СОДЕРЖАНИЕ

[Введение 1](#_Toc163849656)

[1. ТЕОРИТИЧЕСКАЯ ЧАСТЬ 3](#_Toc163849657)

[1.1 Техническое задание 3](#_Toc163849658)

[1.2 Описание выбранного инструментария 8](#_Toc163849659)

[1.3 Анализ существующих решений 17](#_Toc163849660)

[2. ПРАКТИЧЕСКАЯ ЧАСТЬ 20](#_Toc163849661)

[2.1 Макет графического интерфейса 20](#_Toc163849662)

[2.2 Описание исходного кода 21](#_Toc163849663)

Введение

В современном мире операционные системы играют ключевую роль в повседневной деятельности как отдельных пользователей, так и организаций. Операционная система Windows, разработанная корпорацией Microsoft, занимает лидирующее положение среди множества доступных ОС благодаря своей функциональности, удобству использования и широкому спектру поддерживаемого программного обеспечения.

Однако, с течением времени, операционная система может стать менее эффективной из-за накопления временных файлов, кэша, ненужных ресурсов и другого мусора, что приводит к замедлению работы компьютера и нехватке памяти для разных целей. В связи с этим, существует необходимость в разработке программных средств, способных оптимизировать производительность операционной системы Windows путем очистки временных файлов и ненужных ресурсов.

Целью данного курсового проекта является разработка и реализация программного обеспечения, которое будет способно автоматически обнаруживать и удалять временные файлы, кэш и другой мусор, тем самым повышая производительность операционной системы Windows и обеспечивая более гладкую и эффективную работу для пользователей. Так же программное обеспечение будет способно помогать пользователю управлять компьютером из любой точки мира, где есть интернет.

В ходе работы будет проведен анализ существующих подходов к оптимизации производительности операционной системы Windows, изучены основные принципы работы файловой системы и механизмы накопления временных данных. На основе полученных знаний будет разработано программное обеспечение способное решить проблемы с забитыми ненужными файлами дисками.

Задачи:

1. Составить техническое задание
2. Описать выбор методологии
3. Описать выбор языка программирования
4. Описать выбор инструментария
5. Проанализировать предметную область
6. Разработать программное обеспечение
7. Протестировать программное обеспечение
8. Создать инструкцию по установке

1. ТЕОРИТИЧЕСКАЯ ЧАСТЬ

1.1 Техническое задание

1. Введение

1.1. Наименование программного продукта или программы: WinWipe

1.2. Краткая характеристика области применения:

Программа предназначена для очистки операционных систем (далее - ОС) Windows 10 и 11 от временных файлов, ненужных данных и других ресурсов, которые могут замедлять работу системы и занимать дополнительное место на жестком диске;

2. Основания для разработки

2.1. Основания для проведения разработки:

Изучение работы файловой системы и методов накопления временных файлов в ОС Windows, путем создания программного обеспечения (далее - ПО). Развитие навыков программирования на практике, комбинирование двух языков программирования;

2.2. Наименование и условное обозначение разработки:

Разработка программного обеспечения для оптимизации ОС Windows;

3. Назначение разработки

3.1. Функциональное назначение:

Предоставление пользователю возможности осуществлять эффективную очистку операционных систем Windows 10 и 11 от ненужных данных;

Установка софта для управления компьютером;

3.2. Эксплуатационное назначение:

Улучшение производительности операционных систем Windows 10 и 11 путем очистки системы от временных файлов, кэша браузера, устаревших резервных копий и других ненужных данных;

Удаленное управление базовыми функциями компьютера, такими как перемещение по каталогам, копирование файлов, запуск программ

4. Требования к продукту

4.1. Требования к функциональным характеристикам.

4.1.1. Состав выполняемых функций:

* Очистка временных файлов Windows;
* Очистка кэша браузера (для различных браузеров: Chrome, Firefox Yandex и т.д.);
* Удаление ненужных системных логов и отчетов об ошибках;
* Очистка корзины;
* Удаленное управление компьютером.

4.1.2. Организация входных данных:

Название программы логотип, макет программы, краткое описание функционала;

4.1.3. Организация выходных данных:

Папка проекта формата Visual Studio, в которой должны лежать макеты графического интерфейса программы, используемые изображения, файлы исходного кода. А также собранная программа формата исполняемого файла Windows.

4.1.4. Временные характеристики:

* Запуск программы не должен превышать 10 секунд;
* Модуль очистки не должен работать более 5 минут.

4.2. Требования к надежности.

4.2.1. Требования к обеспечению надежного функционирования:  
Некорректные действия пользователя с программой:

* Постоянное нажатие кнопки очистки без надобности;
* Нажатие кнопки очистки до окончания предыдущей очистки;
* Запуск нескольких копий программы одновременно.

4.2.2. Требования к защите

ПО должно предусматривать базовую защиту от редактирования:

Редактирование кода сторонними лицами;

Внедрение в программу вредоносного ПО.

4.3. Условия эксплуатации носителя.

4.3.1. Климатические условия эксплуатации:

Для работы на компьютере в помещении должны быть обеспечены оптимальные параметры микроклимата: температура, относительная и абсолютная влажность.

Оптимальными параметрами микроклимата в помещении с компьютерами считаются:

* температура воздуха - от 19 до 40 градуса;
* относительная влажность - от 62 до 5% соответственно;
* скорость движения воздуха - не более 0,1 м/с.

В производственных помещениях, в которых работа с использованием персонального компьютера (далее - ПК) является вспомогательной, температура, относительная влажность и скорость движения воздуха на рабочих местах должны соответствовать действующим санитарным нормам микроклимата производственных помещений. Содержание вредных химических веществ в воздухе таких помещений не должно превышать предельно допустимых концентраций вредных веществ в воздухе рабочей зоны в соответствии с действующими гигиеническими нормативами.

А вот в помещениях, в которых работа с использованием ПК является основной (например, диспетчерские, операторские, расчетные, кабины и посты управления, залы вычислительной техники и др.), содержание вредных химических веществ в воздухе не должно превышать предельно допустимых концентраций загрязняющих веществ в атмосферном воздухе населенных мест в соответствии с действующими гигиеническими нормативами;

4.3.2. Требования к видам обслуживания:

Для установки и поддержки работоспособности программы требуется пользователь со знанием ПК;

4.3.3. Требования к квалификации пользователя:

Знание ПК на уровне пользователя;

4.4. Технические параметры. Программный продукт требует следующей технической конфигурации персонального компьютера.

4.4.1. Типы поддерживаемых операционных систем:

Windows (с Windows 10 по Windows 11).

4.4.2. Минимальные системные требования:

Процессор 2 ядра, с тактовой частотой: 1600 Mhz;

Объём оперативной памяти: 1 ГБ;

Видеоадаптер с объемом памяти: 16 Mb;

Объем жёсткого диска: не менее 20 МБ свободной памяти;

4.5. Требования к информационной структуре и методам решения.

4.5.1. Требования к исходным кодам и языкам программирования:

* C#;
* Python 3;
* XAML.

4.5.2. Требования к защите информации и программ:

Программа не должна собирать персональные данные пользователя;

4.5.3. Требования к программным средствам, используемым  
программой:

* Любая версия Windows 10 или 11 с установленными пакетами Visual Studio;
* .NET Framework 4.7.2.

4.6. Требования к маркировке и упаковке.

4.6.1. Требования к маркировке:

В главном окне программы указано название и логотип программы;

4.7. Транспортировка и хранение:

Хранение на любом носителе поддерживаемым ОС Windows;

5. Требования к программной документации

5.1. Состав программной документации:

* Руководство по эксплуатации пользователем программного продукта;
* Инструкция по инсталляции;
* Руководство администратора.

6. Календарный план работ

Таблица 1 – Календарный план

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № этапа | Название этапа | Исполнители | Сроки выполнения | Чем заключается этап | Отметка о выполнении этапа |
| 1 | Сбор информации, анализ требований к системе | Войтов Н.Е | 05.05.2024 | Электронный вариант информации |  |
| 2 | Оформление технического задания | Войтов Н.Е | 05.05.2024 | Техническое задание |  |
| 3 | Проектирование | Войтов Н.Е | 05.05.2024 | Написание ПО |  |
| 4 | Тестирование программного продукта | Войтов Н.Е | 05.05.2024 | Тестирование на ошибки ПО |  |
| 5 | Сопровождение (редактирование кода) | Войтов Н.Е | 05.05.2024 | Готовый программный продукт |  |

7. Порядок контроля и приемки

7.1. Виды испытаний:

* Проверка работоспособности модулей программы;
* Проверка правильного расположения графических элементов и текста;
* Проверка работоспособности отображения логов о выполнении.

