**Министерство образования Российской Федерации**

**Пензенский государственный университет**

Кафедра «Математическое обеспечение и применение ЭВМ»

Пояснительная записка к курсовому проекту   
по дисциплине

«Объектно-ориентированное программирование»

Разработка программы с использованием объектно-ориентированного подхода.   
ИС«Конкурс поэтов»

|  |  |
| --- | --- |
| Автор работы: | Головинов М.Ю. |
| Направление бакалавриата | 09.03.04 («Программная инженерия») |
| Обозначение курсовой работы | ПГУ 09.03.04 – 04КР211.19 ПЗ |
| Группа | 21ВП1 |
| Руководитель работы | Афонин А.Ю., к.т.н., доцент |
| Работа защищена «\_\_» \_\_\_\_ 2024г. | Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Пенза 2024

**Содержание**

[Введение 5](#_Toc136299746)

[1 Постановка задачи 6](#_Toc136299747)

[2 Выбор решения 8](#_Toc136299748)

[2.1 Определение необходимых модулей программы 8](#_Toc136299749)

[2.2 Определение структуры файла базы данных 8](#_Toc136299750)

[3 Описание разработки программы 9](#_Toc136299751)

[4 Отладка и тестирование 16](#_Toc136299752)

[5 Описание программы 24](#_Toc136299753)

[5.1 Разработка приложения 24](#_Toc136299754)

[5.2 Разработка меню 26](#_Toc136299755)

[6 Руководство пользователя 31](#_Toc136299756)

[Заключение 32](#_Toc136299757)

[Список используемых источников 33](#_Toc136299758)

[Приложение А - Руководство пользователя 34](#_Toc136299759)

[Приложение В - Исходные тексты программы 43](#_Toc136299760)

# **Введение**

Для эффективного учета поэтов и литературных критиков, зарегистрированных на конкурсе необходимо иметь систему, способную учитывать и анализировать данные о каждом отдельно взятом человеке.

В общем смысле информационные системы – это эффективная организация деятельности, часто позволяющая избавиться от больших объемов бумажного документооборота.

Таким образом информационная система – это рабочая система, деятельность которой направлена на сбор, передачу, хранение, извлечение, обработку и отображение информации.

Цель данной курсовой работы заключается в разработке информационной системы, которая позволит хранить и обрабатывать данные, требующиеся на конкурсе поэтов. Создание такой системы предоставит возможность автоматизировать процессы учета конкурсантов и их работ. Разрабатываемое приложение позволяет пользователям подключаться к единой базе данных, добавлять и удалять записи, осуществлять фильтрацию и поиск записей по заданным критериям.

Для достижения поставленной цели нужно решить следующие задачи:

• изучить особенности работы с коллекциями и файлами на С#;

• разработать модель пользовательского интерфейса программного средства;

• разработать основной функционал приложения;

• провести тестирование разработанной программы;

• разработать руководство пользователя.

Разработка программы осуществлялась под управлением операционной системы Windows 11 в среде программирования Visual Studio Community 2022 на C#.

# **1 Постановка задачи**

Необходимо разработать программу «CourseWork». Программа должна содержать три раздела: раздел таблицы данных о всех участвующих в конкурсе поэтах, раздел критиков, раздел данных о конкурсных работах. В каждом из разделов будет реализован поиск

Раздел таблицы данных о поэтах:

* Поиск по номеру, имени или фамилии;
* Номер;
* Имя;
* Фамилия;
* Дата рождения.

В добавлении и редактировании критиков содержатся следующие информационные поля:

* Поиск по номеру, имени или фамилии;
* Имя;
* Фамилия;
* Номер телефона;
* Дата рождения.

Раздел добавления и редактирования экзаменов содержит следующие информационные поля:

* Номер поэта;
* Номер критика;
* Дата загрузки;
* Время загрузки;
* Текст работы.

Программа должна предоставить возможности: добавления нового участника конкурса и критика, удаления уже добавленного, добавления информации о литературной работе участника на конкурсе, сортировки, фильтрации и поиска по всем полям, а также удаление данных из файла.

Для реализации этих требований нужно разработать интуитивно понятный интерфейс пользователя.

Разработанное приложение должно содержать окно с информацией об авторе и теме проекта.

# **2 Выбор решения**

## 2.1 Определение необходимых модулей программы

Разработка программы осуществлялась под управлением операционной системы Windows 10 в среде программирования Visual Studio Community 2022 на основе шаблона приложения Windows Forms (.NET Framework) для языка программирования C#.

Для работы с файлом базы данных был использован класс SqliteConnection из встроенной библиотеки C#.

## 2.2 Определение структуры файла базы данных

Данные разрабатываемого приложения будут храниться в СУБД SQLite. SQLite является одной из наиболее популярных СУБД. Для работы с базой данной SQLite не требуется наличие сервера, как в случае с MS SQL Server. И .NET предоставляет возможности для работы с SQLite в рамках приложения на языке C#.

Для выполнения запросов к базе данных SQLite применяется класс SqliteCommand, который представляет реализацию интерфейса System.Data.IDbCommand. В базе данных будет представлена реализация классов создания базы данных и классов с подписью Service, которые будут отвечать за обращение к базе данных [1].

В разрабатываемом приложении будет реализована возможность создавать базы данных и загружать уже созданные. Каждая база хранит в себе информацию о поэтах, участвующих в конкурсе, их работах и критиках.

# **3 Описание разработки программы**

Разработанное приложение должно выполнить следующие функции:

* Добавить поэта в базу;
* Удалить поэта из базы;
* Добавить критика в базу;
* Удалить критика из базы;
* Сортировать;
* Искать по буквам, содержащимся в фамилии;
* Добавить работу в базу.

В процессе описания разработки программы была составлена диаграмма вариантов использования.

Диаграмма вариантов использования [2], разработанная с помощью средства UML моделирования – UMLet версии 15, приведена на рисунке 1.

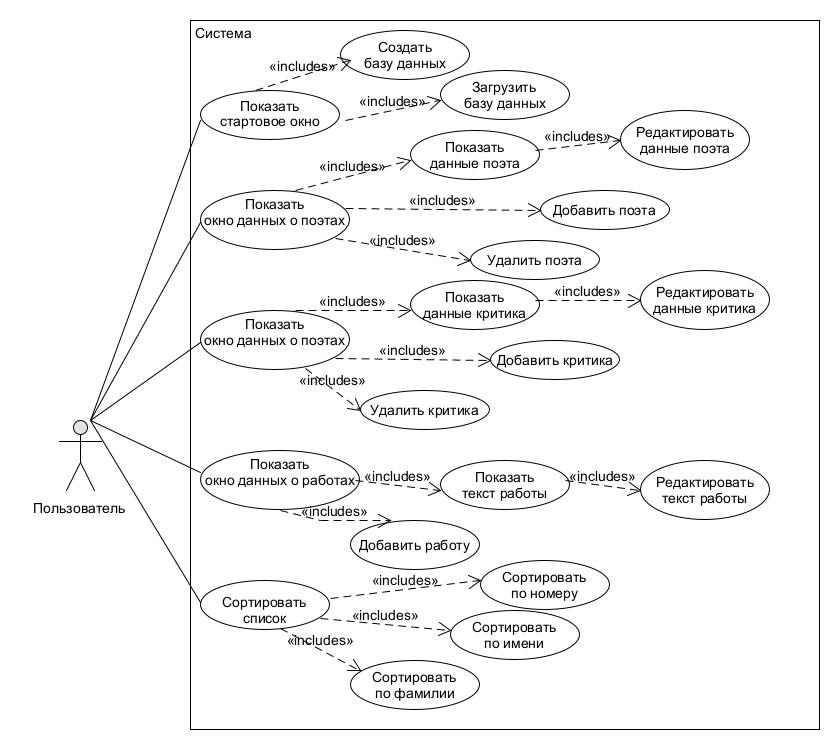


Рисунок 1 – Диаграмма вариантов использования

Описания некоторых спецификаций прецедентов представлены в таблицах 1-3.

Таблица 1 – Спецификация прецедента «**Редактировать данные поэта**»

|  |
| --- |
| **Наименование: Редактировать данные поэта** |
| **ID: 1** |
| **Краткое описание:** пользователь заполняет информационные поля, программа добавляет запись с этой информацией в базу |
| **Действующие лица:** пользователь, программа |
| **Основной поток:**   1. Пользователь заполняет информационные поля. 2. Пользователь нажимает на кнопку «Добавить». 3. Программа проверяет корректность введённых данных. 4. Программа добавляет запись в базу. |
| **Постусловие:** Запись добавлена в базу |

Таблица 2 – Спецификация прецедента «**Удалить поэта**»

|  |
| --- |
| **Наименование: Удалить поэта** |
| **ID: 2** |
| **Краткое описание:** пользователь выбирает поэта для удаления, программа удаляет все данные о нем |
| **Действующие лица:** программа, пользователь |
| **Основной поток:**   1. Пользователь выбирает строку из таблицы с записями. 2. Во всплывшем окне пользователь нажимает на кнопку «Удалить». 3. Программа удаляет данные поэта |
| **Постусловие:** Запись о поэте удалена из базы |

Таблица 3 – Спецификация прецедента «**Поиск поэта по фамилии**»

|  |
| --- |
| **Наименование: Поиск поэта по фамилии** |
| **ID: 3** |
| **Краткое описание:** пользователь вводит фамилию поэта или часть букв из нее, программа отображает записи, содержащие эти буквы |
| **Действующие лица:** программа, пользователь |
| **Основной поток:**  1. Пользователь вводит информацию (букву или несколько букв) в текстовое поле.  2. Программа выводит в таблицу данные, соответствующие введенным пользователем. |
| **Постусловие:** записи, содержащие указанные буквы, выведены в таблицу |

Для описания структуры программного обеспечения была разработана диаграмма классов, представленная на Рисунке 2.

