**СОДЕРЖАНИЕ**

[1 Постановка задачи 5](#_Toc122327100)

[1.1 Анализ предметной области 5](#_Toc122327101)

[1.2 Техническое задание на разработку ГОСТ 19.201-78 5](#_Toc122327102)

[1.2.1 Введение 5](#_Toc122327103)

[1.2.2 Основания для разработки 5](#_Toc122327104)

[1.2.3 Назначение разработки 6](#_Toc122327105)

[1.2.4 Требования к программе или программному изделию 6](#_Toc122327106)

[1.2.5 Технико-экономические показатели 10](#_Toc122327107)

[1.2.6 Стадии и этапы разработки 10](#_Toc122327108)

[1.2.7 Порядок контроля и приемки 10](#_Toc122327109)

[1.3 Алгоритм программы 11](#_Toc122327110)

[1.4 Модель программы 11](#_Toc122327111)

[2 Проектирование 13](#_Toc122327112)

[2.1 Функциональная структура программы 13](#_Toc122327113)

[2.2 Схема модулей программы 14](#_Toc122327114)

[2.3 Паспорта основных модулей 14](#_Toc122327115)

[2.3.1 Модуль получения погоды 14](#_Toc122327116)

[2.3.2 Модуль получения погоды из OpenWeather API 17](#_Toc122327117)

[2.3.3 Модуль получения времени суток 19](#_Toc122327118)

[2.3.4 Модуль получения состояния погоды 20](#_Toc122327119)

[2.3.5 Модуль получения состояния осадков 21](#_Toc122327120)

[2.3.6 Модуль проверки температуры 23](#_Toc122327121)

[2.4 Метрики ПО и анализ кода 23](#_Toc122327122)

[3 Руководство оператора 25](#_Toc122327123)

[3.1 Назначение программы 25](#_Toc122327124)

[3.1.1 Функциональное назначение 25](#_Toc122327125)

[3.1.2 Состав функций 25](#_Toc122327126)

[3.2 Условия выполнения программы 25](#_Toc122327127)

[3.3 Требования к персоналу (пользователю) 26](#_Toc122327128)

[3.4 Выполнение программы 27](#_Toc122327129)

[4 Тестирование 32](#_Toc122327130)

[4.1 Программа, методика и результаты испытаний ГОСТ 19.301-79 32](#_Toc122327131)

[4.1.1 Объект испытаний 32](#_Toc122327132)

[4.1.1.1 Наименование программы 32](#_Toc122327133)

[4.1.1.2 Назначение разработки 32](#_Toc122327134)

[4.1.2 Цель испытаний 32](#_Toc122327135)

[4.1.2.1 Основание для проведения испытаний 32](#_Toc122327136)

[4.1.2.2 Место и продолжительность испытаний 32](#_Toc122327137)

[4.1.2.3 Перечень документов, предъявляемых на испытания 32](#_Toc122327138)

[4.1.2.4 Объем испытаний 33](#_Toc122327139)

[4.1.3 Требования к программе 34](#_Toc122327140)

[4.1.4 Требования к программной документации 34](#_Toc122327141)

[4.1.5 Средства и порядок испытаний 35](#_Toc122327142)

[4.1.5.1 Технические средства, используемые во время испытаний 35](#_Toc122327143)

[4.1.5.2 Условия и порядок проведения испытаний 35](#_Toc122327144)

[4.1.6 Методика испытаний 35](#_Toc122327145)

[4.2 Результаты предварительных испытаний 36](#_Toc122327146)

[4.3 Тестирование при помощи Unit-Test 38](#_Toc122327147)

[4.4 Метрики ПО 39](#_Toc122327148)

[ЗАКЛЮЧЕНИЕ 40](#_Toc122327149)

[ПРИЛОЖЕНИЕ А 41](#_Toc122327150)

1. **Постановка задачи**
   1. **Анализ предметной области**

Основная задача заключается в разработке модуля программы Погода, отвечающего за изображение на экране основного приложения. Также, модуль отвечает за графическое отображение времени суток, погоды.

* 1. **Техническое задание на разработку ГОСТ 19.201-78**
     1. **Введение**
        1. **Наименование программы**

Наименование программы «Погода».

* + - 1. **Краткая характеристика области применения**

Программа предназначена для отображения основной информации по погоде и времени суток в Рязанской области на текущее время использования.

* + 1. **Основания для разработки**
       1. **Основание для проведения разработки**

Основанием для проведения разработки является задание на учебную практику.

* + - 1. **Наименование и условное обозначение темы разработки**

Наименование темы разработки – «Погода». Условное обозначение темы разработки (шифр темы) – «09.02.07».

* + 1. **Назначение разработки**
       1. **Функциональное назначение**

Функциональным назначением программы является отображение сведений о текущей погоде и времени суток в Рязанской области.

* + - 1. **Эксплуатационное назначение**

Для личного пользования, чтобы узнать погоду и время суток в Рязанской области.

* + 1. **Требования к программе или программному изделию**
       1. **Требования к функциональным характеристикам**

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

* отображение температуры;
* отображение времени суток;
* отображение текущего состояния погоды (ясно, облачно, осадки);
* отображение интенсивности осадков, если они есть;
* отображение текущего состояния погоды и интенсивности посредством изображений или анимированных картинок (GIF).
  + - 1. **Требования к надежности**

Программа не должна давать сбой в процессе взаимодействия.

* + - 1. **Время восстановления после отказа**

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы, не должно превышать 20 минут при условии соблюдения условий эксплуатации технических и программных средств.

Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем (крахом) операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

* + - 1. **Отказы из-за некорректных действий оператора**

Отказы программы возможны вследствие некорректных действий оператора (пользователя) при взаимодействии с операционной системой. Во избежание возникновения отказов программы по указанной выше причине следует обеспечить работу пользователя без предоставления ему административных привилегий.

* + - 1. **Условия эксплуатации**

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

* + - 1. **Требования к видам обслуживания**

Программа не требует проведения каких-либо видов обслуживания.

* + - 1. **Требования к численности и квалификации персонала**

Минимальное количество персонала, требуемого для работы программы, должно составлять не менее 2 штатных единиц — системный администратор и пользователь программы оператор. В перечень задач, выполняемых системным администратором, должны входить:

* задача поддержания работоспособности технических средств;
* задачи установки (инсталляции) и поддержания работоспособности системных программных средств — операционной системы;
* задача установки (инсталляции) программы.

Пользователь программы (оператор) должен обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы.

* + - 1. **Требования к составу и параметрам технических средств**

В состав технических средств должен входить IBM-совместимый персональный компьютер (ПЭВМ), включающий в себя:

Процессор: Intel Celeron G1610 2.6 ГГц и более

O3Y: 2 Гб и более

Необходимо место на жестком диске: 1024 Мб

Видеоадаптер: Intel HD Graphics и лучше

Экран: 1028 х 720

* + - 1. **Требования к информационной и программной совместимости**

**Требования к информационным структурам и методам решения**

Информационная структура должна включать отображение входных данных и результата на экране.

**Требования к исходным кодам и языкам программирования**

Исходные коды программы должны быть реализованы на языке C#. В качестве интегрированной среды разработки программы должна быть использована среда Microsoft Visual Studio.

**Требования к программным средствам, используемые программой**

Системные программные средства, используемые программой, должны быть представлены лицензионной локализованной версией операционной системы Windows 7, 8, 8.1, 10 с платформой .NET Framework 4.7.2.

**Требования к защите информации и программ**

Требования к защите информации и программ не предъявляются.

**Специальные требования**

Программа должна обеспечивать взаимодействие с пользователем (оператором) посредством графического пользовательского интерфейса, разработанной согласно рекомендациям лекции.

* + - 1. **Требования к маркировке и упаковке**

**Требования к маркировке**

Программное изделие должно иметь маркировку с обозначением фамилии студента, номера группы, наименования, номера версии. Маркировка должна быть нанесена на программное изделие в виде надписи.

**Требования к упаковке**

Упаковка программного изделия должна осуществляться конвертом для дисков.

