**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

**БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

**Кафедра «Информатика и программное обеспечение»**

**Расчетно-графическая работа**

**по дисциплине «Программирование в среде Microsoft. NET»**

направление подготовки 02.03.03 «Математическое обеспечение и администрирование информационных систем»

на тему:

**Разработка приложения для автоматизированного сбора, хранения и обработки информации о реках с сайта «mostinfo.su»**

Выполнил студ. гр. О-19-МОА:

**Байдекин С.М.**

Проверил:

к.т.н., доц. **Коростелев Д. А.**

**Брянск 2021**

**ЗАДАНИЕ**

В данной расчетно-графической работе необходимо разработать приложение Windows Forms для автоматизированного сбора, хранения и обработки данных информации о реках с выбранного веб-ресурса.

Поставленные задачи:

1. Разработать приложение Windows Forms, которое позволяет загружать в автоматизированном режиме информацию о реках с внешнего источника в БД.
2. Для хранения загруженной информации в программе необходима БД, структуру которой также необходимо разработать.
3. Программа должна иметь меню и соответствующие элементы управления, позволяющие запускать процесс первоначальной загрузки информации с внешнего источника, а также осуществлять дозагрузку. При этом должно быть исключено дублирование информации. Вся загружаемая информация должна помещаться в БД.
4. Программа дополнительно должна иметь элементы управления, позволяющие просматривать, редактировать и удалять загруженные записи из БД, а также позволять добавлять новые записи.
5. Программа должна быть хорошо протестирована, а также должны быть предусмотрены конструкции обработки исключительных ситуаций.

**СОДЕРЖАНИЕ**

[1. ВЫБОР ИСТОЧНИКА ИНФОРМАЦИИ И МЕТОДА РАБОТЫ С НИМ 4](#_Toc87721474)

[1.1. Описание и обоснование выбора источника информации 4](#_Toc87721475)

[1.2. Описание и обоснование метода извлечения данных из источника 5](#_Toc87721476)

[2. АРХИТЕКТУРА РАЗРАБОТАННОЙ ПРОГРАММЫ 6](#_Toc87721477)

[3. ОПИСАНИЕ СТРУКТУРЫ БАЗЫ ДАННЫХ 7](#_Toc87721478)

[4. ОПИСАНИЕ ИЗВЛЕЧЕНИЯ ДАННЫХ С ВНЕШНЕГО ИСТОЧНИКА 11](#_Toc87721479)

[5. ОПИСАНИЕ ИНТЕРФЕЙСА РАЗРАБОТАННОЙ ПРОГРАММЫ 14](#_Toc87721480)

[6. ОПИСАНИЕ ПРОЦЕССА ТЕСТИРОВАНИЯ ПРОГРАММЫ 16](#_Toc87721481)

[ЗАКЛЮЧЕНИЕ 17](#_Toc87721482)

[СПИСОК ЛИТЕРАТУРЫ 18](#_Toc87721483)

# ВЫБОР ИСТОЧНИКА ИНФОРМАЦИИ И МЕТОДА РАБОТЫ С НИМ

## Описание и обоснование выбора источника информации

Главным условием при выборе источника информации была структурированность информации, так как работать со сплошным текстом очень сложно.

Мой выбор пал на сайт «https://mostinfo.su/» (рис. 1).



Рис. 1. Внешний вид источника информации



Рис. 2. Внешний вид итогового файла

## Описание и обоснование метода извлечения данных из источника

В качестве метода извлечения информации был выбран способ позволяющий вставлять ссылку на файл с которого происходит загрузка данных в БД.

Используется HTML Agility Pack.

# АРХИТЕКТУРА РАЗРАБОТАННОЙ ПРОГРАММЫ

Архитектура разработанной программы (рис. ) подходит под описание трёхуровневой модели.

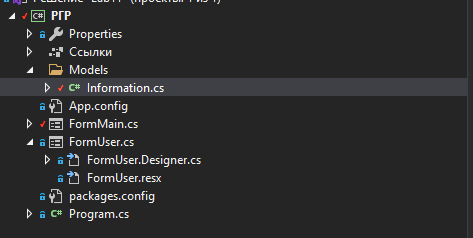


Рис. 3. Архитектура программы

Трёхуровневая архитектура (или же трёхзвенная) — архитектурная модель программного комплекса, предполагающая наличие в нём трёх типов компонентов (уровней, звеньев): клиентских приложений (с которыми работают пользователи), серверов приложений (с которыми работают клиентские приложения) и серверов баз данных (с которыми работают серверы приложений).

