тестирования

Тестирование – процесс исследования и контроль качества, который состоит из планирования, проектирования, собственно проверки и анализа ее результатов.

Протестировать следующие функции:

1. Наличие пустого поля в объекте
2. Ожидание выбора города

## 5.2. Сценарии тестирования с результатами

Таблица №2 «Аннотация теста»

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| Название проекта | WeatherApp |
| Рабочая версия | 1.0 |
| Имя тестирующего | Синюкова Ирина Андреевна |
| Дата(ы) теста | 25.04.2023 |

Таблица №3 «Тест №1»

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| Тестовый пример # | TS\_Null\_1. |
| Приоритет тестирования  (Низкий/Средний/Высокий) | Средний. |
| Заголовок/название теста | Добавить город, оставив поле ввода пустым. |
| Краткое изложение теста | Попытка добавления города, оставив поле пустым. |
| Этапы теста | Нажать на кнопку «Добавить город» и оставить поле пустым, нажав «Добавить». |
| Тестовые данные | Наименование: пустое |
| Ожидаемый результат | Предупреждающее сообщение о незаполненном поле. |
| Фактический результат | Возникновение исключения. |
| Предварительное условие | Веб-сервер должен быть запущен. |
| Постусловие | Приложение закрылось с ошибкой. |
| Статус(Зачет/Незачет) | Незачет. |

Таблица №4 «Тест №2»

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| Тестовый пример # | TS\_period\_slct. |
| Приоритет тестирования  (Низкий/Средний/Высокий) | Средний. |
| Заголовок/название теста | Выбор периода. |
| Краткое изложение теста | Если период выбран ранее, чем город, программа должна ожидать выбор города. |
| Этапы теста | Открыть программу, выбрать период |
| Тестовые данные | Период: month |
| Ожидаемый результат | Ожидание выбора города |
| Фактический результат | Ожидание выбора города |
| Предварительное условие | Веб-сервер должен быть запущен. |
| Постусловие | Программа ожидает выбора города |
| Статус (Зачет/Незачет) | Зачет. |

## 5.3. Предложения по улучшению функциональности

1. Интегрировать отображение графика на главной странице.

2. Добавить обработчика ошибок при добавлении города

3. Выравнивание элементов главной страницы

# **Заключение**

В результате выполнения поставленной задачи разработана программа, предоставляющая данные о температуре в выбранном пользователе городе за выбранный период.

Для обеспечения уверенной работы приложения выполнены тестовые сценарии. Документирование позволило рассмотреть программный модуль с точки зрения программиста, пользователя и тестировщика.

В процессе разработки данной программы, углубил свои знания и навыки в области информационных технологий, изучив множество новых и полезных методик и технологий, которые я успешно применил в ходе работы.

# **Список литературы**

1. Фленов, Михаил Библия C# / Михаил Фленов. - М.: БХВ-Петербург, 2021

2. Сафонов, В. О. Параметризованные типы данных. История, теория, реализация и применение / В.О. Сафонов. - М.: Издательство Санкт-Петербургского университета, 2020.

3. Нейгел, Кристиан C# 5.0 и платформа .NET 4.5 для профессионалов / Кристиан Нейгел и др. - М.: Вильямс, 2020

4. Культин, Н. С# в задачах и примерах / Н. Культин. - М.: БХВ-Петербург, 2020

5. Климов, А. C#. Советы программистам / А. Климов. - М.: БХВ-Петербург, 2018

Интернет-источники:

1. https://metanit.com/

2. https://learn.microsoft.com/ru-ru/

3. https://stackoverflow.com/

4. https://github.com/topics/csharp

# **Приложения**

## Приложение 1. Техническое задание

# **Введение**

Метеорология – наука об атмосфере, ее составе, строении, свойствах, физических и химических процессах, в ней происходящих. Теоретической основой метеорологии служат фундаментальные законы физики и химии.

Температура воздуха – один из термодинамических параметров состояния атмосферы.

Прогноз погоды – научно обоснованное предположение о будущем состоянии погоды в определённом пункте или регионе на определённый период.

# **Основания для разработки**

Разработка проводятся на основании Договора Между ЛДПК и ООО «Evion» № 17 от 1 апреля 2023 г.