* + - 1. **Требования к программной документации**

**Предварительный состав программной документации**

Состав программной документации должен включать в себя:

* техническое задание на разработку ГОСТ 19.201-78;
* руководство оператора ГОСТ 19.505-79;
* программа, методика и результаты испытаний ГОСТ 19.301-79.
  + 1. **Технико-экономические показатели**

**Экономические преимущества разработки**

Коммерческое использование не предусмотрено.

* + 1. **Стадии и этапы разработки**
       1. **Стадии разработки**

Разработка должна быть проведена в две стадии:

* техническое задание;
* технический (и рабочий) проекты.
  + - 1. **Этапы разработки**
* разработка тех. задания;
* разработка программы;
* разработка программной документации;
* испытания программы.
  + 1. **Порядок контроля и приемки**
       1. **Виды испытаний**

Приемочные испытания должны проводиться на объекте заказчика в сроки выполнения для проекта по учебной практике.

Приемочно-сдаточные испытания программы должны проводиться согласно разработанной исполнителем и согласованной заказчиком «Программы и методики испытаний».

* 1. **Алгоритм программы**

Программа должна иметь следующий алгоритм работы, представленный на рисунке 1.

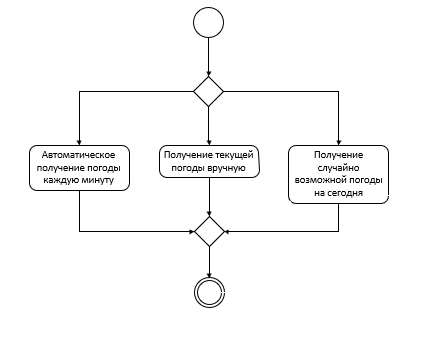


Рисунок 1 – Алгоритм программы

* 1. **Модель программы**

Программа имеет следующую модель, представленную на рисунке 2.

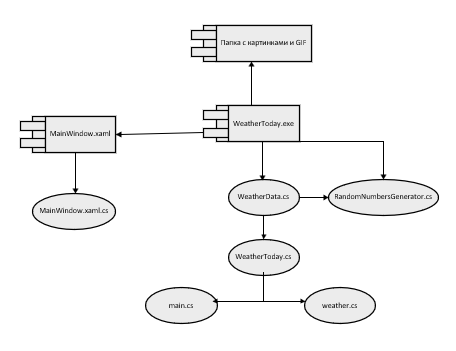


Рисунок 2 – Модель программы

1. **Проектирование**
   1. **Функциональная структура программы**

Описываются функции, которые будет выполнять программа. Согласно технологии структурного программирования, любая составная функция может, и должна быть разложена на элементарные для данного уровня функции.

Ниже Приведена таблица, в которую включаются все возможные функции, которые планируются использовать в программе в соответствии с таблицей 1.

Таблица 1 – Таблица спецификации функций

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Действие** | **Объект** | **Вид** |
| **Отображение погоды** | | | |
| Автосинхронизация погоды | Позволяет получать текущее состояние погоды каждую минуту | Элементы окна | Основная |
| Получить погоду | Позволяет получить текущую погоду | Элементы окна | Основная |
| ГСЧ погоды | Позволяет получить случайные данные о погоде, которые могли бы быть. | Элементы окна | Вспомогательная |
| **Справка** | | | |
| Посмотреть справку | Открывает окно справки, где описаны возможности программы | Окно | Вспомогательная |
| Посмотреть «О программе» | Открывает окно, где выводится информация о разработчике | Окно | Вспомогательная |

* 1. **Схема модулей программы**

Модульная структура программы в соответствии с рисунком 3.

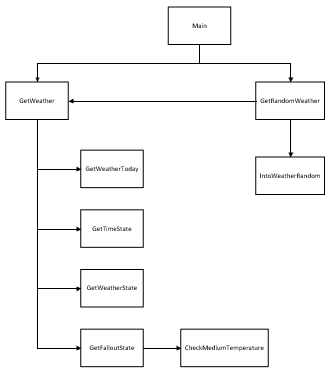
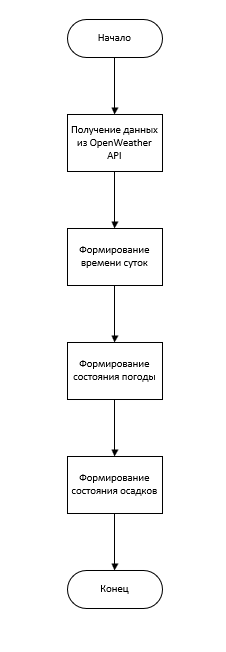


Рисунок 3 – Модульная структура программы

* 1. **Паспорта основных модулей**
     1. **Модуль получения погоды**

Таблица 2 – Структура данных функции «GetWeather»

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| TimeState | string | Свойство, хранящее состояние времени суток |
| WeatherState | string | Свойство, хранящее состояние погоды (ясно, облачно, осадки) |
| FalloutState | string | Свойство, хранящее статус осадков и их интенсивность |
| dateTime | DateTime | Несет в себе время, которое можно получить при использовании ToString(“t”).  Предназначена для формирования своего времени суток.  Также, используется только в одной перегрузке. |



**Рисунок 4** – Структура модуля «GetWeather». Первая перегрузка.

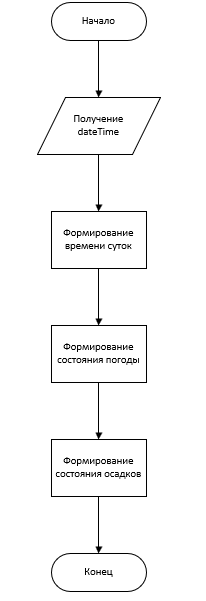


Рисунок 5 – Структура модуля «GetWeather». Вторая перегрузка.

* + 1. **Модуль получения погоды из OpenWeather API**

Таблица 3 – Структура данных функции «GetWeatherToday»

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Описание |
| request | WebRequest | Хранит в себе данные о запросе данных из API |
| response | WebResponse | Представляет собой «ответ» из API. |
| answer | string | Представляет собой строку, в которой будет хранится ответ в строчной форме из request. |
| CurrentWeather | WeatherToday | Представляет собой сформированный результат из строки в объект класса WeatherToday посредством JSON конвертации. Является свойством. Хранит в себе необходимый результат о погоде. |

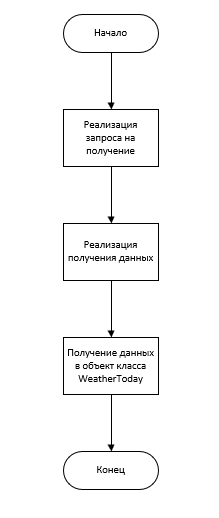


Рисунок 6 – Структура модуля «GetWeatherToday»

* + 1. **Модуль получения времени суток**

Таблица 4 – Структура данных функции «GetTimeState»

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Описание |
| currentTime | DateTime | Несет в себе время, относительно которого будет вестись расчет времени суток. |

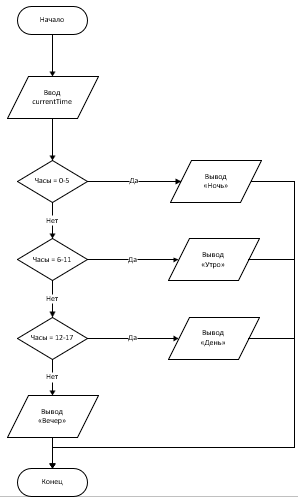


Рисунок 7 – Структура модуля «GetTimeState»

* + 1. **Модуль получения состояния погоды**

Таблица 5 – Структура данных «GetWeatherState»

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Описание |
| main | string | Представляет собой строку, в которой содержится строчная идентификация состояния погоды. |
| DiplayPath | string | Свойство, содержащее в себе путь к файлу, необходимому для идентификации состояния погоды и осадков, в целом. |