7.2. Общие требования к приёму работы

Программный продукт должен считаться работоспособным, если он удовлетворяет всем пунктам данного технического задания;

8. Приложение:

* Накопитель с готовой программой;
* Руководство пользователя;
* Руководство администратора.

1.2 Описание выбранного инструментария

1.2.1 Выбор методологии разработки

Водопадная модель разработки — это линейный подход к разработке программного обеспечения. Она характеризуется строгим, структурированным процессом. Основной принцип — каждый этап проекта должен быть завершен до начала следующего. Как правило, модель состоит из следующих этапов разработки: системного анализа, анализа требований, проектирования, реализации, тестирования, внедрения и сопровождения.

Эта методология подходит для хорошо продуманных проектов, в которых требования строго зафиксированы и вряд ли сильно изменятся в процессе разработки. Она предлагает четкие рамки проекта, что облегчает управление ресурсами и распределение времени. Данная методология не подойдет для курсового проекта, который будет развиваться и меняться в процессе разработки.



Рисунок 1 – Схема водопадной методологии

Функционально-ориентированная разработка (Feature Driven Development, FDD) — это гибкая методология, также основанная на принципах Agile. Она направлена на создание небольших функций или функциональных блоков. FDD — итеративная и инкрементальная (пошаговая) методология, и ее цель — быстро получить ощутимые результаты.

Есть пять основных этапов разработки в рамках FDD: создание общей модели, формирование списка функций, планирование по функциям, дизайн по функциям и разработка по функциям. FDD предусматривает отчетность на всех этапах для отслеживания прогресса и фиксирования результатов. Обычно методология используется в крупномасштабных проектах, поэтому она не подойдет для разработки небольшого проекта.



Рисунок 2 – Схема функционально-адаптированной методологии

Методология Agile — это популярный подход, в котором основное внимание уделяется гибкости, сотрудничеству и оптимизации процессов для реализации качественного проекта. Это итеративный подход, и приоритет в нем отдается обратной связи от владельца продукта и адаптации к изменяющимся требованиям. Цикл разработки ПО по Agile-методологии можно разбить на шесть этапов: планирование, проектирование, разработка, тестирование, развертывание и обслуживание.

Этот подход к разработке широко распространен как в IT-индустрии, так и в других областях — в управлении проектами, разработке продуктов и даже в проектах, не связанных с IT. Организации часто адаптируют принципы Agile под свои нужды. Для данного курсового проекта нужна гибкость, поэтому за основу была взята именно эта методология.

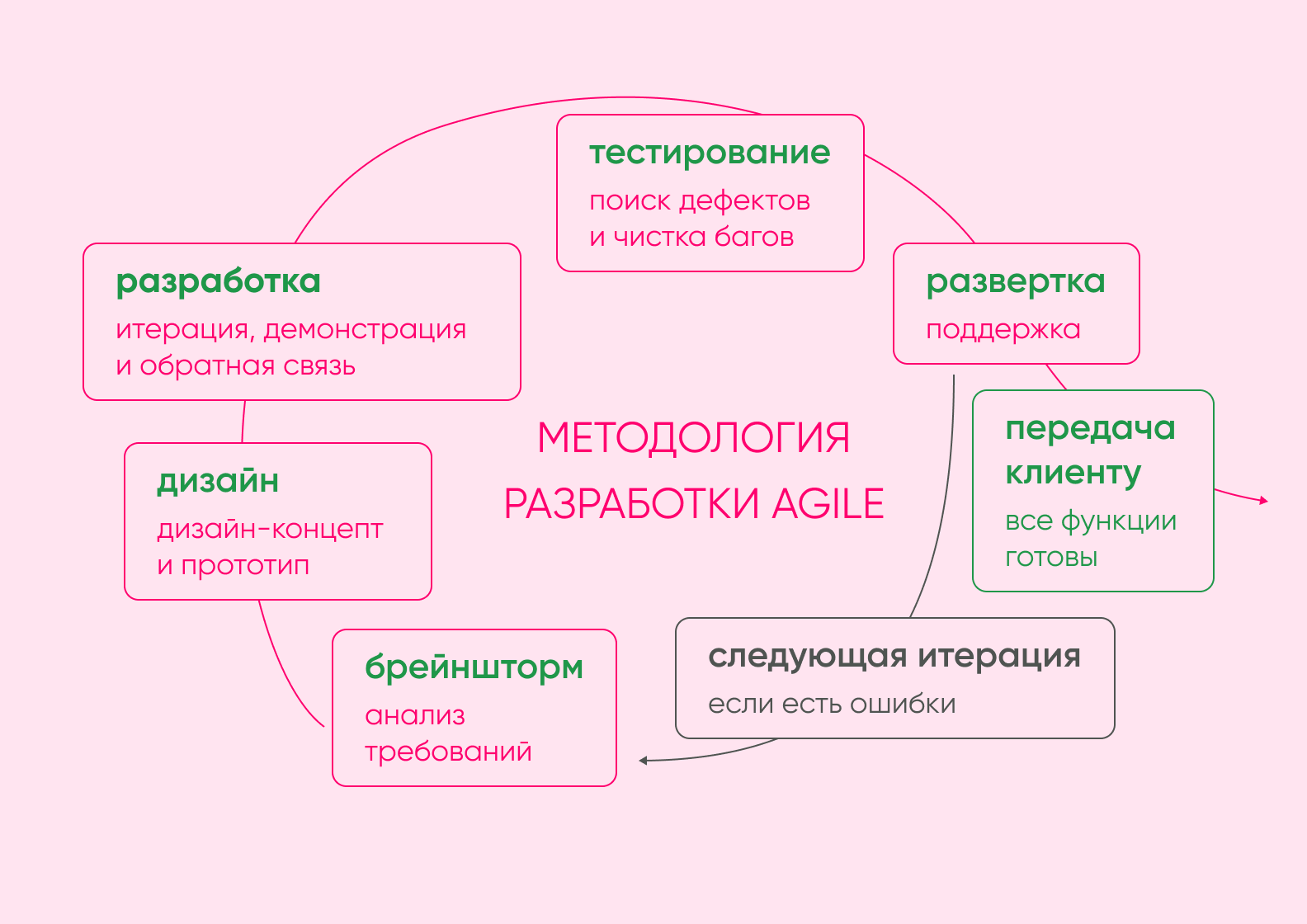


Рисунок 3 – Схема методологии Agile

1.2.2 Выбор языка программирования

Для данной задачи нужно подобрать подходящий язык программирования (далее - ЯП), благодаря которому можно будет реализовать курсовой проект. Язык должен поддерживаться на системах Microsoft Windows 10 и Microsoft Windows 11, возможность работать с файловой системой на приемлемой скорости. Так же для удобства должен быть способ написания графического интерфейса, например фреймворк или встроенные модули языка.

Java — строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle).

Для чего используется язык программирования ЯП Java?

* Разработка игр;
* Облачные вычисления;
* Большие данные;
* Искусственный интеллект;
* Разработка программ для мобильных устройств;

Преимущества:

* Концепция объектно-ориентированного программирования (далее ООП).
* Это значит, что программист сам определяет тип данных, его структуру и набор применяемых к нему функций. Это позволяет многократно использовать одни и те же объекты в разных программах, более удобно для организации структуры программ (особенно с большим объёмом кода), помогает избегать ошибок и упрощает поддержку и модернизацию старого кода;
* Это язык высокого уровня, т.е., он больше похож на человеческую речь, а не на машинный код. Следовательно, у него сравнительно простой синтаксис, что делает его быстрым для освоения и удобным для написания кода, его чтения и обслуживания. Есть и более простые варианты (например, ЯП Python), однако у человека с базовым пониманием основ программирования здесь не должно возникнуть сложностей;
* Безопасность. У ЯП Java есть несколько функций, которые ликвидируют часто встречающиеся уязвимости. В частности, это Security Manager – создаваемая для каждого приложения политика безопасности, в которой можно прописать правила доступа;
* Удобство для распределённого программирования. Этот язык изначально создавался для совместной работы (в том числе удалённой), поэтому он позволяет совместно использовать данные и программы несколькими компьютерами одновременно;
* Принцип «написать один раз и использовать везде» — написанное на ЯП Java приложение можно запустить на любой поддерживающей его платформе;
* Стабильное и постоянно развивающееся сообщество. По многочисленности и активности с ним мало кто может соперничать. В Сети есть масса ресурсов, где на любой вопрос по этой теме либо уже есть ответ, либо найдётся кто-нибудь, кто его подскажет, равно как и сотни курсов, семинаров и обучающих программ, как платных, так и бесплатных.