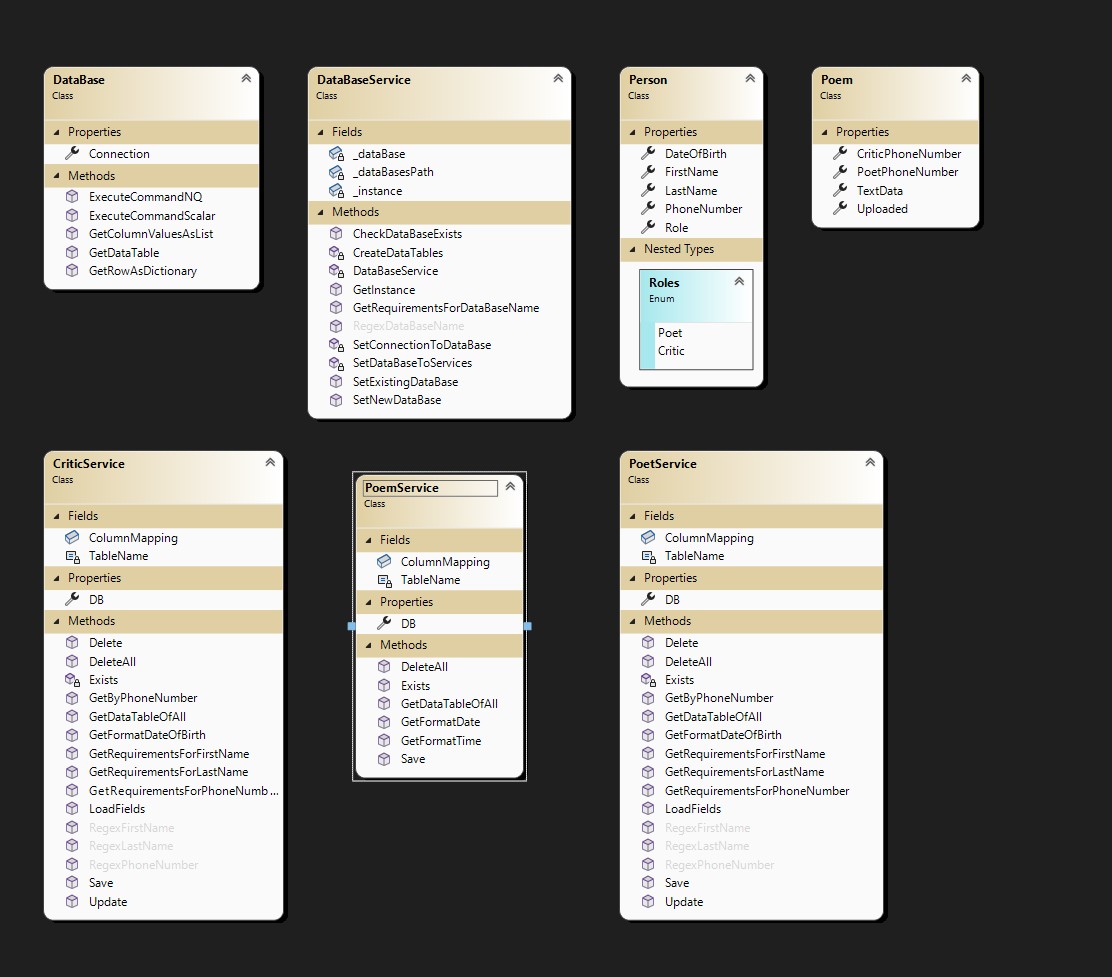


Рисунок 2 – Диаграмма классов

Программа содержит семь основных классов: DataBase, DataBaseService, Person, Poem, CriticService, PoemService и PoetService.

* Класс Data Base

Назначение класса: класс, отвечаюий за соединение с базой данных.

Методы класса:

* ExecuteCommandNQ выполняет команду, которая не возвращает результат (например, INSERT, UPDATE, DELETE). Он открывает соединение с базой данных, выполняет команду и закрывает соединение.
* ExecuteCommandScalar выполняет команду SQL, которая возвращает единственное значение из базы данных.
* GetDataTable возвращает результат в виде DataTable.
* GetRowAsDictionary возвращает результат в виде словаря, где ключи - названия столбцов, а значения - данные из соответствующих столбцов. Он считывает данные с помощью SQLiteDataReader и формирует словарь.
* GetColumnValuesAsList выводит значения из одного столбца как список строк.
* Класс DataBaseService

Назначение класса: класс содержит методы для настройки и работы с базой данных.

Поля класса:

* \_instance – поле, хранящее экземпляр класса DataBaseService
* \_dataBasesPath – строковое поле для хранения пути к папке с базами данных
* \_dataBase – поле класса DataBase для хранения объекта базы данных

Методы класса:

* SetNewDataBase создает новый файл базы данных SQLite с указанным именем.
* SetExistingDataBase устанавливает соединение с существующим файлом базы данных.
* CreateDataTables создает необходимые таблицы (поэты, критики, работы) в базе данных с заданными определениями столбцов и ограничениями внешнего ключа.
* SetConnectionToDataBase устанавливает строку подключения к базе данных с помощью класса SQLiteConnection.
* SetDataBaseToServices назначает экземпляр базы данных другим службам (PoetService, CriticService, PoemService), чтобы разрешить доступ к данным.
* CheckDataBaseExists проверяет, существует ли файл базы данных с указанным именем в базе данных.
* Класс Person

Назначение класса: класс используется для представления физического лица (поэта или критика) в системе.

Свойства класса:

* DateOfBirth – дата рождения
* FirstName – имя
* LastName – фамилия
* PhoneNubmer – номер телефона
* Role – кем лицо является
* Класс Poem

Назначение класса: класс используется для представления конкурсной работы в системе.

Свойства класса:

* CriticPhoneNumber – номер телефона критика
* PoetPhoneNumber – номер телефона поэта
* TextData – текст конкурсной работы
* Uploaded – время загрузки работы в базу
* Класс CriticService

Назначение класса: класс инкапсулирующий логику работы с данными критиков.

Поля класса:

* TableName – константа, содержащая имя таблицы в базе данных
* ColumnMapping – словарь, отображающий имена свойств объекта Person, соответствующие именам столбцов в базе данных [3][4]

Свойства класса:

* DB – доступ к базе данных

Методы класса:

* GetByPhoneNumber(string phoneNumber) – метод для получения информации о критике по его номеру телефона.
* GetDataTableOfAll() – метод для получения всех критиков в виде DataTable.
* Save(Person critic) – метод для добавления нового критика в базу данных.
* Delete(string phoneNumber) – метод для удаления критика по его номеру телефона.
* Update(Person critic) – метод для обновления информации о критике.
* DeleteAll() – метод для удаления всех критиков из базы данных.
* GetFormatDateOfBirth(DateTime dateOfBirth) – вспомогательный метод для форматирования даты рождения критика.
* LoadFields(string phoneNumber) – метод для загрузки полей критика из базы данных.
* Exists(string phoneNumber) – метод для проверки существования критика с заданным номером телефона в базе данных.
* Класс PoemService

Назначение класса: класс инкапсулирующий логику работы с данными конкурсных работ.

Поля класса:

* TableName – константа, содержащая имя таблицы в базе данных
* ColumnMapping – словарь, отображающий имена свойств объекта Person, соответствующие именам столбцов в базе данных

Свойства класса:

* DB – доступ к базе данных

Методы класса:

* Save(Poem poem) – метод для сохранения данных о стихотворении в базу данных. Проверяет, не существует ли уже добавленная работа у данного автора.
* DeleteAll() – метод для удаления всех данных о стихотворениях из базы данных.
* GetDataTableOfAll() – метод для получения всех данных о стихотворениях в виде таблицы.
* Exists(Poem poem) – метод для проверки существования данных о стихотворении в базе данных.
* GetFormatDate(DateTime date) – метод для форматирования даты в формат для записи в базу данных.
* GetFormatTime(DateTime time) – метод для форматирования времени в формат для записи в базу данных.
* Класс PoetService

Назначение класса: класс инкапсулирующий логику работы с данными поэтов.

Поля класса:

* TableName – константа, содержащая имя таблицы в базе данных
* ColumnMapping – словарь, отображающий имена свойств объекта Person, соответствующие именам столбцов в базе данных

Свойства класса:

* DB – доступ к базе данных

Методы класса:

* GetByPhoneNumber(string phoneNumber) – метод, который возвращает объект Person (поэт) по его номеру телефона.
* GetDataTableOfAll() – метод, который возвращает все данные по поэтам в виде DataTable.
* Save(Person poet) – метод для сохранения нового поэта в базе данных.
* Delete(string phoneNumber) – метод для удаления поэта из базы данных по его номеру телефона.
* Update(Person poet) – метод для обновления информации о поэте в базе данных.
* DeleteAll() – метод для удаления всех данных о поэтах из базы данных.

# **4 Отладка и тестирование**

В курсовой работе было выполнено функциональное тестирование разработанного программного обеспечения. План тестирования приведен в таблице 4 [5].

Таблица 4 – План тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Состав теста** | **Ожидаемый результат** | **Наблюдаемый результат** |
| 1 | Добавить поэта | Поэт занесен в таблицу | Данные о поэте отображены в таблице (Рисунок 3) |
| 2 | Удалить поэта | Поэт удален из таблицы | Данные о поэте удалены  (Рисунок 4) |
| 3 | При добавлении критика оставить поля пустыми | Выводится предупреждающая надпись | В пустом поле предупреждение о некорректности  (Рисунок 5) |
| 4 | Добавить работу одному из поэтов | Работа добавляется в базу данных | Работа добавлена в базу (Рисунок 6) |
| 5 | При добавлении поэта оставить поля пустыми | Появление предупреждения в пустом поле | В пустом поле предупреждение о некорректности  (Рисунок 7) |

Продолжение таблицы 4 – План тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | Отсортировать поэтов по первой букве фамилии | Записи отсортированы | Записи в таблице отсортированы по полю фамилии (Рисунок 8) |
| 7 | Ввести в поле телефона латинские буквы | Выводится информация о несоответствии | Вывелась информация о несоответствии (Рисунок 9) |
| 8 | Дважды нажать на поэта в таблице | Данные поэта становятся доступны для редактирования | Данные поэта стали доступны для редактирования  (Рисунок 10) |
| 9 | Запуск поиска при отсутствии данных | Программа проигнорирует запуск поиска | Программа игнорирует запуск поиска  (Рисунок 11) |
| 10 | Выполнить поиск несуществующей записи | Программа выводит пустую таблицу | Программа выводит пустую таблицу  (Рисунок 12) |
| 11 | Выполнить поиск по имени | Таблица отображает записи, в которых присутствует введенное поле | Таблица отображает запись, в которой найдено данное поле  (Рисунок 13) |

На рисунках 3–13, приведены скриншоты, отражающие результаты работы программы в процессе тестирования.