Первый уровень – уровень представления – реализован с помощью компонентов Windows Form, позволяющих создать интерфейс, привычный и понятный пользователю. Кроме того, они позволяют отобразить данные в удобочитаемом формате.

Второй уровень – уровень логики – реализован с помощью функции парсинга который позволяет скачивать информацию с облака и обрабатывать к виду необходимый нам.

Третий уровень – уровень доступа к данным – реализован с помощью СУБД MS SQL, где была создана база данных рек под названием riverDB, связь с которой осуществляется с помощью класса Information, который представляет сущность рек и содержит методы для вставки и получения данных из базы данных.

# ОПИСАНИЕ СТРУКТУРЫ БАЗЫ ДАННЫХ

Для хранения информации о рек была использована СУБД Microsoft SQL Server. Данная СУБД подходит для самых различных проектов: от небольших приложений до больших высоконагруженных проектов.

С помощью средства администрирования SQL Server Management Studio была создана база данных riverDB, включающая в себя одноимённую таблицу (рис. ).

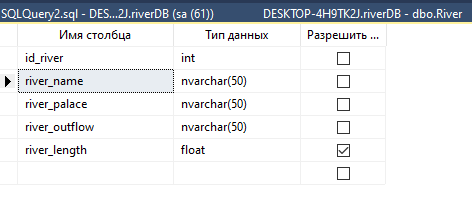


Рис. 4. Таблица рек в базе данных

Таблица состоит из следующих полей:

* Id\_river – уникальный идентификатор записи;
* River\_name – строка с названием реки, максимум 50 символа;
* River\_palace – строка с местоположением реки, максимум 50 символа;
* River\_outflow – строка с местом оттока реки, максимум 50 символа;
* River\_length – площадь реки задаваемая плавующим значением;

База данных была подключена к проекту с помощью sql, представляющего собой кэш данных в памяти.

Для считывания и записи данных в базу данных в программе был разработан класс Information, представляющий собой сущность Реки. Поля класса имеют аналогичные полям в базе данных названия. Само считывание и запись данных реализованы в методах класса List и Insert соответственно (листинг 2).

