Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**РАЗРАБОТКА ПРОГРАММЫ БЫСТРОГО**

**СКАЧИВАНИЯ СТРАНИЦ ИНТЕРНЕТА**

БГУИР КП 6-05-0612-01-025 ПЗ

Студент Крукович А.В.

Руководитель Болтак С.В.

Минск 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc199096055)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc199096056)

[1.1 Обзор аналогов 6](#_Toc199096057)

[1.2 Постановка задачи 8](#_Toc199096058)

[2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 9](#_Toc199096059)

[2.1 Структура программы 9](#_Toc199096060)

[2.2 Проектирование интерфейса программного средства 10](#_Toc199096061)

[2.3 Проектирование функционала программного средства 14](#_Toc199096062)

[3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА](#_Toc199096063) 26

[3.1 Организация процесса загрузки и обработки веб-страниц](#_Toc199096064) 26

[3.2 Обработка HTML-документа и встраивание ресурсов 2](#_Toc199096065)8

[3.3 Сохранение результата и обратная связь](#_Toc199096066) 30

[3.4 Использование внешних библиотек](#_Toc199096067) 31

[4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА](#_Toc199096068) 33

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ](#_Toc199096069) 33

[5.1 Интерфейс программного средства](#_Toc199096070) 33

[5.2 Управление программным средством](#_Toc199096071) 34

[ЗАКЛЮЧЕНИЕ 3](#_Toc199096073)6

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 3](#_Toc199096074)7

[ПРИЛОЖЕНИЕ А. Исходный код программы 3](#_Toc199096075)8

# ВВЕДЕНИЕ

**Современные пользователи интернета ежедневно сталкиваются с необходимостью сохранять полезную информацию из различных источников — от технической документации и научных статей до страниц новостей и рецептов. Однако далеко не всегда доступ к этим материалам возможен в любой момент: нестабильное подключение к сети, удаление страниц, платные ограничения или просто желание работать офлайн — всё это делает задачу скачивания интернет-страниц актуальной.**

Большинство существующих решений, таких как расширения браузера или офлайн-читалки, ориентированы на сохранение одной страницы вручную, либо на загрузку целого сайта, что занимает значительное время и ресурсы. В условиях, когда важна именно скорость получения данных, такие инструменты не всегда удобны.

В рамках данной курсовой работы разработано программное средство для быстрого скачивания страниц интернета, которое позволяет пользователю быстро сохранить нужную веб-страницу в локальное хранилище с возможностью последующего просмотра без подключения к сети. Приложение ориентировано на простоту, скорость и сохранение только действительно нужных элементов: текстового контента и ключевых медиафайлов, без лишнего «веса».

Основное внимание в ходе разработки уделялось эффективности загрузки, устойчивости к ошибкам сети, обработке ссылок и структуре данных при сохранении. Также была реализована возможность хранения нескольких скачанных страниц, их переименования и управления списком сохранений.

Результатом работы стало компактное и удобное веб-приложение, позволяющее в один клик получить локальную копию страницы интернета, пригодную для чтения в любое время.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Обзор аналогов

**На сегодняшний день существует несколько типов программных средств, позволяющих сохранять содержимое веб-страниц для офлайн-доступа. Они различаются по уровню функциональности, удобству и степени сохранения исходного вида страницы. Ниже рассмотрены наиболее известные из них.**

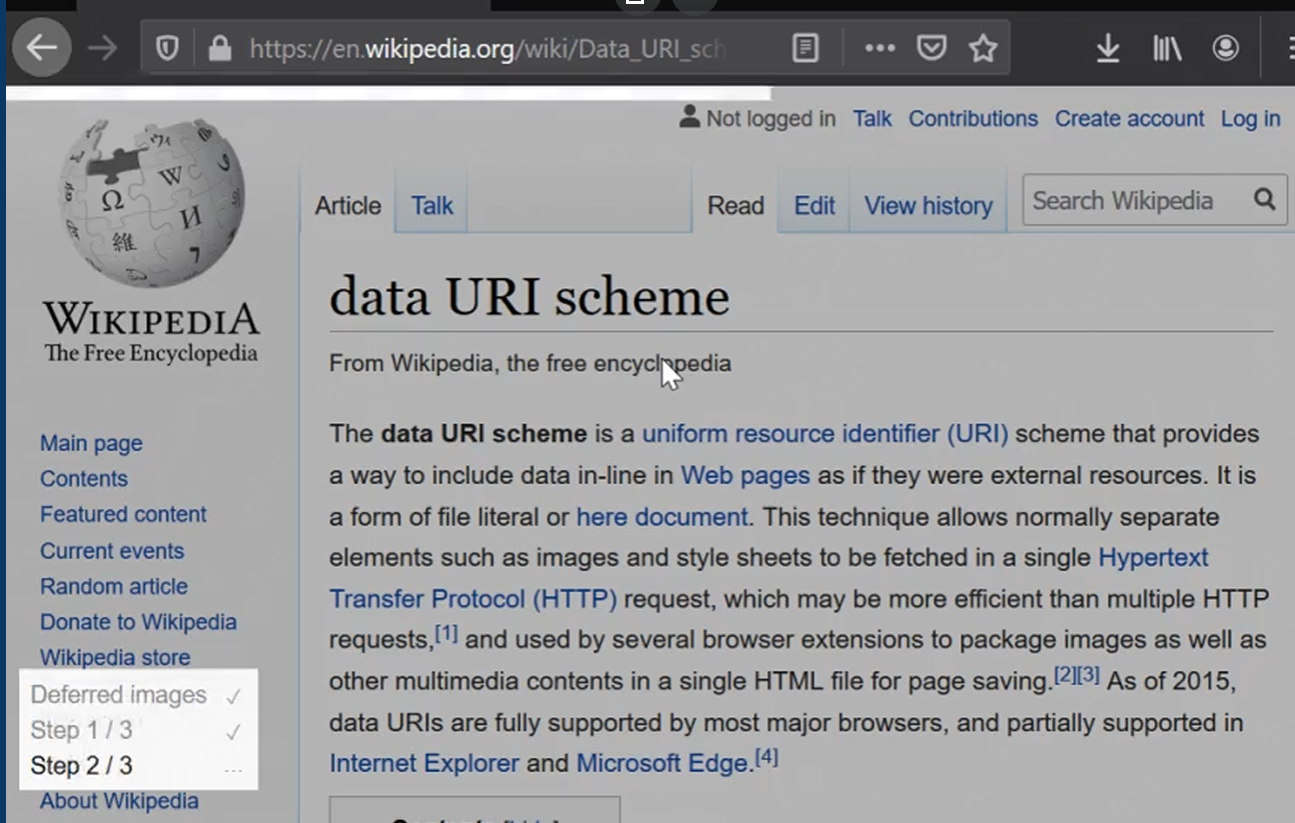
**Расширение для браузеров — SingleFile**

**Это расширение позволяет сохранить всю веб-страницу в один HTML-файл, в который встроены изображения, стили и даже шрифты. Работает в Google Chrome, Firefox и других Chromium-браузерах.**

**Преимущества:**

* **Всё содержимое сохраняется в один файл;**
* **Поддерживает JavaScript и интерактивность;**
* **Имеет расширенные настройки.**

**Недостатки:**

* **Требует ручного запуска для каждой страницы;**
* **Неудобно при массовом сохранении;**
* **Может некорректно работать на тяжёлых или сильно защищённых сайтах;**
* **Не имеет отдельного приложения или инструмента для пакетной загрузки.**

**Рисунок 1.1 – Работа расширения SingleFile.**

**HTTrack Website Copier**

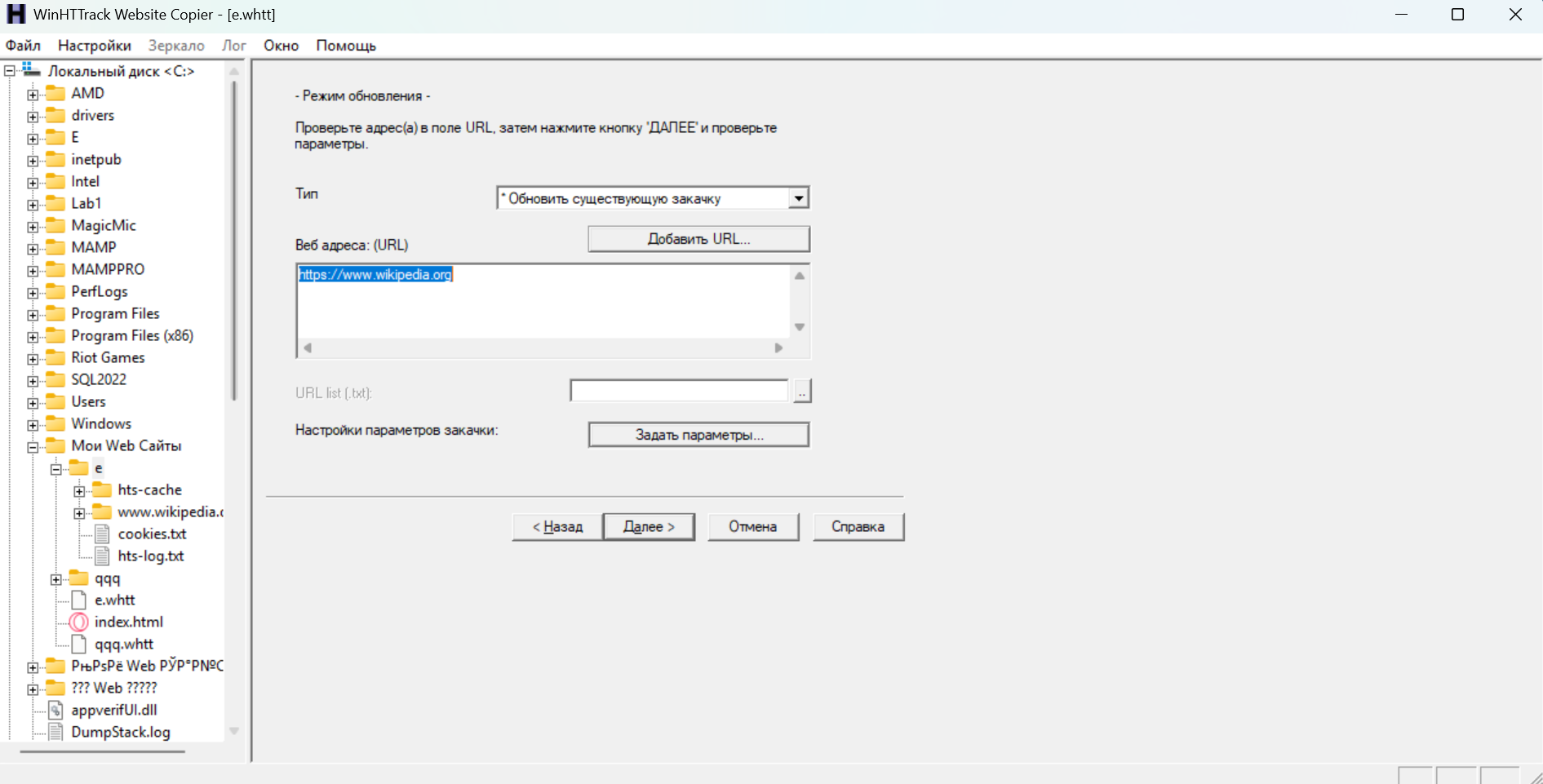
**Это мощная настольная программа, предназначенная для скачивания целых сайтов. Пользователь задаёт URL, и утилита создаёт полную локальную копию с иерархией, изображениями, файлами и переходами между страницами.**

