**Universitatea de Stat din Moldova**

**FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ**

**DEPARTAMENTUL INFORMATICA**

**PROIECT DE SEMESTRU**

**«NewSearcher App»**

**Autor:** Pavlenco Anton

Malai Mihail

Ciornei Oleg

Nartea Nichita

**Chişinău-2023**

**Оглавление**

[Введение 2](#_Toc1801721435)

[1. Используемые технологии 3](#_Toc1530651060)

[1.1. Язык программирования С# 4](#_Toc486133671)

[1.2. User Interface 5](#_Toc768529020)

[1.3. Преимущества использования технологий 5](#_Toc1978423152)

[2. Особенности реализации 6](#_Toc1881317994)

[2.1. Работа с базой данных 7](#_Toc1879303196)

[2.1.2. Класс QueryBuilder. Паттерн Builder 7](#_Toc807539680)

[2.1.4. Обработка условий и ограничений запросов с Where и Limit 8](#_Toc1338460791)

[2.1.3. Класс SQLite Manager 9](#_Toc1077215295)

[2.1.4. Обеспечение безопасности и эффективности работы с базой данных 9](#_Toc1461831067)

[2.1.5. Интеграция с бизнес-логикой приложения 10](#_Toc1942324707)

[2.1.6. Вывод 11](#_Toc1345347786)

[2.2. Фоновые процессы (Workers) 11](#_Toc596653248)

[2.3. Парсер статьи 13](#_Toc1784791292)

[Вывод 16](#_Toc1323018491)

[Библиграфия 17](#_Toc1304211353)

[Приложение 18](#_Toc2097971593)

# Введение

Современный мир информации требует удобных и эффективных способов получения и обработки контента из различных источников. В этом контексте приложение, разработанное на языке программирования C# для парсинга статей с веб-сайтов, таких как Tproger и Habr, представляет собой ценный инструмент для улучшения опыта пользователя.

**Преимущества Приложения**

1. **Удобство Использования**

Пользовательский интерфейс приложения обеспечивает простоту и удобство в работе. Пользователи могут легко находить и читать статьи без лишних усилий.

1. **Сокращение Затрат Времени**

Приложение позволяет пользователям сэкономить время на поиске и отборе интересующего контента, предоставляя собранные статьи в удобном формате.

**Практическое Применение**

1. **Исследования и Образование**

Приложение может быть полезным инструментом для исследователей и студентов, позволяя эффективно собирать информацию для образовательных целей.

1. **Профессиональное Развитие**

Для профессионалов в области информационных технологий приложение предоставляет возможность отслеживать последние тенденции и новости в отрасли.

Далее мы рассмотрим какие технологии были применены для написания данного приложения и иллюстрации его работы.

# Используемые технологии

## Язык программирования С#

C# — современный и многофункциональный язык программирования, разработанный компанией Microsoft. Он является частью технологии Microsoft .NET и предназначен для создания разнообразных приложений, начиная от десктопных и веб-приложений до мобильных приложений и облачных сервисов.

**Основные Характеристики**

1. **Объектно-Ориентированный Язык**

C# поддерживает принципы объектно-ориентированного программирования (ООП), что упрощает создание модульного и структурированного кода.

1. **Интеграция с Платформой .NET**

Язык тесно связан с платформой .NET, предоставляя разработчикам мощные библиотеки классов, которые упрощают разработку и обеспечивают переносимость кода.

1. **Современный Синтаксис**

C# предлагает чистый и современный синтаксис, включая такие возможности, как LINQ (Language Integrated Query) для удобной работы с данными и асинхронное программирование.

1. **Безопасность Типов**

Система строгой типизации и управление памятью делают код на C# более безопасным и предотвращают многие ошибки, связанные с работой с памятью.

1. **Поддержка Параллельного Программирования**

C# предоставляет средства для эффективного использования многозадачности и параллелизма через асинхронные операции и библиотеки для параллельного программирования.

**Области Применения**

1. **Разработка Приложений Windows**

C# широко используется для создания десктопных приложений под управлением операционных систем Windows.

1. **Веб-Разработка**

С использованием ASP.NET, C# применяется для создания веб-приложений и веб-сайтов.

1. **Мобильная Разработка**

Xamarin, платформа для разработки мобильных приложений, позволяет использовать C# для создания кросс-платформенных приложений для iOS и Android.