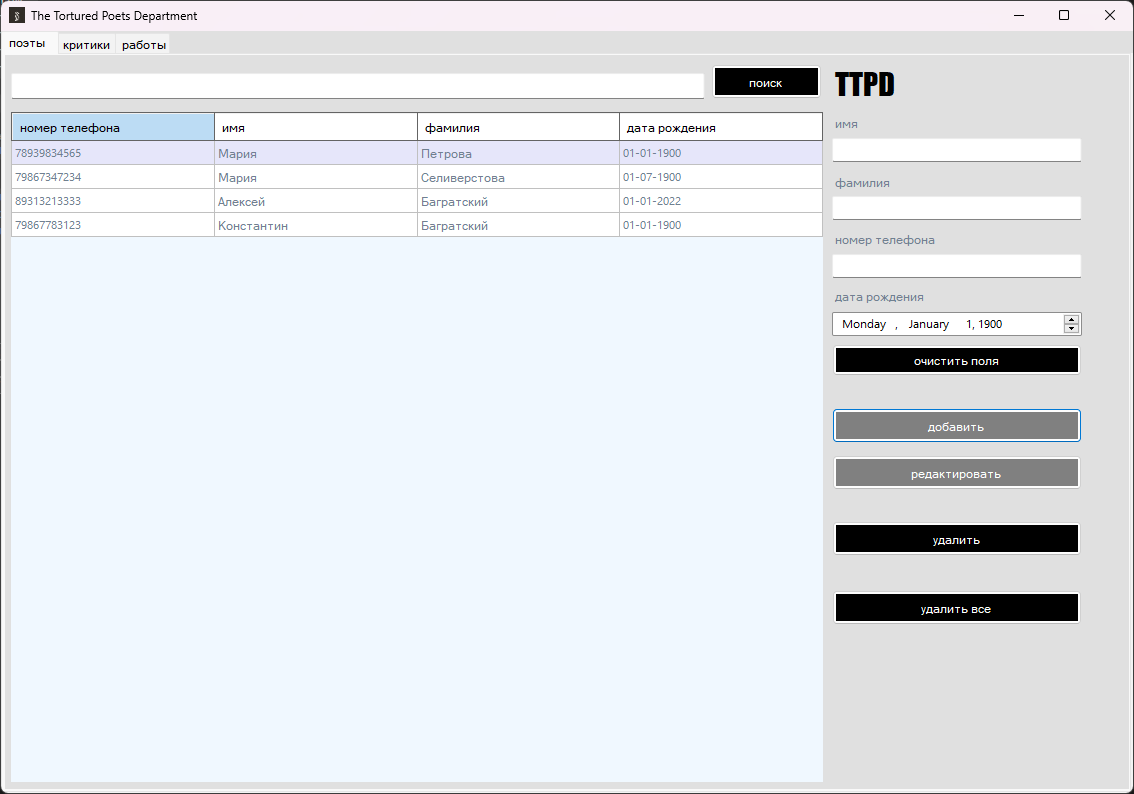


Рисунок 3 – Добавление поэта в таблицу

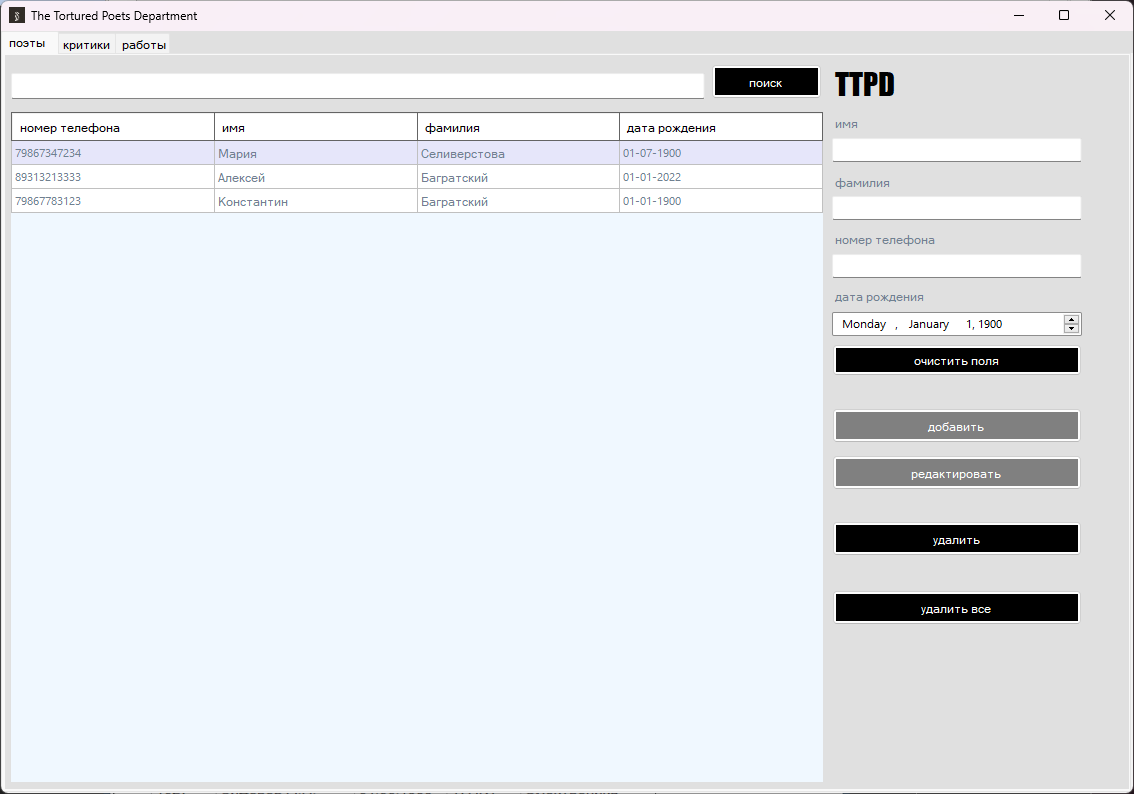


Рисунок 4 – Добавленный поэт удален из таблицы

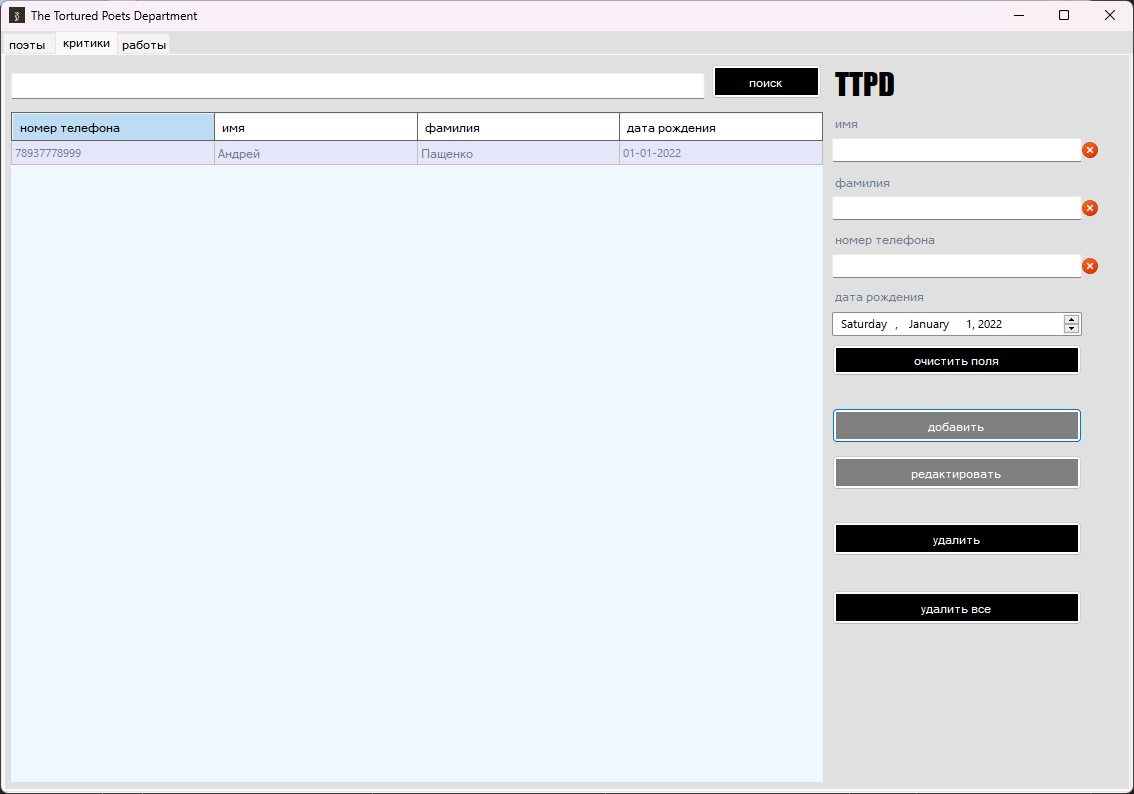


Рисунок 5 – Оставление полей критика пустыми

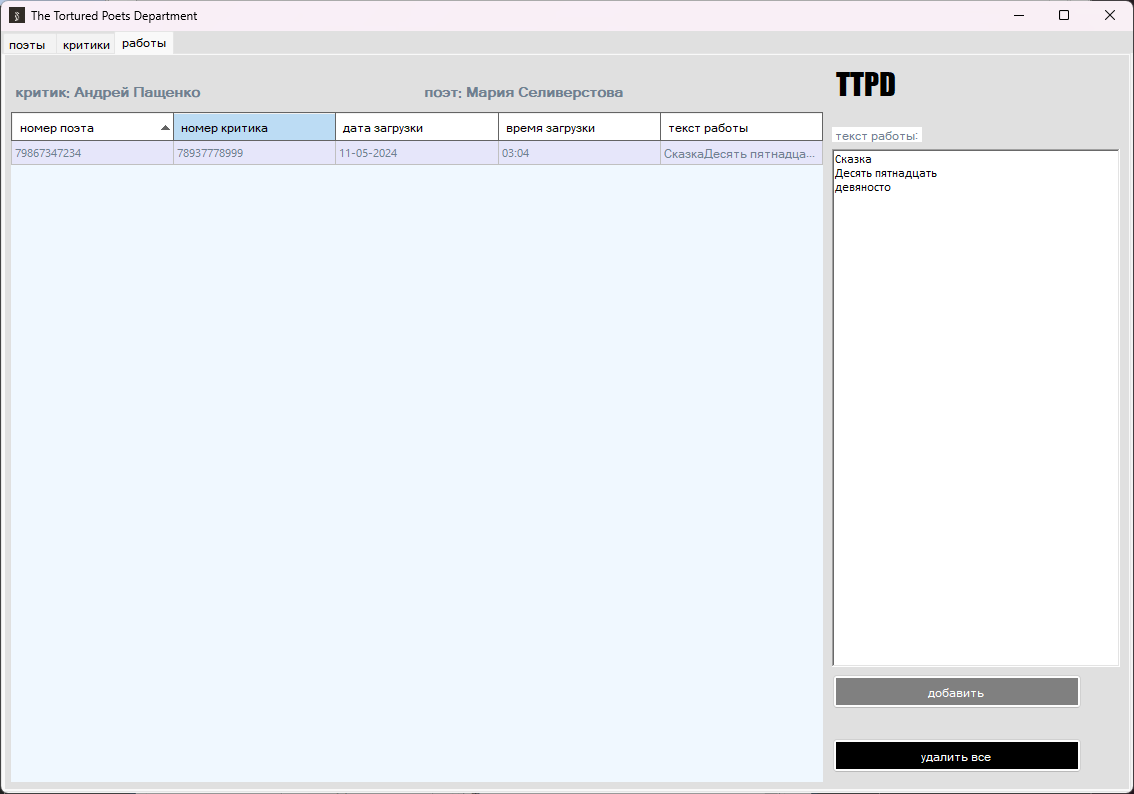


Рисунок 6 – Добавление работы

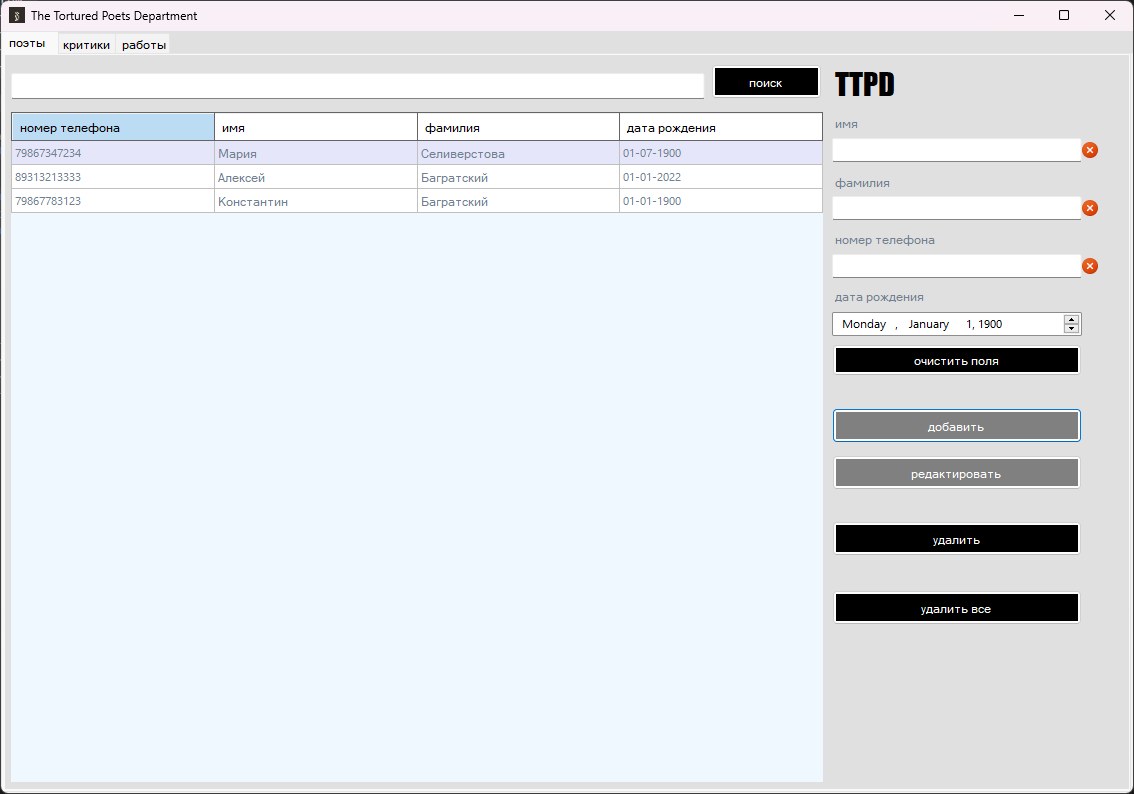


Рисунок 7 – Оставление полей пустыми

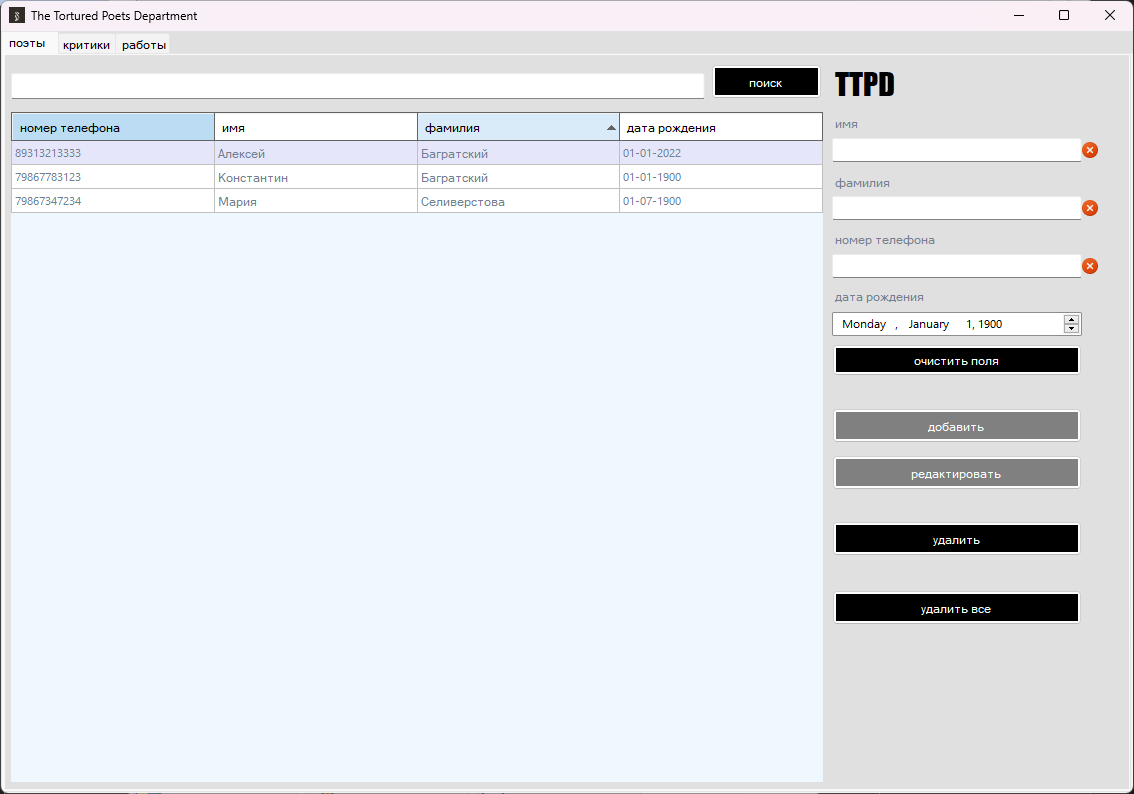


Рисунок 8 – Сортировка по фамилии

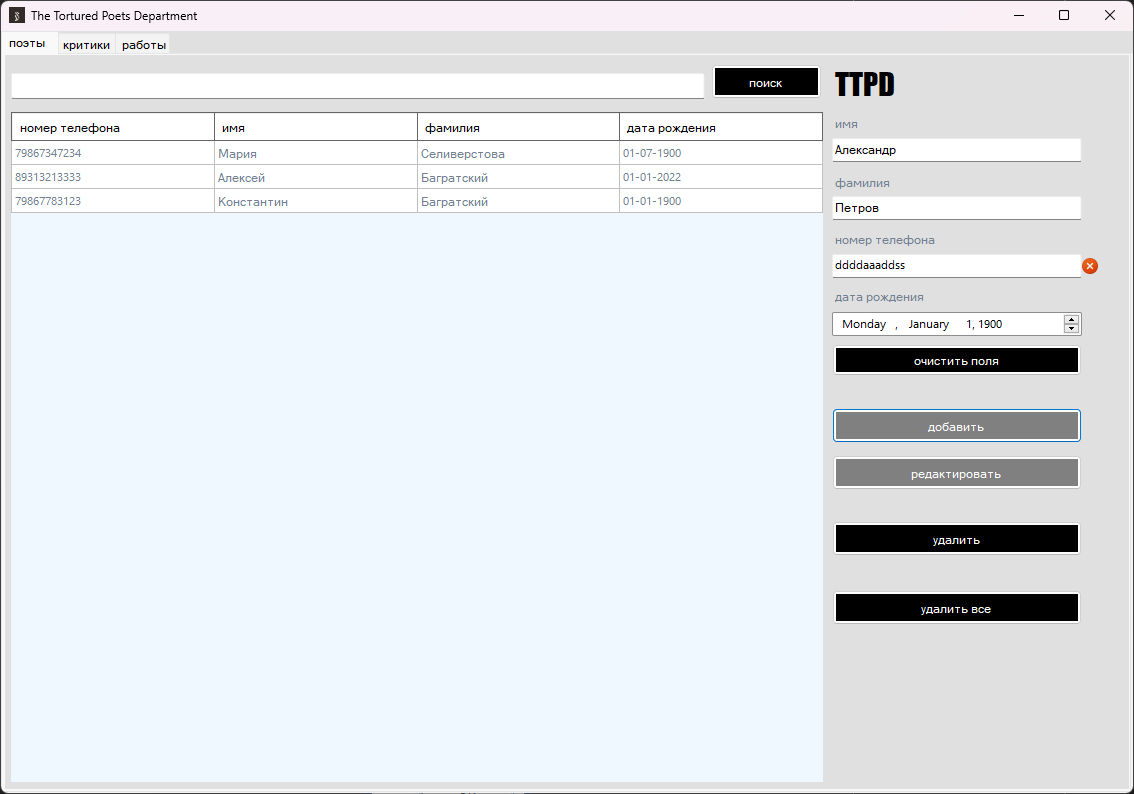


Рисунок 9 – Ввод номера латинскими буквами

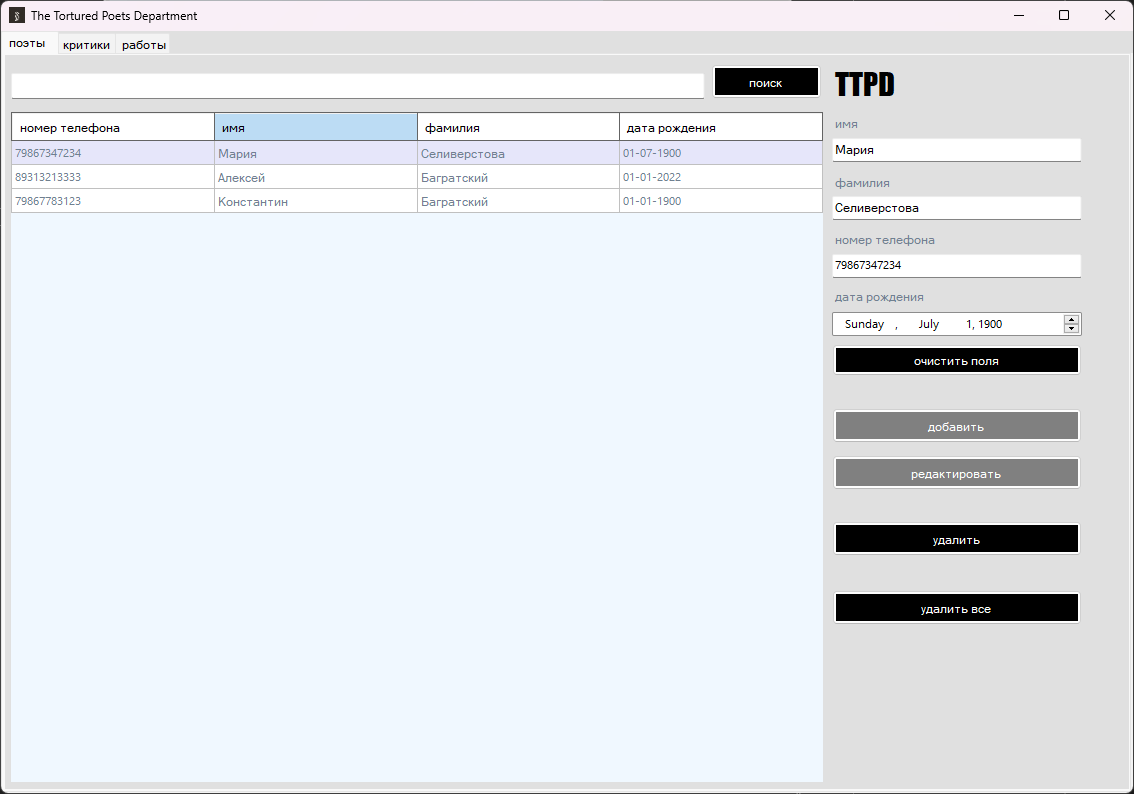


Рисунок 10 – Нажатие на строку поэта

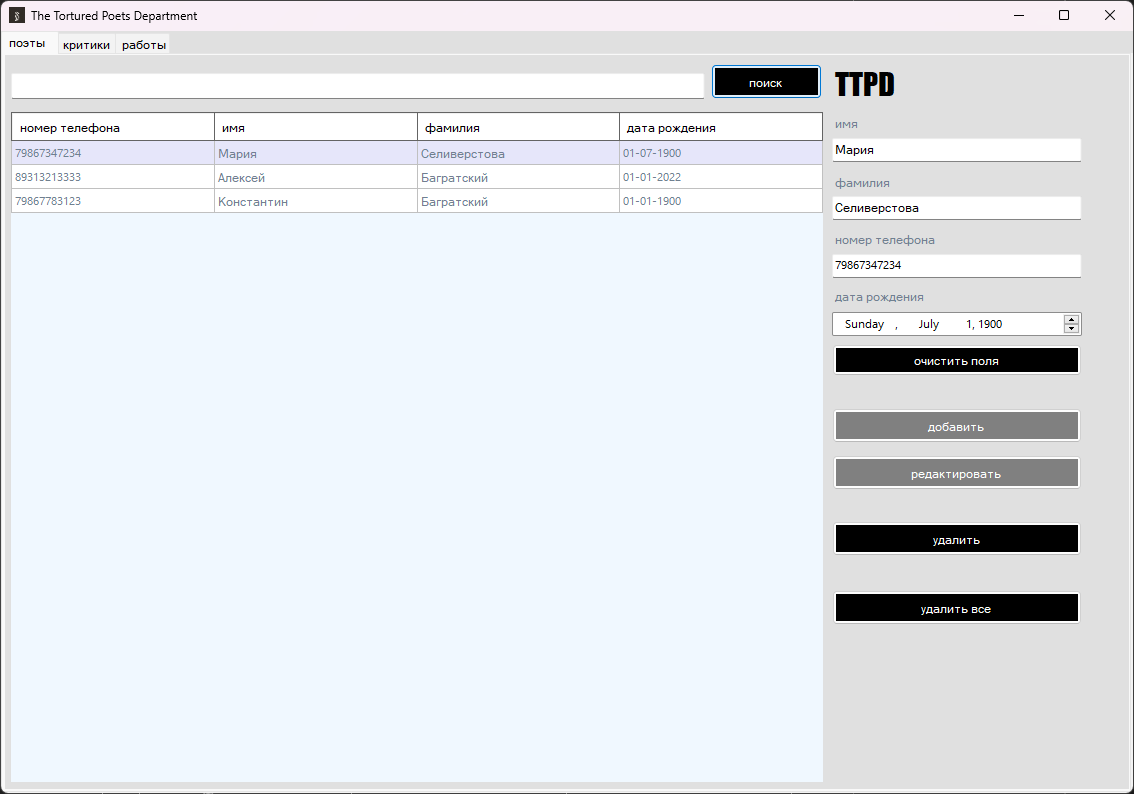


Рисунок 11 – Запуск поиска при отсутствии данных