**Преимущества:**

* **Возможность глубокой загрузки всех связанных страниц;**
* **Гибкая настройка (глубина обхода, фильтры, домены);**
* **Удобен для создания резервных копий сайтов.**

**Недостатки:**

* **Медленная работа на больших сайтах;**
* **Низкая применимость для скачивания одной страницы;**
* **Сложный и устаревший интерфейс;**
* **Требует знания структуры сайта для точной настройки.**

****

**Рисунок 1.2 – Приложение HTT Website Copier**

**Таким образом, существующие решения обладают схожими базовыми возможностями, но отличаются возможностями скачивания нескольких сайтов либо удобностью интерфейса. На основе анализа аналогов можно выделить ключевые требования к разрабатываемой программе: скорость в ущерб красоте, возможность скачивания нескольких сайтов и простота интерфейса.**

## Постановка задачи

**Целью данной курсовой работы является разработка приложения, позволяющего быстро загружать и сохранять отдельные интернет-страницы в виде локальных HTML-копий с базовыми ресурсами, пригодными для просмотра в офлайн-режиме.**

* **Программа должна обеспечивать:**
* **ввод URL-адреса страницы пользователем;**
* **извлечение основной разметки и связанных ресурсов (изображения, стили);**
* **сохранение полученных данных в локальное хранилище;**
* **отображение списка ранее загруженных страниц;**
* **возможность переименовывать и удалять сохранения;**
* **быстрый просмотр скачанных страниц в браузере;**
* **устойчивость к сетевым ошибкам и недоступности страниц;**
* **простой и понятный интерфейс.**

**Ключевыми требованиями являются: простота, скорость загрузки, устойчивость к нестандартной разметке сайтов и нацеленность на одну страницу, а не весь сайт. Разработка предполагается с использованием Python и библиотеки для парсинга HTML (например, BeautifulSoup), а также фреймворка Flask или FastAPI для создания минимального веб-интерфейса.**

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 2.1 Структура программы

**При разработке приложения использовались следующие основные компоненты и модули:**

* **Form1.cs – основной модуль приложения, содержащий графический интерфейс и управляющую логику;**
* **HttpClient – модуль, осуществляющий загрузку HTML-страниц и ресурсов по сети;**
* **AngleSharp.Html.Parser – модуль, обеспечивающий парсинг HTML-документа в объектную модель;**
* **AngleSharp.Dom – модуль, предоставляющий интерфейсы для работы с HTML-элементами;**
* **AngleSharp.Html.Dom – модуль, позволяющий доступ к специфике тегов <link>, <img> и др.;**
* **DownloadPageInternalAsync – метод, реализующий загрузку HTML-страницы и первичную обработку ответа;**
* **EmbedCssAsync – метод, выполняющий извлечение и встраивание CSS в HTML-документ;**
* **EmbedImagesAsync – метод, отвечающий за попытку встраивания изображений в формате Base64;**
* **GenerateFileNameFromUrl – метод, генерирующий корректные имена файлов на основе URL-адресов;**
* **LogMessage – метод, отображающий статусные сообщения в интерфейсе;**
* **FolderBrowserDialog – компонент, позволяющий пользователю выбрать путь для сохранения;**
* **ProgressBar – компонент, визуализирующий ход выполнения загрузки;**
* **ListBox (lstLog) – компонент для отображения логов во время работы;**
* **User-Agent Header – конфигурация заголовка запросов, чтобы избежать блокировки от серверов.**

**Программа реализована как десктопное приложение на Windows Forms, и все действия пользователя и состояния системы отражаются в интерфейсе в режиме реального времени. Взаимодействие с интернетом, парсинг, обработка контента и сохранение файлов организованы с использованием асинхронных методов, обеспечивающих стабильную и отзывчивую работу программы.**

## 

## 2.2 Проектирование интерфейса программного средства

**При разработке программного средства FastPageDownloader1 за основу был взят типовой для настольных приложений Windows Forms подход, обеспечивающий интуитивно понятное взаимодействие пользователя с функциями загрузки и сохранения веб-страниц. Интерфейс спроектирован с акцентом на простоту и эффективность выполнения основной задачи – скачивания веб-контента.**

**2.2.1 Главное окно приложения**

**Главное окно приложения FastPageDownloader1 (см. Рисунок 2.1) предоставляет пользователю все необходимые инструменты для загрузки веб-страниц. Оно логически разделено на несколько функциональных областей:**

1. **Область ввода URL-адресов:**

* **Многострочное текстовое поле (txtUrls), предназначенное для ввода одного или нескольких URL-адресов веб-страниц, которые необходимо скачать. Каждый URL-адрес вводится с новой строки.**

1. **Область выбора папки сохранения:**

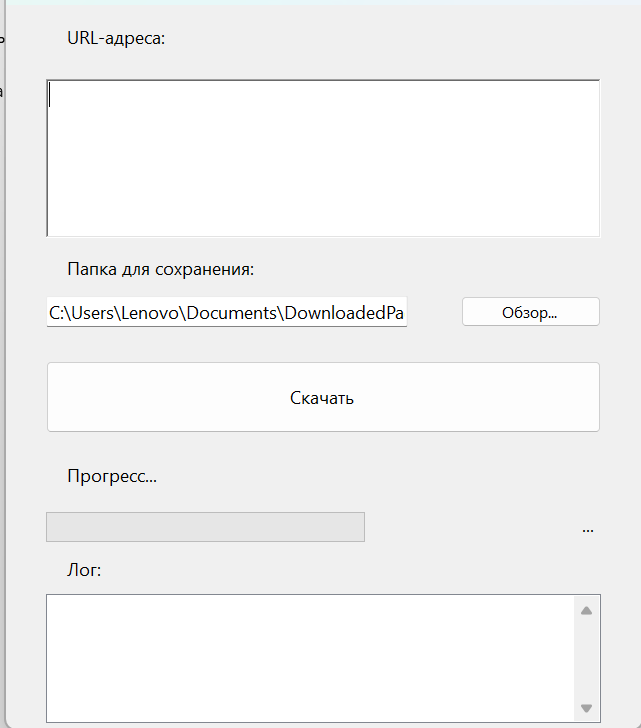
* **Текстовое поле (txtSaveFolder), отображающее текущий путь к папке, в которую будут сохраняться загруженные файлы. По умолчанию предлагается папка "DownloadedPages" в "Документах" пользователя.**
* **Кнопка «Выбрать папку» (btnBrowseFolder), позволяющая пользователю изменить папку назначения через стандартный диалог выбора каталога.**

1. **Область управления загрузкой и отображения статуса:**

* **Кнопка «Скачать» (btnDownload), запускающая процесс загрузки и обработки указанных URL-адресов.**
* **Индикатор прогресса (progressBar), визуально отображающий общий прогресс выполнения задачи по всем указанным URL.**
* **Текстовая метка статуса (lblStatus), информирующая пользователя о текущем состоянии процесса (например, «Скачано X из Y»).**

1. **Область журнала сообщений:**

* **Список (lstLog), в который выводятся подробные сообщения о ходе выполнения операций: начало загрузки каждого URL, статусы HTTP-запросов, сообщения об ошибках, информация о встраивании CSS и изображений, а также путь к сохраненному файлу. Каждое сообщение снабжается временной меткой.**