1. **Серверные Приложения и Облачные Сервисы**

C# эффективно используется для создания серверных приложений и облачных сервисов, благодаря своей интеграции с технологией .NET.

## User Interface

Для разработки приложения был выбран framework языка C# Forms. System.Windows.Forms (WinForms) представляет собой часть .NET Framework и является библиотекой для создания графических пользовательских интерфейсов (GUI) в Windows-приложениях с использованием языка программирования C#. Этот фреймворк обеспечивает разработчиков мощными инструментами для создания интерактивных и интуитивно понятных приложений с использованием стандартных элементов управления.

**Основные Компоненты:**

**1. Формы (Forms)**

Формы представляют собой оконные интерфейсы приложений. Они служат контейнерами для размещения других элементов управления, таких как кнопки, текстовые поля и списки.

**2. Элементы Управления (Controls)**

WinForms предоставляет богатый набор стандартных элементов управления, таких как TextBox, Button, ComboBox и многие другие. Эти элементы обеспечивают базовую функциональность для взаимодействия с пользователем.

**3. Меню и Панели Инструментов**

Фреймворк позволяет легко создавать меню и панели инструментов, что облегчает навигацию и управление приложением.

**4. Графическое Рисование**

System.Windows.Forms поддерживает возможность графического рисования, что полезно для создания пользовательских интерфейсов и визуальных эффектов.

**5. Обработка Событий**

Механизм обработки событий позволяет реагировать на действия пользователя или изменения в приложении, обеспечивая отзывчивость интерфейса.

## Преимущества использования технологий

1. **Простота Использования**

WinForms предоставляет простой и интуитивно понятный способ создания пользовательских интерфейсов, что делает его идеальным выбором для разработчиков с разным уровнем опыта.

1. **Быстрое Развертывание**

Windows-приложения, построенные с использованием WinForms, легко разворачиваются на машинах конечных пользователей, так как .NET Framework включен в большинство версий Windows.

1. **Интеграция с Windows**

Фреймворк интегрирован с функциональностью Windows, что позволяет создавать приложения с естественным пользовательским опытом.

1. **Богатые Возможности Элементов Управления**

WinForms предоставляет обширный выбор стандартных элементов управления и позволяет легко создавать пользовательские элементы.

**Пример Кода:**

|  |
| --- |
| using System;  using System.Windows.Forms;  public class MainForm : Form  {  public MainForm()  {  // Инициализация компонентов формы  InitializeComponent();  }  private void InitializeComponent()  {  // Код для инициализации элементов управления и других компонентов  }  [STAThread]  static void Main()  {  // Запуск главной формы  Application.EnableVisualStyles();  Application.SetCompatibleTextRenderingDefault(false);  Application.Run(new MainForm());  }  } |

# Особенности реализации

## Работа с базой данных

В рамках настоящего исследования акцент будет сделан на аспектах взаимодействия с базой данных в приложении “**NewSearcher**”. Задача эффективной работы с данными стоит перед необходимостью создания структуры запросов, обеспечивающей гибкость и удобство обращения к базе данных. В данном контексте наш анализ сосредоточен на изучении класса **QueryBuilder**, который предоставляет механизм формирования SQL-запросов.

### 2.1.2. Класс QueryBuilder. Паттерн Builder

Класс QueryBuilder играет центральную роль в создании SQL-запросов для взаимодействия с базой данных приложения. Разработанный с использованием паттерна строителя (**Builder**), этот класс предоставляет программисту удобные методы для пошагового формирования запросов различного типа.

Внедрение паттерна проектирования Builder в класс QueryBuilder становится ключевым элементом в обеспечении гибкости и удобства формирования SQL-запросов в приложении. Паттерн Builder предоставляет механизм пошагового построения сложных объектов, позволяя избежать избыточных конструкторов с множеством параметров.

Класс QueryBuilder использует этот паттерн для создания запросов различных типов, таких как SELECT, UPDATE, INSERT и DELETE, в зависимости от требований пользователя. Методы класса предоставляют четкий интерфейс для последовательного добавления параметров запроса, позволяя динамически настраивать условия, поля и ограничения.

