Министерство образования Республики Беларуси

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных систем и сетей  
Кафедра Информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему: "Журналирование процессов и файловой системы в ОС "Windows" по сетевым протоколам"

Дисциплина: Операционные системы и среды (ОСиС)

Выполнил:  
Студент гр. 653505  
Сташинский Г.В.

Проверила:  
ассистент кафедры информатики  
Шнейдер В.

**Содержание**

Введение 3

1. Общие характеристики проекта 5

1.1. Назначение 5  
1.2. Область действия 5  
1.3. Сокращения, обозначения 5  
1.4. Аналоги 6  
1.4. Краткий обзор 6

2. Общее описание 8

2.1. Взаимодействие продукта 8

2.2. Функции продукта 9

2.3. Характеристики пользователя. 11

2.4. Ограничения. 12

3.Инструменты разработки 13

3.1. Мотивация выбора инструментов 13

3.2. Язык С# и платформа .NET Standard 13

3.3. Windows Presenation Foundation (WPF) 14

3.4. Material Design In XAML (MDIX) 16   
4.Программная реализация 18

4.1. Бибилотека System Remote Logger (SRL) API 18   
 4.2. Windows-приложение System Remote Logger (SRL) Cleint 22

5.Демонстрация работы программы 25

Заключение. 29

Использованная литература. 31

**Введение**

Современные компьютерные системы требуют постоянного мониторинга состояния для избежания различных ошибок. К таким ошибкам можно отнести принудительные остановки процессов ОС, скорость работы и многое другое. Механизмы для журанлирования во многом похожи и позволяют производить мониторинг прямо «на месте», например, системным администратором. Существуют также механизмы, которые позволяют проводить журналирование и мониторинг удаленно, однако, зачастую такие системы требуют большого количества оборудования, навыков и, иногда, избыточны в функционале. Например, логгирование состояния СУБД Oracle требует как минимум глубоких навыков администрирования БД. Еще одной проблемой, которая может возникнуть, является отсутствие нужных систем. Например, СУБД Oracle требует собственного SMTP-cервера для отправки сообщений конечному пользователю.

В данной работе поставлена задача разработки системы удаленного журналирования процессов и файловой системы ОС Windows (а так же Linux посредством API), которая позволит без особых навыков системного администрирования задавать журналируемые события и конфигурировать подписчиков и инструменты их оповещения в конфигурационных файлах. В данной работе будут представлены следующие инструменты для оповещения: оповещение по локальной сети (посредством UDP-протокола), а так же отправка сообщений на почту посредством SMTP.

Особенностью данной системы является простота. В частности, она не требует наличия собственного SMTP-cервера. Это важно, например, для организаций, которые не обладаю большим бюджетом или вообще не связаны с автоматизацией процессов и чем-то подобным. Такие организации могут установить в качестве исходящего почтового ящика любой существующий ящик на популярных сервисах: Gmail, Yandex.Mail, Yahoo и т.д. Главное условие – наличие открытого доступа к серверу. Второй особенностью данной системы является, например, возможность широковещательной рассылки клиентам по локальной сети. Такая система удобна, например, при командной разработке ПО или при администрировании массива компьютерных систем сразу на нескольких машинах. Следует отметить, что современные процессы разработчки программного обеспечения включают многочисленные этапы, которые позволяют не только ускорять процесс разработки, но и обеспечить стабильность и корректность работы ПО. Такими этапами являются логгирование, удаленные и локальные репозитории, тестирование и многое другое. Однако, существуют ситуации, которые не покрыты тестами и они выявляются в процессе локальной разработки. Такие ошибки анализируются прямо на месте (в окне браузера, в лог-файлах), однако такой подход не является удобным. Именно для таких нужд может и потребоваться

**1. Общие характеристики проекта**

1.1. Назначение

Основной областью применения программного продукта, разработанного в рамках данной работы, является логгирование состояния операционной системы и её внутренних процессов в рамках локальной сети посредством протокола UDP, а также удаленно посредством рассылки журналов логгирования на электронную почту через SMTP. Данное решение может применятся при отладке кода, удаленной и локальной диагностике состояния компьютера/системы компьютеров.

1.2. Область действия

В данной работе библиотека реализована в качестве готовой кроссплатформенной .NET Standard Library сборки в виде API-сервисов, поэтому решение может быть использовано как на ОС Windows, так и на Linux (клиентское UI приложение доступно под ОС Windows, терминальный клиент доступен через dotnetcore в Linux и Windows). Следует отметить, что клиентская часть на WPF может быть перекомпилирована в клиент на ОС Linux с выходом .NET Core 3.

Логгирование в локальной сети c широковещательной рассылкой может быть полезно в закрытых системах, в которых нет доступа к сети Интернет, однако крайне важно наличие системы диагностики. К таким закрытым системам можно причислить лаборатории компаний, в которых изучаются отдельные вредоносные программы (выход в глобальную сеть отсутствует в целях безопасности), а также организации или предприятия с большим количеством взаимосвязанного оборудования, которые соединены через локальную сеть.

Логгирование через рассылку электронных журналов на электронную почту также представляет интерес, т.к. журналы могут рассылаться на независимые почтовые сервера и в случае повреждения компьютера или сети компьютеров данные не будут утеряны и можно будет провести диагностику. Также подход удобен для получения оперативной инфомрации о состоянии системы, например при атаке посредством сети (например, получение оперативной информации о нагрузке на процессор или оперативную память во время DDos-атаки).

1.3. Определения, сокращения

Application Programming Interface (API) - программный интерфейс приложения,  описание способов которыми одна компьютерная программа может взаимодействовать с другой программой. Может быть реализована как готовая сборка, так и позволять работать посредством сетевых протоколов.

System Remote Logger – удаленный системный журналист (логгер). Название продукта, реализованного в рамках данной работы.

System Remote Logger (SRL) API – API-сервисы, реализованные в рамках данной работа. Представлены в качестве .NET Standard сборки SystemRemoteLogger.Services. Для уточнения протокола взаимодействия со сборкой можно обратиться к тестам SystemRemoteLogger.Services.Tests.