Недостатки:

* Низкая скорость. Все высокоуровневые языки приходится компилировать с помощью виртуальной машины, что плохо сказывается на их производительности. ЯП Java – не исключение, кроме того, у него есть и некоторые собственные особенности, вызывающие дополнительные проблемы с производительностью;
* Многословие. Сходство с естественными языками делает ЯП Java проще для изучения и понимания, но также ведёт и к тому, что он содержит много лишней информации и довольно громоздок;
* Существующие фреймворки не позволяют сделать легко красивый UI нативного вида для программ под Windows.

Python — это язык программирования, который широко используется в интернет-приложениях, разработке программного обеспечения, науке о данных и машинном обучении. Разработчики используют ЯП Python, потому что он эффективен, прост в изучении и работает на разных платформах.

Преимущества ЯП Python:

* Разработчики могут легко читать и понимать программы на ЯП Python, поскольку язык имеет базовый синтаксис, похожий на синтаксис английского.
* ЯП Python помогает разработчикам быть более продуктивными, поскольку они могут писать программы на Python, используя меньше строк кода, чем в других языках.
* ЯП Python имеет большую стандартную библиотеку, содержащую многократно используемые коды практически для любой задачи. В результате разработчикам не требуется писать код с нуля.
* Разработчики могут легко сочетать ЯП Python с другими популярными языками программирования: Java, C и C++.
* Активное сообщество ЯП Python состоит из миллионов поддерживающих разработчиков со всего мира. При возникновении проблем сообщество поможет в их решении.
* Кроме того, в Интернете доступно множество полезных ресурсов для изучения ЯП Python. Например, вы можете легко найти видеоролики, учебные пособия, документацию и руководства для разработчиков.
* ЯП Python можно переносить на различные операционные системы: Windows, macOS, Linux и Unix.

Главным недостатком ЯП Python будет являться медленная скорость работы, для работы с большими операциями скорость ответа будет слишком большой. Так же очень мало фреймворков для написания своего интерфейса.

C++ — это язык программирования, который был разработан в 80-х годах прошлого века как расширение языка C. Этот язык отличается от Си тем, что имеет больший набор возможностей, включая объектно-ориентированное программирование и шаблоны.

Преимущества C++:

* Высокая производительность, потому что он не накладывает никакой избыточной нагрузки на программу, не использующую какие-либо возможности.
* Поддержка множества стилей программирования (процедурное программирование, абстракцию данных, объектно-ориентированное программирование и обобщенное программирование). Поэтому разработчик может сам выбрать, в каком стиле ему писать программу.
* Большое сообщество.

Главным недостатком будет являться высокий порог входа. Сложная читаемость кода. Сложность языка не оправдана для текущей цели. Аналогом с похожими характеристиками будет являться следующий язык программирования.

C# — это язык программирования от компании Microsoft. Изначально его создавали для проектов под Windows, но теперь это по-настоящему универсальный язык: на нём пишут игры, десктопные приложения, веб-сервисы, нейросети.

Поскольку C# — это язык от Microsoft, на нём удобно писать настольные приложения для Windows. Для этого разработчики придумали специальную платформу — называется .NET Framework.

Преимущество платформы в том, что она может исполнить любую C#-команду на любом процессоре — а на Windows работает не меньше тысячи разных моделей. Если бы не .NET, пришлось бы компилировать код под каждое железо отдельно. А ещё .NET поддерживает много плагинов, библиотек и шаблонов для наглядной разработки интерфейсов — это фактически целый мир и экосистема для программиста на Windows.

Таким образом можно понять, что для разработки программы с графическим интерфейсом под Windows отлично подходит C#, а в дополнение к нему ЯП Python для написания некоторых функций программы.

1.2.3 Выбор среды разработки

Для C# предусмотрена официальная среда разработки Microsoft Visual Studio, в которой есть весь инструментарий для написания графического интерфейса, установки библиотек и отладки приложения.

Есть 3 разных выпуска IDE (Integrated Development Environment) Microsoft Visual Studio: Enterprise, Professional, Community. Они отличаются функционалом отладки и тестирования приложения. Среди них есть только одна бесплатная версия -Community. Поэтому был выбран выпуск Community.

1.2.4 Выбор библиотек и фреймворков

System

Содержит фундаментальные и базовые классы, определяющие часто используемые типы значений и ссылочных данных, события и обработчики событий, интерфейсы, атрибуты и исключения обработки. С помощью классов внутри этого пространства реализовано большинство функций работы с файлами и системой в целом.

WinForms

Windows Forms — это платформа пользовательского интерфейса для создания классических приложений Windows. Она обеспечивает один из самых эффективных способов создания классических приложений с помощью визуального конструктора в Visual Studio. Такие функции, как размещение визуальных элементов управления путем перетаскивания, упрощают создание классических приложений.

В Windows Forms можно разрабатывать графически сложные приложения, которые просто развертывать, обновлять, и с которыми удобно работать как в автономном режиме, так и в сети. Приложения Windows Forms могут получать доступ к локальному оборудованию и файловой системе компьютера, на котором работает приложение.

Microsoft.Win32.TaskSheduler

Библиотека, которая позволяет работать с Планировщиком заданий ОС Windows, с помощью нее реализована установка ПО в автозапуск для удаленного управления ПК.

Windows Presentation Foundation (WPF)

Технология Windows Presentation Foundation (Далее WPF) является часть экосистемы платформы .NET и представляет собой подсистему для построения графических интерфейсов.

Если при создании традиционных приложений на основе WinForms за отображение элементов управления и графики отвечали такие части ОС Windows, как User32 и GDI+, то приложения WPF основаны на DirectX. В этом состоит ключевая особенность рендеринга графики в WPF: используя WPF, значительная часть работы по отрисовке графики, как простейших кнопочек, так и сложных 3D-моделей, ложиться на графический процессор на видеокарте, что также позволяет воспользоваться аппаратным ускорением графики.

Одной из важных особенностей является использование языка декларативной разметки интерфейса XAML, основанного на XML: вы можете создавать насыщенный графический интерфейс, используя или декларативное объявление интерфейса, или код на управляемых языках C#, VB.NET и F#, либо совмещать и то, и другое.

1.3 Анализ существующих решений

1.3.1 ССleaner

Пожалуй, самый популярный софт, который распространяется на бесплатной основе. Но функционал утилиты может быть расширен, если приобрести премиум версию. Однако, и стартового набора будет достаточно для очистки системы. Под нож попадут не только временные файлы, но и остаточные ключи в реестре. Также можно настроить автозагрузку, обновить программы и выполнить стирание дисков. При желании, можно скачать портативную версию и запускать её с флешки на разных компьютерах.



Рисунок 4 – Окно CCleaner

Достоинства:

* Большое кол-во функций;
* Качественная и при этом быстрая очистка/анализ;
* Удобный и приятный интерфейс;
* Поддерживается русский язык.

Недостатки:

* Очищать реестр с помощью CCleaner бесполезно и даже вредно.

Есть мнение, что реестр Windows попросту необходимо регулярно чистить и оптимизировать. В процессе эксплуатации системы (в особенности после удаления программ) он заполняется мусорными данными и ошибками. От этого компьютер якобы начинает страшно тормозить, и свободное место в оперативной памяти и на диске стремительно заканчивается.

Однако на самом деле необходимость чистки реестра — это миф, который тянется со времён Windows 95. Возможно, тогда оптимизация действительно имела смысл, но не теперь;

Microsoft официально заявила, что эта процедура бесполезна и зачастую даже вредна.

Корпорация утверждает, что не может рекомендовать никакие программы для очистки и не гарантирует работоспособность Windows после их применения, так как реестр не предназначен для правки пользователем.

И да, даже если вы удалите все «лишние» фрагменты реестра, то высвободите максимум десяток‑другой килобайт места на диске. Никакой пользы это не принесёт, компьютер не будет загружаться быстрее;

* CCleaner назойливо предлагает программное обеспечение.

CCleaner предназначен для очистки ПК, но он же постоянно старается запихнуть в систему что‑нибудь лишнее.

Например, при установке программа предлагает загрузить антивирусы Avast или AVG. Это ожидаемо, потому что чистильщик принадлежит Avast, но всё равно не очень приятно;

* CCleaner занимает оперативную память и показывает рекламу.

Ранние версии CCleaner запускались только по требованию пользователя. У вас осталось мало места на диске, вы запустили чистильщик, освободили пространство и забыли о CCleaner до следующего раза. Теперь же он по умолчанию работает в фоновом режиме, расходуя лишнюю оперативку.

Вдобавок программа регулярно предлагает купить Pro‑версию за 19,95 доллара в год. Но это вряд ли можно назвать удачным вложением. CCleaner Professional умеет только автоматически выполнять очистку ненужных данных, истории браузеров и файлов cookie;

* CCleaner собирает пользовательские данные.