Преимуществом использования паттерна Builder является возможность легкого расширения функциональности класса без изменения его основной структуры. Это особенно важно в контексте приложения, где требования к запросам могут часто меняться. Паттерн Builder обеспечивает высокую степень абстракции и управляемости процессом формирования запросов, что способствует поддерживаемости кода и расширяемости системы.

Начнем с рассмотрения операции SELECT. С использованием метода Select(string fields), разработчик может указать необходимые поля для извлечения из базы данных. Например:

|  |
| --- |
| QueryBuilder queryBuilder = new QueryBuilder(); string selectQuery = queryBuilder.Select("Title, Content").From("News").AndWhere(new Where("Category", "Technology")).Get(); |

В данном примере формируется запрос на выборку заголовков и содержания новостей из таблицы "News" с дополнительным условием, где категория новости равна "Technology".

Далее рассмотрим операцию вставки данных в базу данных с использованием метода Insert(string table)

|  |
| --- |
| QueryBuilder queryBuilder = new QueryBuilder(); string insertQuery = queryBuilder.Insert("News").Values("'New Technology News', 'Exciting content about the latest tech advancements'").Get(); |

Этот пример создает запрос на вставку новой записи в таблицу "News" с указанным заголовком и содержанием.

Таким образом, класс QueryBuilder обеспечивает удобный и гибкий способ формирования SQL-запросов, что является ключевым элементом для эффективной работы с базой данных в приложении.

### 2.1.4. Обработка условий и ограничений запросов с Where и Limit

Для еще более тщательной настройки SQL-запросов и определения критериев поиска, в приложении IT News Reader были разработаны дополнительные классы Where и Limit. Класс Where предоставляет абстракцию для формирования условий в SQL-запросах, в то время как Limit позволяет управлять ограничениями на количество возвращаемых записей.

Пример использования класса Where для добавления условия в запрос:

|  |
| --- |
| QueryBuilder queryBuilder = new QueryBuilder(); string selectQuery = queryBuilder.Select("Title, Content").From("News")  .AndWhere(new Where("Category", "Technology"))  .AndWhere(new Where("Date", DateTime.Now, ">="))  .Get(); |

В данном примере добавляются два условия: выборка новостей из категории "Technology" и новостей, опубликованных после текущей даты.

Класс Limit позволяет контролировать количество возвращаемых записей и определять смещение:

|  |
| --- |
| QueryBuilder queryBuilder = new QueryBuilder(); string selectQuery = queryBuilder.Select("Title, Content").From("News")  .AddLimit(new Limit(0, 10))  .Get(); |

В этом примере указывается, что необходимо вернуть первые 10 записей из результата выборки.

Таким образом, классы Where и Limit в совокупности с QueryBuilder предоставляют разработчикам гибкий инструментарий для детализации SQL-запросов и точного контроля над объемом возвращаемых данных. Этот подход позволяет эффективно адаптировать запросы к требованиям приложения IT News Reader и обеспечивает более точные и релевантные результаты.

### 2.1.3. Класс SQLite Manager

После того как запросы были успешно сформированы с использованием QueryBuilder, необходим механизм выполнения этих запросов в базе данных. Здесь на сцену выходит класс SqliteManager. Он предоставляет абстракцию для работы с базой данных SQLite и реализует методы для выполнения запросов типа SELECT, INSERT и DELETE.

Пример использования **SqliteManager** для выполнения запроса на выборку данных:

|  |
| --- |
| SqliteManager sqliteManager = new SqliteManager(); string selectQuery = "SELECT \* FROM News WHERE Category = 'Technology';"; DataTable result = sqliteManager.Select(selectQuery); |

В этом примере SqliteManager используется для выполнения SQL-запроса на выборку всех записей из таблицы "News", где категория новостей равна "Technology". Возвращенные данные представлены в виде DataTable, обеспечивая удобство дальнейшей обработки.

Таким образом, взаимодействие между QueryBuilder и SqliteManager обеспечивает комплексный и эффективный механизм работы с базой данных в приложении. Класс QueryBuilder предоставляет гибкий способ формирования запросов, в то время как SqliteManager обеспечивает их исполнение и взаимодействие с базой данных, создавая единый механизм работы с данными.

### 2.1.4. Обеспечение безопасности и эффективности работы с базой данных

Важным аспектом взаимодействия с базой данных в приложении IT News Reader является обеспечение безопасности и эффективности операций. Класс SqliteManager внедряет принципы безопасности, используя параметризованные запросы для предотвращения атак SQL-инъекций. Этот подход обеспечивает надежность обработки данных, предостерегая от возможных угроз безопасности.