1.4. Аналоги

Т.к. двумя ключевыми особенностями данного ПО является имено рассылка по локальной сети и по почте, но разделим аналоги по двум катеориям:

* По почте:
  + СУБД Oracle: позволяет производить рассылку сообщений всем подписчикам. Основной недостаток: сложность конфигурирование и обязательные технические навыки по работе с данной СУБД
  + Библиотека Logging (Python): позволяет логгировать действия на рантайме в Python в файл или по почте. Основной недоcтаток: работает только с скриптами Python, нельзя запускать на уровне приложения ОС
* По рассылке по локальной сети:
  + Аналогов рассылки именно журналов ОС не было обнаружено, однако существуют различные инструменты по рассылке файлов по локальной сети. Из недостатков: слишком шировкая область применения и отстутствует гибкая система конфигурации

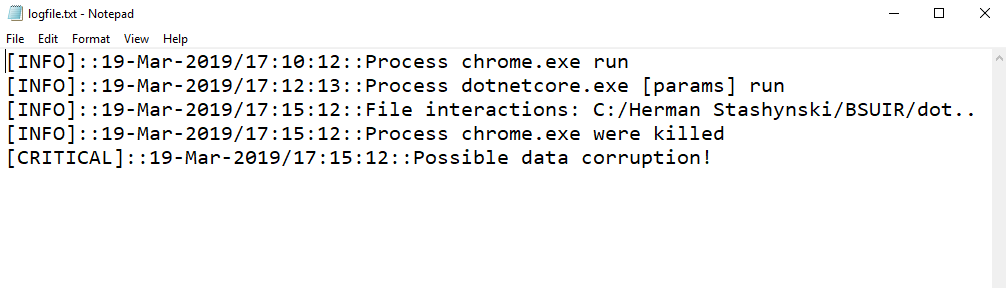
1.5. Краткий обзор

Данное решение состоит из 2 основных компонент:

* API-сервисов в виде сборки **SystemRemoteLogger.Services**
* WPF-клиента в виде сборки **SystemRemoteLogger.WPF**

Также к данной работе прилагаются Unit и интеграционные тесты в сборке **SystemRemoteLogger.Services.Tests**. Они позволят наглядно разобрать как устроен SRL API, а также гарантируют исправность работы кода на данных тестовых случаях.

Результатом работы являются текстовые сообщения (для поддерживания актуальной информации и обеспечения стабильности передаваемых данных) с системной информацией, которые присылаются по локальной сети всем подписчиками, файлы с журналами диагностики на электронной почте. Данная информация форматируется как стандартыне журналы с лог файлами. Ниже представлен возможный пример лог-файла, который присылается по почте:

  
Рис. 1.1. Пример лог-файла присылаемого на почтовый сервер

Клиентская часть реализована при помощи библиотеки MDIX (Material Design In XAML), которая позволяет реализовать Material Design компоненты для .NET приложений.

**2. Общее описание**2.1 Взаимодействие продукта (с другими продуктами и компонентами)

Данная система сервисов включает в себя ряд зависимостей, как низкоуровневых, так и высокоуровневых, которые позволяют реализовывать определенные задачи.

В частности, для получения информации о состоянии системы используются как высокоуровневые инструменты платформы .NET (FileSystemWatcher, Process), так и низкоуровневые запросы к ОС посредством языка запросов. Также используются инструменты WinAPI.

Говоря о низкоуровневых компонентах, стоит упомянуть реализацию сетевого общения в данном ПО:

1. UDP Multicast. В данной программе для реализации широковещательной трансляции всем подписчикам в локальной сети используются низкоуровневые инструменты, такие как System.Net.Sockets.UdpClient. Сборка System.Net.Sockets представляет из себя обертку, которая предоставляет базовый функционал для создания веб-сокетов. Сама сборка работает с неуправляемым кодом и создает соединения, которые нужно высвобождать. Однако, часть забот берет на себя платформа .NET.
2. SMTP. В данной работе журналирование посредством электронной почты реализовано на уровне схожего с предыдщим пунктом инструмента: System.Net.Mail.SmtpClient. Данная сборка имеет наименьший возможный функционал из доступных на рынке для .NET. Альтернатива – использование MailKit, MimeKit (атрибутировано Obsolete компанией Microsoft).

Также следует поговорить и высокоуровневых компонентах для создания интерфейса. Таким инструментом сдал MDIX (Material Design In XAML). Данный фреймворк позволяет создавать полноценные десктопные приложения в соответствии с Google Material Design. Это решение включает в себя не только систему визуальных стилей для стандартных компонентов XAML, но и анимации, которые применяются в Google Material Design.

Сам же итотовый продукт данной работы представляет из себя (говоря именно о ядре проекта) – сборку из API-сервисов «System Remote Logger (SRL) API». Данная сборка легко интегрируется в любые решения и полностью соотвествует принципам SOLID.

2.2 Функции продукта

В рамках кратакого обзора и описания проекта были перечислены основные направления, в рамках которого осуществляется деятельность данного проекта.

Говоря о функциях проекта, следует говорить о предоставленном сборкой SystemRemoteLogger.Services API, т.к. его методы раскрывают основные функции и цели данного проекта.

Отметим, что в данной таблице будут перечислены не все методы и сервисы, предоставленные в данной библиотеке. Данная ставит перед собой цель предоставить представление о возможностях данного API, которые выражают функции данного проекта:

|  |  |  |
| --- | --- | --- |
| MailSender [IMailSender] | Данный сервис ответственен за отправку сообщений по протоколу SMTP | SendMail представляет из себя обработчик для события сервиса RemoteLoggingService. Отправляет сообщения в соответствии с именем хоста и порта.  Сервис требует информацию для конфигурирования SMTP Client, а также экземпляр провайдера конфигурации. |
| UdpConnectionService | Данный сервис предоставляет возможность логгирования сообщений в поток передачи UDP Multicast | Обладает таким же методом SendMail, создан для тех же целей. Внутри также присутствует метод Сonnect для установки соединения.  Сервис подавляет возникшие в процессе работы исключения и логгирует эут ифнормацию в поток журнала (по почте или в UDP Multicast). |
| RemoteLoggingService | Является ключевым Core элементом данного проекта. | Соедержит набор из делегата EncryptedDataEventHandler и события NewMessageOn. Данный подход позволяет решить проблему дублирования кода и сделать код слабосвязным.  Основным методом данного сервиса является асинхронный метод-слушатель Start. Данный метод начинает «прослушивать» заданную в ConfigurationProvider директорию и исходя из происходящих событий отправляет информацию его подписчикам посредстом почты или UDP Multicast |
| ConfigurationProvider | Сервис для работы с конфигурационными данными | Является .NET Standard сборкой, поэтому реализация ConfigurationManager предоставляется NuGet и из коробки поддерживает лишь считывание данных (требуется для обратной совместимости).  Предоставляет набор публичных свойств-геттеров для получения информации о конфигурации приложения, свойствах сервисов и т.п.  Сервис получает ифнормацию с уровня предсталвения (Presentation Layer). |
| EncryptionHelper | Stateless-сервис, используемый в качестве инструмента для преобразования строковых данных в массив байт (для последующей передачи по UDP) и наоборот | Предоставляет публчиные методы Encode и Decode. Входные и выходные данные соответствуют примитиву string и массиву байт. |

Исходя из вышеперечисленного, функциями данного проекта являются глубокая поддержка API-сервисов для журналирования заранее определенных событий, происходящих в рамках ОС Windows и Linux. Вторичной функией данного проекта стоит демонстрация возможностей API посредством клиента для ОС Windows, построенного при помощи актуальной технологии MDIX. Именно демонстрация посредством визуального клиента может углубить понимание целей и функций этого программного продукта.