Эта скрытая функция называется Heartbeat. Приложение отправляет статистику использования CCleaner на серверы Avast каждые 12 часов.

Более того, собранную информацию Piriform и Avast имеют право продавать третьим лицам, о чём прямо сообщают в лицензионном соглашении.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Макет графического интерфейса

Макет графического интерфейса был написать с использованием языка разметки XAML, который сделан на основе XML. Он содержит логотип программы, название, автора. Так же потребуется несколько кнопок управления и флаги для выбора параметров очистки. Для большинства элементов нужно задать атрибуты стиля, размера, цвета, а также уникальное имя для работы с классами. Для программы понадобится несколько окон, для них так же будут написаны макеты (смотреть рисунок 5, 6, 7).

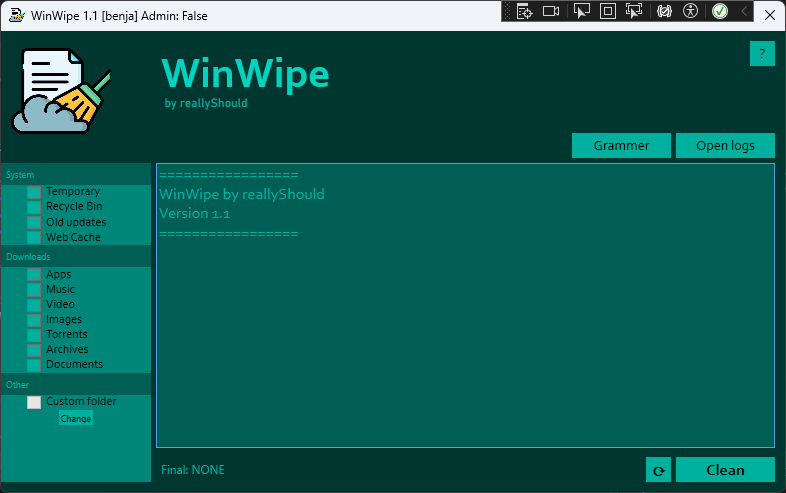


Рисунок 5 – Главное окно

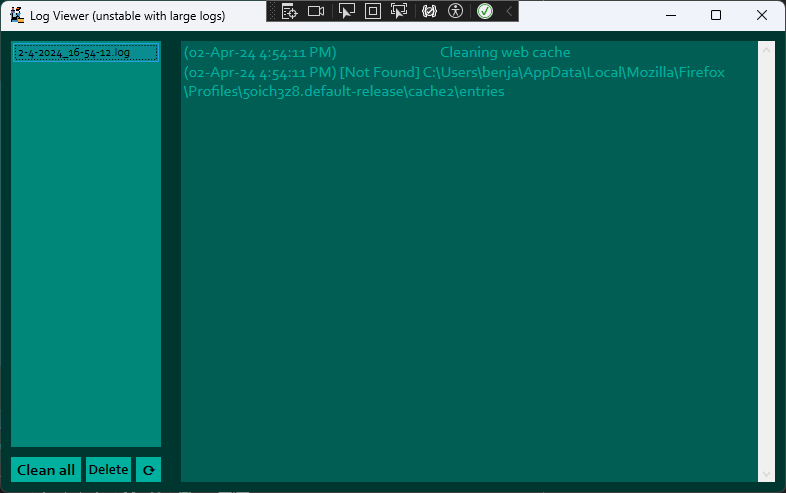


Рисунок 6 – Окно просмотра логов

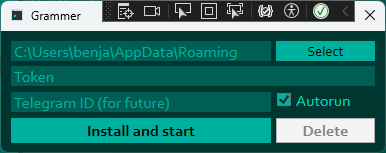


Рисунок 7 – Окно установки удаленного доступа

2.2 Описание исходного кода

Для начала нужно написать графический интерфейс на языке разметки XAML, укажем основные параметры окна (смотреть рисунок 8). Отрисуем панель выбора параметров очистки (смотреть рисунок 9), так же понадобится отображение логов и кнопка начала (смотреть рисунок 10). И в завершение сделаем отображение логотипа программы и название (смотреть рисунок 11).