Пример безопасного выполнения SQL-запроса с использованием параметров:

|  |
| --- |
| SqliteManager sqliteManager = new SqliteManager(); string selectQuery = "SELECT \* FROM Users WHERE Username = @Username AND Password = @Password;"; DataTable result = sqliteManager.Select(selectQuery, username, password); |

В данном случае метод Select класса SqliteManager позволяет передать параметры напрямую в запрос, что делает его устойчивым к возможным атакам SQL-инъекций.

Также, класс SqliteManager обеспечивает эффективное управление подключением к базе данных, открывая и закрывая соединение в соответствующих частях кода. Это способствует оптимизации ресурсов и предотвращению утечек памяти.

Таким образом, в главе "Работа с Базой данных" подчеркивается важность принципов безопасности и эффективности, реализованных в классах SqliteManager и QueryBuilder. Этот подход обеспечивает не только гибкость и удобство работы с данными, но также гарантирует надежность и производительность операций с базой данных в контексте приложения.

### 2.1.5. Интеграция с бизнес-логикой приложения

Важным аспектом в работе с базой данных в приложении является интеграция с бизнес-логикой. Классы QueryBuilder и SqliteManager обеспечивают не только удобные инструменты для формирования и выполнения запросов, но также легко интегрируются с остальной частью приложения.

Пример использования класса QueryBuilder в бизнес-логике приложения:

|  |
| --- |
| public List<Article> GetTechnologyNews() {  QueryBuilder queryBuilder = new QueryBuilder();  string selectQuery = queryBuilder.Select("Title, Content, Date")  .From("News")  .AndWhere(new Where("Category", "Technology"))  .Order("Date", "DESC")  .AddLimit(new Limit(0, 5))  .Get();   DataTable result = \_sqliteManager.Select(selectQuery);  return ConvertDataTableToList(result); } |

В этом примере метод бизнес-логики использует QueryBuilder для формирования запроса на получение последних пяти новостей из категории "Technology", отсортированных по дате. Результат запроса преобразуется в список объектов Article для дальнейшего использования в приложении.

Также, взаимодействие с базой данных легко адаптируется к изменениям в бизнес-логике приложения. Если требуется изменить критерии выборки или добавить новые параметры, класс QueryBuilder позволяет вносить изменения без необходимости модификации других частей кода.

Таким образом, интеграция классов QueryBuilder и SqliteManager с бизнес-логикой приложения обеспечивает единый и гибкий механизм работы с данными. Этот подход позволяет разработчикам легко адаптировать систему к новым требованиям и поддерживать высокую степень модульности в коде приложения.

### 2.1.6. Вывод

В результате проведенного исследования и анализа работы с базой данных в приложении IT News Reader можно сделать ряд ключевых выводов. Применение классов QueryBuilder и SqliteManager обеспечивает не только эффективное взаимодействие с базой данных, но и реализует важные принципы проектирования, такие как гибкость, безопасность и модульность.

Класс QueryBuilder, используя паттерн Builder, предоставляет удобный и интуитивно понятный интерфейс для формирования SQL-запросов. Это позволяет разработчикам легко создавать запросы различных типов, подстраиваясь под требования бизнес-логики приложения. Кроме того, интеграция с классами Where и Limit добавляет дополнительную гибкость и точность в определении условий запросов.

## Фоновые процессы (Workers)

Класс Worker [Приложение 1] является важным компонентом в разрабатываемом приложении для сбора новостей с различных веб-сайтов. Рассмотрим его функциональность и роль в приложении:

* **Метод StartAsync**

Этот метод является точкой входа для асинхронной работы Worker. Он выполняет роль основного цикла сбора новостей и обработки данных.

Использование ключевого слова async указывает на то, что метод выполняет асинхронные операции, что позволяет приложению продолжать выполнение других задач во время ожидания завершения асинхронной операции.

Принимает в качестве аргумента CancellationToken, который предоставляет возможность отслеживать запросы на отмену выполнения операции. Это важно для корректного завершения работы приложения.

* **Инициализация объекта SqliteManager**

Создается экземпляр класса SqliteManager, который служит для управления взаимодействием с базой данных SQLite.