* 1. Характеристики пользователя

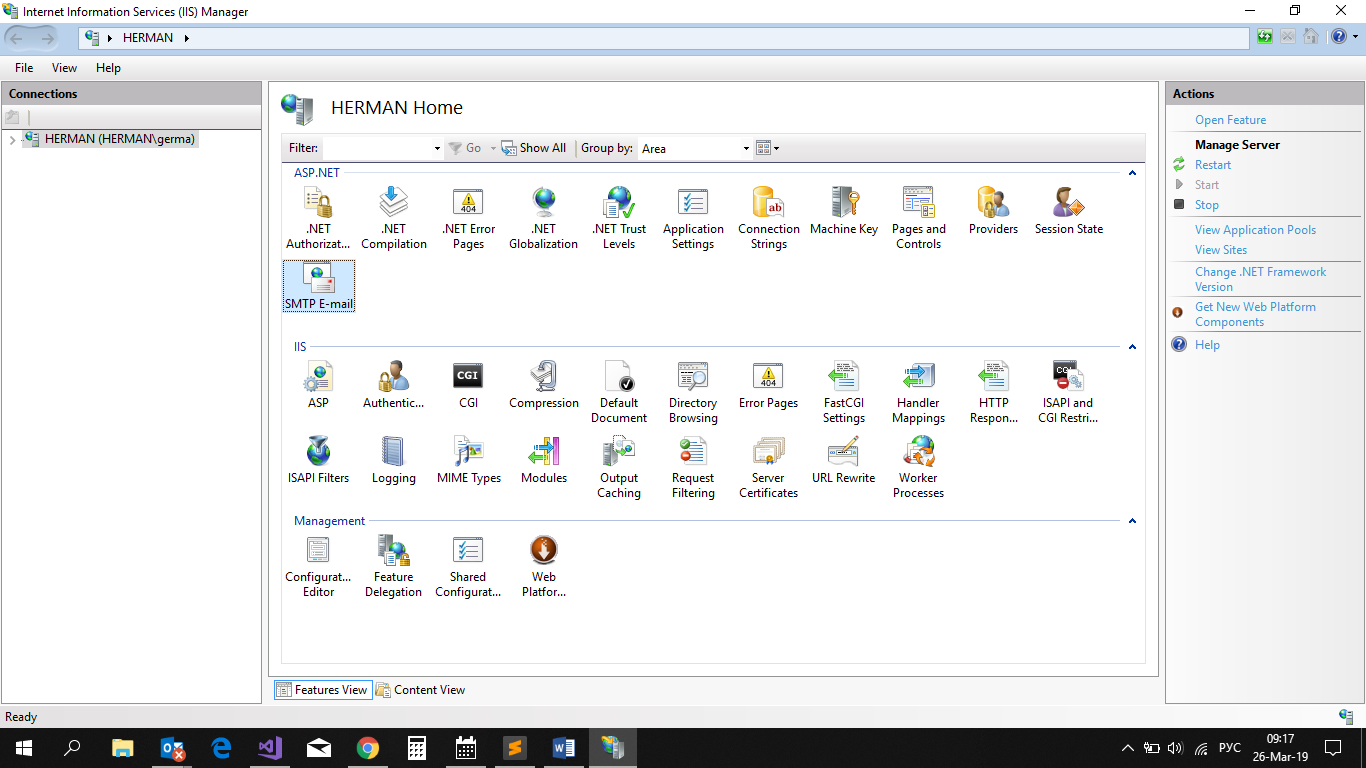
Целевая аудитория данного сервиса – разработчики, системные администраторы, пользователи ПК с большим опытом работы с ОС Windows и Linux.

В особенности следует выделить именно группу разработчиков и администраторов систем, так как данный инструмент позволяет устранять системные ошибки различного уровня на реальных проектах во время разработки, а также для поддержки уже запущенных веб/настольных приложений.

Минимальные требования к пользователю: знание понятий операционная система, процессы в ОС, файловая система, журналирование/логгирование, компьютерные сети; опыт работы с ОС Windows/Linux

Дополнительные требования (для команд разработки и системных администраторов): опыт работы с компьютерными сетями, понимание протокола UDP, опыт работы с почтовыми серверами и локальными сетями.

Затрагивая дополнительные требования, опыт работы с почтовыми серверами может быть разнообразным. Одним из возможных вариантов – использование стандартного (предустановленного) в ОС Windows сервера IIS (Internet Information Services) (отметим, что данный возможности сервера являются доступными после его активации в «Панели управления). Данный сервер представлен здесь как рекомендуемый, так как предоставялет из коробки возможности для работы с почтой. На рис. 2 представлен скриншот «домашней страницы» IIS с выделенным пунктом для работы с SMTP E-mail.

  
Рис. 2.1. Сервер IIS (Internet Information Services)

* 1. Ограничения

Данный продукт не рекомендуется использовать на системах, отличных от ОС Windows и ОС Ubuntu, т.к. даная библиотека была протестирована именно на данных операционный системах. Однако, отметим, что 2 данные системы покрывают огромную долю рынка ПК.

Данное ПО было протестировано на:

* ОС Windows 10 Home 1803
* ОС Ubuntu 18.04

Также не желательно использование данного ПО пользователями, которые не удовлетворяют требованиям, описанным выше (см. пункт 2.3. Характеристики пользователя).