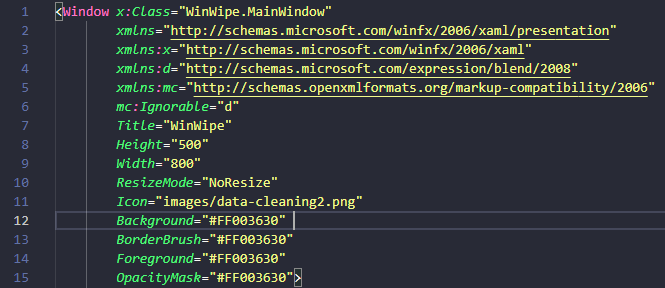


Рисунок 8 – Параметры главного окна

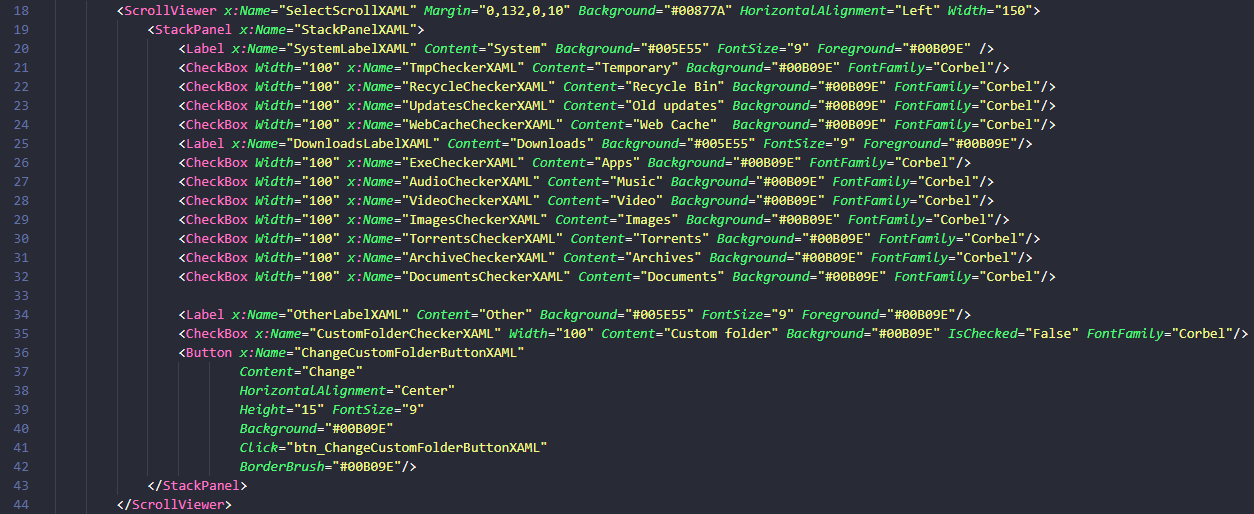


Рисунок 9 – Параметры очистки

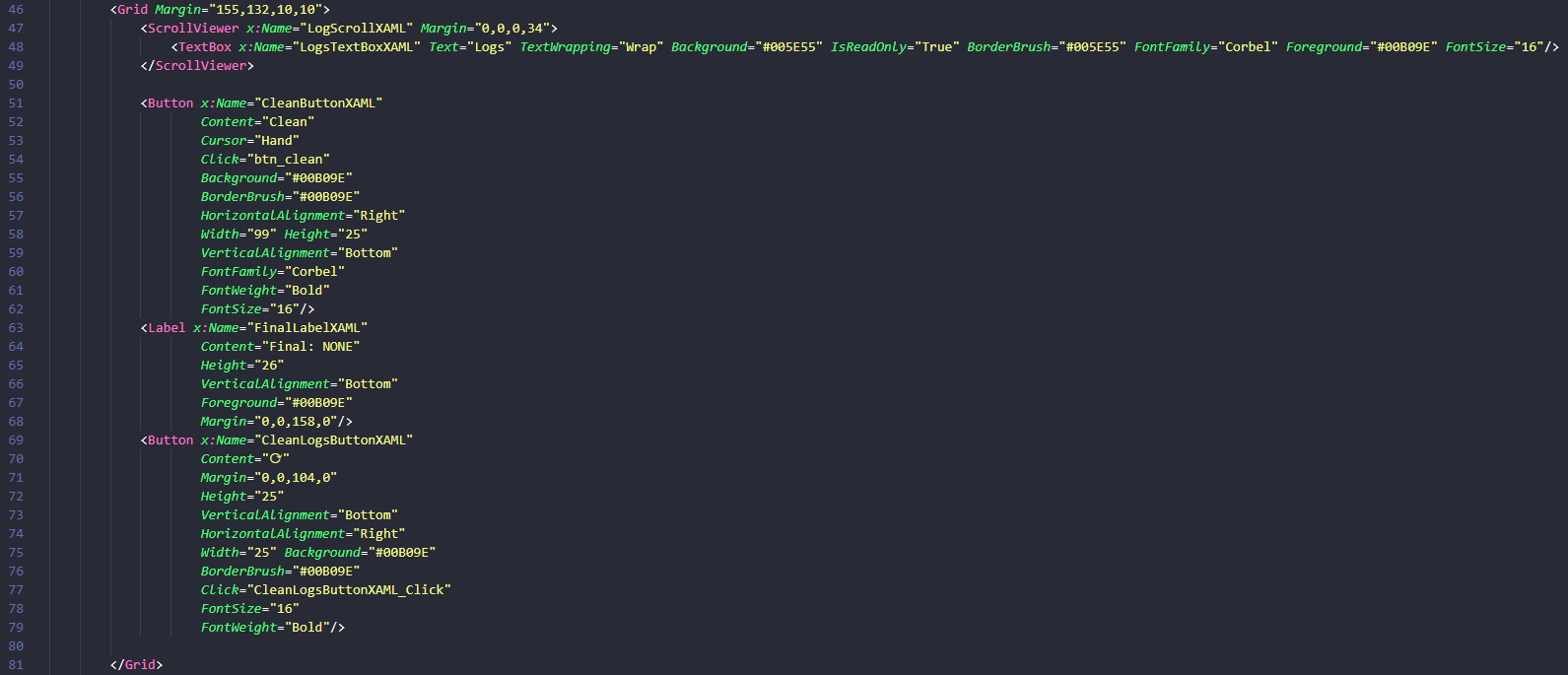


Рисунок 10 – Панель логов и кнопка начала

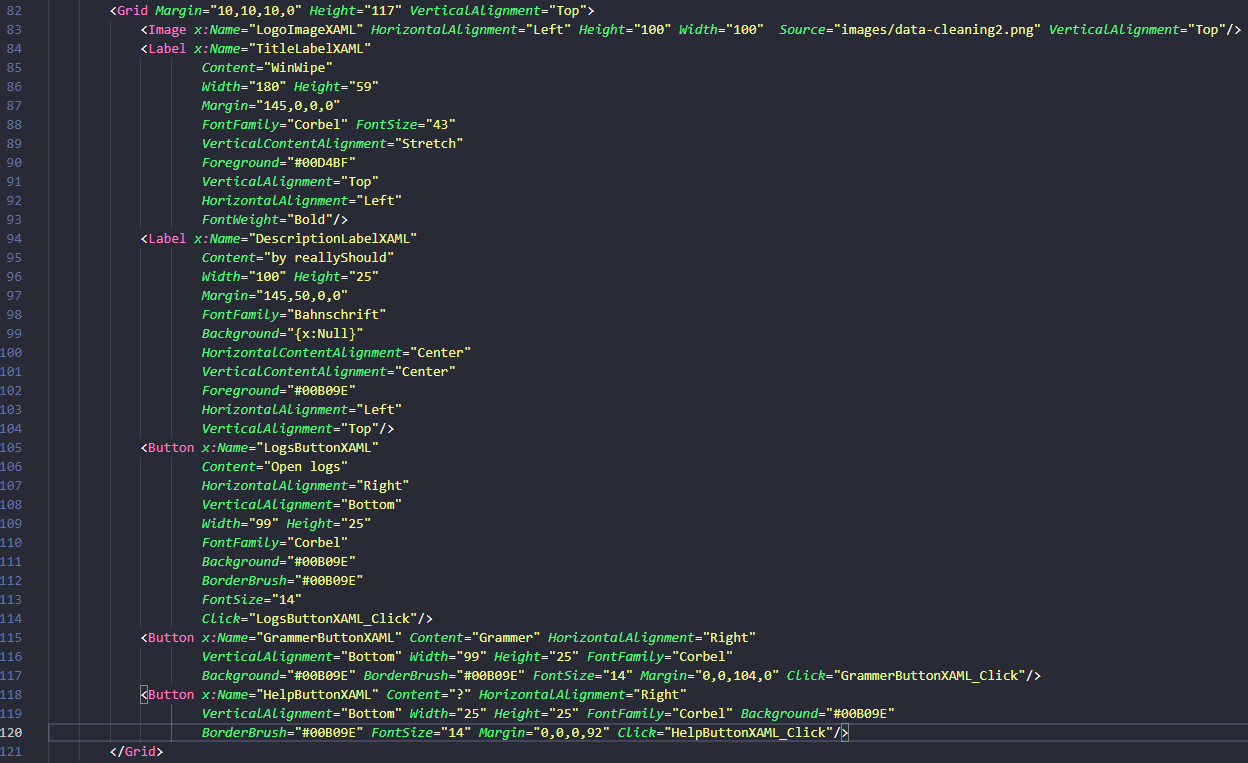


Рисунок 11 – Отрисовка логотипа и заголовка программы

После создания интерфейса понадобится класс, который будет определять пути логов, статус пользователя, диск с файлами пользователя и запись самих логов. Так же он будет переводить единицы измерения файловой системы, определять установленные программы и считать общий размер очищенных данных. Назовем его SystemAdd и подключим все пространства имен (смотреть рисунок 12).

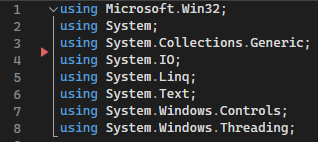


Рисунок 12 – Пространства имен

Начнем с инициализации класса и обозначения основных переменных (смотреть рисунок 13), напишем функцию init(), в которой будут задаваться нужные переменные для работы программы, а так же создаваться папки для логов (смотреть рисунок 14). Этот класс будет помещен в отдельную папку libs для удобства.

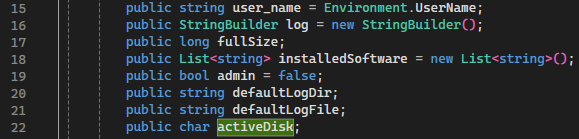


Рисунок 13 – Основные переменные

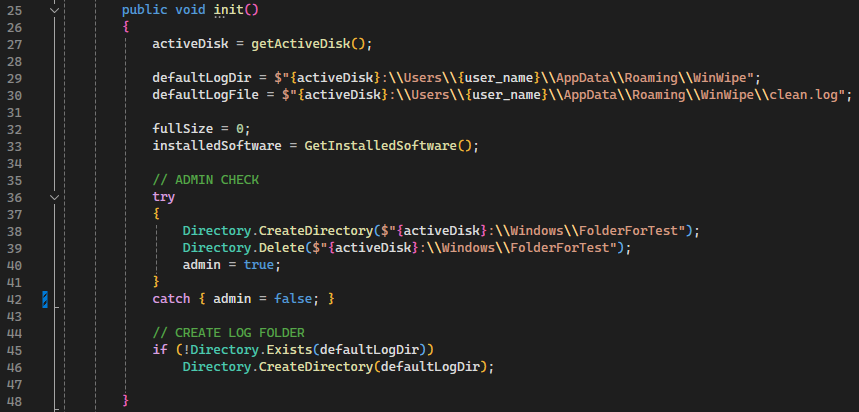


Рисунок 14 – Функция init()

Напишем функции для поиска диска с данными пользователя (смотреть рисунок 15) и установленных программ (смотреть рисунок 16).

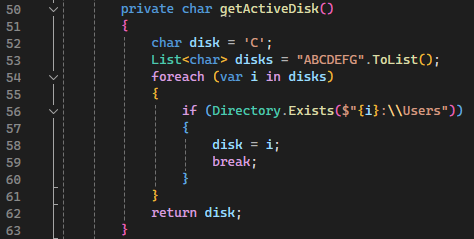


Рисунок 15 – Функция поиска диска с данными пользователя

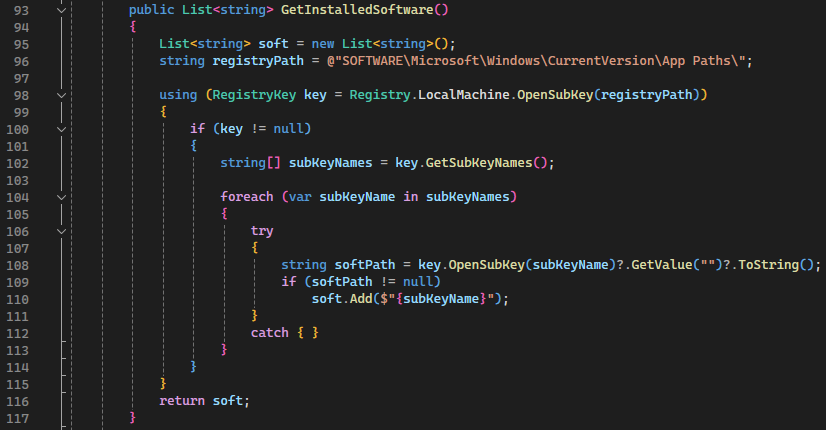


Рисунок 16 – Функция поиска установленных программ

Помимо этого, напишем функции, которые пригодятся позже. А именно перевод единиц измерения (смотреть рисунок 17), запись логов (смотреть рисунок 18), методы для считывания удаленных данных (смотреть рисунок 19).