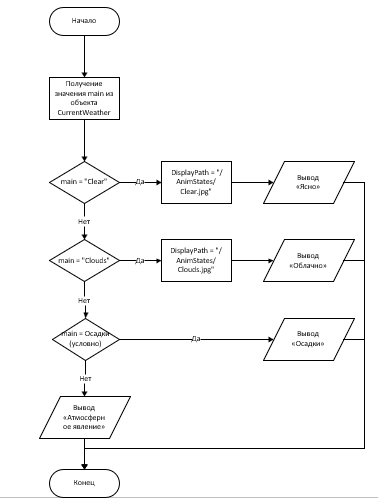


Рисунок 8 – Структура модуля «GetWeatherState»

* + 1. **Модуль получения состояния осадков**

Таблица 6 – Структура данных «GetFalloutState»

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Описание |
| description | string | Представляет собой строку, в которой содержится подробное состояние погоды. Используется для идентификации типа осадков и их интенсивности. |
| DiplayPath | string | Свойство, содержащее в себе путь к файлу, необходимому для идентификации состояния погоды и осадков, в целом. |

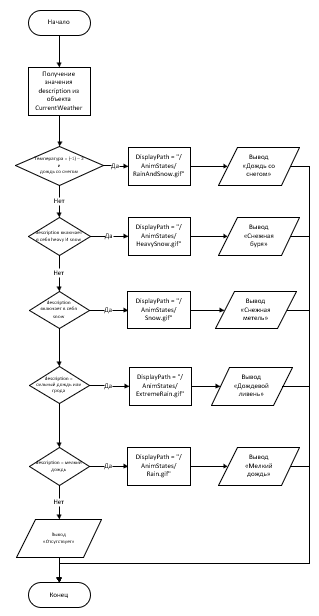


Рисунок 9 – Структура модуля «GetFalloutState»

* + 1. **Модуль проверки температуры**

Таблица 7 – Структура данных «CheckMediumTemperature»

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Описание |
| Temperature | int | Представляет собой свойство, хранящее в себе температуру, полученную в результате получения данных о погоде. |

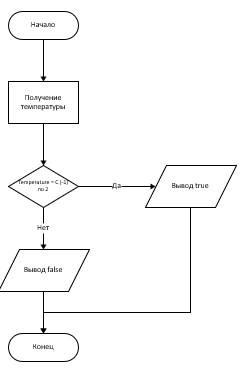


Рисунок 10 – Структура модуля «CheckMediumTemperature»

* 1. **Метрики ПО и анализ кода**

Метрики ПО следующие, как на рисунке 11.

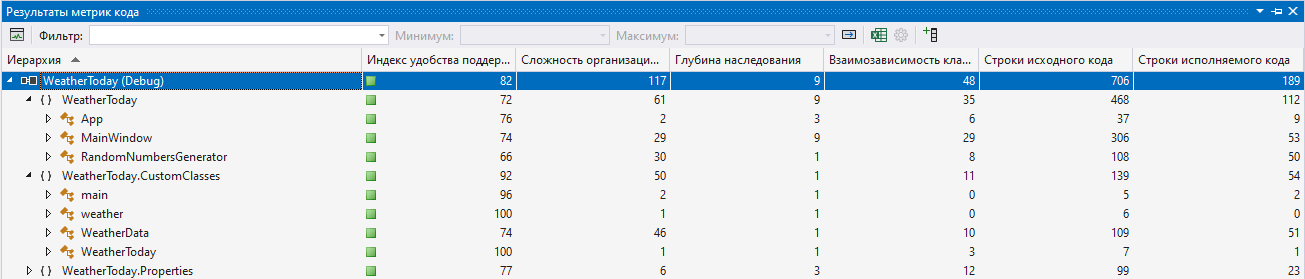


Рисунок 11 – Метрики ПО

При завершении проектирования была выполнена очистка кода встроенными возможностями Visual Studio. Неиспользуемые пространства имен и переменные были удалены успешно. Оптимизация далее не требуется. Метрики ПО удовлетворительные.

1. **Руководство оператора**

### Назначение программы

Программа предназначена для отображения основной информации по погоде и времени суток в Рязанской области на текущее время использования.

### Функциональное назначение

Функциональным назначением программы является отображение сведений о текущей погоде и времени суток в Рязанской области.

### Состав функций

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

* отображение температуры;
* отображение времени суток;
* отображение текущего состояния погоды (ясно, облачно, осадки);
* отображение интенсивности осадков, если они есть;
* отображение текущего состояния погоды и интенсивности посредством изображений или анимированных картинок (GIF).

### Условия выполнения программы

* + 1. **Климатические условия эксплуатации**

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

* + 1. **Требования к составу и параметрам технических средств**

В состав технических средств должен входить IBM-совместимый персональный компьютер (ПЭВМ), включающий в себя:

Процессор: Intel Celeron G1610 2.6 ГГц и более O3Y: 2 Гб и более

Необходимо место на жестком диске: 1024 Мб

Видеоадаптер: Intel HD Graphics и лучше

Экран: 1028 х 720

* + 1. **Требования к программным средствам, используемым программой**

Системные программные средства, используемые программой, должны быть представлены лицензионной локализованной версией операционной системы Windows 7, 8, 8.1, 10 с платформой .NET Framework 4.7.2.

### Требования к персоналу (пользователю)

Минимальное количество персонала, требуемого для работы программы, должно составлять не менее 2 штатных единиц — системный администратор и пользователь программы оператор. В перечень задач, выполняемых системным администратором, должны входить:

* задача поддержания работоспособности технических средств;
* задачи установки (инсталляции) и поддержания работоспособности системных программных средств — операционной системы;
* задача установки (инсталляции) программы.

Пользователь программы (оператор) должен обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы.

### Выполнение программы

* загрузка и запуск программы;
* выполнение программы;
* завершение работы программы.
  + 1. **Загрузка и запуск программы**

Загрузка и запуск программы осуществляется способами, детальные сведения о которых изложены в руководстве пользователя операционной системы.

В случае успешного запуска программы на рабочем столе будет отображено окно браузера как на рисунке 12.

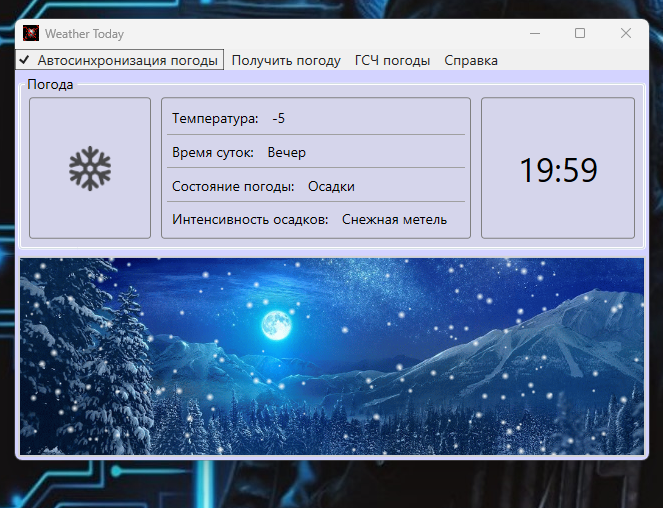


Рисунок 12 – Окно программы

* + 1. **Основные функции программы**
       1. **Автосинхронизация**

При необходимости, можно отключить автосинхронизацию, нажав соответствующую клавишу как на рисунке 13.

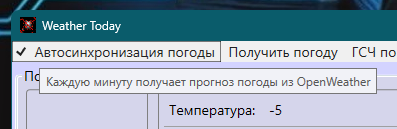


Рисунок 13 – Клавиша автосинхронизации

По результату нажатия автосинхронизация будет отключена, а кнопка будет иметь следующее состояние как на рисунке 14.

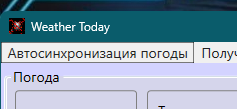


Рисунок 14 – Состояние кнопки после нажатия

После этого, автосинхронизация времени будет прекращена. Время последнего получения погоды указано в правой части окна как на рисунке 15.

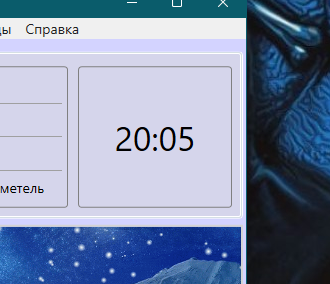


Рисунок 15 – Время последнего получения

* + - 1. **Получить погоду**

При нажатии кнопки «Получить погоду» будет получена погода на текущей момент времени в Рязани. На 18.12.2022 20:10 погода была такая как на рисунке 16.