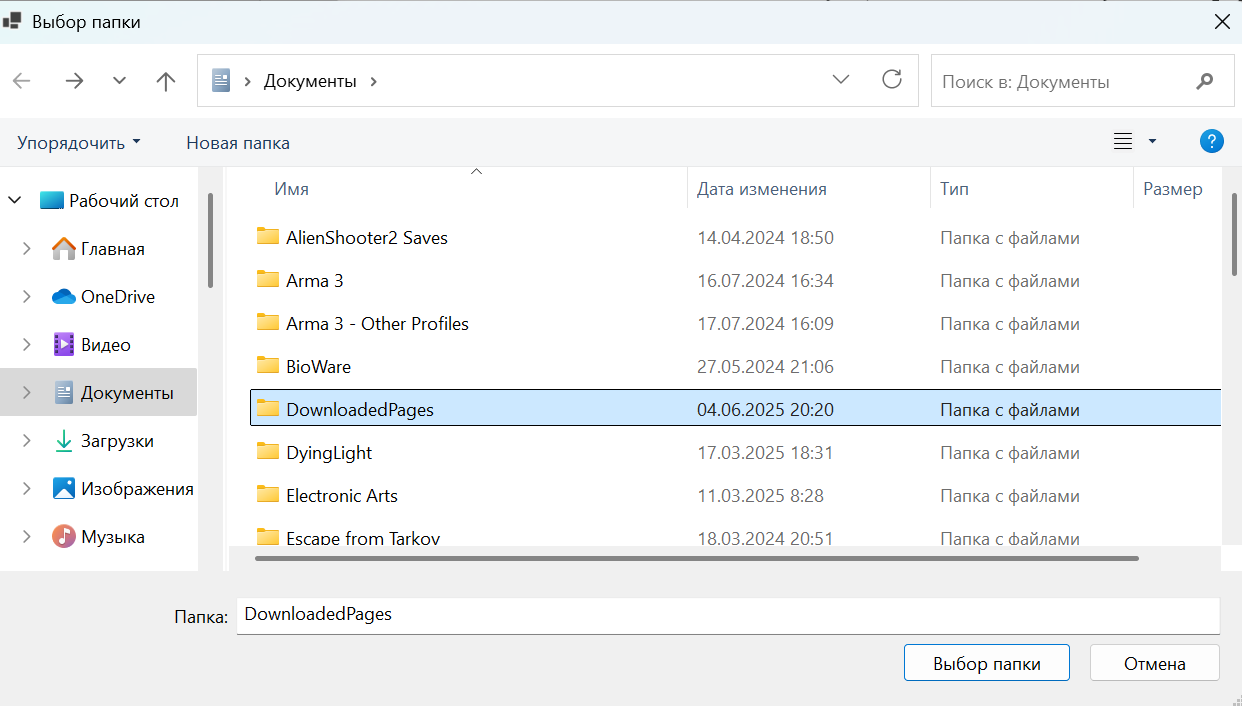
****

**Рисунок 2.1 – Главное окно**

2.2.2 **Диалоговые окна и уведомления**

**Для обеспечения интерактивности и информирования пользователя FastPageDownloader1 использует стандартные диалоговые окна и системные уведомления Windows Forms:**

* **Диалог выбора папки сохранения: При нажатии на кнопку «Выбрать папку» (btnBrowseFolder) пользователю отображается стандартный системный диалог FolderBrowserDialog. Этот элемент интерфейса позволяет пользователю наглядно выбрать или создать директорию на локальном диске для сохранения загруженных веб-страниц. Выбранный путь немедленно отображается в текстовом поле txtSaveFolder, предоставляя пользователю подтверждение его выбора. (См. Рисунок 2.2 – Диалог выбора папки для сохранения).**
* **Информационные сообщения и предупреждения: Для обратной связи с пользователем о результатах операций, ошибках или необходимости выполнить определенные действия, приложение использует модальные окна MessageBox. Эти окна требуют внимания пользователя и подтверждения перед продолжением работы:**
* **Предупреждение об отсутствии URL: Если пользователь инициирует загрузку без указания URL-адресов в поле txtUrls, выводится сообщение «Пожалуйста, введите хотя бы один URL.» с иконкой предупреждения, предотвращая запуск пустого процесса.**
* **Предупреждение об отсутствии папки сохранения: Аналогично, если поле txtSaveFolder не заполнено, отображается сообщение «Пожалуйста, выберите папку для сохранения.».**
* **Уведомление об ошибке создания папки: В случае, если приложение не может создать указанную папку сохранения (например, из-за ограничений прав доступа или недействительного пути), пользователь получает информативное сообщение об ошибке вида «Не удалось создать папку: [текст системной ошибки]».**
* **Уведомление о завершении загрузки: По успешному окончании всех запланированных операций загрузки и обработки страниц, пользователю выводится подтверждающее сообщение «Скачивание завершено!» с информационной иконкой..**

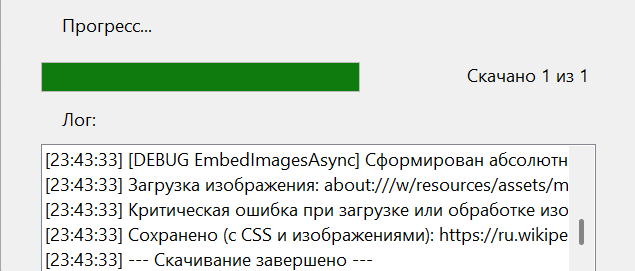
****

**Рисунок 2.2 – Диалоговое окно выбора папки для сохранения**

2.2.3 **Обратная связь и управление процессом загрузки**

**Программное средство FastPageDownloader1 предоставляет пользователю исчерпывающую обратную связь на всех этапах процесса загрузки и обработки веб-страниц, а также реализует механизмы управления этим процессом для обеспечения стабильности работы:**

* **Визуализация прогресса выполнения:**
* **Индикатор прогресса (progressBar): Динамически отображает общий прогресс выполнения всех задач по загрузке и обработке URL-адресов. Его длина заполняется пропорционально количеству завершенных операций от общего числа URL, указанных пользователем.**
* **Текстовая метка статуса (lblStatus): Дополняет индикатор прогресса, предоставляя количественную информацию в формате «Скачано X из Y», где X – количество обработанных URL, а Y – общее количество. Обновляется синхронно с progressBar.**
* **Детализированный журнал операций (lstLog):**
* **В реальном времени в список lstLog выводятся подробные сообщения о ходе выполнения каждой операции. Это включает:**
* **Время начала загрузки для каждого URL.**
* **Статус HTTP-запроса (например, успешное получение, код ошибки).**
* **Ключевые этапы обработки контента, такие как начало и завершение внедрения CSS и изображений.**
* **Сообщения об ошибках, возникших при парсинге HTML, загрузке ресурсов или других непредвиденных ситуациях, с указанием проблемного URL.**
* **Информацию о конечном имени файла и пути его сохранения.**
* **Каждое сообщение в журнале снабжается временной меткой ([ЧЧ:мм:сс]), что облегчает анализ последовательности событий и диагностику возможных проблем. Список автоматически прокручивается для отображения последних сообщений.**
* **Управление доступностью элементов интерфейса:**
* **Во время активного процесса загрузки и обработки данных (после нажатия кнопки «Скачать»), основные управляющие элементы интерфейса – поле ввода URL (txtUrls), кнопка выбора папки (btnBrowseFolder) и сама кнопка «Скачать» (btnDownload) – временно деактивируются (Enabled = false). Это предотвращает случайные или некорректные действия пользователя, которые могли бы нарушить текущий процесс.**
* **По завершении всех операций (успешном или с ошибками) эти элементы управления вновь становятся активными, позволяя пользователю инициировать новую сессию загрузки или скорректировать параметры.**

****

**Рисунок 2.3 – Обратная связь**

## 2.3 Проектирование функционала программного средства

**Программное средство FastPageDownloader предназначено для автоматизированного скачивания веб-страниц и их ресурсов (CSS, изображения) с целью создания локальных, автономных копий, полностью работоспособных в офлайн-режиме. Для этого оно реализует следующий набор ключевых функций:**

* **Ввод и обработка списка URL-адресов для загрузки.**
* **Выбор и подготовка папки для сохранения файлов.**
* **Асинхронная загрузка основного HTML-контента веб-страниц.**
* **Парсинг HTML-документа и встраивание внешних CSS-стилей.**
* **Встраивание изображений непосредственно в HTML-документ в формате Data URI.**
* **Формирование корректных имен файлов и сохранение результатов.**

2.3.1 **Ввод и обработка списка URL-адресов**

**Пользователь предоставляет приложению список URL-адресов веб-страниц, которые необходимо загрузить. Это осуществляется путем ввода или вставки URL в многострочное текстовое поле txtUrls, где каждый URL располагается на новой строке. Перед началом загрузки приложение выполняет базовую обработку:**

* **Пустые строки и строки, состоящие только из пробельных символов, игнорируются.**
* **Для URL, не имеющих явного указания протокола (http:// или https://), автоматически добавляется префикс http:// (реализовано в методе DownloadPageInternalAsync и косвенно в DownloadPageAsync). Это упрощает ввод для пользователя и обеспечивает корректную работу HttpClient.**

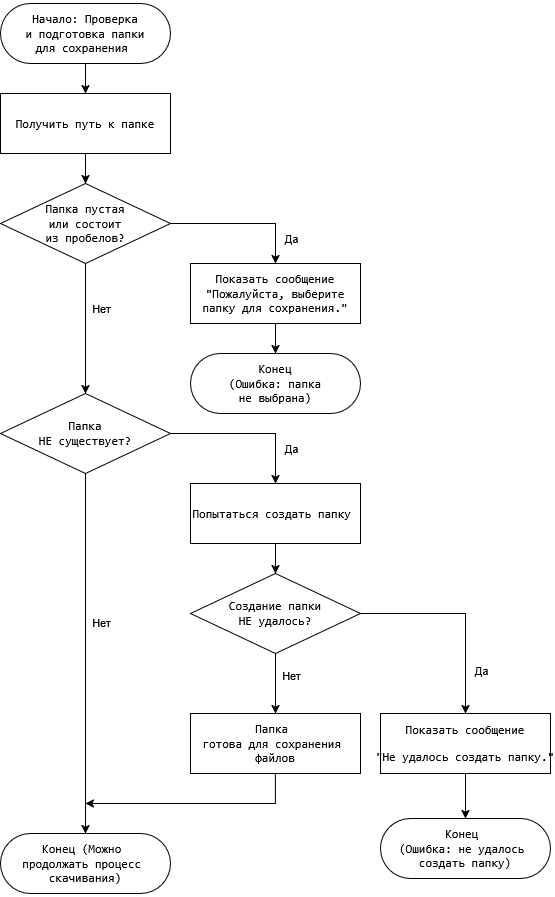
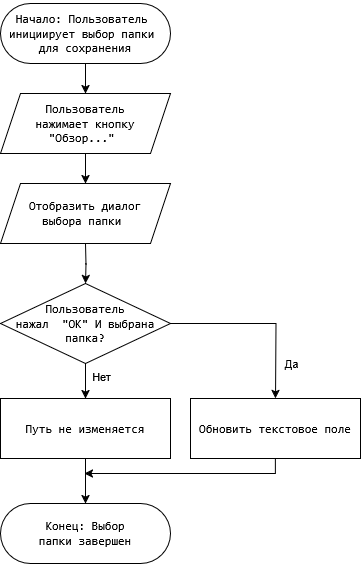
****