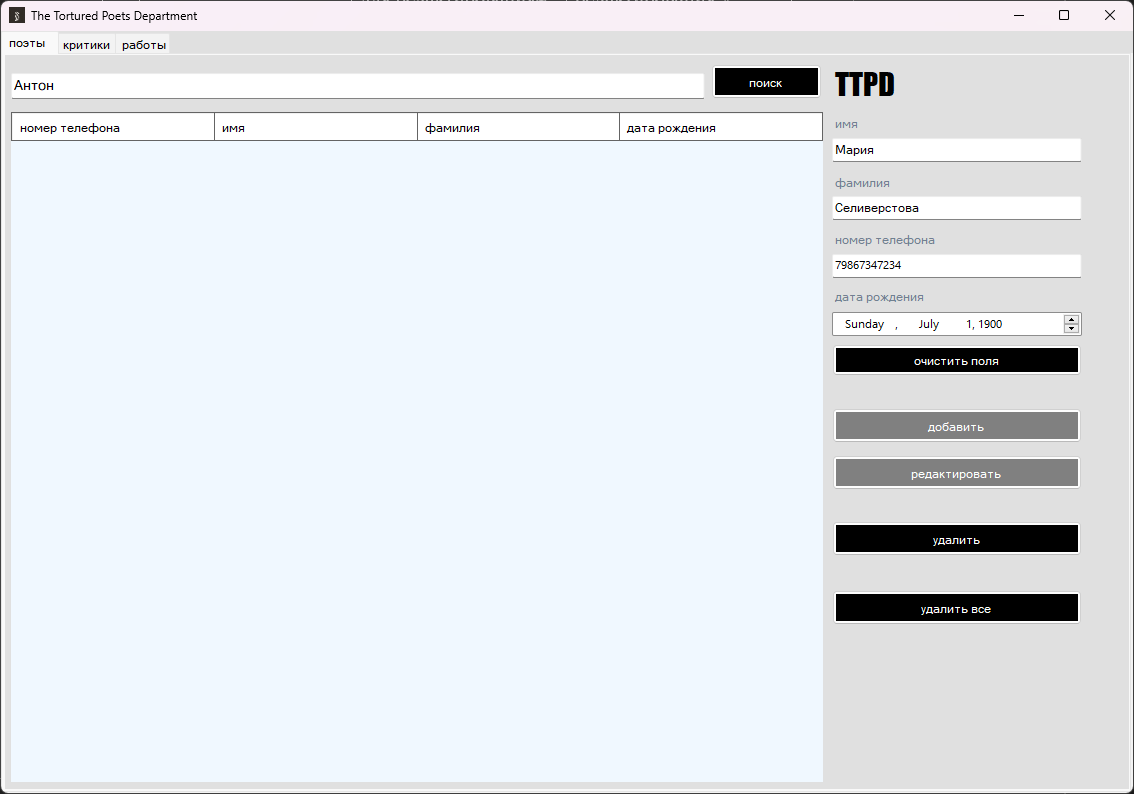


Рисунок 12 – Поиск несуществующей записи

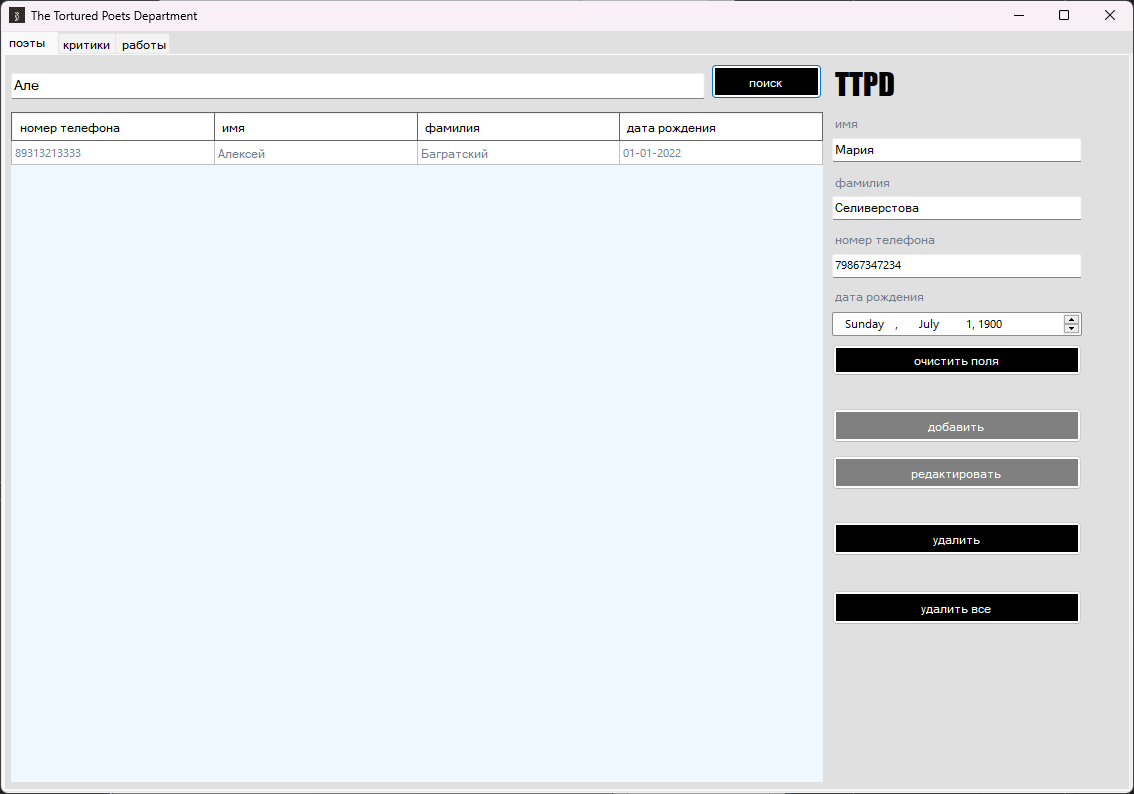


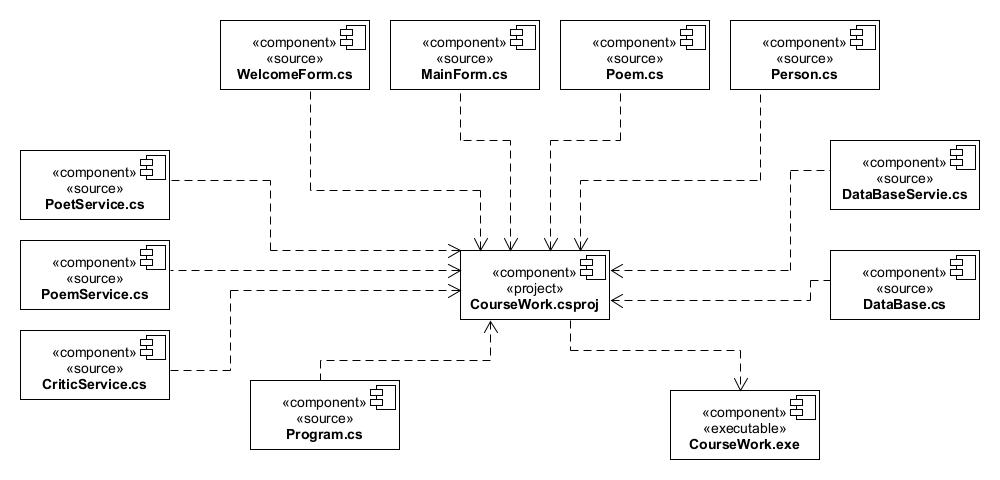
Рисунок 13 – Поиск по имени

В ходе выполнения тестирования несовпадения ожидаемого и наблюдаемого результата не выявлены. Следовательно, можно сделать вывод, что программа работает корректно.

# **5 Описание программы**

## 5.1 Разработка приложения

В процессе выполнения курсовой работы была составлена диаграмма компонентов, которая отображает разбиение программной системы на структурные компоненты и связи между ними. Диаграмма компонентов приведена на рисунке 14.

Рисунок 14 – Диаграмма компонентов

Описание компонентов приведено в таблице 5. [6]

Таблица 5 – Описание компонентов