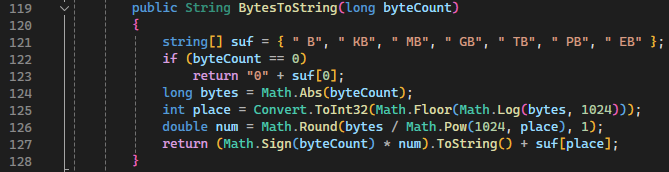


Рисунок 17 – Функция перевода байтов в другие единицы измерения

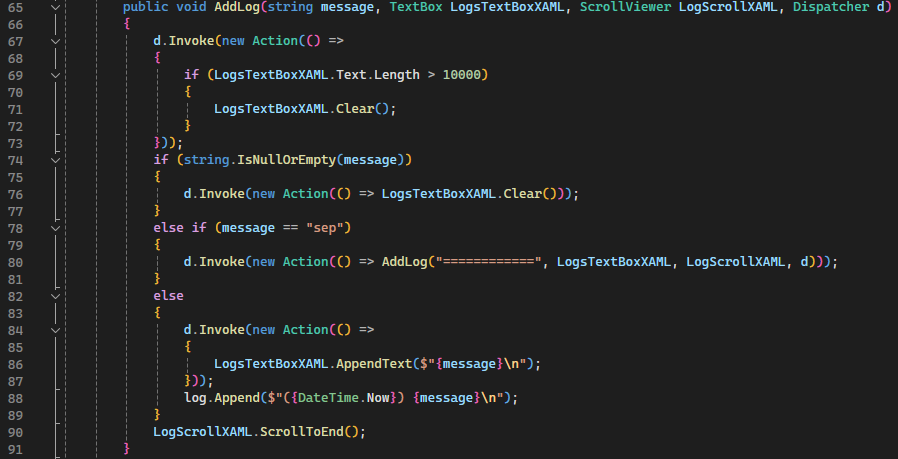


Рисунок 18 – Функция записи логов

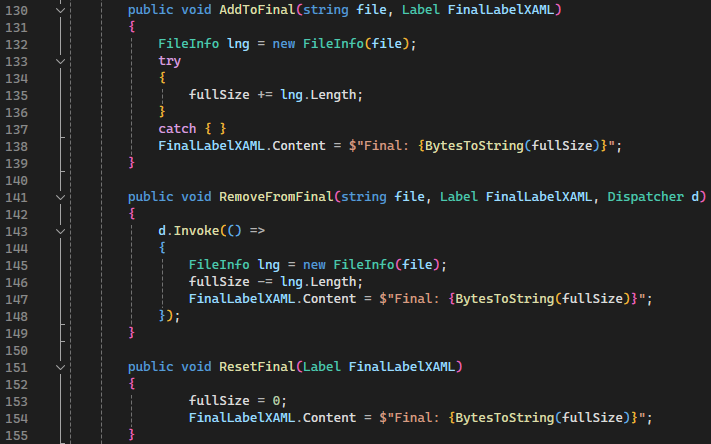


Рисунок 19 – Функции для определения размера очищенных данных

Создадим класс со всеми функциями очистки Cleaner, для этого подключим класс SystemAdd и обозначим все переменные, которые понадобятся для работы функций (смотреть рисунок 20).

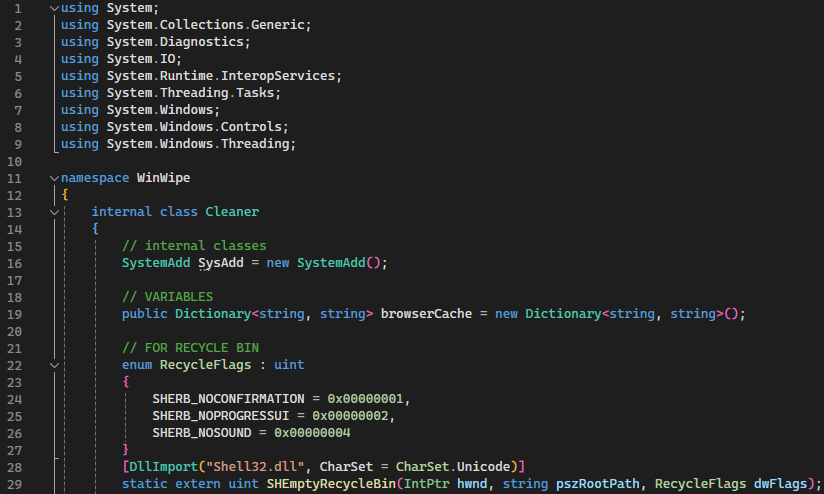


Рисунок 20 – Обозначение переменных в классе Cleaner

Так же, как и для другого класса SystemAdd нужно добавить метод, который будет задавать переменным значения. В нем будут заданы пути до кэша браузеров и инициализация SystemAdd (смотреть рисунок 21).

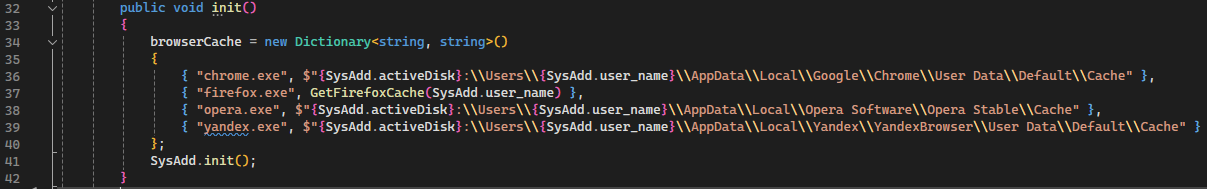


Рисунок 21 – Функция инициализации

Если посмотреть на указанные пути браузер (смотреть рисунок 21), то можно увидеть, что для браузера Firefox указана функция. Дело в том, что этот браузер записывает все данные пользователя в случайно сгенерированные папки, названия которых нельзя знать заранее, и этих папок может быть большое множество, зависит от количества профилей пользователя, нам же нужна только одна папка с основным профилем пользователя, для этого была написана функция, которая будет возвращать название этой папки (смотреть рисунок 22).



Рисунок 22 – Функция поиска кэша Firefox

Теперь можно написать главное, начнем с функции, которая будет удалять папки и файлы внутри рекурсивно, помимо этого будут записываться лог-сообщения для будущего анализа очистки. Большинство действий требует использовать Dispatcher, который уместит поток в основной процесс (смотреть рисунок 23).

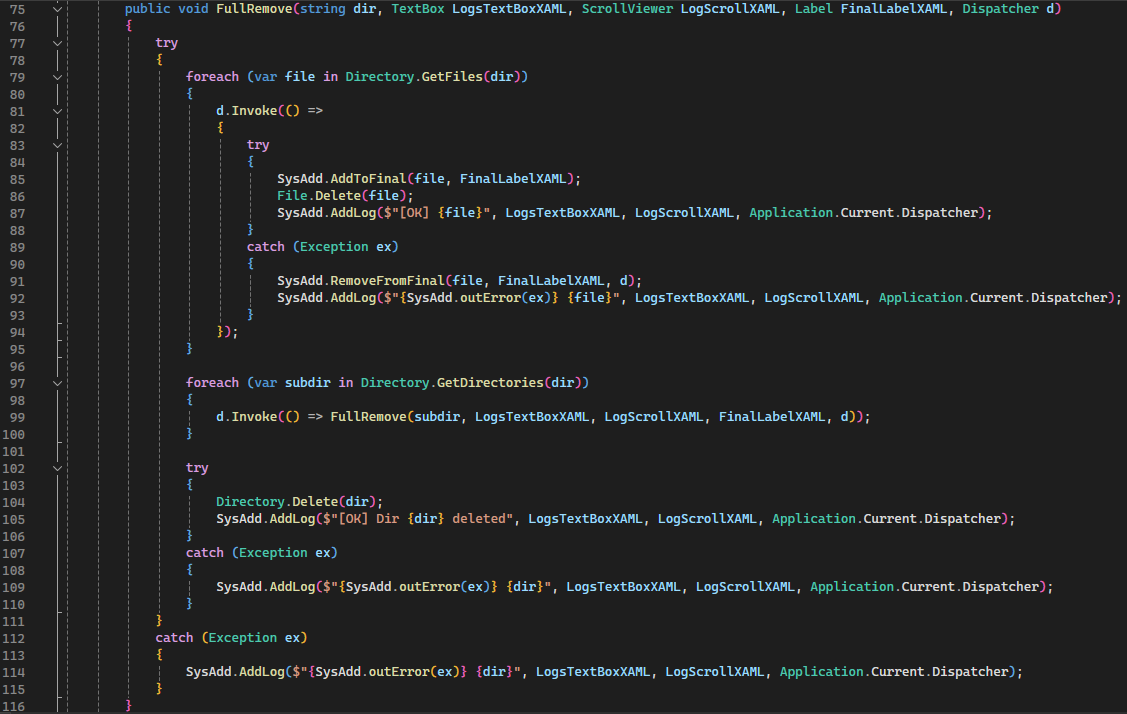


Рисунок 23 – Функция удаления

Далее были написаны функции очистки основных мест скопления мусора, например, локальная и глобальная папка Temp (смотреть рисунок 24), корзина (смотреть рисунок 25), старые версии Windows (смотреть рисунок 26).