*Листинг 2*

|  |
| --- |
| **using** System;  **using** System.Collections.Generic;  **using** System.Data;  **using** System.Data.SqlClient;      **namespace** Lab11.Models  {  **public** **class** Information  {  **private** **static** **readonly** **string** \_selectUsersCommand = @"SELECT [id\_river], [river\_name],[river\_palace], [river\_outflow],[river\_length] FROM [dbo].[River]";  **private** **static** **readonly** **string** \_insertUserCommand = @"INSERT INTO [dbo].[River] ([river\_name],[river\_palace], [river\_outflow], [river\_length]) VALUES (@river\_name,@river\_palace, @river\_outflow,@river\_length)";  **private** **static** **readonly** **string** \_updateUserCommand = @"UPDATE [dbo].[River] SET [river\_name] = @river\_name,[river\_palace]=@river\_palace, [river\_outflow] = @river\_outflow, [river\_length] = @river\_length WHERE [id\_river] = @id\_river";  **private** **static** **readonly** **string** \_deleteUserCommand = @"DELETE FROM [dbo].[River] WHERE [id\_river] = @id\_river";  **private** **static** **readonly** **string** \_checkDuplicates = @"SELECT [id\_river], [river\_name],[river\_palace], [river\_outflow],[river\_length] FROM [dbo].[River] WHERE [river\_name]=@river\_name ";  **public** **int** RiverId { **get**; **set**; }  **public** **string** RiverName { **get**; **set**; }  **public** **string** RiverOutflow { **get**; **set**; }  **public** **string** RiverPalace { **get**; **set**; }  **public** **double** RiverLength { **get**; **set**; }  **public** **static** List<Information> List(SqlConnection connection)  {  List<Information> rivers = new List<Information>();  **using** (SqlCommand command = new SqlCommand())  {  **try**  {  command.Connection = connection;  command.CommandText = \_selectUsersCommand;  command.CommandType = CommandType.Text;  connection.Open();  **var** reader = command.ExecuteReader();  **while** (reader.Read())  {  Information river = new Information  {  RiverId = (**int**)reader["id\_river"],  RiverName = (**string**)reader["river\_name"],  RiverPalace = (**string**)reader["river\_palace"],  RiverOutflow = (**string**)reader["river\_outflow"],  RiverLength= (**double**)reader["river\_length"]      };  rivers.**Add**(river);  }  }  **finally**  {  **if** (connection != **null** && connection.State == ConnectionState.Open) connection.Close();  }  }  **return** rivers;  }    **public** **static** **void** Insert(SqlConnection connection, Information river)  {  **using** (SqlCommand command = new SqlCommand())  {  **try**  {  command.Connection = connection;  command.CommandText = \_insertUserCommand;  command.CommandType = CommandType.Text;  command.Parameters.**Add**("@river\_name", SqlDbType.NVarChar, 50).**Value** = river.RiverName;  command.Parameters.**Add**("@river\_palace", SqlDbType.NVarChar, 50).**Value** = river.RiverPalace;  command.Parameters.**Add**("@river\_outflow", SqlDbType.NVarChar, 50).**Value** = river.RiverOutflow;  command.Parameters.**Add**("@river\_length", SqlDbType.**Float**).**Value** = river.RiverLength;  command.Parameters.**Add**("@id\_river", SqlDbType.**Int**, 32).**Value** = river.RiverId;  connection.Open();  command.ExecuteNonQuery();  }  **finally**  {  **if** (connection != **null** && connection.State == ConnectionState.Open) connection.Close();  }  }  }    **public** **static** **void** Update(SqlConnection connection, Information river)  {  **using** (SqlCommand command = new SqlCommand())  {  **try**  {    command.Connection = connection;  command.CommandText = \_updateUserCommand;  command.CommandType = CommandType.Text;  command.Parameters.**Add**("@river\_name", SqlDbType.NVarChar, 50).**Value** = river.RiverName;  command.Parameters.**Add**("@river\_palace", SqlDbType.NVarChar, 50).**Value** = river.RiverPalace;  command.Parameters.**Add**("@river\_outflow", SqlDbType.NVarChar, 50).**Value** = river.RiverOutflow;  command.Parameters.**Add**("@river\_length", SqlDbType.**Float**).**Value** = river.RiverLength;  command.Parameters.**Add**("@id\_river", SqlDbType.**Int**, 32).**Value** = river.RiverId;  connection.Open();  command.ExecuteNonQuery();  }  **finally**  {  **if** (connection != **null** && connection.State == ConnectionState.Open) connection.Close();  }  }  }  **public** **static** **string** CheckDuplicates(SqlConnection connection, **string** name) {  **String** Title = **null**;  **using** (SqlCommand command = new SqlCommand())  {  **try**  {  command.Connection = connection;  command.CommandText = \_checkDuplicates;  command.CommandType = CommandType.Text;  command.Parameters.**Add**("@river\_name", SqlDbType.NVarChar, 32).**Value** = name;  connection.Open();  **var** reader = command.ExecuteReader();  **while** (reader.Read())  {  Title = (**string**)reader["river\_name"];  }  }  **finally**  {  **if** (connection != **null** && connection.State == ConnectionState.Open) connection.Close();  }  }  **return** Title;      }  **public** **static** **void** Delete(SqlConnection connection, **int** riverID)  {  **using** (SqlCommand command = new SqlCommand())  {  **try**  {  command.Connection = connection;  command.CommandText = \_deleteUserCommand;  command.CommandType = CommandType.Text;  command.Parameters.**Add**("@id\_river", SqlDbType.**Int**).**Value** = riverID;  connection.Open();  command.ExecuteNonQuery();  }  **finally**  {  **if** (connection != **null** && connection.State == ConnectionState.Open) connection.Close();  }  }  }  }  } |

*Листинг 2. Класс* Information

# ОПИСАНИЕ ИЗВЛЕЧЕНИЯ ДАННЫХ С ВНЕШНЕГО ИСТОЧНИКА

Для того, чтобы форма не «зависала» во время извлечения данных из внешнего источника. Эта функция позволяет выполнять операцию, требующую много времени, в нашем случае асинхронную загрузку данных (листинг 3).

Используется HTML Agility Pack.

При выборе в меню пункта «Загрузить данные» используется функция которая позволяет скачать данные с сервера (листинг 4). После завершения работы метода парсинга, данные которые были получены загружаются в БД.