|  |  |
| --- | --- |
| **Компонент** | **Назначение** |
| CourseWork.exe | Исполняемый файл приложения |
| Program.cs | Главный файл программы |
| CriticService.cs | Файл для работы с данными критиков |
| PoetService.cs | Файл для работы с данными поэтов |
| PoemService.cs | Файл для работы с данными по работам |
| WelcomeForm.cs | Форма создателя проекта и входа в бд |
| MainForm.cs | Форма основного меню и поиска |
| Poem.cs | Файл класса конкурсной работы |
| Person.cs | Файл физ. лица для представления поэта и критика |
| DataBase.cs | Файл для подключения к базе данных |
| DataBaseService.cs | Файл реализации сервиса работы с базой данных |

## 5.2 Разработка меню

Приложение состоит из 4 различных форм.

На рисунке 15 демонстрируется интерфейс формы представления.

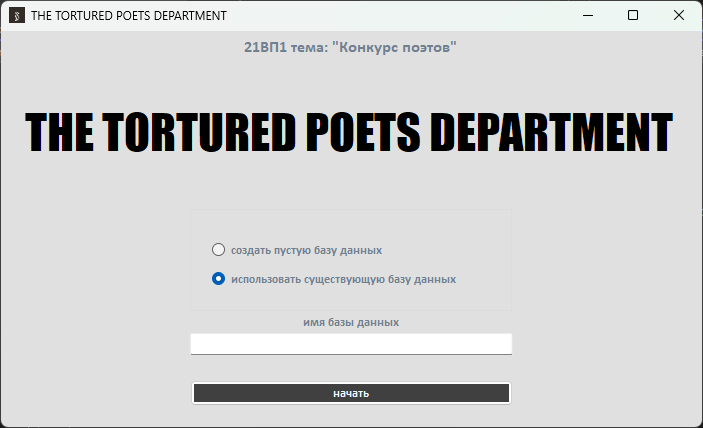


Рисунок 15 – Интерфейс приветственной формы

На рисунке 16 представлен интерфейс формы основной страницы. Раздел базы данных критиков выглядит аналогично.

Данная форма содержит следующие компоненты:

1, 2, 3 – переключение между разделами поэтов, критиков и работ;

4 – кнопка для активации поиска

5, 6, 7, 8 – поля данных о поэте

9 – кнопка для очистки полей

10, 11 – кнопки для добавления нового поэта или редактирования добавленного

12, 13 – удалить выбранного поэта или удалить всех

14 – таблица данных о поэтах

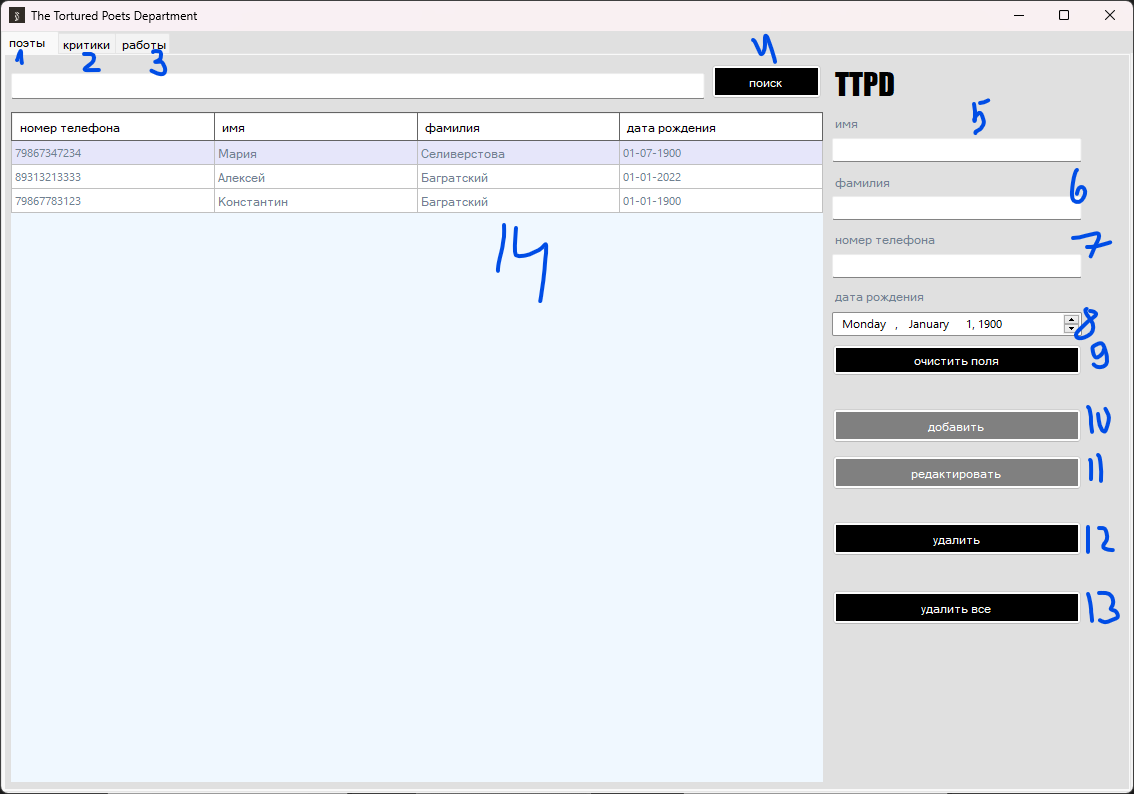


Рисунок 16 – Интерфейс формы данных о поэтах

На рисунке 17 представлен интерфейс страницы информации о критиках.