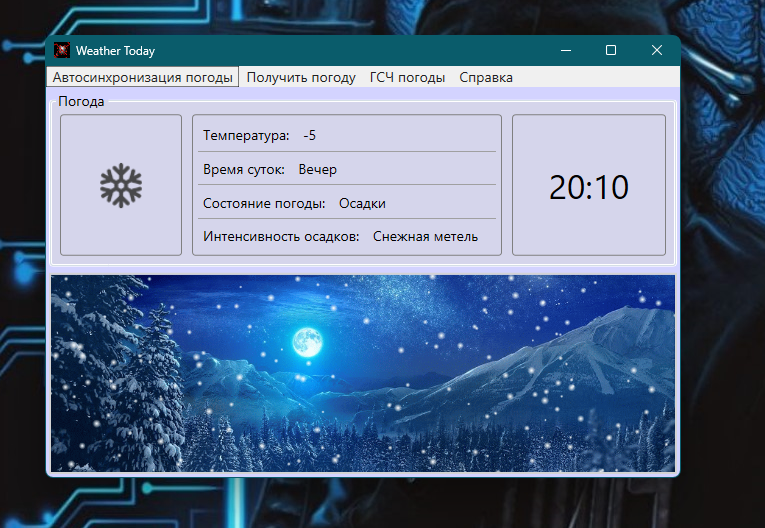


Рисунок 16 – Результаты получения

* + - 1. **ГСЧ погоды**

При желании, можно воспользоваться ГСЧ погоды, чтобы посмотреть ее возможные состояния. Примером результата может служить информация как на рисунке 17.

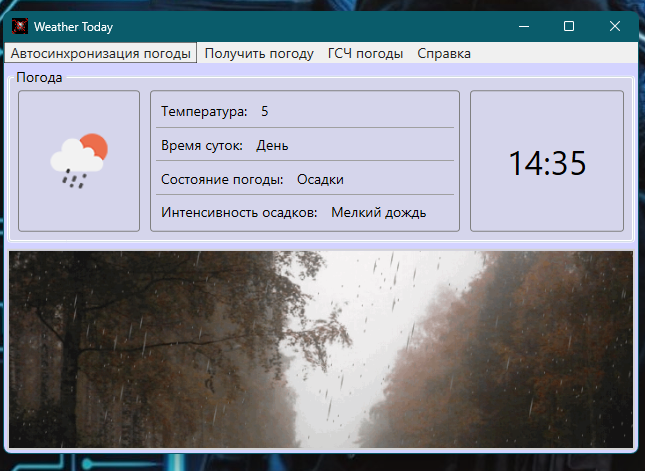


Рисунок 17 – Демонстрация главного окна

* + - 1. **Функция справки**

При необходимости, пользователь может воспользоваться справкой, если требуется узнать о возможностях программы. Для этого достаточно кликнуть на клавишу «Справка» и выбрать необходимый пункт. В первом случае пользователь может получить справку как на рисунке 18.

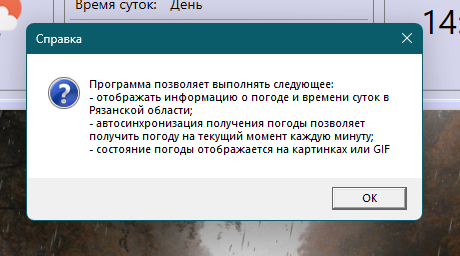


Рисунок 18 – Справка по программе

Во втором случае можно узнать информацию о разработчике как на рисунке 19.

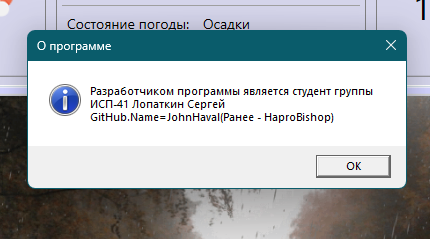


Рисунок 19 – О программе

1. **Тестирование**
   1. **Программа, методика и результаты испытаний ГОСТ 19.301-79**
      1. **Объект испытаний**
         1. **Наименование программы**

Наименование программы «Погода».

* + - 1. **Назначение разработки**

Программа предназначена для получения текущего состояния погоды в Рязанской области.

* + 1. **Цель испытаний**

Цель проведения испытаний – проверка соответствия характеристик разработанной программы (программного изделия) функциональным и отдельным иным видам требований, изложенным в документе Техническое задание.

* + - 1. **Основание для проведения испытаний**

Основание: задание на учебную практику.

* + - 1. **Место и продолжительность испытаний**

Дом, колледж. Срок: 1 неделя.

* + - 1. **Перечень документов, предъявляемых на испытания**

Состав программной документации должен включать в себя:

* техническое задание;
* руководство оператора;
* программа, методика и результаты испытаний.
  + - 1. **Объем испытаний**

**Перечень этапов испытаний**

Испытания проводятся в два этапа:

* ознакомительный;
* испытания.
  + - * 1. **Перечень проверок, проводимых на 1 этапе испытаний**

Перечень проверок, проводимых на 1-м этапе испытаний, должен включать в себя:

* проверку комплектности программной документации;
* проверку комплектности состава технических и программных средств.
  + - * 1. **Перечень проверок, проводимых на 2 этапе испытаний**

Перечень проверок, проводимых на 2-м этапе испытаний, должен включать в себя:

* проверку соответствия технических характеристик программы;
* проверку степени выполнения требований функционального назначения программы.
  + - * 1. **Количественные и качественные характеристики, подлежащие оценке**

В ходе проведения приемосдаточных испытаний оценке подлежат качественные (функциональные) характеристики программы. Проверке подлежит возможность выполнения программой перечисленных ниже функций:

* отображение температуры;
* отображение времени суток;
* отображение текущего состояния погоды (ясно, облачно, осадки);
* отображение интенсивности осадков, если они есть;
* отображение текущего состояния погоды и интенсивности посредством изображений или анимированных картинок (GIF).
  + 1. **Требования к программе**

При проведении испытаний функциональные характеристики (возможности) программы подлежат проверке на соответствие требованиям, изложенным в п. «Требования к функциональным характеристикам» технического задания.

* + 1. **Требования к программной документации**

Состав программной документации должен включать в себя:

* техническое задание;
* руководство оператора;
* программа, методика и результаты испытаний.
  + 1. **Средства и порядок испытаний**
       1. **Технические средства, используемые во время испытаний**

В состав технических средств должен входить IBM-совместимый персональный компьютер (ПЭВМ), включающий в себя:

Процессор: Intel Celeron G1610 2.6 ГГц и более

O3Y: 2 Гб и более

Необходимо место на жестком диске: 1024 Мб

Видеоадаптер: Intel HD Graphics и лучше

Экран: 1028 х 720

* + - * 1. **Порядок проведения испытаний**

Испытания должны проводиться поэтапно согласно п. «Перечень этапов испытаний» настоящего документа.

* + - 1. **Условия и порядок проведения испытаний**
         1. **Условия начала и завершения отдельных этапов испытаний**

Необходимым и достаточным условием завершения 1 этапа испытаний и начала 2 этапа испытаний является успешное завершение проверок, проводимых на 1 этапе (см. п. «Перечень проверок, проводимых на 1 этапе испытаний»).

Условием завершения 2 этапа испытаний является успешное завершение проверок, проводимых на 2 этапе испытаний.

* + - * 1. **Требования к техническому обслуживанию**

Требования к техническому обслуживанию не предъявляются.

* + 1. **Методика испытаний**

**Методы проведения проверки комплектности программной документации**

Проверка комплектности программной документации на программное изделие производится визуально преподавателем. В ходе проверок сопоставляется состав и комплектность программной документации, представленной исполнителем, с перечнем программной документации.

Проверка считается завершенной в случае соответствия состава и комплектности программной документации, представленной исполнителем, перечню программной документации, приведенному в указанном выше пункте.