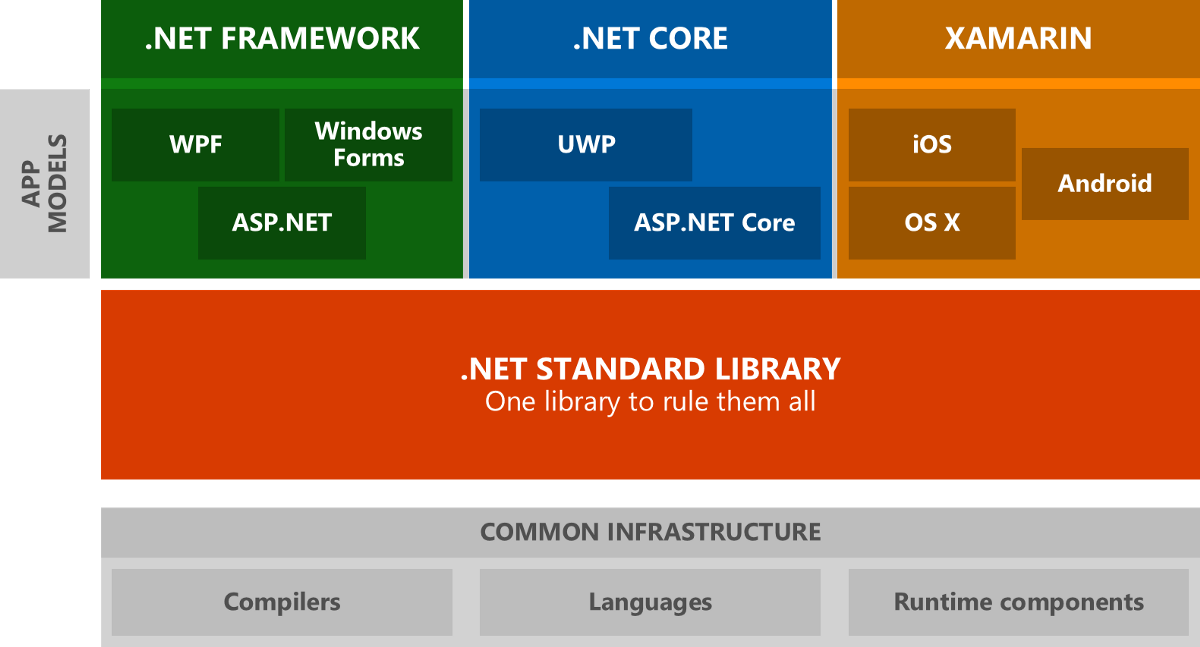
1. **Инструменты разработки**

3.1 Мотивации выбора инструментов

Основными мотивациями для выбора той или иной технологии при разработке программного обеспечения являются скорость разработки, порог вхождения, оптимизации языков и платформ под работу с целевыми ОС.

Для решения поставленных в данной курсовой работке задач были выбраны следующие технологии:

1. Язык C# в составе платформы .NET Core и .NET Standard.
2. Технология Windows Presentation Foundation (WPF) в составе .NET Framework.
3. Библиотека для создания интерфейса Material Design для WPF

  
Рис 3.1. Соотношение .NET Core, .NET Standard и .NET Framework, используемых в проекте

Также одной из причин выбора стала поддержка низкоуровневых инструментов для сетевого программирования (TCP/IP для UDP-мультикастинга, HTTP, SMTP), поддержка инструментов для работы с конфигурационными файлами (XML, JSON) и т.д.

3.2 Язык C# и платформа .NET Standard

Для реализации данных задач был выбран язык C#, ввиду полной поддержки .NET Standard. Среда разработки – Visual Studio 2017 Community. Данная среда является оптимальной, ввиду полной совместимости со всеми актуальными версиями платформы .NET Framework и последними стандартами языка C#.

Особенностью платформы .NET является независимость от целевой платформы и, следовательно, возможность запускать программу под любую ОС и любой процессор, который поддерживает работу с .NET CLR (clr.dll).

Также данная платформа поддерживает работу с пакетами посредством менеджера пакетов NuGet.

.NET Framework — это зрелая и полнофункциональная платформа, которая поставляется вместе с Windows. Экосистему .NET Framework разработчики используют уже много лет. Она поддерживает разработку современных приложений и обладает высочайшей совместимостью с существующими приложениями и библиотеками.

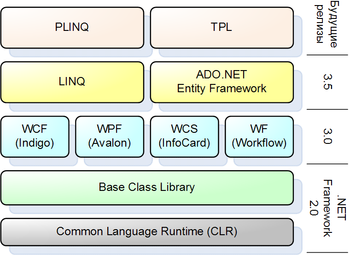
Платформа .NET — это совокупность всех средств разработки, распространения и выполнения приложений .NET, включая операционные системы, серверы, сервисы, стандарты, спецификации и документацию.

3.3 Windows Presentation Foundation (WPF)

Для создания программного интерфейса приложения были исследованы следующие варианты: Windows Forms, WPF, Qt (C++). Однако именно технология Windows Presentation Foundation удовлетворяла всем требованиям применяемым к разработке ПО.

Одной из причин использования платформы .NET Framework стало наличие технологии WPF с поддержкой языка разметки XAML. Данная технология позволяет создавать десктопные приложения для ОС Windows посредством языка C# с поддержкой очень гибких возможностей для настройки интерфейса.

WPF предустановлена в [Windows Vista](https://ru.wikipedia.org/wiki/Windows_Vista) ([.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework) 3.0), [Windows 7](https://ru.wikipedia.org/wiki/Windows_7) ([.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework) 3.5 SP1), [Windows 8](https://ru.wikipedia.org/wiki/Windows_8)(.NET Framework 4.0 и 4.5), [Windows 8.1](https://ru.wikipedia.org/wiki/Windows_8.1) (.NET Framework 4.5.1). С помощью WPF можно создавать широкий спектр как автономных, так и запускаемых в [браузере](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) приложений.

  
Рис. 3.2. WPF в составе .NET Framework

Для работы с WPF требуется любой [.NET](https://ru.wikipedia.org/wiki/.NET_Framework)-совместимый язык. В этот список входит множество языков: [C#](https://ru.wikipedia.org/wiki/C_Sharp), [F#](https://ru.wikipedia.org/wiki/F_Sharp), [VB.NET](https://ru.wikipedia.org/wiki/VB.NET), [C++](https://ru.wikipedia.org/wiki/Visual_C%2B%2B), [Ruby](https://ru.wikipedia.org/wiki/IronRuby), [Python](https://ru.wikipedia.org/wiki/IronPython), [Delphi (Prism)](https://ru.wikipedia.org/wiki/Oxygene_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [Lua](https://ru.wikipedia.org/w/index.php?title=Lua.NET&action=edit&redlink=1) и многие другие. Для полноценной работы может быть использована как [Visual Studio](https://ru.wikipedia.org/wiki/Visual_Studio), так и [Expression Blend](https://ru.wikipedia.org/wiki/Expression_Blend). Первая ориентирована на программирование, а вторая — на дизайн и позволяет делать многие вещи, не прибегая к ручному редактированию [XAML](https://ru.wikipedia.org/wiki/XAML). Примеры этому — анимация, стилизация, состояния, создание элементов управления и так далее.

Каждое окно .NET WPF представляет собой экземпляр класса-наследника Window, который является частью библиотеки PresentationFramework.dll. При добавлении нового окна в проект автоматически создаются 2 файла по следующим правилам:

* Файл для разметки GUI при помощи языка XAML с расширением .xaml для декларативного описания окна
* Файл для описание окна посредством кода с расширением .xaml.cs