Инициализация происходит при создании экземпляра класса Worker. Это гарантирует наличие подготовленного объекта для взаимодействия с базой данных на протяжении всего жизненного цикла Worker.

* **Асинхронная Работа**

Метод StartAsync выполняет асинхронные операции, такие как сбор новостей и сохранение их в базе данных. Это важно, поскольку сбор новостей может занимать значительное время, и блокировка потока выполнения в ожидании завершения операции может негативно сказаться на производительности приложения.

Асинхронность также позволяет использовать операции ввода-вывода без блокировки основного потока, что повышает отзывчивость приложения.

Таким образом, благодаря асинхронному методу StartAsync и использованию SqliteManager для взаимодействия с базой данных, класс Worker обеспечивает эффективное и гибкое управление жизненным циклом приложения, сосредотачиваясь на сборе, обработке и сохранении новостей.

Периодический сбор новостей в приложении, реализованный через цикл while с использованием cancellationToken и Task.Delay, представляет собой ключевую часть функционала. Этот механизм обеспечивает непрерывный процесс обновления данных, гарантируя актуальность информации в базе данных. Цикл while обеспечивает постоянное прослушивание новостных источников до тех пор, пока не поступит запрос на отмену операции.

Оптимизация использования ресурсов достигается с помощью Task.Delay, который устанавливает интервал между итерациями цикла. Такой подход позволяет эффективно использовать ресурсы системы, предотвращая постоянную активность и избыточное потребление ресурсов.

Цель периодического сбора новостей заключается в обеспечении постоянного обновления базы данных и поддержании актуальности информации для пользователей приложения. Этот процесс фундаментально важен для приложения, собирающего новости, поскольку он обеспечивает пользователей свежей и актуальной информацией, повышая качество пользовательского опыта.

* **Обработка исключений и отмены**

В рамках обработки исключений и отмены, код в Worker предусматривает набор мер, направленных на обеспечение стабильной и безопасной работы приложения.

Обработка исключений означает, что в коде предусмотрены механизмы для обнаружения и корректной реакции на неожиданные ситуации. Это может включать в себя сценарии, где, например, сбой в сети или недоступность источника данных могут привести к ошибке. Обработка исключений способствует избежанию аварийного завершения приложения и позволяет восстановить его работоспособность.

Корректная обработка запросов на отмену операции является важным аспектом в управлении потоком выполнения. При наличии механизма отмены, приложение может корректно завершить свою работу в случае, если пользователь или система выражают желание прервать операцию. Это важно не только для безопасности, но и для эффективного управления ресурсами, предотвращая выполнение лишних операций.

* **Логирование и отчетность**

Вывод информации о выполняемых операциях в консоль играет роль инструмента мониторинга и отладки приложения. При разработке и поддержке приложения такие сообщения могут служить ключевой информацией для выявления проблем, отслеживания хода выполнения кода, а также для мониторинга и оптимизации процессов.

Комментарии в коде, поясняя назначение и действия, служат важным элементом документации и обеспечивают легкость понимания кода другими разработчиками. Это особенно важно в случае совместной работы над проектом, а также при последующем внесении изменений или обслуживании кодовой базы.

В общем, эти аспекты обеспечивают надежность и удобство работы приложения, гарантируя безопасное завершение операций, возможность мониторинга и легкость поддержки кода в долгосрочной перспективе.

## Парсер статьи

Класс ArticlesParser [Приложение 2] является важным компонентом в разрабатываемом приложении для сбора новостей с различных веб-сайтов. Рассмотрим его функциональность и роль в приложении.

Класс ArticlesParser является абстрактным и содержит статический метод GetArticles, предназначенный для парсинга статей (или новостей) с веб-сайтов, представленных в виде списка объектов Sites. Каждый объект Sites содержит информацию о конкретном веб-сайте, такую как URL, теги для извлечения заголовков, текста, ссылок и других элементов.  
 Обьект Site представлен следующим образом:

|  |
| --- |
| public string Domain { get; private set; } public string Url { get; private set; } public string ArticleTag { get; private set; } public string TitleTag { get; private set; } public string TextTag { get; private set; } public string LinkTag { get; private set; } public string StartLinkPattern { get; private set; } public string EndLinkPattern { get; private set; } public string ImageStartLinkPatter { get; private set; } public string ImageEndLinkPatter { get; private set; } |