**Методы проведения проверки степени выполнения требований функционального назначения программы**

Тестирование серого ящика или поведенческое тестирование – стратегия (метод) тестирования функционального проведения программы с точки зрения внешнего мира, при котором используется знание о внутреннем устройстве тестируемого объекта. Под методом понимаются систематические методы отбора и создания тестов для тестового набора. Стратегия поведенческого теста исходит из технических требований и их спецификаций.

* 1. **Результаты предварительных испытаний**

Результаты предварительного тестирования в соответствии с таблицей 8.

Таблица 8 - Функциональное тестирование «серого ящика»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Тестовая спецификация** | | | | | | |
| **Разработчик:**  Лопаткин Сергей | | **Начальные установки:** нет | | | | |
| **Источник тестовых данных:** | | **Цели:**  Проверки степени выполнения требований функционального назначения программы | | | | |
| **Test Case №** | **Описание** | **Шаги теста** | **Ожидаемые результаты** | **Реальные результаты** | **Прошел/Провалился** | **Тестер/Дата** |
| **Тестирование функций показа погоды** | | | | | | |
| 1 | Тестирование автосинхронизации погоды | Нажатие кнопки «Автосинхронизация погоды» | Получение погоды каждую минуту | Получение погоды каждую минуту | Прошел | Лопаткин Сергей 20.12.2022 |
| 2 | Тестирование получения погоды | Нажатие кнопки «Получить погоду» | Получение реальных данных погоды на экране | Получение реальных данных погоды на экране | Прошел | Лопаткин Сергей 20.12.2022 |
| 3 | Тестирование ГСЧ погоды | Нажатие кнопки «ГСЧ погоды» | Получение случайно возможных данных погоды на экране | Получение случайно возможных данных погоды на экране | Прошел | Лопаткин Сергей 20.12.2022 |
| **Тестирование функций справки** | | | | | | |
| 4 | Тестирование открытия окна справки | Нажать кнопку «Посмотреть справку» | Появление окна справки, где описаны возможности программы | Появление окна справки, где описаны возможности программы | Прошел | Лопаткин Сергей 20.12.2022 |
| 5 | Тестирование открытия окна о разработчике | Нажать кнопку «О программе» | Появление окна о разработчике | Появление окна о разработчике | Прошел | Лопаткин Сергей 20.12.2022 |

* 1. **Тестирование при помощи Unit-Test**

Для того, чтобы убедиться в правильности отображаемых данных, можно воспользоваться классом RandomNumbersGenerator (также, является простой вспомогательной функцией программы под видом ГСЧ), который, при разработке, был предназначен для тестирования правильного формирования отображаемых данных, взятых из OpenWeather API. Он включает в себя основные возможные значения для проверки верности отображения. Используя его, мы можем найти недочеты в отображении погодной информации.

Также, класс является полностью отлаженным в соответствии с логикой возможной погоды, так как он изначально являлся запланированной расширенной функцией для юнит-теста (не просто случайные числа).

Разработанные юнит-тесты проверяют каждое условие со всеми его возможными удовлетворениями. Каждому тестовому методу соответствует условие.

Результаты тестирования изображены на рисунке 20. Все успешно. Работает корректно.

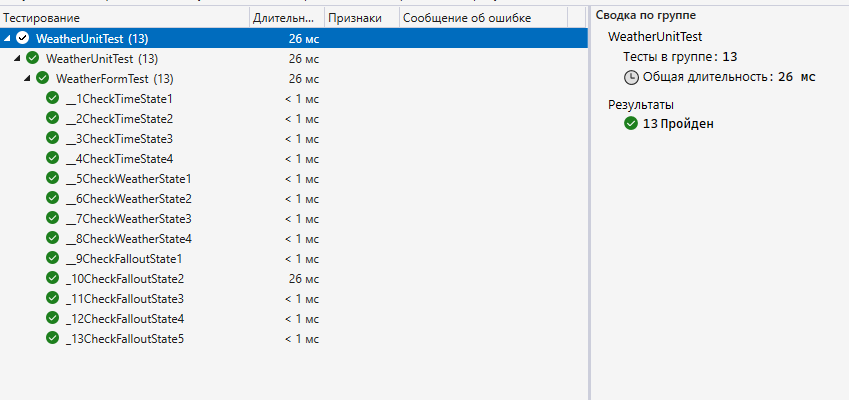


Рисунок 20 – Юнит-тестирование

Листинг кода юнит-тестов можно посмотреть в п. «ПРИЛОЖЕНИЕ А».

**Протоколы тестирования методов класса WeatherData**

Протоколы тестирования представлены в таблице 9.

Таблица 9 - Тестирование при помощи Unit-Test

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Тестер и разработчик:** Лопаткин Сергей | | | | | | |
| Unit Test № | Назначение | Начальные значения | Ожидаемый результат | Фактический результат | Успешность |
| 1 | Используется для проверки правильности условия в GetTimeState | 5 часов | TimeState = «Ночь» | TimeState = «Ночь» | Не успешно |
| 2 | Используется для проверки правильности условия в GetTimeState | 6 часов | TimeState = «Утро» | TimeState = «Утро» | Не успешно |
| 3 | Используется для проверки правильности условия в GetTimeState | 12 часов | TimeState = «День» | TimeState = «День» | Не успешно |
| 4 | Используется для проверки правильности условия в GetTimeState | 22 часа | TimeState = «Вечер» | TimeState = «Вечер» | Не успешно |
| 5 | Используется для проверки правильности условия в GetWeatherState | main = «Clear» | WeatherState = «Ясно» | WeatherState = «Ясно» | Не успешно |
| 6 | Используется для проверки правильности условия в GetWeatherState | main = «Clouds» | WeatherState = «Облачно» | WeatherState = «Облачно» | Не успешно |
| 7 | Используется для проверки правильности условия в GetWeatherState | main = «Mist» | WeatherState = «Атмосферное явление» | WeatherState = «Атмосферное явление» | Не успешно |
| 8 | Используется для проверки правильности условия в GetWeatherState | 1) main = «Rain»  2) main = «Snow»  3) main = «Drizzle»  4) main = «Thunderstorm» | 1) WeatherState = «Осадки»  2) WeatherState = «Осадки»  3) WeatherState = «Осадки»  4) WeatherState = «Осадки» | 1) WeatherState = «Осадки»  2) WeatherState = «Осадки»  3) WeatherState = «Осадки»  4) WeatherState = «Осадки» | Не успешно |
| 9 | Используется для проверки правильности условия в GetFalloutState | 1) temp = 273.15  description = «rain and snow»  2) temp = 273.15  description = «sleet»  3) temp = 273.15 – 25 | 1) FalloutState = «Дождь со снегом»  2) FalloutState = «Дождь со снегом»  3) FalloutState <> «Дождь со снегом» | 1) FalloutState = «Дождь со снегом»  2) FalloutState = «Дождь со снегом»  3) FalloutState <> «Дождь со снегом» | Не успешно |
| 10 | Используется для проверки правильности условия в GetFalloutState | 1) description = «heavy snow»  2) description = «snow» | 1) FalloutState = «Снежная буря»  2) FalloutState = «Снежная метель» | 1) FalloutState = «Снежная буря»  2) FalloutState = «Снежная метель» | Не успешно |
| 11 | Используется для проверки правильности условия в GetFalloutState | 1) description = «extreme rain»  2) description = «thunderstorm»  3) description = «shower rain» | 1) FalloutState = «Дождевой ливень»  2) FalloutState = «Дождевой ливень»  3) FalloutState = «Дождевой ливень» | 1) FalloutState = «Дождевой ливень»  2) FalloutState = «Дождевой ливень»  3) FalloutState = «Дождевой ливень» | Не успешно |
| 12 | Используется для проверки правильности условия в GetFalloutState | 1) description = «drizzle»  2) description = «rain» | 1) FalloutState = «Мелкий дождь»  2) FalloutState = «Мелкий дождь» | 1) FalloutState = «Мелкий дождь»  2) FalloutState = «Мелкий дождь» | Не успешно |
| 13 | Используется для проверки правильности условия в GetFalloutState | description = «clear sky» | FalloutState = «Отсутствует» | FalloutState = «Отсутствует» | Не успешно |

* 1. **Метрики ПО**

Метрики ПО следующие, с учетом добавления проекта Unit Test, как на рисунке 21.