Данная форма содержит аналогичные предыдущей компоненты:

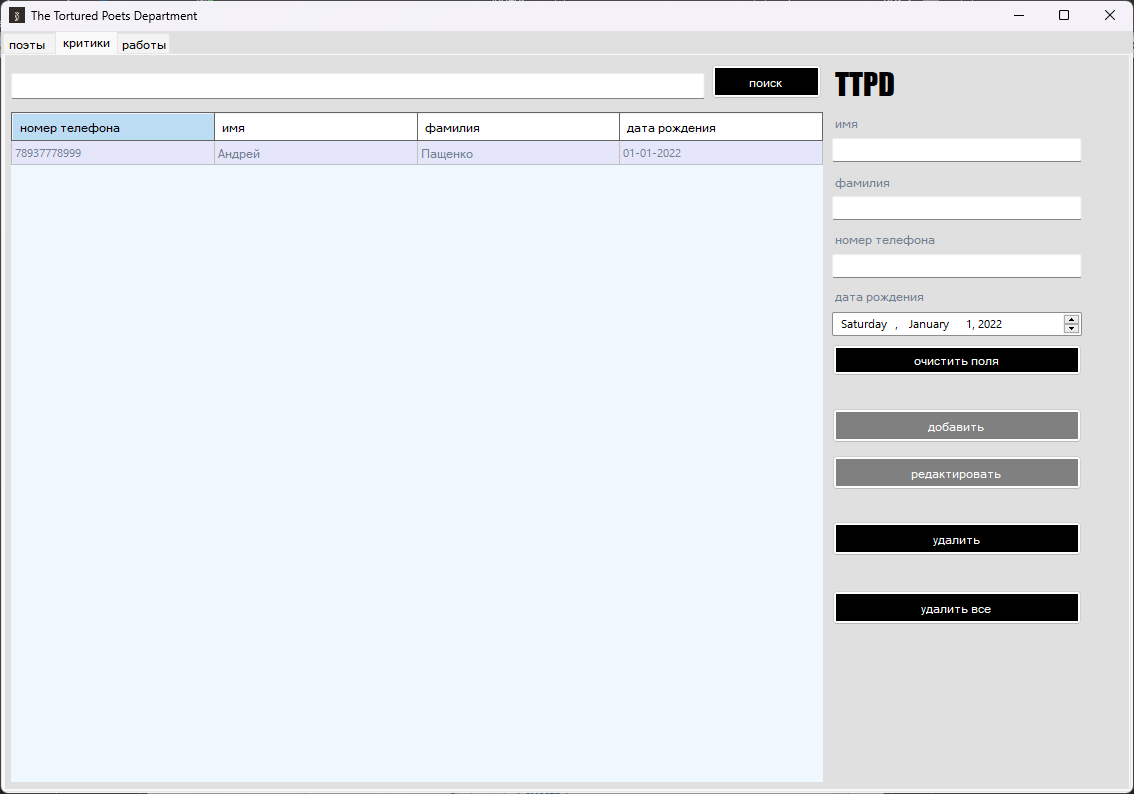


Рисунок 17 – Интерфейс формы отображения таблицы данных критиков

На рисунке 18 представлено редактирования информации о выбранном лице.

Данная форма содержит следующие компоненты:

1 – данные о принятых на конкурс работах;

2 – поле для отображения текста выбранной работы;

3 – кнопка для добавления новой работы в базу;

4 – кнопка для удаления всех загруженных работ

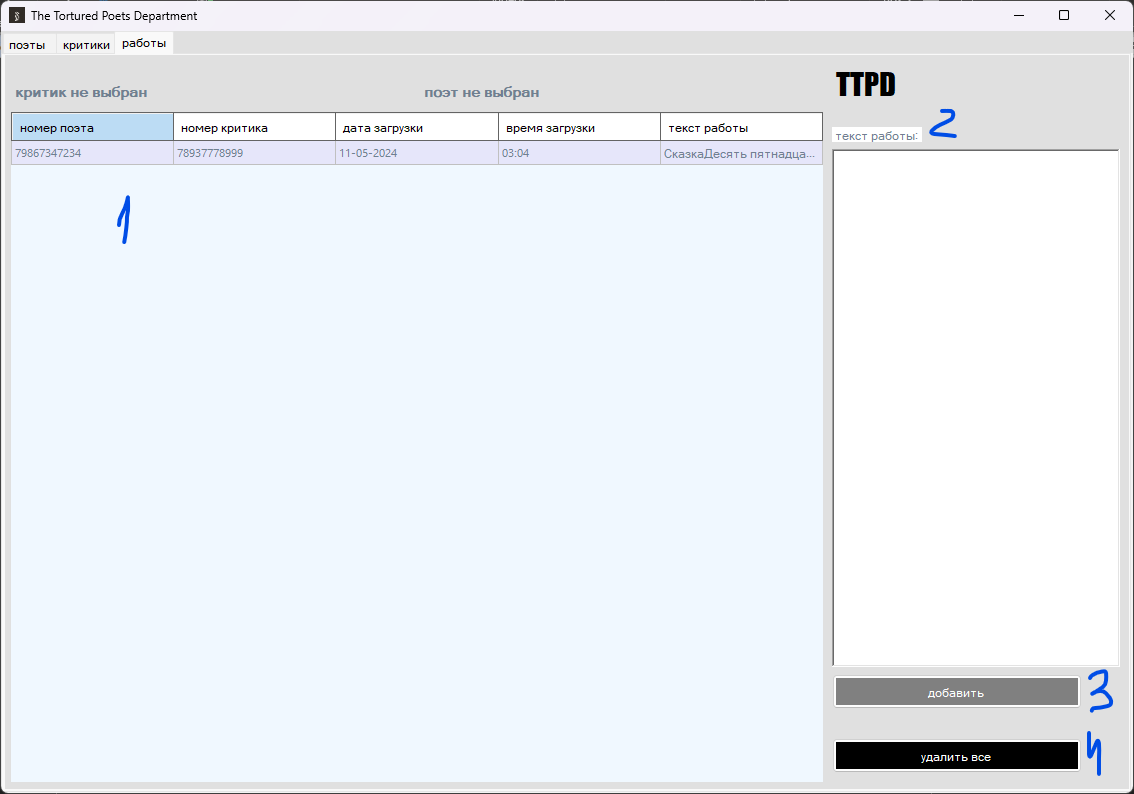


Рисунок 18 – Редактирование полей

# **6 Руководство пользователя**

Основная цель руководства пользователя заключается в обеспечении пользователя необходимой информацией для самостоятельной работы с программой, оно содержит описание шагов, которые необходимо выполнить для достижения пользователем конкретной цели. Текст руководства пользователя приведен в приложении A.

# **Заключение**

В ходе выполнения курсового проекта была разработана программа «CourseWork», позволяющая пользователям работать с базой данных, хранящей записи о поэтах, участвующиз в конкурсе и их работах. Приложение предоставляет основные инструменты для следующих действий: добавление записей в БД, удаление записей, поиск и сортировка записей. Сконструированное программное средство отвечает всем поставленным требованиям. Результаты тестирования показывают корректную работу приложения. Код программы приведен в приложении Б.

В процессе выполнения курсового проекта было разработано руководство пользователя, которое позволит быстро разобраться с функциональностью разработанного приложения.

Заключительным этапом разработки приложения стало создание инсталлятора.

Репозиторий проекта: <https://github.com/ntstlk/CourseWork_C-_21VP1_Golovinov>

# **Список используемых источников**

1. Подключение к базе данных SQLite: [caйт] URL: <https://metanit.com/sharp/adonetcore/4.1.php?ysclid=lw1bk4i3wl590986173> (Дата обращения: 01.05.2024)
2. Балашова, И. Ю. Современные информационные технологии в проектировании программных систем и комплексов: учеб. пособие / И. Ю. Балашова, Д. В. Такташкин; под ред. П. П. Макарычева. – Пенза : Изд-во ПГУ, 2019. − 106 с. (Дата обращения: 01.05.2024)
3. Коллекция Dictionary<K, V>: [caйт] URL: https://metanit.com/sharp/tutorial/4.9.php (Дата обращения 01.05.2024)
4. Шилдт, Герберт. C# 4.0: полное руководство.: Пер. с англ. – М.: ООО "И.Д. Вильямс", 2011. – 1056 с.: ил. (Дата обращения: 02.05.2024)
5. Функциональное тестирование программного обеспечения: [caйт] URL: <https://daglab.ru/funkcionalnoe-testirovanie-programmnogo-obespechenija/> (дата обращения: 03.05.2024)
6. Стиллмен Эндрю, Грин Дженнифер Head First. Изучаем C#. 4-е изд. / Пер. с англ. Е. Матвеева. — СПб.: Питер, 2022. — 768 с.: ил. (Дата обращения: 03.05.2024)

# **Приложение А - Руководство пользователя**

Программа CourseWork.exe предназначена для хранения информации о проводимом конкурсе поэтов. Программа имеет интуитивно понятный интерфейс, оснащена начальной страницей, где пользователь может выбрать необходимый раздел. Поддерживает такие операции как добавление нового лица и конкурсной работы, удаление выбранныз лиц из базы данных, а также несколько второстепенных операций, таких как фильтрация по полям или поиск по ним.

При запуске программы появляется окно приветствия.