*Листинг 3*

|  |
| --- |
| List<Information> DownList = new List<Information>();  **using** (WebClient client = new WebClient())  {    **var** htmlWeb = new HtmlWeb();  htmlWeb.OverrideEncoding = Encoding.UTF8;  **var** doc = htmlWeb.Load("https://mostinfo.su/53-samye-bolshie-reki-mira.html");      **var** HTMLTableTRList = **from** table **in** doc.DocumentNode.SelectNodes("//tbody").Cast<HtmlNode>()  **from** row **in** table.SelectNodes("tr").Cast<HtmlNode>()  **from** cell1 **in** row.SelectNodes("th|td").Cast<HtmlNode>()  **select** new { Cell\_Text = cell1.InnerText };          **int** x = 0;  **string**[] ss1 = new **string**[4];  **int** lineId = 1;  **int** j = 0;  **int** check = 0;    **foreach** (**var** cell1 **in** HTMLTableTRList)  {  **if** (check >4) {    **var** ls = cell1.Cell\_Text;    **var** lines = cell1.Cell\_Text;  **if** (lineId == 1)  {  ss1[0] = cell1.Cell\_Text;  }  **if** (lineId == 2)  {  ss1[1] = cell1.Cell\_Text;  }  **if** (lineId == 3)  {  ss1[2] = cell1.Cell\_Text;  }  **if** (lineId == 5)  {  ss1[3] = cell1.Cell\_Text;    }  lineId++;  j++;  **if** (lineId > 5)  {  lineId = 1;  Information river = new Information();  river.RiverName = ss1[0];  river.RiverPalace = ss1[1];  river.RiverOutflow = ss1[2];  river.RiverLength = **double**.Parse(ss1[3]);  DownList.**Add**(river);  }  }  check ++;  }    }      **for** (**int** i = 0; i < DownList.Count; i++)  {  **if** (Information.CheckDuplicates(\_connection, DownList[i].RiverName) == **null**)  {  Information.Insert(\_connection, DownList[i]);  }        }    **var** rivers = Information.List(\_connection);  listViewUsers.Items.Clear();  **for** (**int** i = 0; i < rivers.Count; i++)  {  **var** river = rivers[i];  **var** listListViewItem = listViewUsers.Items.**Add**(river.RiverId.ToString());  listListViewItem.Tag = river;  listListViewItem.SubItems.**Add**(river.RiverName);  listListViewItem.SubItems.**Add**(river.RiverPalace);  listListViewItem.SubItems.**Add**(river.RiverOutflow);  listListViewItem.SubItems.**Add**(river.RiverLength.ToString());  } |

*Листинг 3. Асинхронная загрузка данных с сервера и вставка в WPF*

Когда получена вся информация о реке, проводится проверка на существование записи с таким названием реки в базе данных с помощью метода CheckDuplicates. В случае отсутствия такой записи вызывается метод Insert, который добавляет объект Information в базу данных.

Выглядит следующим образом.

*Листинг 4*

|  |
| --- |
| **public** **static** **void** Insert(SqlConnection connection, Information river)  {  **using** (SqlCommand command = new SqlCommand())  {  **try**  {  command.Connection = connection;  command.CommandText = \_insertUserCommand;  command.CommandType = CommandType.Text;  command.Parameters.**Add**("@river\_name", SqlDbType.NVarChar, 50).**Value** = river.RiverName;  command.Parameters.**Add**("@river\_palace", SqlDbType.NVarChar, 50).**Value** = river.RiverPalace;  command.Parameters.**Add**("@river\_outflow", SqlDbType.NVarChar, 50).**Value** = river.RiverOutflow;  command.Parameters.**Add**("@river\_length", SqlDbType.**Float**).**Value** = river.RiverLength;  command.Parameters.**Add**("@id\_river", SqlDbType.**Int**, 32).**Value** = river.RiverId;  connection.Open();  command.ExecuteNonQuery();  }  **finally**  {  **if** (connection != **null** && connection.State == ConnectionState.Open) connection.Close();  }  }  } |

# ОПИСАНИЕ ИНТЕРФЕЙСА РАЗРАБОТАННОЙ ПРОГРАММЫ

Для создания интерфейса было использовано приложение Windows Forms, в котором была создана форма (рис. ), включающая в себя следующие компоненты:

* MenuStrip – система меню, группирующая команды приложения и делающая их легко доступными. В меню имеются пункты для загрузки данных с веб-сайта и выхода из приложения.
* TabController – создать элемент управления с несколькими вкладками.
* ToolStrip – представляет панель инструментов позволяющий настроить меню.
* ListView – инструмент позволяющий отображать данные в табличной форме.

Изначально подгружается информации с БД.

Если информация есть в БД.