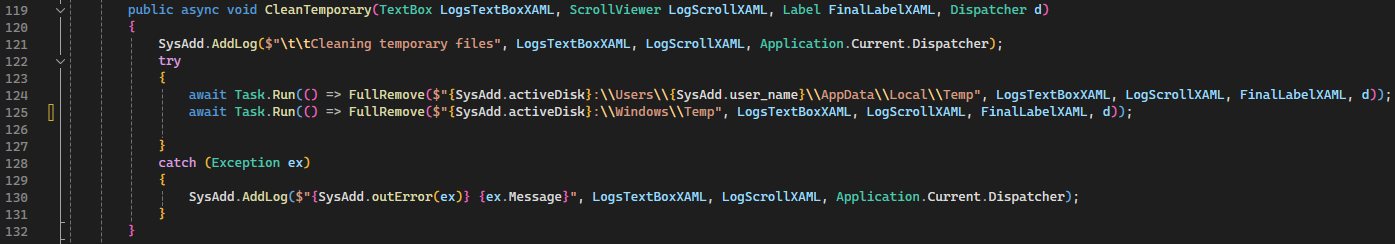


Рисунок 24 – Очистка временных файлов

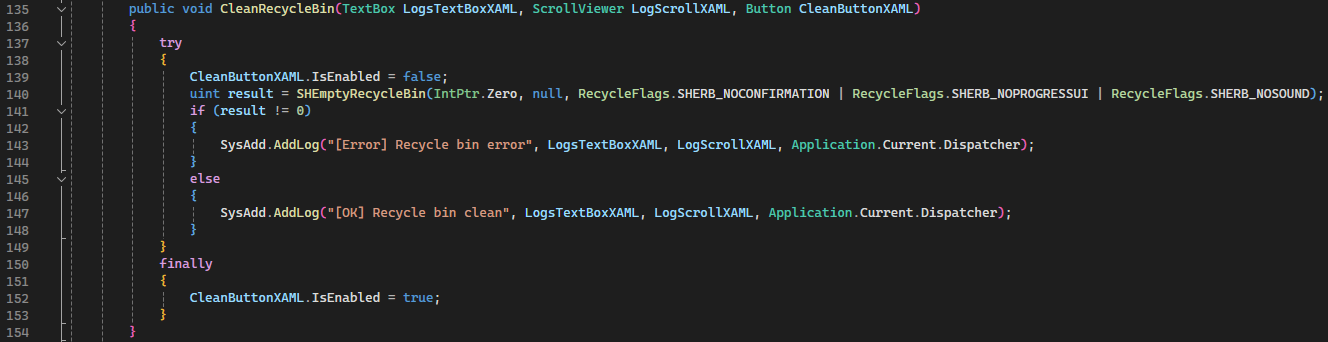


Рисунок 25 – Очистка корзины

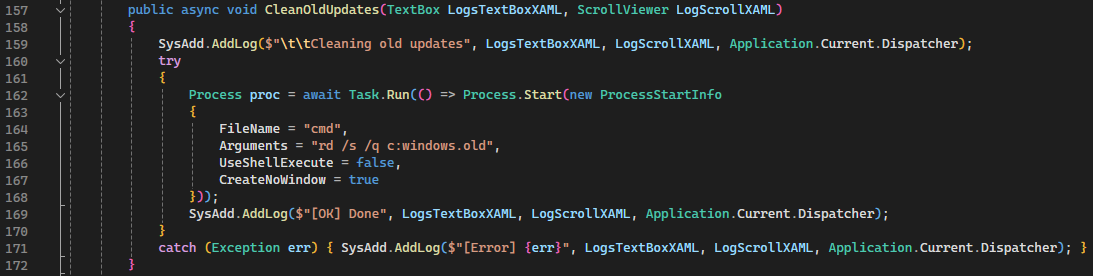


Рисунок 26 – Очистка старых версий Windows

Так же нужно очищать кэш браузера, для этого сверяемся с установленными программами и очищаем нужную папку. Для этого используются переменные, которые были обозначены ранее (смотреть рисунок 20, 21). Напишем функцию (смотреть рисунок 27).

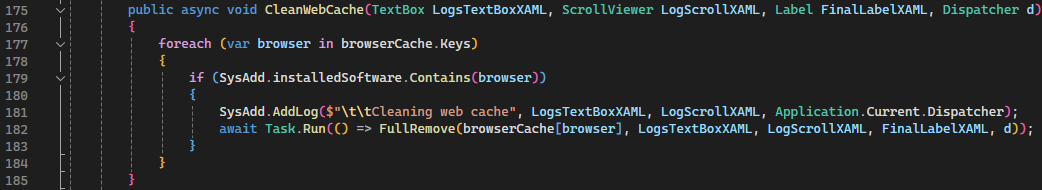


Рисунок 27 – Очистка кэша браузера

У некоторых пользователей, которые часто что-то скачивают - забита папка загрузок разными ненужными файлами, например ненужными документами, фотографиями, торрент файлами и т.д. Иногда количество этих файлов может достигать большого числа, для этого было решено написать функцию, которая будет удалять файлы исходя из расширения файла. Она будет принимать список этих файлов и удалять их (смотреть рисунок 28).

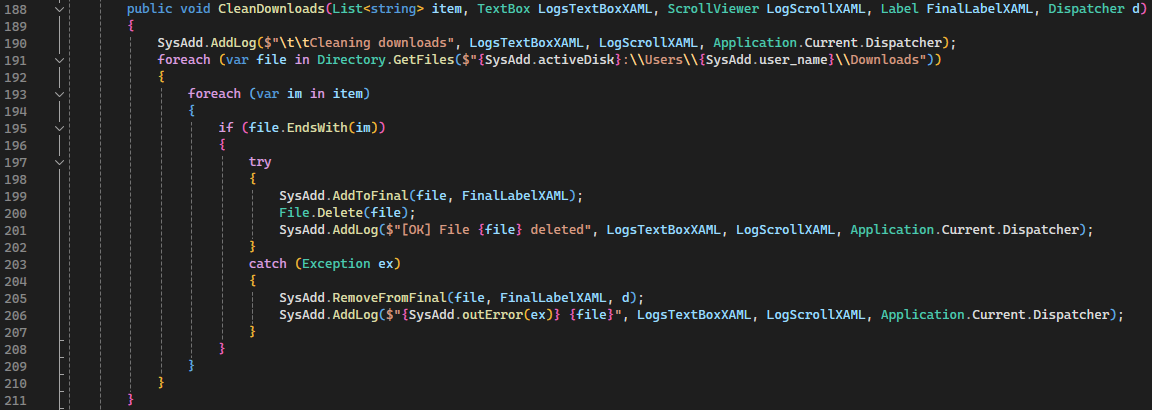


Рисунок 28 – Функция очистки загрузок

Все основные функции написаны, поэтому можно начать подключать все к нашему окну. Для этого нужно инициализировать все классы и обозначить переменные, в которых будет находится информация о версии и файлах, которые нужно очистить (смотреть рисунок 29). Любые ошибки будут обрабатываться с помощью конструкции try-catch и выводится на экран.