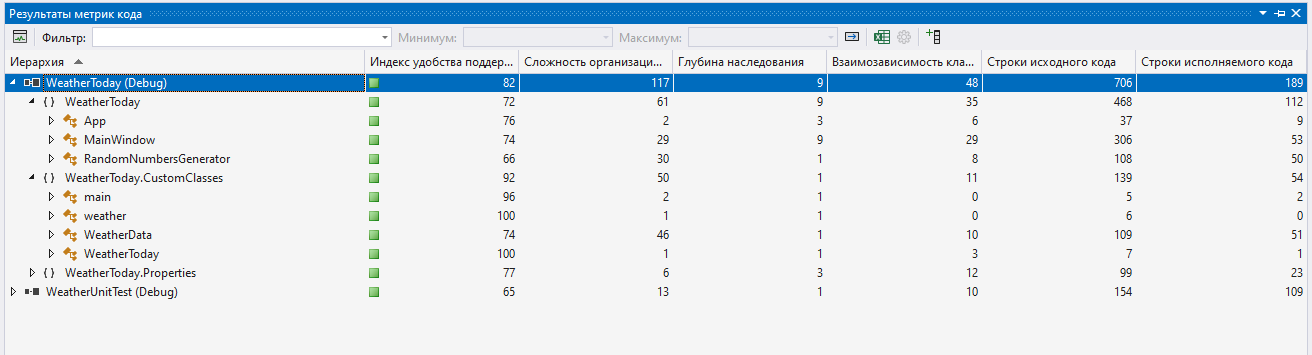


Рисунок 21 – Метрики ПО с проектом Unit Test

**ЗАКЛЮЧЕНИЕ**

При выполнении проекта по учебной практике удалось достигнуть все поставленные цели. Программа «Погода» реализована и работает корректно. Работают все основные функции исправно, а также добавлены дополнительные для удобства пользователя.

Приобретены навыки работы с API и файлами JSON с учетом возможностей ООП.

**ПРИЛОЖЕНИЕ А**

**Листинг Unit-Test**

**Листинг класса WeatherFormTest**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using WeatherToday;

using System;

using WeatherToday.CustomClasses;

namespace WeatherUnitTest

{

[TestClass]

public class WeatherFormTest

{

WeatherData \_weatherData;

DateTime currentTime = new DateTime();

[TestMethod]

public void \_\_1CheckTimeState1()

{

RandomNumbersGenerator.GetRandomWeather();

\_weatherData = RandomNumbersGenerator.WeatherRandom;

currentTime = new DateTime() + new TimeSpan(5, 0, 0);

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.TimeState == "Ночь");

}

[TestMethod]

public void \_\_2CheckTimeState2()

{

RandomNumbersGenerator.GetRandomWeather();

\_weatherData = RandomNumbersGenerator.WeatherRandom;

currentTime = new DateTime() + new TimeSpan(6, 0, 0);

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.TimeState == "Утро");

}

[TestMethod]

public void \_\_3CheckTimeState3()

{

RandomNumbersGenerator.GetRandomWeather();

\_weatherData = RandomNumbersGenerator.WeatherRandom;

currentTime = new DateTime() + new TimeSpan(12, 0, 0);

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.TimeState == "День");

}

[TestMethod]

public void \_\_4CheckTimeState4()

{

RandomNumbersGenerator.GetRandomWeather();

\_weatherData = RandomNumbersGenerator.WeatherRandom;

currentTime = new DateTime() + new TimeSpan(22, 0, 0);

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.TimeState == "Вечер");

}

[TestMethod]

public void \_\_5CheckWeatherState1()

{

RandomNumbersGenerator.GetRandomWeather();

\_weatherData = RandomNumbersGenerator.WeatherRandom;

\_weatherData.CurrentWeather.weather[0].main = "Clear";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.WeatherState == "Ясно");

}

[TestMethod]

public void \_\_6CheckWeatherState2()

{

RandomNumbersGenerator.GetRandomWeather();

\_weatherData = RandomNumbersGenerator.WeatherRandom;

\_weatherData.CurrentWeather.weather[0].main = "Clouds";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.WeatherState == "Облачно");

}

[TestMethod]

public void \_\_7CheckWeatherState3()

{

RandomNumbersGenerator.GetRandomWeather();

\_weatherData = RandomNumbersGenerator.WeatherRandom;

\_weatherData.CurrentWeather.weather[0].main = "Mist";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.WeatherState == "Атмосферное явление");

}

[TestMethod]

public void \_\_8CheckWeatherState4()

{

RandomNumbersGenerator.GetRandomWeather();

\_weatherData = RandomNumbersGenerator.WeatherRandom;

\_weatherData.CurrentWeather.weather[0].main = "Rain";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.WeatherState == "Осадки");

\_weatherData.CurrentWeather.weather[0].main = "Snow";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.WeatherState == "Осадки");

\_weatherData.CurrentWeather.weather[0].main = "Drizzle";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.WeatherState == "Осадки");

\_weatherData.CurrentWeather.weather[0].main = "Thunderstorm";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.WeatherState == "Осадки");

}

[TestMethod]

public void \_\_9CheckFalloutState1()

{

RandomNumbersGenerator.GetRandomWeather();

\_weatherData = RandomNumbersGenerator.WeatherRandom;

\_weatherData.CurrentWeather.main.temp = 273.15;

\_weatherData.CurrentWeather.weather[0].description = "rain and snow";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.FalloutState == "Дождь со снегом");

\_weatherData.CurrentWeather.weather[0].description = "sleet";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.FalloutState == "Дождь со снегом");

\_weatherData.CurrentWeather.main.temp = 273.15 - 25;

\_weatherData.GetWeather(currentTime);

Assert.IsFalse(\_weatherData.FalloutState == "Дождь со снегом");

}

[TestMethod]

public void \_10CheckFalloutState2()

{

RandomNumbersGenerator.GetRandomWeather();

\_weatherData = RandomNumbersGenerator.WeatherRandom;

\_weatherData.CurrentWeather.weather[0].description = "heavy snow";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.FalloutState == "Снежная буря");

\_weatherData.CurrentWeather.weather[0].description = "snow";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.FalloutState == "Снежная метель");

}

[TestMethod]

public void \_11CheckFalloutState3()

{

RandomNumbersGenerator.GetRandomWeather();

\_weatherData = RandomNumbersGenerator.WeatherRandom;

\_weatherData.CurrentWeather.weather[0].description = "extreme rain";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.FalloutState == "Дождевой ливень");

\_weatherData.CurrentWeather.weather[0].description = "thunderstorm";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.FalloutState == "Дождевой ливень");

\_weatherData.CurrentWeather.weather[0].description = "shower rain";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.FalloutState == "Дождевой ливень");

}

[TestMethod]

public void \_12CheckFalloutState4()

{

RandomNumbersGenerator.GetRandomWeather();

\_weatherData = RandomNumbersGenerator.WeatherRandom;

\_weatherData.CurrentWeather.weather[0].description = "drizzle";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.FalloutState == "Мелкий дождь");

\_weatherData.CurrentWeather.weather[0].description = "rain";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.FalloutState == "Мелкий дождь");

}

[TestMethod]

public void \_13CheckFalloutState5()

{

RandomNumbersGenerator.GetRandomWeather();

\_weatherData = RandomNumbersGenerator.WeatherRandom;

\_weatherData.CurrentWeather.weather[0].description = "clear sky";

\_weatherData.GetWeather(currentTime);

Assert.IsTrue(\_weatherData.FalloutState == "Отсутствует");

}

}

}

**ПРИЛОЖЕНИЕ Б**

**Листинг программы**

**Листинг главного окна**

<Window x:Class="WeatherToday.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:gif="http://wpfanimatedgif.codeplex.com"

xmlns:local="clr-namespace:WeatherToday"

mc:Ignorable="d"