**Рисунок 2.4 – Блок-схема обработки URL**

2.3.2 **Выбор и подготовка папки для сохранения**

**Пользователь имеет возможность указать директорию на локальном диске, куда будут сохраняться загруженные и обработанные веб-страницы.**

* **Выбор папки осуществляется через стандартный диалог FolderBrowserDialog, вызываемый кнопкой btnBrowseFolder\_Click.**
* **Выбранный путь отображается в текстовом поле txtSaveFolder. По умолчанию предлагается подпапка "DownloadedPages" в системной папке "Мои документы".**
* **Перед началом процесса загрузки приложение проверяет существование указанной папки. Если папка не существует, предпринимается попытка её создания (в btnDownload\_Click). В случае неудачи (например, из-за отсутствия прав доступа) пользователь уведомляется об ошибке.**

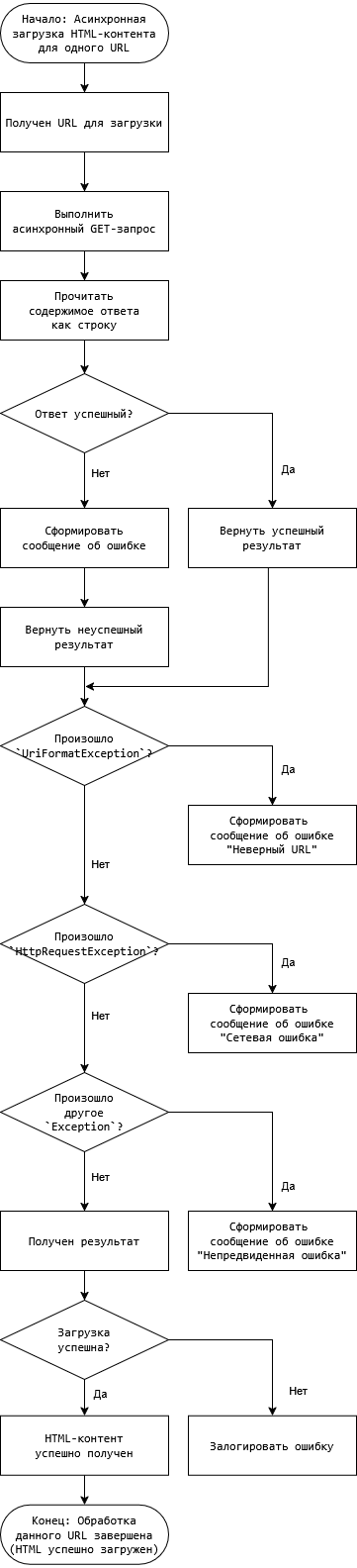
****

**Рисунки 2.5 и 2.6 – Блок-схемы выбора и подготовки папки сохранения**

2.3.3 **Асинхронная загрузка основного HTML-контента**

**Загрузка основного HTML-содержимого каждой указанной веб-страницы выполняется асинхронно для каждого URL, что позволяет обрабатывать несколько страниц параллельно и не блокировать интерфейс пользователя.**

* **Для сетевых запросов используется статический экземпляр HttpClient, что является рекомендуемой практикой для повышения производительности и экономии ресурсов. В конструкторе Form1 устанавливается заголовок User-Agent, имитирующий браузер, для лучшей совместимости с веб-серверами.**
* **Метод DownloadPageInternalAsync отвечает за непосредственное выполнение GET-запроса к указанному URL.**
* **Приложение обрабатывает HTTP-ответы:**
* **В случае успешного ответа (код состояния 2xx), содержимое страницы считывается как строка.**
* **В случае ошибки HTTP (коды 4xx, 5xx), информация об ошибке (код состояния, причина, часть тела ответа) логируется.**
* **Обрабатываются также исключения, связанные с сетевыми проблемами (HttpRequestException) или некорректным форматом URL (UriFormatException).**

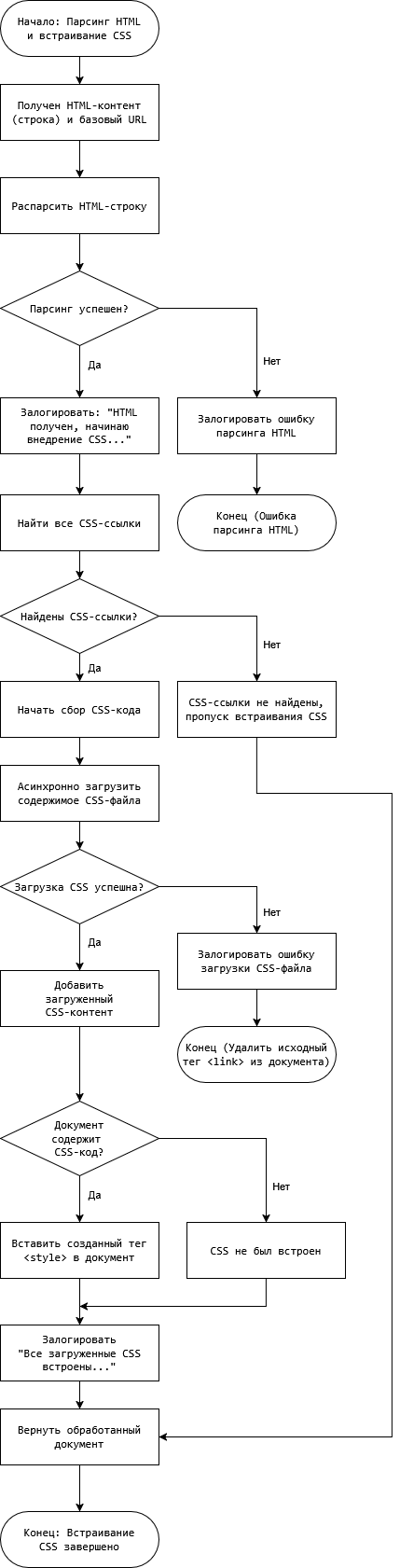
****

**Рисунок 2.7 – Блок-схема загрузки HTML-контента**

2.3.4 **Парсинг HTML-документа и встраивание CSS-стилей**

**После успешной загрузки HTML-содержимого страницы оно подвергается обработке с целью встраивания внешних CSS-стилей для обеспечения корректного отображения в офлайн-режиме.**

* **Для парсинга HTML используется библиотека AngleSharp. Метод HtmlParser.ParseDocumentAsync() преобразует строку HTML в объектную модель документа (DOM).**
* **Метод EmbedCssAsync выполняет следующие действия:**
* **Находит все теги <link> с атрибутами rel="stylesheet" и непустым href в <head> документа.**
* **Для каждой найденной CSS-ссылки:**
* **Формируется абсолютный URL на основе базового URL страницы и относительной ссылки на CSS-файл.**
* **С помощью HttpClient асинхронно загружается содержимое CSS-файла.**
* **В случае успешной загрузки, содержимое CSS-файла добавляется внутрь нового тега <style type="text/css">, который, в свою очередь, вставляется в начало элемента <head> HTML-документа.**
* **Исходный тег <link> удаляется из документа.**
* **Логируются ошибки, возникшие при загрузке или обработке CSS-файлов.**

****

**Рисунок 2.8 – Блок-схема прасинга HTML-документа и встраивания CSS-стилей**

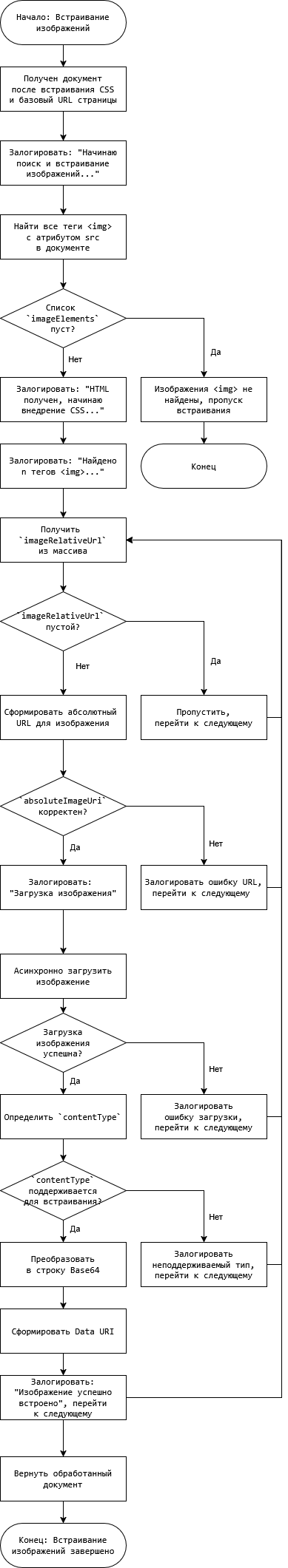
2.3.5 **Встраивание изображений в HTML-документ**

**Аналогично CSS, внешние изображения, на которые ссылаются теги <img>, встраиваются непосредственно в HTML-документ.**

**Метод EmbedImagesAsync отвечает за эту функциональность:**

* **Используя AngleSharp, находятся все теги <img> с непустым атрибутом src.**
* **Изображения, уже представленные в формате Data URI (атрибут src начинается с data:), пропускаются.**
* **Для каждого внешнего изображения:**
* **Формируется абсолютный URL изображения.**
* **Изображение асинхронно загружается с помощью HttpClient как массив байт.**
* **Определяется MIME-тип изображения на основе заголовка Content-Type ответа сервера. Если заголовок отсутствует, предпринимается попытка угадать тип по расширению файла. Поддерживаются распространенные форматы изображений (JPEG, PNG, GIF и т.д.); SVG-изображения (которые могут содержать ссылки на другие ресурсы) и неподдерживаемые типы пропускаются.**
* **Загруженные байты изображения кодируются в строку Base64.**
* **Атрибут src тега <img> заменяется на строку Data URI, содержащую MIME-тип и Base64-представление изображения (например, data:image/png;base64,...).**
* **Логируются ошибки при загрузке или обработке изображений.**