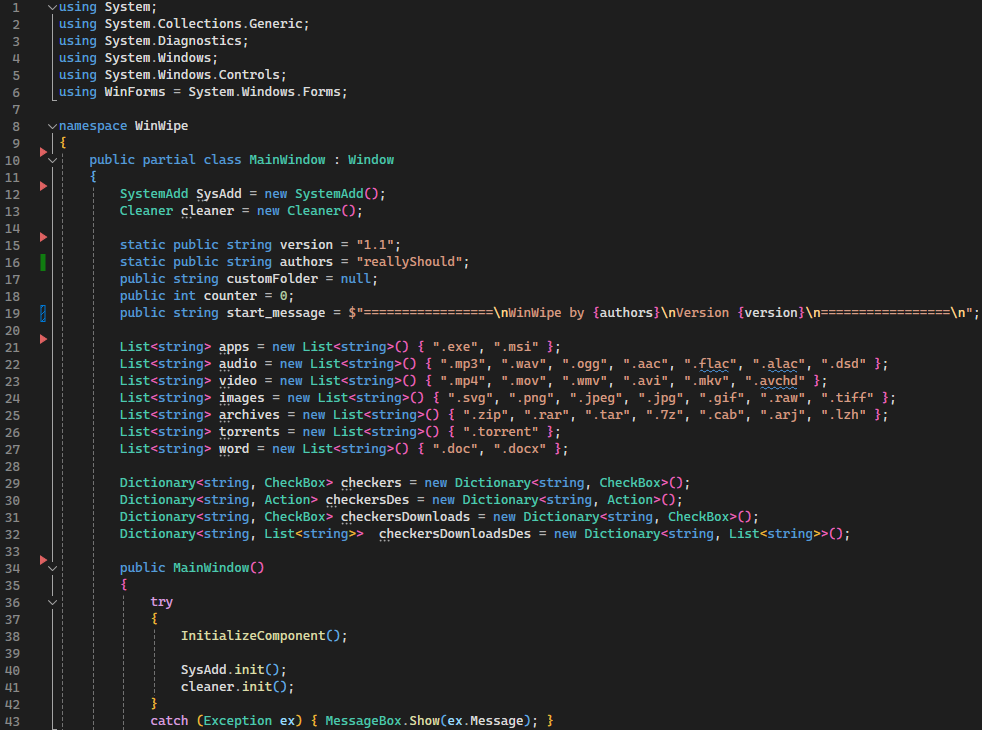


Рисунок 29 – Обозначение переменных главного окна

Что бы подключить флаги очистки было сделано несколько словарей (смотреть рисунок 30), которые позволяют не повторять один и тот же код несколько раз и не использовать огромные условия.

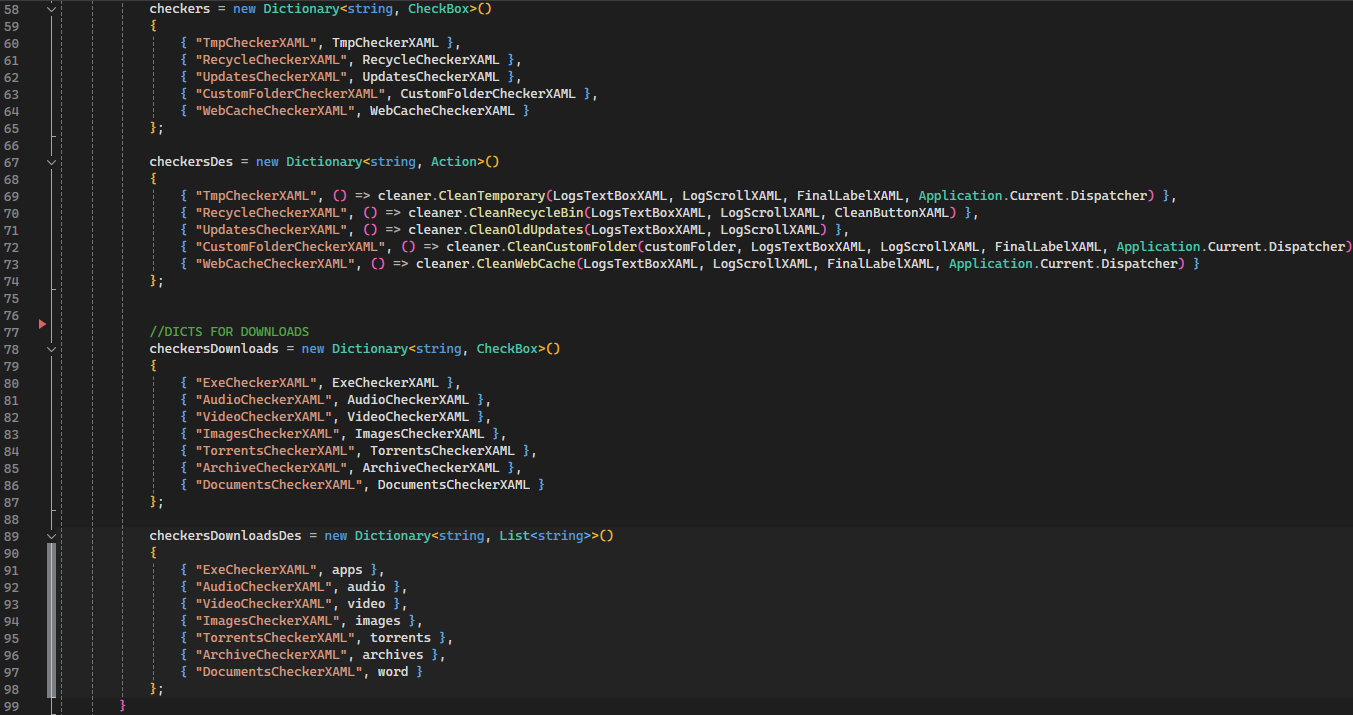


Рисунок 30 – Словари флагов

Теперь подключим самую главную кнопку очистки и запишем в нее очистку по выделенным флагам, сначала очистим основные флаги (смотреть рисунок 31), затем и загрузки (смотреть рисунок 32).

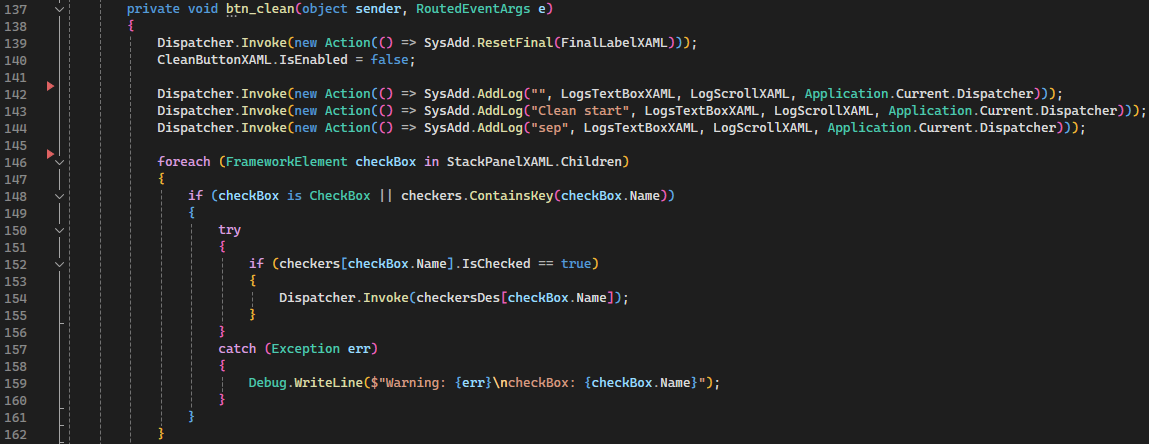


Рисунок 31 – Очистка основных флагов

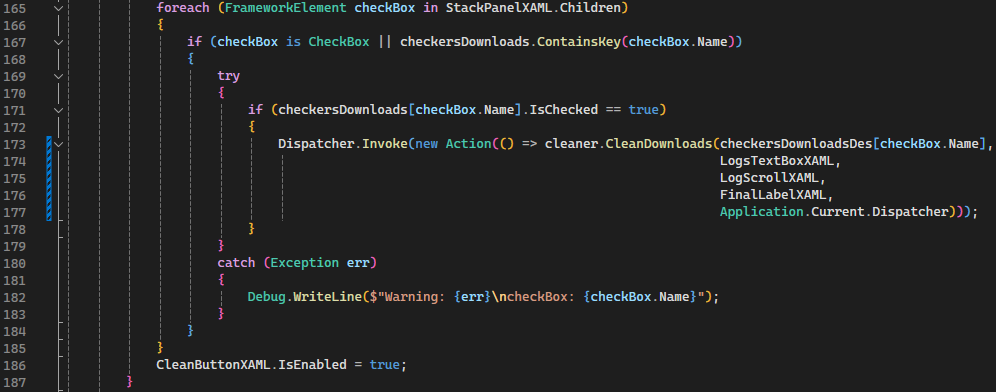


Рисунок 32 – Очистка флагов загрузок