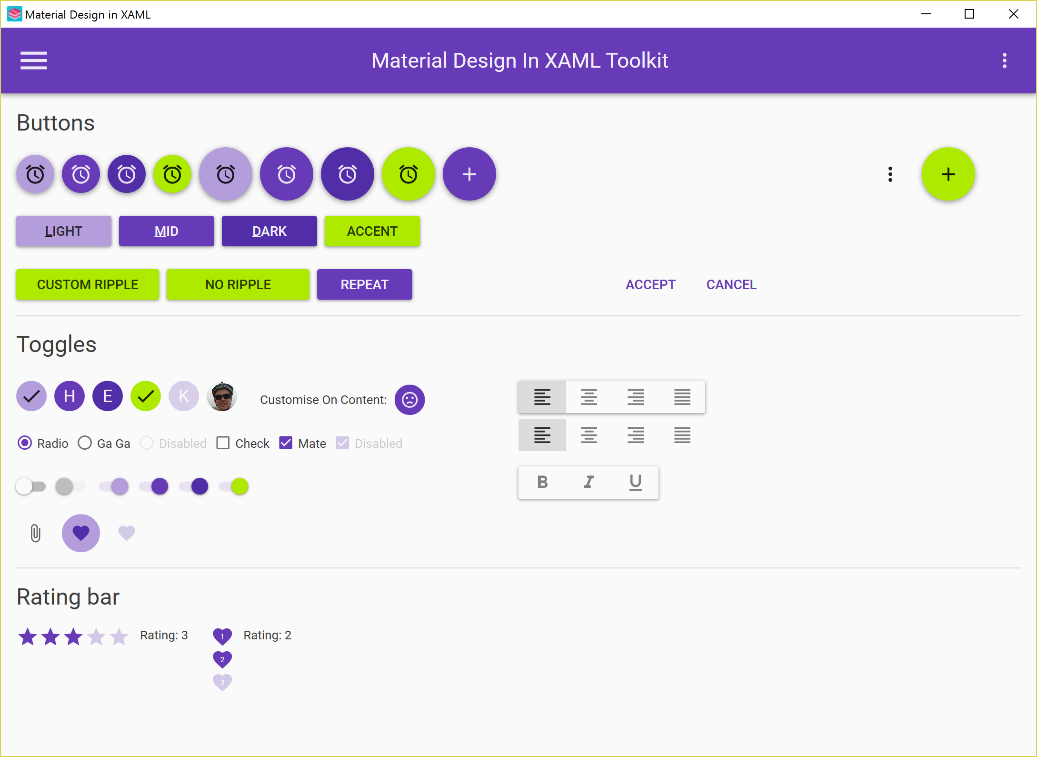
В файле с кодом на языке С# хранятся подключаемые пространства имен, обработчики событий, отдельные служебные классы, однако, следует отметить, что вся логика взаимодействия с окном должна находится в классе окна. Также для реализации некоторых паттерном (MVVM и др.) cледует воздержаться от написания кода в этом файле и уделить внимание файлу XAML.

Windows Presentation Foundation (WPF) — система для построения клиентских приложений [Windows](https://ru.wikipedia.org/wiki/Windows) с визуально привлекательными возможностями взаимодействия с пользователем, графическая (презентационная) подсистема в составе [.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework) (начиная с версии [3.0](https://ru.wikipedia.org/wiki/.NET_Framework_3.0)), использующая язык [XAML](https://ru.wikipedia.org/wiki/XAML).

WPF поддерживается IoC-контейнером Ninject, который применяется для разрешения зависимостей, что и использовалось в данном проекте.

* 1. Material Design In XAML (MDIX)

Для реализации программного интерфейса использовалась библиотека Material Design In XAML (MDIX). Данный проект явялется Open Source проектом с богатым набором инструментов и компонентов для реализации WPF-приложения.

  
Рис. 3.3. Компоненты MDIX

По-умолчанию приложение конфигурируется в App.xaml соотвествующими ссылками на ресурсы, а также на уровне окон приложения:  
<ResourceDictionary.MergedDictionaries>  
 <ResourceDictionary Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.Light.xaml" />  
</ResourceDictionary.MergedDictionaries>

В рамках предудыщей курсовой работы (дисциплина АВС, 5 семестр) для реализации клиентской части использовалась библиотека Metro UI, которая использует аналогичный подход к конфигурированию визуальных компонентов и ресурсов ПО.

Одним из преимуществ этой библиотеки является также и поддержка единых стилей во всем приложении, как того и требуют гайдлайны Material Design. Ввиду этого в приложении легко можно настроить тему из файла App.config (темную, светлую, выбрать Primary Color и т.д.).

1. **Программная реализация**4.1 Библиотека System Remote Logger (SRL) API

Центральным, корневым звеном приложения является библиотека System Remote Logger (SRL) API, которая входит в состав сборки SystemRemoteLogger.Services. Данная сборка представляет из себя .NET Standard библиотеку, которая совместима как с .NET Framework, так и с .NET Core. Ввиду поддержки этих двух стандартов платформы .NET, библиотека может работать как в ОС Windows, так и в Linux.

Основными сервисами в данной библоитеке стали:

* MailSender
* UdpConnectionService
* RemoteLoggingService
* MailService
* Побочные сервисы-хелперы:
  + ConfigurationProvider
  + EncryptionHelper
* А также опредления их интерфейсов

Так как основной особенностяью проекта является именно работа с компьютерными сетями, то рассмотрим детально реализацию работы с стевыми протоколами:

MailSender – сервис, ответственный за работу с протоколом SMTP. Данный сервис конфигурируется с параметрами SMTP-cервера, такими как: порт (например, 587), хост (smtp.yandex.ru) и данными для аутентикации пользователя (логин, почта, и пароль), которые предоставляются посредством провайдера конфигурации ConfigurationProvider:

Для отправки почты в среде интернет используется протокол SMTP (Simple Mail Transfer Protocol). Данный протокол указывает, как почтовые сервера взаимодействуют при передаче электронной почты. Для работы с протоколом SMTP и отправки электронной почты в .NET предназначен класс SmtpClient из пространства имен System.Net.Mail. Центральной сущностью, которая описывает сам объект письма является MailMessage. Она содержит ряд свойств, которые позволяет максимально удобно и подробно описывать все нужные нам свойства.

public MailSender(int port, string host, IConfigurationProvider provider)

{

\_port = port;

\_host = host;

\_provider = provider;

}

Следует отметить, что можно настроить работу данного сервиса под собственный сервер предоставляя такие данные. Сообещения, которые формирует данный сервис являются HTML-разметкой. По умолчанию все данные сконфигурированы и готовы к работе без дополнительных действий пользователя. Такими данными являются подпись (signtature) и стили подписи, текст сообщения, текст заголовка и т.п. Сервис выставляет публичны асинхронный метод SendMail и событие, которое использует данный метод - SendMessage. При возникновении каких-либо специфичных ошибок при работе с API, библиотека генерирует кастомное исключение MailLoggingException.

UdpConnectionService – сервис, ответственный за работу с протоколом UDP. Интерес данного сервиса заключается в том, что он отвечает за трансляцию сообщений для всех подписчиков посредством инструмента UDP Multicast. Данный подход удобен для журналирования взаимодействия с ОС в рамках офиса или проекта, например, если локальная копия веб-приложения логгирует ошибки, то любой пользователь проекта может получить доступ к лог-файлу с ошибками даже не зная пароля от компьютера, на котором произошла ошибка.

Протокол UDP позволяет рассылать сообщения с помощью широковещательной групповой рассылки. При такой рассылке клиенту достаточно отправить одно сообщение, и его получат все остальные клиенты, которые подключены к группе. И нет надобности отправлять одно сообщение каждому отдельному клиенту.