**Блок-схема процесса встраивания изображений представлена на Рисунке 2.9.**

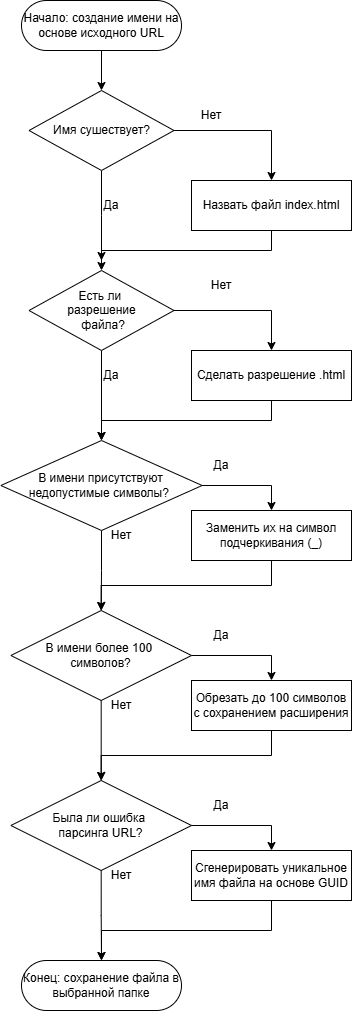
****

**Рисунок 2.9 – Блок-схема процесса встраивания изображений**

2.3.6 **Формирование имени файла и сохранение результата**

**После обработки HTML, встраивания CSS и изображений, итоговый документ сохраняется на локальный диск.**

* **Метод GenerateFileNameFromUrl отвечает за создание имени файла на основе исходного URL:**
* **Из URL извлекается путь и имя файла. Если имя отсутствует (например, для корневого URL домена), используется "index.html".**
* **К имени файла без расширения добавляется ".html", если оно отсутствует.**
* **Недопустимые для имен файлов символы заменяются на подчеркивания (\_).**
* **Имя файла обрезается до 100 символов (с сохранением расширения), чтобы избежать проблем с максимальной длиной пути в файловой системе.**
* **В случае ошибки парсинга URL генерируется уникальное имя файла на основе GUID.**
* **Модифицированный HTML-контент (полученный из объекта IDocument с помощью document.ToHtml()) записывается в файл с сгенерированным именем в выбранную пользователем папку (txtSaveFolder.Text). Запись производится в кодировке UTF-8 для корректного отображения различных символов.**

****

**Рисунок 2.10 – Блок-схема формирования имени файла и встраивания результата**

**\**

# РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

**3.1 Организация процесса загрузки и обработки веб-страниц**

**Основной задачей программного средства является загрузка веб-страниц по указанным URL-адресам, их последующая модификация путем встраивания внешних CSS-стилей и изображений, и сохранение результата в виде единого HTML-файла. Этот процесс управляется из главного окна приложения и выполняется асинхронно для обеспечения отзывчивости интерфейса.**

3.1.1 **Инициация и управление загрузкой**

**Процесс загрузки инициируется пользователем нажатием кнопки «Скачать» (btnDownload\_Click). Перед запуском основных операций выполняются следующие проверки:**

1. **Проверка наличия URL: Список URL-адресов, введенных пользователем в txtUrls.Lines, фильтруется для удаления пустых строк. Если итоговый список пуст, выводится предупреждение, и процесс не запускается.**
2. **Проверка папки сохранения: Проверяется, указана ли папка для сохранения в txtSaveFolder.Text. Если путь не указан, выводится предупреждение.**
3. **Создание папки сохранения: Если указанная папка сохранения не существует, предпринимается попытка ее создать (Directory.CreateDirectory). В случае ошибки создания (например, из-за прав доступа), выводится сообщение об ошибке, и процесс прерывается.**

**После успешного прохождения проверок интерфейс пользователя подготавливается к длительной операции: кнопка «Скачать», поле ввода URL и кнопка выбора папки становятся неактивными. Очищается журнал сообщений (lstLog), сбрасывается и настраивается индикатор прогресса (progressBar.Maximum устанавливается равным количеству URL).**

**Для каждого URL из списка создается и запускается отдельная задача (Task.Run), инкапсулирующая весь цикл загрузки и обработки одной страницы. Это позволяет параллельно обрабатывать несколько URL, что значительно ускоряет общий процесс. Список всех задач (downloadTasks) собирается, и их общее завершение ожидается с помощью await Task.WhenAll(downloadTasks)**

3.1.2 **Асинхронная загрузка HTML-контента (DownloadPageInternalAsync)**

**Для каждого URL-адреса вызывается метод DownloadPageInternalAsync, ответственный за непосредственное получение HTML-содержимого.**

* **Нормализация URL: Если URL не начинается с http:// или https://, к нему добавляется префикс http://.**
* **HTTP-запрос: Используется статический экземпляр HttpClient (сконфигурированный с User-Agent для имитации браузера) для выполнения асинхронного GET-запроса (httpClient.GetAsync(uri)).**
* **Обработка ответа:**
* **При успешном ответе (response.IsSuccessStatusCode) содержимое ответа считывается как строка (response.Content.ReadAsStringAsync()) и возвращается вместе со статусом успеха.**
* **При неуспешном статусе ответа (например, 404 Not Found, 500 Internal Server Error) формируется сообщение об ошибке, включающее статус-код, причину и, по возможности, часть тела ответа.**
* **Обработка исключений: Метод обрабатывает специфические исключения, такие как UriFormatException (неверный формат URL) и HttpRequestException (сетевые ошибки), а также общие Exception. В каждом случае формируется соответствующее сообщение об ошибке.**

**Результат работы метода – кортеж, содержащий HTML-контент (или null в случае ошибки), HTTP-статус код (если был получен), флаг успеха и сообщение об ошибке (если есть).**

**// Фрагмент метода DownloadPageInternalAsync**

**private async Task<(string Content, System.Net.HttpStatusCode? StatusCode, bool IsSuccess, string ErrorMessage)> DownloadPageInternalAsync(string url)**

**{**

**// ... нормализация URL ...**

**try**

**{**

**Uri uri = new Uri(url);**

**HttpResponseMessage response = await httpClient.GetAsync(uri);**

**string responseContent = await response.Content.ReadAsStringAsync();**

**if (response.IsSuccessStatusCode)**

**{**

**return (responseContent, response.StatusCode, true, null);**

**}**

**else**

**{**

**string errorMessage = $"Статус: {response.StatusCode} ({response.ReasonPhrase}). Ответ сервера: {responseContent.Substring(0, Math.Min(responseContent.Length, 200))}";**

**return (null, response.StatusCode, false, errorMessage);**

**}**

**}**

**// ... обработка исключений ...**

**}**

## 3.2 Обработка HTML-документа и встраивание ресурсов

После успешной загрузки основного HTML-содержимого страницы оно подвергается анализу и модификации с использованием библиотеки AngleSharp.

3.2.1 **Парсинг HTML**

Строковое HTML-содержимое преобразуется в объектную модель документа (DOM) с помощью HtmlParser из библиотеки AngleSharp

var parser = new HtmlParser();

IDocument document = await parser.ParseDocumentAsync(downloadResult.Content);

**Это позволяет программно манипулировать структурой и элементами HTML-документа.**

3.2.2 **Встраивание CSS-стилей**

**Метод EmbedCssAsync отвечает за нахождение, загрузку и встраивание внешних CSS-файлов.**

1. **Поиск CSS-ссылок: В DOM-дереве документа (в секции <head>) производится поиск всех элементов <link> с атрибутами rel="stylesheet" и непустым href.**
2. **Загрузка CSS-контента: Для каждой найденной ссылки:**

* **Формируется абсолютный URL CSS-файла на основе базового URL страницы и относительного пути из href.**
* **С помощью httpClient.GetAsync загружается содержимое CSS-файла.**
* **В случае успешной загрузки, исходный элемент <link> удаляется из DOM.**
* **Загруженный CSS-код добавляется в общий StringBuilder.**

* **Ошибки загрузки CSS логируются.**

1. **Создание элемента <style>: Если был загружен хотя бы один CSS-файл, создается новый элемент <style type="text/css">, его текстовым содержимым становится накопленный CSS-код, и этот элемент вставляется в начало секции <head> документа.**

// Фрагмент метода EmbedCssAsync

private async Task<IDocument> EmbedCssAsync(IDocument document, string baseUrl)

{

var cssLinks = document.QuerySelectorAll("link[rel='stylesheet'][href]")

.OfType<IHtmlLinkElement>()

.ToList();

foreach (var linkElement in cssLinks)

{

HttpResponseMessage cssResponse = await httpClient.GetAsync(absoluteCssUri);

if (cssResponse.IsSuccessStatusCode)

{

string cssContent = await cssResponse.Content.ReadAsStringAsync();

embeddedCssBuilder.AppendLine(cssContent);

linkElement.Remove();

} // ...

}

if (embeddedCssBuilder.Length > 0)

{

var styleElement = document.CreateElement("style");

styleElement.TextContent = embeddedCssBuilder.ToString();

head.InsertBefore(styleElement, head.FirstChild);

}

return document;

}

**3.2.3** Встраивание изображений

Метод EmbedImagesAsync выполняет поиск, загрузку и встраивание изображений в виде Data URI.

1. Поиск элементов <img>: В DOM-документе ищутся все элементы <img> с непустым атрибутом src.
2. Обработка каждого изображения:

* Пропускаются изображения, src которых уже является Data URI (data:).
* Формируется абсолютный URL изображения.
* С помощью httpClient.GetAsync изображение загружается как массив байт (ReadAsByteArrayAsync).
* Определяется MIME-тип изображения: сначала из заголовка Content-Type ответа сервера, а если он отсутствует или некорректен, то по расширению файла (например, ".jpg" -> "image/jpeg"). Пропускаются изображения с неподдерживаемыми типами (например, не "image/\*" или "image/svg+xml").
* Загруженные байты изображения конвертируются в строку Base64 (Convert.ToBase64String).
* Атрибут src элемента <img> заменяется на Data URI вида data:[MIME-тип];base64,[Base64-строка].
* Ошибки загрузки или обработки изображений логируются.

// Фрагмент метода EmbedImagesAsync

private async Task<IDocument> EmbedImagesAsync(IDocument document, string baseUrl)

{

// ...

var imageElements = document.QuerySelectorAll("img[src]")

.OfType<IHtmlImageElement>()

.ToList();

// ...

foreach (var imgElement in imageElements)

{

// ... формирование абсолютного URL изображения, пропуск data: URI ...