ООО «Evion»

Наименование работы: Настольно приложение “WeatherApp”.

# **Назначение разработки**

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.

# **Требования к программе или программному изделию**

## 4.1 Требования к функциональным характеристикам

## 4.1.1. Требования к составу выполняемых функций:

* Отображение списка городов
* Возможность выбора города для отображения статистики
* Возможность добавлять города
* Возможность 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. **Требования к организации входных данных:**

Входными данными являются логин и пароль пользователя, а также выбранные им города.

* + 1. **Требования к организации выходных данных:**

Выходные данные организованы в виде интерфейса приложения с отображением необходимых данных.

## 4.2 Требования к надежности.

Программа должна быть в достаточной степени надёжна от сбоев. На крайний случай предусмотрено сохранение данных БД в приложении «PostgreSQL» или восстановление данных в случае завершения работы.

## 4.3 Условия эксплуатации

Программа не требует специального обслуживания. Для ознакомления с полным функционалом пользователь должен прочесть Руководство пользователя. Для работы с программой требуются хотя бы малейшие навыки работы с приложениями с похожим интерфейсом.

**Климатические условия эксплуатации:**

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации.

## 4.4 Требования к составу и параметрам технических средств

|  |  |
| --- | --- |
| Процессор | Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz 2.70 GHz |
| Оперативная Память | 4,00 ГБ, DDR4, DIMM, 2133 МГц |
| Разрешение экрана | 1920x1080 |
| Размер экрана | От 15 дюймов |
| Устройства ввода | Мышь, клавиатура |
| Дисковое пространство | 0.5 Гб |
| Операционная система | Windows 10 |

## 4.5 Требования к информационной и программной совместимости

## 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.

## 4.6 Специальные требования

Программа должна обеспечивать взаимодействие с пользователем посредством графического пользовательского интерфейса.

## Требования к программной документации

В ходе разработки программы должны быть подготовлены следующие программные документы: текст программы, описание программы, программа и методика испытаний, руководство пользователя, руководство программиста, технико-экономическое обоснование.

## Приложение 2. Руководство программиста

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**

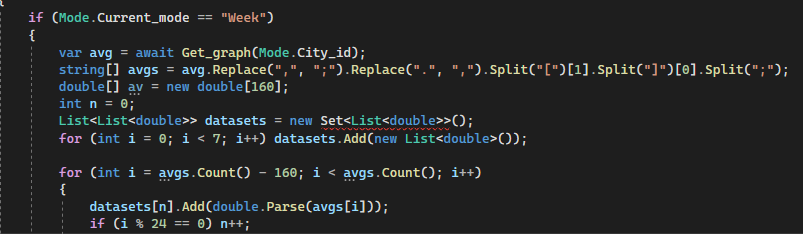
Участок кода с получением данных о температуре за неделю, работающий некорректно:

Рис.1 «Неявное присваивание»

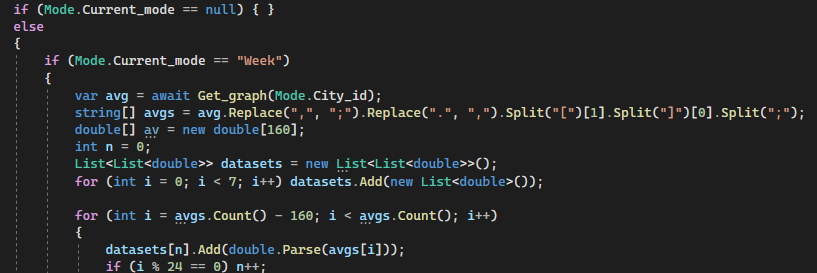
Исправленный участок кода с получением данных о температуре за неделю, работающий корректно: 

Рис.2 «Присваивание списка списков чисел»

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

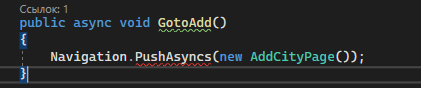


Рис.3 «Неправильно указано имя модуля»

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

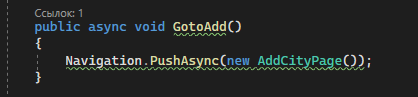


Рис.4 «Указания правильного имени модуля»

## Приложение 3. Руководство пользователя