Ключевым в данном случае является метод JoinMulticastGroup(), который позволяет присоединиться клиенту к группе в локальной сети. Благодаря этому объект UdpClient сможет получать дейтаграммы, которые предназначаются всей группы объектов UdpClient.

Традиционно адресами для мульткастинга выбирают адреса в диапазоне 224.0.0.0 - 239.255.255.255. В нашем случае я выбрал 239.0.0.222. Все подписчки в локальной сети обязаны прослушивать порт 65000, чтобы получать рассылку журналирования.

Основным методом, как и в случае с «почтовым» сервисом, стал метод SendMessage. В данном случае это хэндлер (обработчик события), который реагирует на логгирование по UDP. Данный метод обладает такой же сигнатурой и поддерживает асинхронную парадигму, что важно в приложении с пользовательским интерфейсом. Следует отметить, что доставка сообщения, строго говоря, не является гарантированной ввиду использования широковещательной рассылки по UDP, однако ввиду малых объемов данных это не является проблемой.

public async Task SendMessage(object sender, EncodingEventArgs e)

{

NLog.ILogger logger = NLog.LogManager.GetCurrentClassLogger();

logger.Info(e.data);

try

{

if (e.data.Length > 33)

e.data = e.data.Substring(0, 33) + "...";

string formattedMesage = string.Format(e.data);

byte[] buffer = EncryptionHelper.Encode(formattedMesage);

senderUdpClient.Send(buffer, buffer.Length, remoteEndPoint);

}

catch (Exception ex)

{

byte[] buffer = EncryptionHelper.Encode("Exception);

senderUdpClient.Send(buffer, buffer.Length, remoteEndPoint);

}

}

Для обоих сервисов важно отметить, что они поддерживают расширение и могут быть дополнены возможностью отправлять по UDP и SMTP целые файлы (т.к. особой разницы с точки зрения потоков нету – в обоих случаях это поток массива байт).

Ядром для комфортного и агрегированного использования всех методов данной библитеки стал сервис RemoteLoggingService.

В сущности данный сервис является наблюдателем за папками, файловой системой, агрегирует конфигурацию и настраивает все сервисы библоитеки. Следует отметить, что несмотря на ощущение, что данный сервис нарушает принцип SRP, это не так. Данный сервис выполняет паттерн «Фасад» и является посредником между клиентской частью и серверной/библоитечной.

Центральным методом стал метод Start(). Он запускает работы сервисов, начинает отслеживать влазимодействия с ОС. Метод поддеррживает асинхронную парадигму. Это особенно важно именно для этого сервиса, т.к. он является «прослойкой» между UI и билоитекой.

Метод предсталяет из себя конфигурирование низкоуровневой библиотеки FileSystemWatcher и бесконечный цикл с ожиданием в 5 секунд (это никак не скажется на отслеживании файлов: все события станут в очередь и освободятся, когда поток продолжит исполнение). Продемонстрируем небольшой отрывок ниже:

var directoryPath = \_configurationProvider.Directory;

CheckDirectory(directoryPath);

\_watcher = new FileSystemWatcher(directoryPath);

\_watcher.NotifyFilter = NotifyFilters.LastAccess | NotifyFilters.FileName | NotifyFilters.LastWrite | NotifyFilters.DirectoryName | NotifyFilters.CreationTime;

\_watcher.Created += new FileSystemEventHandler(DirectoryChanged);

\_watcher.Changed += new FileSystemEventHandler(DirectoryChanged);

\_watcher.Error += new ErrorEventHandler(DirectoryChangedError);

\_watcher.EnableRaisingEvents = true;

while (true)

{

await Task.Delay(5000);

}

С точки зрения побочных сервисов я отмечу провайдер конфигурации ConfigurationProvider. Данный провайдер ответственнен за работу с XML в файле App.config, а также appSettings.json. Именно благодаря этому мы можем работать с нашей библотекой как в ОС Windows, так и в Linux, т.к. данные легко настроить даже при работе из терминала.

Работа с App.config ведется при помощи билилотеки System.Configuration.dll. Данная библиотека доступна в классическом .NET Framework, а также посредстом NuGet Manager в .NET Core и .NET Standard (что характерно для нашего случая). Для поддержки обратной совместимости с ОС семейства Unix данная библиотека не поддерживает редактироване свойств конфигурационого файла на рантайме («времени исполнения»).

Аналогично вместе с .NET Core представлена более продвинутая библиотека Microsoft.Extensions.Configuration. В нашем случае она используется для работы с JSON.

Конфигурирование происходит следующим образом. Запуск этого метода происходит автоматически при загрузке сборки и первой попытке обратиться к статическому полю – автоматически вызывается статический конструктор static ConfigurationProvider():

private static void BuildConfig()

{

IConfiguration configuration = new ConfigurationBuilder().AddJsonFile("appSettings.json", optional: true, reloadOnChange: true).Build();

config = configuration;

}

Работа со свойствами конфигурацонный файлов на рантайе ведется при помощи удобный статических свойств определнных в классе-провайдере. Отметим, что свойства не поддерживают мутаторы («сеттеры»), причины были описаны выше:

public string MailFrom { get => ConfigurationManager.AppSettings["mailFrom"]; }

public string Password { get => ConfigurationManager.AppSettings["password"]; }

4.2 Windows-приложение System Remote Logger (SRL) Client

В рамках данной работы было разработано клиентское приложение, которое позволяет работать с System Remote Logger (SRL) API. Для реализации использовался фреймворк Windows Presentation Foundation (WPF) и сторонняя библиотека Material Design In XAML (MDIX).

Основными компонентами клинетского приложения стали следующие окна:

* MainWindow.xaml
* Settings.xaml
* UdpLoggingScreen.xaml

Следует отметить, что число окон не влияет на функциональность приложения: все основные виджеты для конфигурирования и просмотра текущей инфомрации находятся на «домашней странцие приложения».

Основная зона ответственности главного экрана – это не только виджеты и настройки, а также запуск всех сервисов:

RemoteLoggingService loggingService = new RemoteLoggingService(config);

UdpConnectionService udpService = new UdpConnectionService(true, UserPrincipal.Current.DisplayName);

int port = config.Port;

string host = config.SmtpHost;

MailSender smtpService = new MailSender(port, host, config);

udpService.Connect();

if (config.UdpLoggingOn)

loggingService.NewMessageOn += udpService.SendMessage;

if (config.EmailLoggingOn)

loggingService.NewMessageOn += smtpService.SendMessage;

loggingService.Start();

В окне работы с UDP-рассылкой находится экран, который позвоялет подключится к прослушке, а также добавлять специфичные сообщения в общий чат-рассылку.

Итогом работы стала следующая система проектов в одном решении (Solution), приведенная на рис. 4.1.