HttpResponseMessage imageResponse = await httpClient.GetAsync(absoluteImageUri);

if (imageResponse.IsSuccessStatusCode)

{

byte[] imageBytes = await imageResponse.Content.ReadAsByteArrayAsync();

string contentType = imageResponse.Content.Headers.ContentType?.MediaType;

// ... определение/угадывание contentType ...

// ... проверка на поддерживаемый тип и пустые байты ...

string base64Image = Convert.ToBase64String(imageBytes);

string dataUri = $"data:{contentType};base64,{base64Image}";

imgElement.Source = dataUri;

} // ...

}

return document;

}

## 3.3 Сохранение результата и обратная связь

**3.3.1** Формирование итогового HTML и сохранение файла

После завершения всех модификаций DOM-объект IDocument сериализуется обратно в строку HTML с использованием AngleSharp.Html.PrettyMarkupFormatter для удобочитаемости.

string finalHtml;

var formatter = AngleSharp.Html.PrettyMarkupFormatter.Instance;

using (var stringWriter = new System.IO.StringWriter())

{

document.ToHtml(stringWriter, formatter);

finalHtml = stringWriter.ToString();

}

Имя для сохраняемого файла генерируется функцией GenerateFileNameFromUrl:

* Извлекается имя файла из пути URL. Если отсутствует, используется "index.html".
* Добавляется расширение ".html", если необходимо.
* Недопустимые символы заменяются на "\_".
* Имя усекается до 100 символов с сохранением расширения.
* В случае ошибки парсинга URL генерируется GUID-имя.
* Готовый HTML-код записывается в файл с использованием StreamWriter в кодировке UTF-8 в указанную пользователем папку.

**3.3.2** Обновление интерфейса и логирование

На протяжении всего процесса работы в список lstLog выводятся информационные сообщения о ходе выполнения: начало загрузки, статусы, ошибки, этапы встраивания ресурсов, путь к сохраненному файлу. Каждое сообщение снабжается временной меткой. Метод LogMessage обеспечивает потокобезопасное обновление ListBox из асинхронных задач с помощью lstLog.InvokeRequired и lstLog.Invoke.

Индикатор прогресса progressBar и метка статуса lblStatus обновляются после завершения обработки каждого URL, отражая количество успешно обработанных страниц.

По завершении всех задач выводится итоговое сообщение, и элементы управления (кнопки, текстовые поля) снова становятся активными.

// Фрагмент метода LogMessage

private void LogMessage(string message)

{

if (lstLog.InvokeRequired)

{

lstLog.Invoke((MethodInvoker)delegate {

lstLog.Items.Add($"[{DateTime.Now:HH:mm:ss}] {message}");

// ... автопрокрутка ...

});

}

else

{

lstLog.Items.Add($"[{DateTime.Now:HH:mm:ss}] {message}");

// ... автопрокрутка ...

}

}

## 3.4 Использование внешних библиотек

Для реализации функционала программного средства используются следующие ключевые внешние библиотеки:

* **AngleSharp (AngleSharp.Html.Parser, AngleSharp.Dom, AngleSharp.Html.Dom): Мощная .NET библиотека для парсинга HTML и CSS. Используется для разбора загруженного HTML-кода в объектную модель документа (DOM), поиска элементов (CSS-ссылок, изображений) и модификации DOM (удаление старых ссылок, добавление <style>-тегов, изменение атрибутов src у <img>).**
* **System.Net.Http: Стандартная библиотека .NET для выполнения HTTP-запросов. Класс HttpClient используется для асинхронной загрузки HTML-страниц, CSS-файлов и изображений.**
* **System.IO: Стандартная библиотека .NET для работы с файловой системой. Используется для создания папок (Directory.CreateDirectory), генерации имен файлов (Path.GetFileName, Path.Combine и т.д.) и записи итоговых HTML-файлов на диск (StreamWriter).**
* **System.Text.RegularExpressions: Используется в методе GenerateFileNameFromUrl для удаления недопустимых символов из имен файлов.**

**Эти библиотеки обеспечивают надежную и эффективную реализацию основных функций приложения, таких как сетевое взаимодействие, работа с HTML и файловой системой.**

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В процессе тестирования программы для быстрого скачивания страниц интернета были выявлены и устранены различные ошибки и недочёты, касающиеся как клиентской, так и серверной части программного средства.

**Программа корректно сохраняет HTML и встроенные CSS-файлы. Большинство страниц отображаются читаемо в офлайн-режиме. Встраивание изображений частично работает, но возникает ряд ограничений:**

* **не все форматы изображений обрабатываются (например, SVG часто игнорируется);**
* **некоторые сайты (например, Википедия) возвращают нестандартные MIME-типы, что мешает корректному внедрению;**
* **если изображение не загружается, программа не прерывает выполнение, а логирует ошибку.**

**Интерфейс остаётся стабильным при любом результате загрузки, что подтверждает корректную обработку исключений.**

**5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Данное руководство пользователя предназначено для пользователей программы «FastPageDownloader1». Оно описывает интерфейс программы, основные шаги для скачивания веб-страниц с последующим встраиванием CSS-стилей и изображений непосредственно в HTML-файл для офлайн-просмотра. Программа является приложением для операционной системы Windows и не требует сложной установки.

**5.1 Интерфейс программного средства**

Интерфейс программы «FastPageDownloader1» разработан для удобства пользователя и сосредоточен в одном главном окне, предоставляющем доступ ко всем функциям.

Главное окно программы (рис. 5.1) содержит следующие элементы:

– Поле для ввода URL-адресов: Многострочное текстовое поле, предназначенное для ввода или вставки URL-адресов веб-страниц, которые необходимо скачать. Каждый URL-адрес должен располагаться на новой строке.

– Поле «Папка для сохранения»: Текстовое поле, отображающее текущий путь к папке, в которую будут сохраняться загруженные и обработанные веб-страницы. По умолчанию предлагается папка DownloadedPages в «Документах» пользователя.

– Кнопка «Обзор...»: При нажатии открывает стандартное диалоговое окно Windows для выбора папки на диске, куда будут сохраняться файлы.

– Кнопка «Скачать»: Запускает процесс загрузки и обработки всех URL-адресов, указанных в соответствующем поле. Во время работы кнопка становится неактивной.

– Область логирования: Список, в котором отображаются подробные сообщения о ходе выполнения операций: начало загрузки каждой страницы, статусы обработки CSS и изображений, информация о сохранении файлов, а также возможные ошибки.

– Индикатор прогресса: Визуально отображает общий прогресс скачивания всех указанных URL-адресов.

– Статус скачивания: Текстовая метка, показывающая количество обработанных страниц относительно общего числа заданных URL (например, «Скачано 0 из 3»).

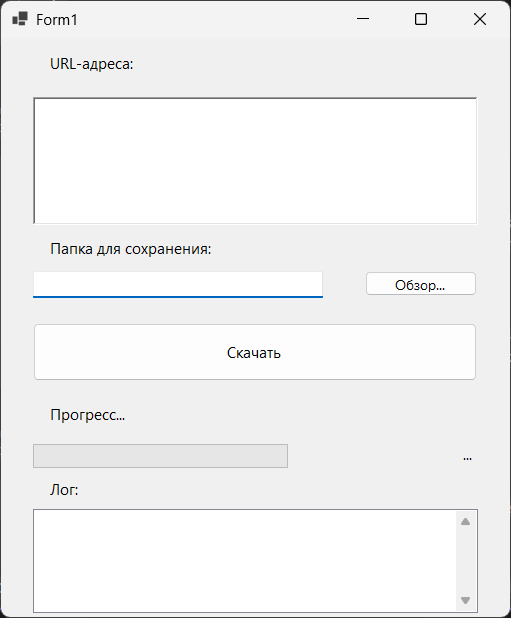


Рисунок 5.1 – Главное окно программы

**5.2 Управление программным средством**

Требования:

* Операционная система Windows с установленным .NET Framework (рекомендуется версия 4.5 или выше, на которой приложение было скомпилировано).
* Активное подключение к сети Интернет для загрузки веб-страниц и их ресурсов.

Запуск программы:

Запустите исполняемый файл программы (например, FastPageDownloader1.exe). Откроется главное окно программы, описанное в разделе 5.1.

Начало работы и скачивание страниц:

1. После запуска программы вы увидите главное окно. Поле для ввода URL-адресов может уже содержать примеры (например, https://example.com/).
2. Введите URL-адреса**:** В поле «URL-адреса для скачивания» введите или вставьте список веб-страниц, которые вы хотите загрузить. Каждый URL должен быть на новой строке. Пустые строки или строки, состоящие только из пробелов, будут проигнорированы.
3. Выберите папку для сохранения:

* По умолчанию программа предлагает сохранить файлы в папку DownloadedPages внутри ваших «Документов».
* Если вы хотите изменить папку, нажмите кнопку «Обзор...». В открывшемся диалоговом окне выберите желаемую директорию и нажмите «ОК». Путь к выбранной папке отобразится в поле «Папка для сохранения».
* Если указанная папка не существует, программа попытается создать её автоматически.

1. Начните скачивание: Нажмите кнопку «Скачать».

* Кнопка «Скачать» и другие элементы управления вводом станут неактивными на время процесса.
* В области «Лог выполнения» будут отображаться сообщения о ходе загрузки и обработки каждой страницы.
* «Индикатор прогресса» и метка «Статус скачивания» будут обновляться, показывая текущее состояние.

1. Завершение скачивания:

* По окончании обработки всех URL-адресов появится информационное сообщение «Скачивание завершено!».
* Кнопка «Скачать» и другие элементы управления снова станут активными.
* Скачанные HTML-файлы (с встроенными CSS-стилями и изображениями в формате Data URI) будут находиться в ранее указанной папке сохранения. Имя каждого файла генерируется на основе его URL-адреса (например, example.com.html или index.html).

Управление элементами интерфейса:

* Мышь: Используется для нажатия кнопок («Обзор...», «Скачать»), а также для взаимодействия с диалоговым окном выбора папки.
* Клавиатура: Используется для ввода и редактирования URL-адресов в соответствующем текстовом поле и, при необходимости, для ручного ввода пути к папке сохранения.