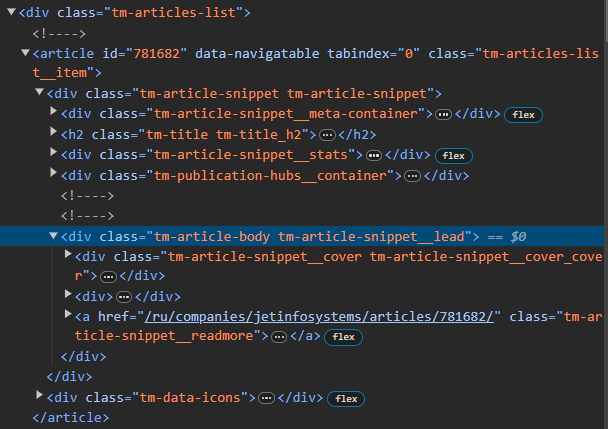
Данный класс был добавлен для обеспечения гибкости и универсальности класса ArticlesParser. Достигается это тем, что если разработчик решит добавить интеграцию нового сайта, будет необходимо лишь создать экземпляр класса. Значения полей заполняются вручную, по тегам HTML разметки на сайте. Пример:  


Рис 1. Разметка сайта

При добавлении нового сайта важно правильно опредлить его теги, что было проделано для двух сайтов, а именно Habr и Tproger. Habr это популярный веб-ресурс, представляющий собой платформу для обмена информацией, знаний и опытом в области информационных технологий (IT), программирования, стартапов, науки, и других связанных тем. На Хабре пользователи могут публиковать статьи (посты), обсуждать материалы других авторов, комментировать публикации, задавать вопросы в сообществах (хабах) и так далее. Хабр имеет большое количество хабов (тематических разделов), охватывающих различные аспекты IT и смежных областей. Tproger это IT-ресурс, который фокусируется на публикации о программировании, информационных технологиях и смежных темах. Основными форматами контента на Tproger являются статьи, новости, обзоры, а также материалы о профессиональной деятельности программистов. Сайт часто предоставляет разнообразные ресурсы, полезные для разработчиков, и регулярно обновляется актуальными материалами.

|  |
| --- |
| public static Sites Habr => new("https://habr.com", "<https://habr.com/ru/news/>",  "//article",  "//a[@class='tm-title\_\_link']",  "//div[@class='tm-article-body tm-article-snippet\_\_lead']",  "//a[@class='tm-article-snippet\_\_readmore']",  "href=\"","\"",  "<img src=\"", "\"");  public static Sites Tproger => new("https://tproger.ru", "<https://tproger.ru/>",  "//div[@class='tp-post-card\_\_content']",  "//h2[@class='tp-post-card\_\_title']",  "//p[@class='tp-post-card\_\_text']",  "//a[@class='tp-post-card\_\_link']",  "<a class=\"tp-post-card\_\_link\" href=\"", "\"",  "img class=\"tp-image\_\_image\" src=\"", "\""); |

При реализации парсера была использована библиотека HTML Agility Pack (HAP), предназначенная для анализа и обработки HTML-документов. Она предоставляет удобные инструменты для навигации, выборки и модификации элементов HTML, а также извлечения данных из веб-страниц.

HTML Agility Pack обеспечивает объектную модель для представления HTML-документа, позволяя вам проводить различные операции с его содержимым.

В методе главном GetArticles класса ArticlesParser выполняются следующие шаги:

1. Для каждого сайта из списка выполняется следующий набор действий.
   1. Используется библиотека HTML Agility Pack для загрузки HTML-страницы с указанного в site.Url URL.
   2. Из HTML-документа извлекаются элементы, соответствующие статьям, с использованием тега, указанного в site.ArticleTag.
   3. Заголовки, тексты, ссылки и изображения извлекаются с использованием тегов, указанных в site.TitleTag, site.TextTag, site.LinkTag, site.ImageStartLinkPattern, и site.ImageEndLinkPattern. Некоторые обработки, такие как удаление лишних символов или замена некоторых подстрок, также выполняются.
   4. Для каждой извлеченной статьи создается объект Article, который затем добавляется в список articles. Объект Article содержит информацию о заголовке, тексте, ссылке и изображении статьи.
   5. Если на текущем сайте не удалось найти статьи, выводится сообщение об ошибке.
2. Возврат списка статей (return articles;): После обработки всех сайтов возвращается список статей.