Также следует отметить важность сущностей-хелперов, которые производят перевод POCO-сущностей в массивы байт для транспортировки посредством UDP-мультикастинга через датаграммы:

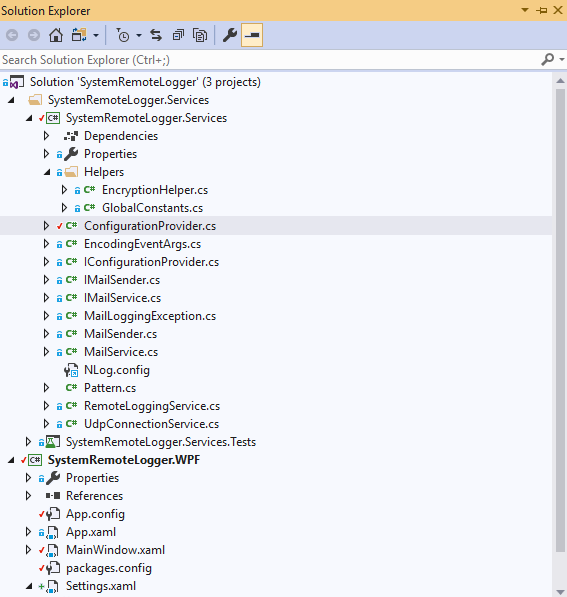
public static class EncryptionHelper

{

public static byte[] Encode(string data) => Encoding.UTF8.GetBytes(data);

public static string Decode(byte[] data) => Encoding.UTF8.GetString(data);

}

  
Рис. 4.1. Демонстрация системы типов в решении

1. **Демонстрация работы программы**

Для просмотра анимированной (gif) версии демонстрации можно посетить личный репозиторий данного курсового проекта на GitHub: <https://github.com/stashinskii/BSUIR.SystemRemoteLogger>

Как и было описаны ваше – реализация пользовательского интерфейса легла на библиотеку MDIX и ввиду этого были задействованы ряд возможностей этой библиотеки, в частности анимированные виджеты, анимации переходов и событий, общая настройка цветовой гаммы приложения (похожий подход реализовывался в проекте SystemStastusMonitor в библиотеке Metro UI).

Главный экран:

На данном экране расположены следующие компоненты:

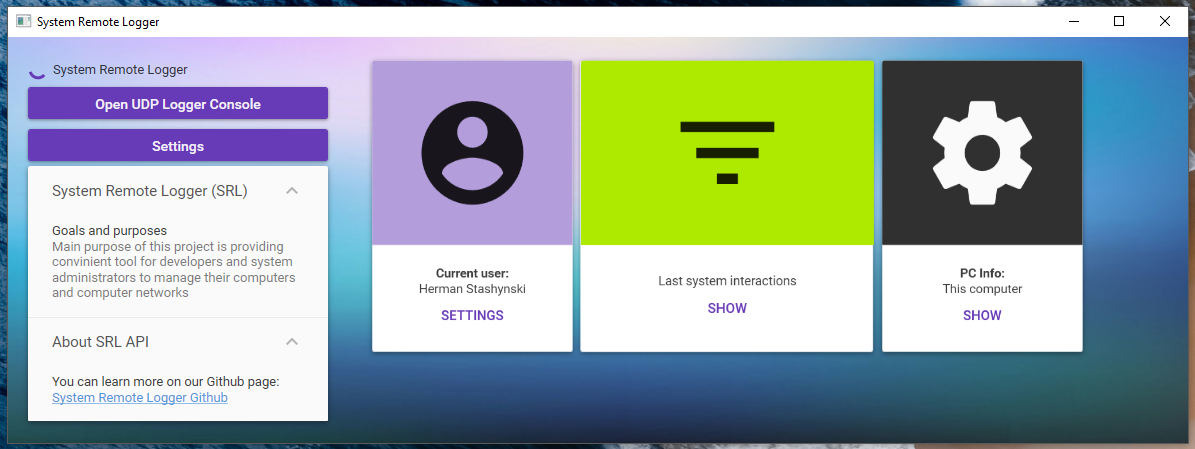
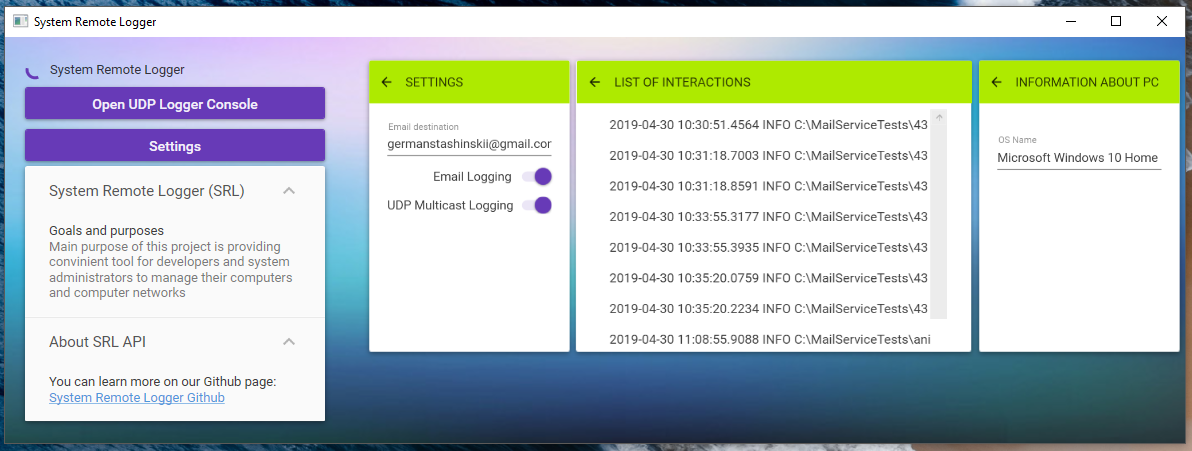
* Виджеты для демонстрации важной информации:
  + Текущий пользователь и информация о нем (доступна после развертывания виджета)
  + Виджет последних логов
  + Виджет информации о ПК
* Кнопки для открытия настроек и консоли UDP-чата
* Аккордеоны для общего описания приложения: ссылка на репозиторий, цели и задачи проекта

На первом виджете расположена информация о пользователе и текущих настройках приложения: почта для рассылки по SMTP, переключатели уровней логгирования.

На втором виджете расположен список последних взаимодействий с системой, которые были залогированны и отправлены по почте.

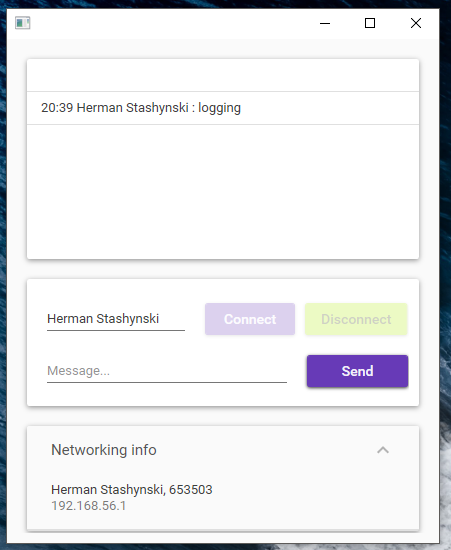
Третий виджет является опциональным и хранит имя операционной системы. Данный виджет предполагает расширение функциональности и добавление новых свйоств из библиотеки SystemStatusMonitor (мой личный курсовой проект по дисциплине АВС, 5 семестр).