Для завершения работы с программой «FastPageDownloader1» просто закройте её главное окно стандартным для Windows способом (например, нажав на значок «X» в правом верхнем углу окна).

# ЗАКЛЮЧЕНИЕ

**В ходе выполнения курсового проекта было разработано и протестировано программное средство, предназначенное для быстрого скачивания и хранения интернет-страниц. Основное внимание при этом уделялось скорости работы, простоте интерфейса и надёжности сохранения данных в условиях нестабильного соединения или отсутствия доступа к сети.**

**Разработанное решение позволяет пользователю без необходимости загружать целые сайты — что не всегда рационально — быстро сохранить отдельную страницу и открыть её позже в автономном режиме. Подход с акцентом на «только нужное» позволяет сэкономить трафик, снизить нагрузку и избежать хранения лишней информации.**

**В ходе работы были реализованы ключевые функции: скачивание страницы по URL, сохранение базового контента (HTML, изображения), управление локальными копиями и базовый интерфейс для взаимодействия с пользователем. Были также проанализированы существующие подходы и реализована оптимизированная логика извлечения содержимого.**

**Разработанное приложение может быть полезно для широкого круга пользователей — от студентов и исследователей до журналистов и обычных читателей. В дальнейшем возможны такие направления развития, как поддержка пакетной загрузки, организация скачанных страниц по папкам и более глубокая фильтрация контента при сохранении.**

**Таким образом, поставленная цель была достигнута: создано функциональное и удобное средство для быстрого офлайн-доступа к нужной информации из интернета.**

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

**[1] Фримен А. Pro ASP.NET Core 6. – Sebastopol: Apress, 2022. – 1271 p.**

**[2]Microsoft Corporation. Документация ASP.NET Core [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/aspnet/core/**

**[3] AngleSharp. AngleSharp Documentation [Электронный ресурс]. – Режим доступа: https://anglesharp.github.io/**

**[4] Клири С. Параллельное программирование в C#. – 2-е изд. – Sebastopol: O’Reilly Media, 2019. – 246 p.**

**[5] Microsoft Corporation. HttpClient Class Documentation [Электронный ресурс]. – Режим доступа: [https://docs.microsoft.com/dotnet/api/system.net.http.httpclient](https://docs.microsoft.com/dotnet/api/system.net.http.httpclient) [6] Троелсен Э., Джапиксе Ф. Язык программирования C# 10 и платформа .NET 6. – 11-е изд. – New York: Apress, 2021. – 1905 p.**

# ПРИЛОЖЕНИЕ А. Исходный код программы

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Net.Http;

using System.IO;

using System.Text.RegularExpressions;

using AngleSharp.Html.Parser;

using AngleSharp.Dom;

using AngleSharp.Html.Dom;

using System.Net.Mime;

namespace FastPageDownloader1

{

public partial class Form1 : Form

{

private static readonly HttpClient httpClient = new HttpClient();

public Form1()

{

InitializeComponent();

httpClient.DefaultRequestHeaders.UserAgent.ParseAdd("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36");

txtSaveFolder.Text = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments), "DownloadedPages");

}

private void btnBrowseFolder\_Click(object sender, EventArgs e)

{

using (var fbd = new FolderBrowserDialog())

{

fbd.SelectedPath = txtSaveFolder.Text;

DialogResult result = fbd.ShowDialog();

if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(fbd.SelectedPath))

{

txtSaveFolder.Text = fbd.SelectedPath;

}

}

}

private async void btnDownload\_Click(object sender, EventArgs e)

{

string[] urls = txtUrls.Lines.Where(line => !string.IsNullOrWhiteSpace(line)).ToArray();

string saveFolderPath = txtSaveFolder.Text;

if (urls.Length == 0)

{

MessageBox.Show("Пожалуйста, введите хотя бы один URL.", "Нет URL", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

if (string.IsNullOrWhiteSpace(saveFolderPath))

{

MessageBox.Show("Пожалуйста, выберите папку для сохранения.", "Нет папки", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

if (!Directory.Exists(saveFolderPath))

{

try

{

Directory.CreateDirectory(saveFolderPath);

}

catch (Exception ex)

{

MessageBox.Show($"Не удалось создать папку: {ex.Message}", "Ошибка папки", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

}

btnDownload.Enabled = false;

btnBrowseFolder.Enabled = false;

txtUrls.Enabled = false;

lstLog.Items.Clear();

progressBar.Value = 0;

progressBar.Maximum = urls.Length;

lblStatus.Text = $"Скачано 0 из {urls.Length}";

List<Task> downloadTasks = new List<Task>();

int completedCount = 0;

foreach (string urlString in urls)

{

string currentUrl = urlString;

downloadTasks.Add(Task.Run(async () =>

{

try

{

LogMessage($"Начало загрузки: {currentUrl}");

var downloadResult = await DownloadPageInternalAsync(currentUrl);

if (downloadResult.IsSuccess && downloadResult.Content != null)

{

var parser = new HtmlParser();

IDocument document = await parser.ParseDocumentAsync(downloadResult.Content);

LogMessage($"HTML для {currentUrl} получен. Начинаю внедрение CSS...");

document = await EmbedCssAsync(document, currentUrl);

LogMessage($"CSS для {currentUrl} обработан. Начинаю внедрение изображений...");

document = await EmbedImagesAsync(document, currentUrl);

string finalHtml;

var formatter = AngleSharp.Html.PrettyMarkupFormatter.Instance;

using (var stringWriter = new System.IO.StringWriter())

{

document.ToHtml(stringWriter, formatter);

finalHtml = stringWriter.ToString();

}

string fileName = GenerateFileNameFromUrl(currentUrl);

string filePath = Path.Combine(saveFolderPath, fileName);

using (var streamWriter = new StreamWriter(filePath, false, Encoding.UTF8))

{

await streamWriter.WriteAsync(finalHtml);

}

LogMessage($"Сохранено (с CSS и изображениями): {currentUrl} -> {fileName}");

}

else

{

if (downloadResult.StatusCode.HasValue)

{

LogMessage($"Ошибка HTTP: {currentUrl} - {downloadResult.StatusCode} - {downloadResult.ErrorMessage}");

}

else

{

LogMessage($"Ошибка загрузки: {currentUrl} - {downloadResult.ErrorMessage}");

}

}

}

catch (Exception ex)

{

if (ex is AngleSharp.Dom.DomException domEx)

{

LogMessage($"Ошибка AngleSharp DOM при обработке {currentUrl}: {domEx.Message} (Код ошибки: {domEx.Code})");

}

else if (ex is System.Xml.XmlException xmlEx)

{

LogMessage($"Ошибка XML при обработке {currentUrl}: {xmlEx.Message}");

}

else

{

LogMessage($"Непредвиденная ошибка при обработке URL {currentUrl}: {ex.Message}");

}

}

finally

{

this.Invoke((MethodInvoker)delegate {

completedCount++;

progressBar.Value = completedCount;

lblStatus.Text = $"Скачано {completedCount} из {urls.Length}";

});

}

}));

}

await Task.WhenAll(downloadTasks);

LogMessage("--- Скачивание завершено ---");

MessageBox.Show("Скачивание завершено!", "Готово", MessageBoxButtons.OK, MessageBoxIcon.Information);

btnDownload.Enabled = true;

btnBrowseFolder.Enabled = true;

txtUrls.Enabled = true;

}

private async Task<string> DownloadPageAsync(string url)

{

if (!url.StartsWith("http://") && !url.StartsWith("https://"))

{

url = "http://" + url;

}

Uri uri = new Uri(url);

HttpResponseMessage response = await httpClient.GetAsync(uri);

response.EnsureSuccessStatusCode();

return await response.Content.ReadAsStringAsync();

}

private string GenerateFileNameFromUrl(string url)

{

try

{

Uri uri = new Uri(url);

string path = uri.AbsolutePath;

string fileName = Path.GetFileName(path);

if (string.IsNullOrEmpty(fileName) || fileName == "/")

{

fileName = "index.html";

}

else

{

if (!Path.HasExtension(fileName))

{

fileName += ".html";

}

}

string invalidChars = new string(Path.GetInvalidFileNameChars()) + new string(Path.GetInvalidPathChars());

Regex r = new Regex(string.Format("[{0}]", Regex.Escape(invalidChars)));

fileName = r.Replace(fileName, "\_");

if (fileName.Length > 100)

{

string ext = Path.GetExtension(fileName);

string nameWithoutExt = Path.GetFileNameWithoutExtension(fileName);

int maxNameLength = 100 - (ext?.Length ?? 0);

if (nameWithoutExt.Length > maxNameLength)

{

nameWithoutExt = nameWithoutExt.Substring(0, maxNameLength);

}

fileName = nameWithoutExt + ext;

}

return fileName;

}

catch (UriFormatException)

{

return Guid.NewGuid().ToString() + ".html";

}

}

private async Task<(string Content, System.Net.HttpStatusCode? StatusCode, bool IsSuccess, string ErrorMessage)> DownloadPageInternalAsync(string url)

{

if (!url.StartsWith("http://") && !url.StartsWith("https://"))

{

url = "http://" + url;

}

try

{

Uri uri = new Uri(url);

HttpResponseMessage response = await httpClient.GetAsync(uri);

string responseContent = await response.Content.ReadAsStringAsync();

if (response.IsSuccessStatusCode)

{

return (responseContent, response.StatusCode, true, null);

}

else

{

string errorMessage = $"Статус: {response.StatusCode} ({response.ReasonPhrase}). Ответ сервера: {responseContent.Substring(0, Math.Min(responseContent.Length, 200))}";

return (null, response.StatusCode, false, errorMessage);

}

}

catch (UriFormatException uriEx)

{

return (null, null, false, $"Неверный URL '{url}': {uriEx.Message}");

}

catch (HttpRequestException httpEx)

{

return (null, null, false, $"Сетевая ошибка для '{url}': {httpEx.Message}");

}

catch (Exception ex)

{

return (null, null, false, $"Непредвиденная ошибка для '{url}': {ex.Message}");