Также в классе есть приватный метод ExtractSubstringBetweenPatterns, который используется для извлечения подстроки между заданными паттернами в тексте. Этот метод применяется, например, для извлечения ссылок и изображений с использованием заданных шаблонов начала и конца, так как они хранятся не в innerHtml а в outerHtml.

Данный класс является основной бизнес логикой проекта, поэтому он создан расширяемым, универсальным и открытым для изменений.

# Вывод

# Библиграфия

# Приложение

Приложение 1. Класс Worker

|  |
| --- |
| using it\_news\_reader.parser; using it\_news\_reader.parser.entities;  namespace it\_news\_reader.job {  internal class Worker  {  private readonly SqliteManager \_sqliteManager = new();   public async Task StartAsync(CancellationToken cancellationToken, List<Sites> sites)  {  while (!cancellationToken.IsCancellationRequested)  {  DeleteOldData();  Console.WriteLine(@"Old articles deleted from database.");   ArticlesParser.GetArticles(sites).ForEach(i =>  {  var insertQuery = new QueryBuilder().Insert("articles")  .Field("title,text,link,image")  .Values(ToStringList(new[]  {  i.GetTitle().Replace("'", "’"),  i.GetText().Replace("'", "’"),  i.GetLink(), i.GetImageLink()  })).Get();  \_sqliteManager.Insert(insertQuery);  });   Console.WriteLine(@"New articles are persisted in database.");  *// await Task.Delay(TimeSpan.FromHours(4));* await Task.Delay(TimeSpan.FromSeconds(1 \* 60), cancellationToken);  }  }   private static string ToStringList(string[] values)  {  return "'" + string.Join("','", values) + "'";  }   private void DeleteOldData()  {  string query = new QueryBuilder().Delete("articles").AndWhere(new Where("date", "$param1", "<")).Get();  DateTime cutoffDate = DateTime.Now.AddHours(-24);  \_sqliteManager.Delete(query, cutoffDate);  }  } } |

Приложение 2. Класс ArticlesParser

|  |
| --- |
| using System.Text.RegularExpressions; using HtmlAgilityPack; using it\_news\_reader.parser.entities;  namespace it\_news\_reader.parser;  abstract class ArticlesParser {  public static List<Article> *GetArticles*(List<Sites> sites)  {  List<Article> articles = new List<Article>();  foreach (var site in sites)  {  string url = site.Url;  var web = new HtmlWeb();  var doc = web.Load(url);  var articlesParsed = doc.DocumentNode.SelectNodes(site.ArticleTag);  var titles = doc.DocumentNode.SelectNodes(site.TitleTag);  var texts = doc.DocumentNode.SelectNodes(site.TextTag);  var links = doc.DocumentNode.SelectNodes(site.LinkTag);   if (articlesParsed != null)  {  for (int currentArticle = 0; currentArticle < articlesParsed.Count; currentArticle++)  {  string title = titles[currentArticle].InnerText.Trim();  string text = texts[currentArticle].InnerText  .Replace("Читать далее", "")  .Replace("&nbsp;", " ")  .Trim();  string link = *ExtractSubstringBetweenPatterns*(  links[currentArticle].OuterHtml, site.StartLinkPattern, site.EndLinkPattern);   string imageLink = null!;  if (articlesParsed[currentArticle].OuterHtml.Contains(site.ImageStartLinkPattern))  {  imageLink = *ExtractSubstringBetweenPatterns*(  articlesParsed[currentArticle].OuterHtml, site.ImageStartLinkPattern,  site.ImageEndLinkPattern);  }   articles.Add(new Article(title, text, site.Domain + link, imageLink!));  }  }  else  {  Console.*WriteLine*(@"Unable to find news on " + site.Domain);  }  }   return articles;  }   private static string *ExtractSubstringBetweenPatterns*(string input, string startPattern, string endPattern)  {  Regex regex = new Regex($"{Regex.*Escape*(startPattern)}(.\*?){Regex.*Escape*(endPattern)}");  Match match = regex.Match(input);   if (match.Success)  {  return match.Groups[1].Value;  }   return string.*Empty*;  } } |