Данные виджеты никак не влияют на конфигурирование. Например, переключатели не достпуны пользователю, т.к. это бы нарушало обратную совместимость с ОС Linux. Подробнее с этой особенностью можно ознакомиться на официальной странице библиотеки для провайдера конфигурации Microsoft.Extensions.Configuration от компании Microsoft: <https://www.nuget.org/packages/Microsoft.Extensions.Configuration/>.

  
  
Рис. 5.1. Главный экран приложения. Развернутые и свернутые виджеты

Экран UDP-чата:

Для работы с широковещательной рассылкой по UDP и просмотров логов был разработан своеобразный чат, в который попадают все сообщения от системы, а также от администратора. При желании чат может быть расширен возможность прикреплять файлы и высылать push-уведомления в ОС Windows.

  
Рис. 5.2. Экран UDP-чата

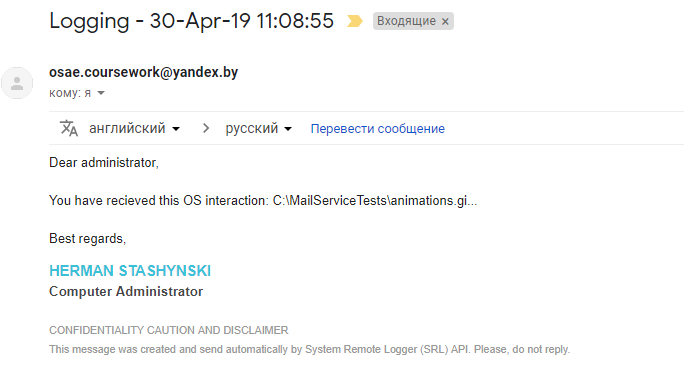
Получение логов по почте:

В предыдущих процессах был описан процесс разработки SMTP-клиента для рассылки логов по почте. Сейчас опишем процеес конфигурации рассылки и результаты ее работы.

В качестве SMTP-серверы был выбран сервер компании «Яндекс». Данные конфигурирования представлены в файле App.config:

|  |  |
| --- | --- |
| Port | 587 |
| Host | <http://smtp.yandex.ru> |
| Login | [osae.coursework@yandex.by](mailto:osae.coursework@yandex.by) |
| Password | Osae-1 |
| Recipient | [germanstashinskii@gmail.com](mailto:germanstashinskii@gmail.com) |

В результате по сконфигурированному адресу приходит автоматическое письмо с заголовком и подписью в формате HTML, которое можно настраивать по желанию клиента. В тему письма вставляется дата и общая кодовая фраза «Logging» для удобной фильтрации писем в почтовом ящике:

  
Рис. 5.3. Скриншот полученного письма

**Заключение**

В рамках данной курсовой работы был разработан программный продукт, который предсталвяет из себя инструмент для логгирования состояния компьютера и ОС Windows и Linux, а также различных взаимодействий с ней посредством сетевых протоколов через широковещательную рассылку по локальной сети с помощью UDP Multicast и глобальная рассылка по сети посредством SMTP на почтовые ящики. Центральная (core) часть проекта реализована как система сервисов System Remote Logger (SRL) API в составе сборке SystemRemoteLogger.Services.

Помимо реализации всех необходимых бибилотек SRL API, был реализован Windows клиент System Remote Logger Client для удобного мониторинга процесса логгирования и просмотра информации о конфигурации библиотеки. Клиентская часть представляет из себя минималистичное приложение, не нагруженное элементами управления. Этого удалось добиться ввиду использования библиотеки для создания Material Design в WPF - MDIX. Возможности клиентской части в теории не ограничены лишь ОС Windows – современные версии .NET Core (в частности, .NET Core 3) предполагают возможность полной перекомпиляции Windows Presentation Foundation для работы в ОС Linux на базе .NET Core.

Система обладает гибкостью и возможностью работать с различными инструментами. В частности, система сервисов предполагает работу не только с собственным SMTP-сервером (например, на базе Internet Information Services (IIS)), но и со сторонними SMTP-серверами, например Gmail или Yandex.Mail. Удобство системы проявляется также в том, что для работы UDP-рассылки не требуется дополнительной конфигурации: рассылка логов будет производиться автоматически в сети, в которой запущены процессы данного приложения. Такие особенности могут оказать важное влияние на выбор особенно для тех клиентов, кто обладает ограниченными ресурсами (финансовыми или техническими).

Система типов, реализованная в данной работе, является кросс-платформенной и может работать как библиотека .NET Standard на ОС Linux. Запуск её возможен посредством запуска приложения dotnet с соответствующими параметрами в bash-терминале. Такая система будет исполнятся в среде CoreCLR.

Итогом проделанной работы стала полноценно работающая библиотека и клиентская часть, которые выполняют поставленные задачи и могут быть расширены в будущем. Помимо изучения особенностей компьютерных сетей и работы с кроссплатформенным программированием для ОС Linux и Windows под платформу .NET были применены инструменты для автоматизации и мониторинга процессов разработки программного продукта (в частности, использование Trello), что является, несомненно, важным элементом современной разработки ПО.

Цели, которые были поставлены перед началом работы, были достигнуты, а в некоторых аспектах превышают поставленные ожидания.

**Использованная литература**

1. Goldstein S. Pro .NET Perfomance. Optimize Your C# Applications. Apress – 361 c.
2. Microsoft Developer Network [Электронный ресурс] / Microsoft Corparation - Вашингтон, США – Режим доступа: https://msdn.microsoft.com/. – Дата доступа: 17.03.2019
3. Official home of .NET on GitHub [Электронный ресурс] / GitHub.com - США – Режим доступа: <https://github.com/Microsoft/dotnet>. – Дата доступа: 02.04.2019
4. How SMTP works [Электронный ресурс] / Medium.com - США – Режим доступа: <https://medium.com/@imranshaikh2124/how-smtp-works-3c5aa6e81779>. – Дата доступа: 02.04.2019
5. Enable and Set Up IIS [Электронный ресурс] / dnnsoftware.com – Режим доступа: <https://www.dnnsoftware.com/docs/administrators/setup/set-up-iis.html>. – Дата доступа: 02.03.2019
6. What computer networks are and how to actually understand them [Электронный ресурс] / Medium.com – Режим доступа: <https://medium.freecodecamp.org/computer-networks-and-how-to-actually-understand-them-c1401908172d> . – Дата доступа: 02.04.2019