}

}

private async Task<IDocument> EmbedCssAsync(IDocument document, string baseUrl)

{

if (document == null)

{

LogMessage($"Документ для {baseUrl} не предоставлен для внедрения CSS. Пропуск.");

return null;

}

try

{

var head = document.Head;

if (head == null)

{

LogMessage($"Не найден <head> на странице {baseUrl}, CSS не будет встроен.");

return document;

}

var cssLinks = document.QuerySelectorAll("link[rel='stylesheet'][href]")

.OfType<IHtmlLinkElement>()

.ToList();

if (!cssLinks.Any())

{

return document;

}

StringBuilder embeddedCssBuilder = new StringBuilder();

Uri baseUri = new Uri(baseUrl);

LogMessage($"Найдено {cssLinks.Count} CSS ссылок на странице {baseUrl} (EmbedCssAsync). Начинаю обработку...");

foreach (var linkElement in cssLinks)

{

string cssRelativeUrl = linkElement.Href;

if (string.IsNullOrWhiteSpace(cssRelativeUrl))

{

LogMessage($"Пустой href у CSS ссылки на {baseUrl} (EmbedCssAsync)");

continue;

}

Uri absoluteCssUri;

try

{

absoluteCssUri = new Uri(baseUri, cssRelativeUrl);

}

catch (UriFormatException ex)

{

LogMessage($"Некорректный CSS URL: '{cssRelativeUrl}' на странице {baseUrl} (EmbedCssAsync). Ошибка: {ex.Message}");

continue;

}

try

{

LogMessage($"Загрузка CSS: {absoluteCssUri} (EmbedCssAsync)");

HttpResponseMessage cssResponse = await httpClient.GetAsync(absoluteCssUri);

if (cssResponse.IsSuccessStatusCode)

{

string cssContent = await cssResponse.Content.ReadAsStringAsync();

embeddedCssBuilder.AppendLine($"/\* CSS from {absoluteCssUri} \*/");

embeddedCssBuilder.AppendLine(cssContent);

embeddedCssBuilder.AppendLine();

linkElement.Remove();

LogMessage($"CSS {absoluteCssUri} успешно загружен и ссылка удалена (EmbedCssAsync).");

}

else

{

LogMessage($"Ошибка загрузки CSS {absoluteCssUri} (EmbedCssAsync): {cssResponse.StatusCode}");

}

}

catch (Exception ex)

{

LogMessage($"Критическая ошибка при загрузке или обработке CSS {absoluteCssUri} (EmbedCssAsync): {ex.Message}");

}

}

if (embeddedCssBuilder.Length > 0)

{

var styleElement = document.CreateElement("style");

styleElement.SetAttribute("type", "text/css");

styleElement.TextContent = embeddedCssBuilder.ToString();

head.InsertBefore(styleElement, head.FirstChild);

LogMessage($"Все загруженные CSS встроены в <style> для {baseUrl} (EmbedCssAsync)");

}

return document;

}

catch (Exception ex)

{

LogMessage($"Общая ошибка при обработке HTML для внедрения CSS ({baseUrl}, EmbedCssAsync): {ex.Message}");

return document;

}

}

private async Task<IDocument> EmbedImagesAsync(IDocument document, string baseUrl)

{

if (document == null)

{

LogMessage($"Документ для {baseUrl} не предоставлен для EmbedImagesAsync. Пропуск.");

return null;

}

LogMessage($"Начинаю поиск и встраивание изображений для {baseUrl} (EmbedImagesAsync)...");

var imageElements = document.QuerySelectorAll("img[src]")

.OfType<IHtmlImageElement>()

.ToList();

LogMessage($"[DEBUG EmbedImagesAsync] Найдено IHtmlImageElement: {imageElements.Count}");

if (!imageElements.Any())

{

LogMessage($"Изображения <img> не найдены на {baseUrl} или не являются IHtmlImageElement (EmbedImagesAsync).");

return document;

}

LogMessage($"Найдено {imageElements.Count} тегов <img> на {baseUrl} (EmbedImagesAsync). Начинаю обработку...");

Uri baseUri = new Uri(baseUrl);

foreach (var imgElement in imageElements)

{

string imageRelativeUrl = imgElement.Source;

LogMessage($"[DEBUG EmbedImagesAsync] Обработка img.Source: '{imageRelativeUrl}'");

if (string.IsNullOrWhiteSpace(imageRelativeUrl))

{

LogMessage($"Пустой src у тега <img> на {baseUrl} (EmbedImagesAsync). Пропуск.");

continue;

}

if (imageRelativeUrl.StartsWith("data:", StringComparison.OrdinalIgnoreCase))

{

LogMessage($"Пропуск уже встроенного изображения (Data URI): {imageRelativeUrl.Substring(0, Math.Min(imageRelativeUrl.Length, 60))}... (EmbedImagesAsync)");

continue;

}

Uri absoluteImageUri;

try

{

absoluteImageUri = new Uri(baseUri, imageRelativeUrl);

LogMessage($"[DEBUG EmbedImagesAsync] Сформирован абсолютный URL: {absoluteImageUri}");

}

catch (UriFormatException ex)

{

LogMessage($"Некорректный URL изображения: '{imageRelativeUrl}' на странице {baseUrl} (EmbedImagesAsync). Ошибка: {ex.Message}");

continue;

}

catch (ArgumentNullException ex)

{

LogMessage($"ArgumentNullException при создании Uri для изображения: '{imageRelativeUrl}' на {baseUrl} (EmbedImagesAsync). Ошибка: {ex.Message}");

continue;

}

try

{

LogMessage($"Загрузка изображения: {absoluteImageUri} (EmbedImagesAsync)");

HttpResponseMessage imageResponse = await httpClient.GetAsync(absoluteImageUri);

LogMessage($"[DEBUG EmbedImagesAsync] Ответ от сервера для {absoluteImageUri}: {imageResponse.StatusCode}");

if (imageResponse.IsSuccessStatusCode)

{

byte[] imageBytes = await imageResponse.Content.ReadAsByteArrayAsync();

string contentType = imageResponse.Content.Headers.ContentType?.MediaType;

if (string.IsNullOrWhiteSpace(contentType) && absoluteImageUri.Segments.Length > 0)

{

string ext = Path.GetExtension(absoluteImageUri.AbsolutePath).ToLowerInvariant();

switch (ext)

{

case ".jpg":

case ".jpeg":

contentType = "image/jpeg";

break;

case ".png":

contentType = "image/png";

break;

case ".gif":

contentType = "image/gif";

break;

case ".bmp":

contentType = "image/bmp";

break;

case ".webp":

contentType = "image/webp";

break;

default:

contentType = "application/octet-stream";

break;

}

LogMessage($"[DEBUG EmbedImagesAsync] ContentType не был предоставлен, угадан по расширению '{ext}': {contentType}");

}

else if (string.IsNullOrWhiteSpace(contentType))

{

contentType = "image/jpeg";

LogMessage($"[DEBUG EmbedImagesAsync] ContentType не был предоставлен и не удалось угадать, по умолчанию: {contentType}");

}

LogMessage($"[DEBUG EmbedImagesAsync] ContentType для {absoluteImageUri}: {contentType}, Размер: {imageBytes.Length} байт");

if (!contentType.StartsWith("image/", StringComparison.OrdinalIgnoreCase) ||

contentType.Equals("image/svg+xml", StringComparison.OrdinalIgnoreCase))

{

LogMessage($"Неподдерживаемый ContentType '{contentType}' для встраивания изображения {absoluteImageUri} (EmbedImagesAsync). Пропуск.");

continue;

}

if (imageBytes.Length == 0)

{

LogMessage($"Изображение {absoluteImageUri} пустое (0 байт). Пропуск встраивания.");

continue;

}

string base64Image = Convert.ToBase64String(imageBytes);

string dataUri = $"data:{contentType};base64,{base64Image}";

imgElement.Source = dataUri;

LogMessage($"Изображение {absoluteImageUri} успешно встроено как Data URI (EmbedImagesAsync).");

}

else

{

LogMessage($"Ошибка загрузки изображения {absoluteImageUri} (EmbedImagesAsync): {imageResponse.StatusCode}");

string errorContent = await imageResponse.Content.ReadAsStringAsync();

if (!string.IsNullOrWhiteSpace(errorContent))

{

LogMessage($"[DEBUG EmbedImagesAsync] Тело ошибки для {absoluteImageUri}: {errorContent.Substring(0, Math.Min(errorContent.Length, 200))}");

}

}

}

catch (HttpRequestException httpEx)

{

LogMessage($"HttpRequestException при загрузке изображения {absoluteImageUri} (EmbedImagesAsync): {httpEx.Message}. StatusCode: {httpEx.StatusCode}");

}

catch (TaskCanceledException ex)

{

LogMessage($"TaskCanceledException (возможно, таймаут) при загрузке изображения {absoluteImageUri} (EmbedImagesAsync): {ex.Message}");

}

catch (Exception ex)

{

string errorMessage = $"Критическая ошибка при загрузке или обработке изображения {absoluteImageUri} (EmbedImagesAsync): {ex.GetType().FullName} - {ex.Message}";

if (ex.InnerException != null)

{

errorMessage += $"\n Внутреннее исключение: {ex.InnerException.GetType().FullName} - {ex.InnerException.Message}";

}

LogMessage(errorMessage);

}

}

return document;

}

private void LogMessage(string message)

{

if (lstLog.InvokeRequired)

{

lstLog.Invoke((MethodInvoker)delegate {

lstLog.Items.Add($"[{DateTime.Now:HH:mm:ss}] {message}");

lstLog.TopIndex = Math.Max(0, lstLog.Items.Count - lstLog.ClientSize.Height / lstLog.ItemHeight);

});

}

else

{

lstLog.Items.Add($"[{DateTime.Now:HH:mm:ss}] {message}");

lstLog.TopIndex = Math.Max(0, lstLog.Items.Count - lstLog.ClientSize.Height / lstLog.ItemHeight);

}

}

private void Form1\_Load(object sender, EventArgs e)

{

txtUrls.Text = "https://example.com/\r\nhttps://www.google.com/\r\nhttps://nonexistentpage12345.com/";

}

}

}