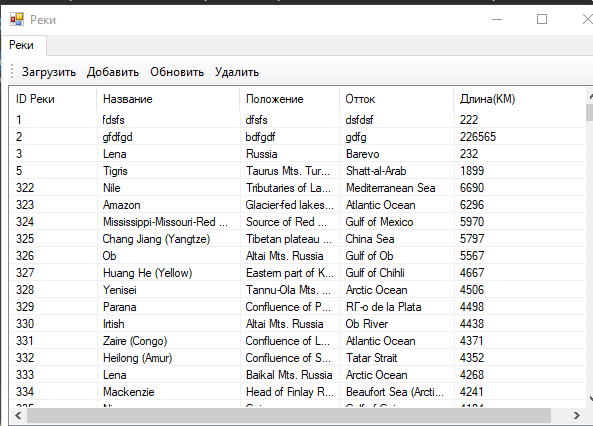


Рис. 5. Интерфейс приложения до загрузки данных

После загрузки данных с веб-сайта приложение принимает вид, показанный на рис. . Полученная информация о выбранной реки может быть изменена с помощью нового диалогового окна.

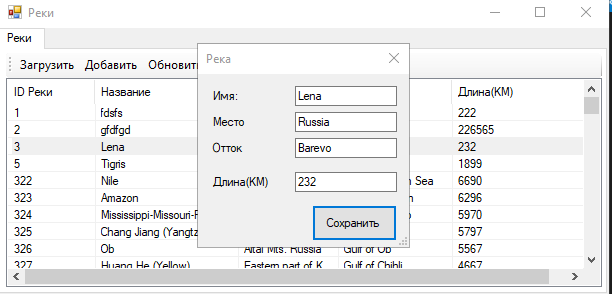


Рис. 6. Интерфейс приложения после загрузки данных с сервера

# ОПИСАНИЕ ПРОЦЕССА ТЕСТИРОВАНИЯ ПРОГРАММЫ

В ходе тестирования работоспособности программы было обнаружено, что при вводе пользователем данных может возникнуть дублирование информации Также в программу была добавлена обработка исключений, связанных с этими индексами (рис. ).

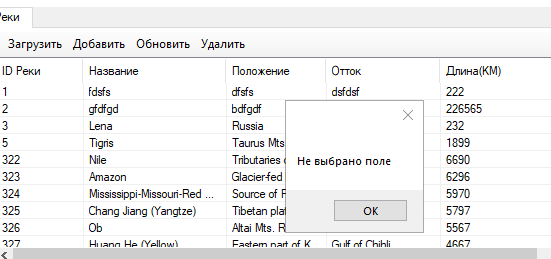


Рис. 7. Попытка удалить пустое место

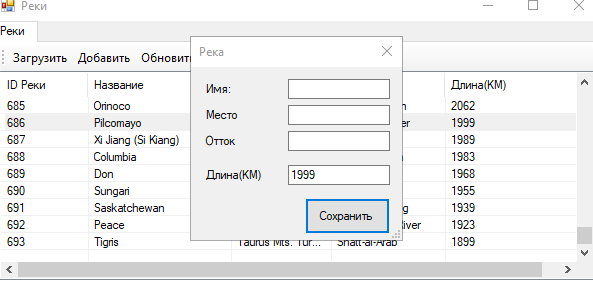


Рис. 8. Попытка удалить данные выдаст ошибку обновления

# ЗАКЛЮЧЕНИЕ

В результате проделанной работы было разработано приложение Windows Forms для автоматизированного сбора, хранения и обработки данных информации о реках с выбранного веб-ресурса.

Приложение работает без критических ошибок и полностью соответствует требованиям поставленной задачи.

Также в результате проделанной работы мною были получены навыки по работе с базами данных, пользовательским интерфейсом приложений и парсингом информации с сервера.

# СПИСОК ЛИТЕРАТУРЫ

1. Официальный сайт «mostinfo.su» [Электронный ресурс]. – Режим доступа: https://mostinfo.su/ свободный (12.11.2021).
2. Официальный сайт «HTML Agility Pack» [Электронный ресурс]. – Режим доступа: [http://html-agility-pack.net](http://html-agility-pack.net/) свободный (12.11.2021).
3. Нейгел, К. C# 5.0 и платформа .NET 4.5 для профессионалов / К.Нейгел, Б.Ивьен, Дж.Глинн, М.Скиннер, К.Уотсон. – М.: Диалектика, 2013. – 1440с.
4. Троелсен, Э. С# 2008 и платформа .NET 3.5 Framework / Э.Троелсен. – 4-е изд. – М.: Вильямс, 2009. – 1368 с.
5. Неш, Т. C# 2008: ускоренный курс для профессионалов: [пер. с англ.] / Т.Неш – М.: Вильямс, 2008. – 576 с.