Title="Weather Today" MinHeight="450" MinWidth="650" Height="450" Width="650" WindowStartupLocation="CenterScreen" Loaded="Window\_Loaded">

<DockPanel Background="#FFD3D3FF">

<Menu DockPanel.Dock="Top" FontSize="14">

<MenuItem x:Name="AutoSync" Header="Автосинхронизация погоды" IsCheckable="True" IsChecked="True" BorderBrush="Black" BorderThickness="0.5" ToolTip="Каждую минуту получает прогноз погоды из OpenWeather" Checked="AutoSync\_Checked" Unchecked="AutoSync\_Unchecked"/>

<MenuItem x:Name="GetToday" Header="Получить погоду" ToolTip="Позволяет получить информацию о реальной погоде в Рязанской области." Click="GetToday\_Click"/>

<MenuItem x:Name="GetRandom" Header="ГСЧ погоды" ToolTip="Позволяет получить случайные данные погоды, какие могли бы быть вообще." Click="GetRandom\_Click"/>

<MenuItem Header="Справка">

<MenuItem x:Name="Help" Header="Посмотреть справку" Click="Help\_Click"/>

<MenuItem x:Name="AboutProgram" Header="О программе" Click="AboutProgram\_Click"/>

</MenuItem>

</Menu>

<GroupBox Header="Погода" DockPanel.Dock="Top" Margin="3" FontSize="14" Background="#7FD6D6D6">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition/>

<ColumnDefinition Width="Auto" MinWidth="164"/>

</Grid.ColumnDefinitions>

<Border CornerRadius="3" BorderThickness="1" BorderBrush="Gray" Margin="5,5,5,5">

<Image x:Name="IconState" Width="120" Height="110" Source="http://openweathermap.org/img/wn/04d@2x.png"/>

</Border>

<Border CornerRadius="3" BorderThickness="1" BorderBrush="Gray" Margin="5,5,5,5" Grid.Column="1">

<StackPanel Margin="5">

<StackPanel Orientation="Horizontal">

<Label Content="Температура: "/>

<Label x:Name="Temperature" Content="N/A"/>

</StackPanel>

<Separator/>

<StackPanel Orientation="Horizontal">

<Label Content="Время суток: "/>

<Label x:Name="TimeState" Content="N/A"/>

</StackPanel>

<Separator/>

<StackPanel Orientation="Horizontal">

<Label Content="Состояние погоды: "/>

<Label x:Name="WeatherState" Content="N/A"/>

</StackPanel>

<Separator/>

<StackPanel Orientation="Horizontal">

<Label Content="Интенсивность осадков: "/>

<Label x:Name="FalloutState" Content="N/A"/>

</StackPanel>

</StackPanel>

</Border>

<Border CornerRadius="3" BorderThickness="1" BorderBrush="Gray" Margin="5,5,5,5" Grid.Column="2" ToolTip="Последнее время получения погоды">

<Label x:Name="LastTime" Content="00:00" HorizontalContentAlignment="Center" VerticalContentAlignment="Center" FontSize="34"/>

</Border>

</Grid>

</GroupBox>

<Border BorderBrush="LightGray" BorderThickness="2" Margin="3">

<Image x:Name="DisplayState" gif:ImageBehavior.AnimatedSource="/AnimStates/Clouds.jpg" gif:ImageBehavior.AutoStart="True" Stretch="UniformToFill"/>

</Border>

</DockPanel>

</Window>

**ПРИЛОЖЕНИЕ В**

**Листинг кода главного окна**

using System;

using System.Windows;

using System.Windows.Media.Imaging;

using System.Windows.Threading;

using WeatherToday.CustomClasses;

using WpfAnimatedGif;

namespace WeatherToday

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

Timer.Tick += Timer\_Tick;

Timer.Interval = new TimeSpan(0, 1, 0);

Timer.IsEnabled = true;

}

WeatherData WeatherToday = new WeatherData();

private void Timer\_Tick(object sender, EventArgs e)

{

WeatherToday.GetWeather();

IconState.Source = GetIconState(false);

GetDisplayState(false);

LastTime.Content = DateTime.Now.ToString("t");

Temperature.Content = WeatherToday.Temperature;

TimeState.Content = WeatherToday.TimeState;

WeatherState.Content = WeatherToday.WeatherState;

FalloutState.Content = WeatherToday.FalloutState;

}

DispatcherTimer Timer = new DispatcherTimer();

private BitmapImage GetIconState(bool isRandom)

{

BitmapImage imageState = new BitmapImage();

imageState.BeginInit();

if (!isRandom) imageState.UriSource = new Uri(WeatherToday.IconState);

else imageState.UriSource = new Uri(RandomNumbersGenerator.WeatherRandom.IconState);

imageState.EndInit();

ImageBehavior.SetAnimatedSource(DisplayState, imageState);

return imageState;

}

private void GetDisplayState(bool isRandom)

{

BitmapImage imageState = new BitmapImage();

imageState.BeginInit();

if (!isRandom) imageState.UriSource = new Uri(WeatherToday.DisplayPath, UriKind.RelativeOrAbsolute);

else imageState.UriSource = new Uri(RandomNumbersGenerator.WeatherRandom.DisplayPath, UriKind.RelativeOrAbsolute);

imageState.EndInit();

ImageBehavior.SetAnimatedSource(DisplayState, imageState);

}

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

Timer\_Tick(sender, e);

}

private void AutoSync\_Checked(object sender, RoutedEventArgs e)

{

Timer.IsEnabled = true;

}

private void AutoSync\_Unchecked(object sender, RoutedEventArgs e)

{

Timer.IsEnabled = false;

}

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

{

Timer\_Tick(sender, e);

}

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

{

RandomNumbersGenerator.GetRandomWeather();

IconState.Source = GetIconState(true);

GetDisplayState(true);

LastTime.Content = RandomNumbersGenerator.RandomTime.ToString("t");

Temperature.Content = RandomNumbersGenerator.WeatherRandom.Temperature;

TimeState.Content = RandomNumbersGenerator.WeatherRandom.TimeState;

WeatherState.Content = RandomNumbersGenerator.WeatherRandom.WeatherState;

FalloutState.Content = RandomNumbersGenerator.WeatherRandom.FalloutState;

}

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

{

MessageBox.Show("Программа позволяет выполнять следующее:\n" +

"- отображать информацию о погоде и времени суток в Рязанской области;\n" +

"- автосинхронизация получения погоды позволяет получить погоду на текущий момент каждую минуту;\n" +

"- состояние погоды отображается на картинках или GIF", "Справка", MessageBoxButton.OK, MessageBoxImage.Question);

}

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

{

MessageBox.Show("Разработчиком программы является студент группы ИСП-41 Лопаткин Сергей\n" +

"GitHub.Name=JohnHaval(Ранее - HaproBishop)", "О программе", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

}

**Листинг класса WeatherData**

using Newtonsoft.Json;

using System;

using System.IO;

using System.Net;

namespace WeatherToday.CustomClasses

{

public class WeatherData

{

public WeatherToday CurrentWeather { get; set; }

public string IconState { get => $"http://openweathermap.org/img/wn/{CurrentWeather.weather[0].icon}@2x.png"; }

public int Temperature { get; private set; }

public string TimeState { get; private set; }

public string WeatherState { get; private set; }

public string FalloutState { get; private set; }

public string DisplayPath { get; private set; }//Путь к GIF или картинкам

private string GetTimeState(DateTime currentTime)

{

if (currentTime.Hour >= 0 && currentTime.Hour <= 5) return "Ночь";

if (currentTime.Hour >= 6 && currentTime.Hour <= 11) return "Утро";

if (currentTime.Hour >= 12 && currentTime.Hour <= 17) return "День";

return "Вечер";

}

private string GetWeatherState()

{

string main = CurrentWeather.weather[0].main;

if (main == "Clear")

{

DisplayPath = "/AnimStates/Clear.jpg";

return "Ясно";

}

if (main == "Clouds")

{

DisplayPath = "/AnimStates/Clouds.jpg";

return "Облачно";

}

if (main == "Snow" || main == "Rain" || main == "Drizzle" || main == "Thunderstorm")

