**Universitatea de Stat din Moldova**

**FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ**

**DEPARTAMENTUL INFORMATICA**

**PROIECT DE SEMESTRU**

**«NewSearcher App»**

**Autor:** Pavlenco Anton

Malai Mihail

Oleg Ciornei

Nartea Nichita

**Chişinău-2023**

**Оглавление**

[Введение 3](#_Toc153836209)

[1. Используемые технологии 4](#_Toc153836210)

[1.1. Язык программирования С# 4](#_Toc153836211)

[1.2. User Interface 5](#_Toc153836212)

[1.3. Преимущества использования технологий 6](#_Toc153836213)

[2. Особенности реализации 8](#_Toc153836214)

[2.1. Работа с базой данных 8](#_Toc153836215)

[2.1.2. Класс QueryBuilder. Паттерн Builder 8](#_Toc153836216)

[2.1.4. Обработка условий и ограничений запросов с Where и Limit 9](#_Toc153836217)

[2.1.3. Класс SQLite Manager 10](#_Toc153836218)

[2.1.4. Обеспечение безопасности и эффективности работы с базой данных 10](#_Toc153836219)

[2.1.5. Интеграция с бизнес-логикой приложения 11](#_Toc153836220)

[2.1.6. Вывод 12](#_Toc153836221)

[2.2. Фоновые процессы (Workers) 12](#_Toc153836222)

[Вывод 15](#_Toc153836223)

[Библиграфия 16](#_Toc153836224)

[Приложение 17](#_Toc153836225)

# 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.

# 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);  }  } } |