{

return "Осадки";

}

DisplayPath = "/AnimStates/Clouds.jpg";

return "Атмосферное явление";

}

private string GetFalloutState()

{

string description = CurrentWeather.weather[0].description;

if (CheckMediumTemperature() && (description.ToLower().Contains("rain and snow") || description.ToLower().Contains("sleet")))

{

DisplayPath = "/AnimStates/RainAndSnow.gif";

return "Дождь со снегом";

}

if (description.ToLower().Contains("snow") && description.ToLower().Contains("heavy"))

{

DisplayPath = "/AnimStates/HeavySnow.gif";

return "Снежная буря";

}

if (description.ToLower().Contains("snow"))

{

DisplayPath = "/AnimStates/Snow.gif";

return "Снежная метель";

}

if (description == "extreme rain" || description.ToLower().Contains("thunderstorm") || (description.Contains("shower") && description.Contains("rain")))

{

DisplayPath = "/AnimStates/ExtremeRain.gif";

return "Дождевой ливень";

}

if (description.Contains("rain") || description.Contains("drizzle"))

{

DisplayPath = "/AnimStates/Rain.gif";

return "Мелкий дождь";

}

return "Отсутствует";//Зависит от данных из OpenWeather

}

private bool CheckMediumTemperature()

{

Temperature = Convert.ToInt32(CurrentWeather.main.temp);

if (Temperature >= -1 && Temperature <= 2) return true;

return false;

}

/// <summary>

/// Отвечает за полное формирование погоды последовательно. Результаты в свойствах! Используется текущие дата и время.

/// </summary>

/// <returns></returns>

public void GetWeather()

{

GetWeatherToday();

GetWeather(DateTime.Now);

}

/// <summary>

/// Использовать для ГСЧ или самостоятельного формирования по дате.

/// </summary>

/// <param name="dateTime"></param>

public void GetWeather(DateTime dateTime)

{

TimeState = GetTimeState(dateTime);

WeatherState = GetWeatherState();

FalloutState = GetFalloutState();

}

private void GetWeatherToday()

{

WebRequest request = WebRequest.Create("https://api.openweathermap.org/data/2.5/weather?q=Ryazan&appid=cced05e23a8fd9054a38c1de61d639ea");

request.Method = "POST";

request.ContentType = "application/x-www-urlcoded";

WebResponse response = request.GetResponse();

string answer = "";//Answer of request

using (Stream stream = response.GetResponseStream())

{

using (StreamReader reader = new StreamReader(stream))

{

answer = reader.ReadToEnd();

}

}

response.Close();

CurrentWeather = JsonConvert.DeserializeObject<WeatherToday>(answer);

}

}

}

**Листинг класса WeatherToday**

using Newtonsoft.Json;

namespace WeatherToday.CustomClasses

{

public class WeatherToday

{

[JsonProperty("base")]

public string Base;

public main main;

public weather[] weather;

}

}

**Листинг класса main**

namespace WeatherToday.CustomClasses

{

public class main

{

private double \_temp;

public double temp { get => \_temp; set => \_temp = value - 273.15; }

}

}

**Листинг класса weather**

namespace WeatherToday.CustomClasses

{

public class weather

{

public string main;

public string description;

public string icon;

}

}

**Листинг класса RandomNumbersGenerator**

using System;

using WeatherToday.CustomClasses;

namespace WeatherToday

{

public static class RandomNumbersGenerator

{

public static WeatherData WeatherRandom = new WeatherData();

public static DateTime RandomTime { get; private set; }

private static string[] WeatherStates { get => GetWeatherStates(); }

private static string[] GetWeatherStates()

{

var weatherStates = new string[7];

weatherStates.SetValue("Clear", 0);

weatherStates.SetValue("Clouds", 1);

weatherStates.SetValue("Rain", 2);

weatherStates.SetValue("Snow", 3);

weatherStates.SetValue("Drizzle", 4);

weatherStates.SetValue("Thunderstorm", 5);

weatherStates.SetValue("Other", 6);//Может быть любое значение для алгоритма вычисления

return weatherStates;

}

private static string[] FalloutStates { get => GetFalloutStates(); }

private static string[] GetFalloutStates()

{

var falloutStates = new string[5];

falloutStates.SetValue("heavy snow", 0);

falloutStates.SetValue("Snow", 1);

falloutStates.SetValue("thunderstorm", 2);

falloutStates.SetValue("rain", 3);

falloutStates.SetValue("rain and snow", 4);

return falloutStates;

}

public static int GetRandom(int min, int max)

{

Random rnd = new Random();

return rnd.Next(min, max);

}

/// <summary>

/// Производит формирование всех результативных значений в WeatherRandom свойстве. Основа работы - WeatherData. Смотреть там работу.

/// </summary>

public static void GetRandomWeather()

{

RandomTime = new DateTime() + new TimeSpan(GetRandom(0, 24), GetRandom(0, 60), 0);

IntoWeatherRandom();

WeatherRandom.GetWeather(RandomTime);

}

/// <summary>

/// Производит рандомное занесение основной (необходимой) информации для работы формирования значений в классе WeatherData.

/// </summary>

private static void IntoWeatherRandom()

{

int temp = GetRandom(-30, 30) + Convert.ToInt32(273.15);//Необходимо добавить значение по Клельвину для правильности отображения (Свойство минусует значение на получении. ООП)

WeatherRandom.CurrentWeather = new CustomClasses.WeatherToday();

WeatherRandom.CurrentWeather.main = new main();

WeatherRandom.CurrentWeather.main.temp = temp;

temp = Convert.ToInt32(WeatherRandom.CurrentWeather.main.temp);

string rndWeatherState = WeatherStates[GetRandom(0, 7)];

WeatherRandom.CurrentWeather.weather = new weather[1];

WeatherRandom.CurrentWeather.weather.SetValue(new weather(), 0);

WeatherRandom.CurrentWeather.weather[0].main = rndWeatherState;

if (rndWeatherState != "Other" && rndWeatherState != "Clear" && rndWeatherState != "Clouds")

{

string rndFalloutState = FalloutStates[4];//"rain and snow"//Используется для удовлетворения правильности рандома

if (temp < -1 || temp > 2)

{

if (rndWeatherState.ToLower().Contains("snow"))

{

if (RandomTime.Hour >= 0 && RandomTime.Hour <= 5)

WeatherRandom.CurrentWeather.weather[0].icon = "13n";

else WeatherRandom.CurrentWeather.weather[0].icon = "13d";

rndFalloutState = FalloutStates[GetRandom(0, 2)];

}

else

{

if (RandomTime.Hour >= 0 && RandomTime.Hour <= 5)

WeatherRandom.CurrentWeather.weather[0].icon = "10n";

else WeatherRandom.CurrentWeather.weather[0].icon = "10d";

rndFalloutState = FalloutStates[GetRandom(2, 4)];

}

}

else

{

if (RandomTime.Hour >= 0 && RandomTime.Hour <= 5)

WeatherRandom.CurrentWeather.weather[0].icon = "09n";

else WeatherRandom.CurrentWeather.weather[0].icon = "09d";

}

WeatherRandom.CurrentWeather.weather[0].description = rndFalloutState;

}

else

{

if (rndWeatherState == "Clear")

{

if (RandomTime.Hour >= 0 && RandomTime.Hour <= 5)

WeatherRandom.CurrentWeather.weather[0].icon = "01n";

else WeatherRandom.CurrentWeather.weather[0].icon = "01d";

}

if (rndWeatherState == "Clouds")

{

if (RandomTime.Hour >= 0 && RandomTime.Hour <= 5)

WeatherRandom.CurrentWeather.weather[0].icon = "04n";

else WeatherRandom.CurrentWeather.weather[0].icon = "04d";

}

if (rndWeatherState == "Other")

{

if (RandomTime.Hour >= 0 && RandomTime.Hour <= 5)

WeatherRandom.CurrentWeather.weather[0].icon = "50n";

else WeatherRandom.CurrentWeather.weather[0].icon = "50d";

}

WeatherRandom.CurrentWeather.weather[0].description = "Отсутствует";

}

}

}

}