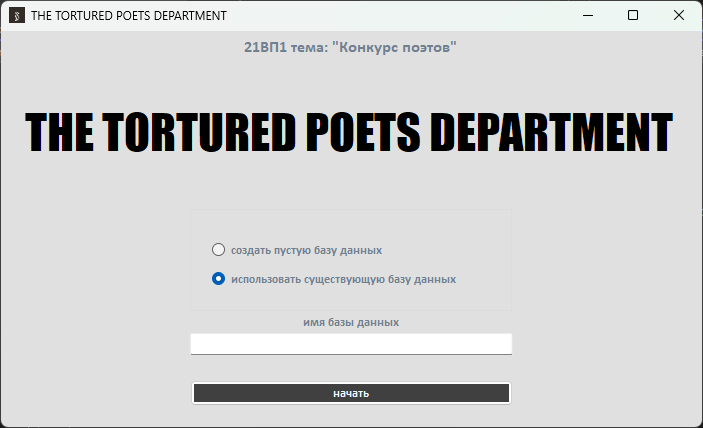


Рисунок А.1 – Окно приветствия

Далее появляется окно меню, где отображается таблица с данными и раздел фильтрации и поиска

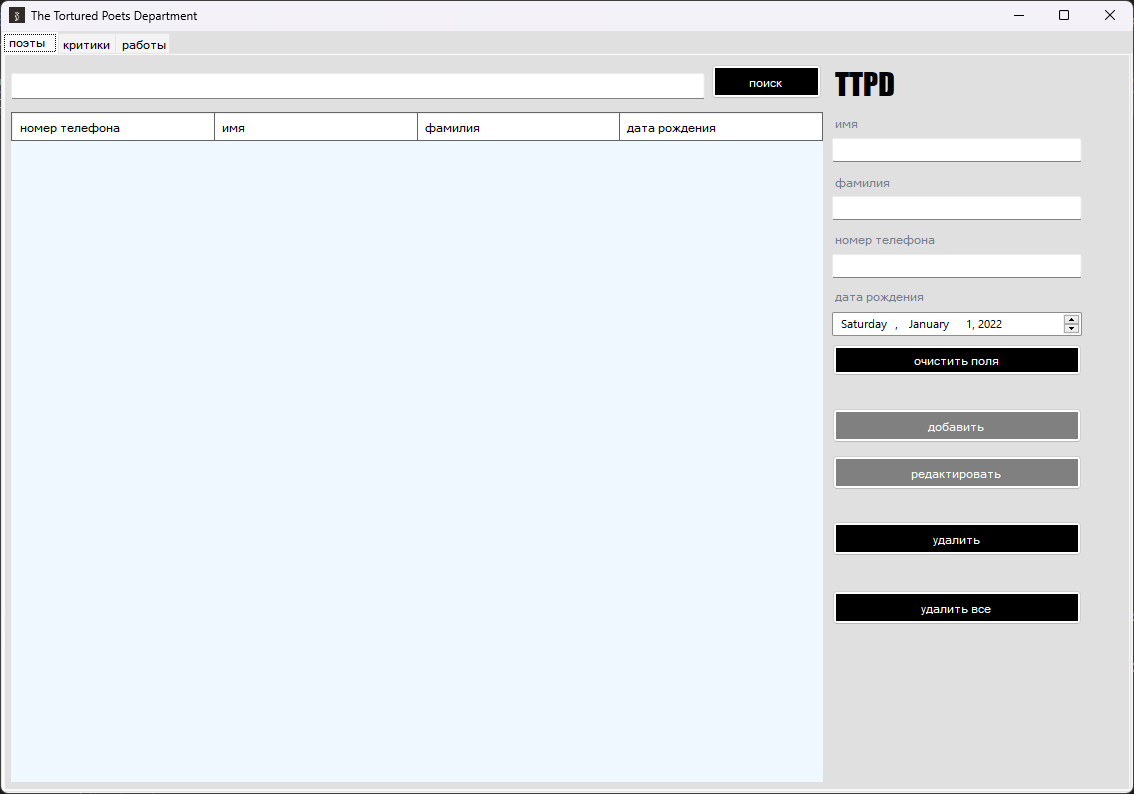


Рисунок А.2 – Окно меню

Данная таблица пуста, заполнить ее можно вручную с нуля, введя данные о поэте или о критике на соответствующих страницах, либо пользователь может заранее выбрать в приветственном окне «использовать существующую базу данных» и ввести название.

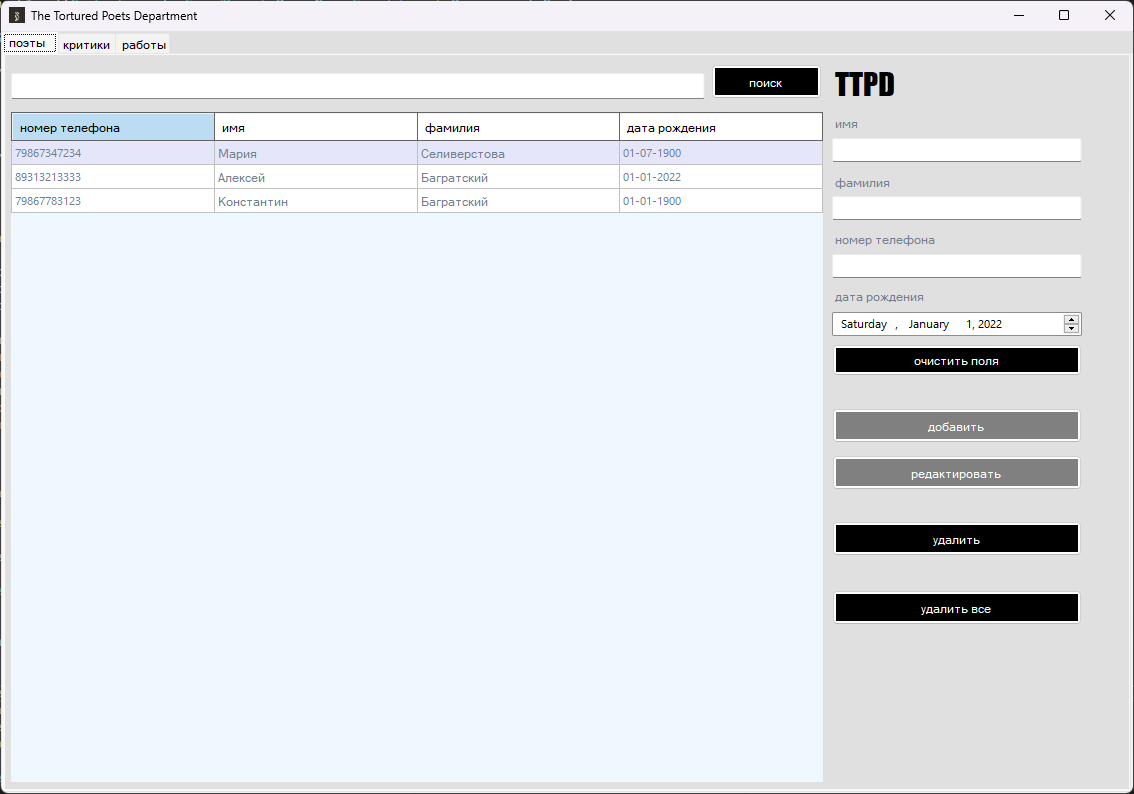


Рисунок А.3 – Заполненная таблица при загрузке уже существующей

При неудачной загрузке файла появится оповещение о том, что такой базы данных нет.

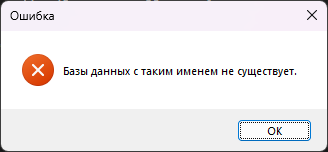


Рисунок А.4 – Оповещение о том, что введенное название некорректно

Для добавления нового критика в таблицу необходимо также вписать необходимые данные в текстовые поля и нажать кнопку «добавить».

В заполенной таблице пользователь может прибегнуть к редактированию полей.

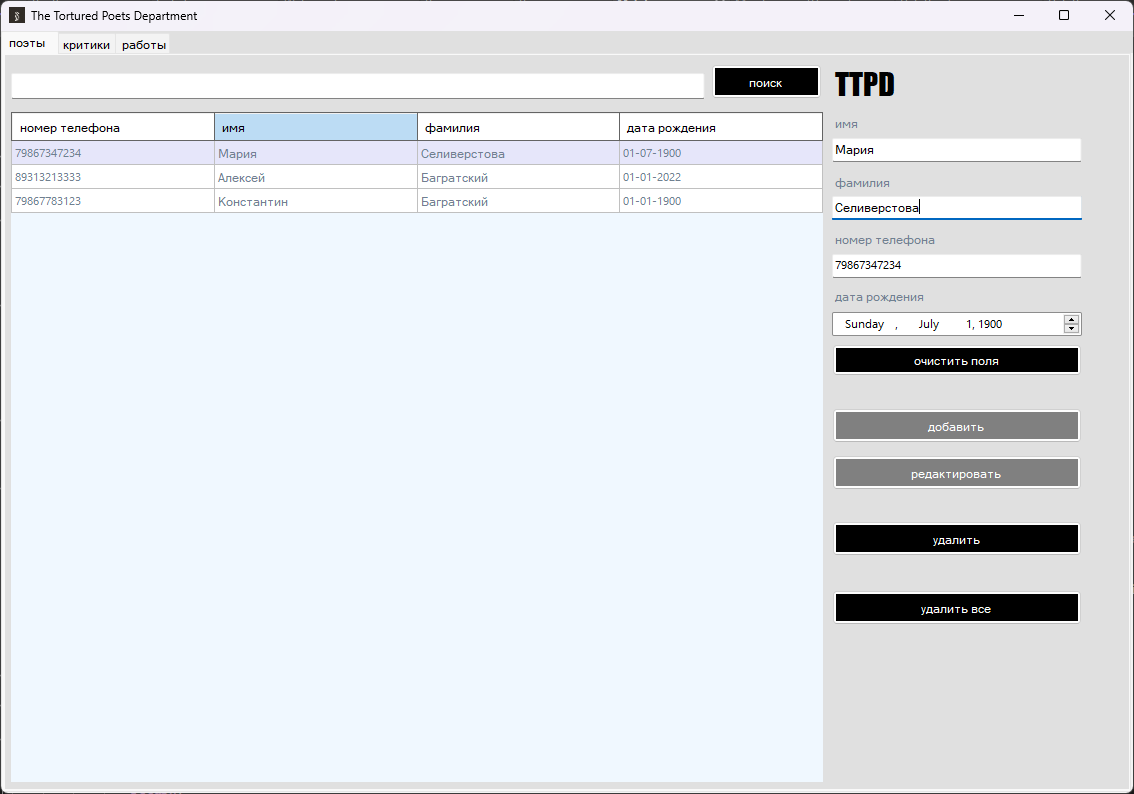


Рисунок А.5 – Редактирование данных

С помощью кнопки «Редактировать» мы можем сохранить обновленные данные о поэте или критике.

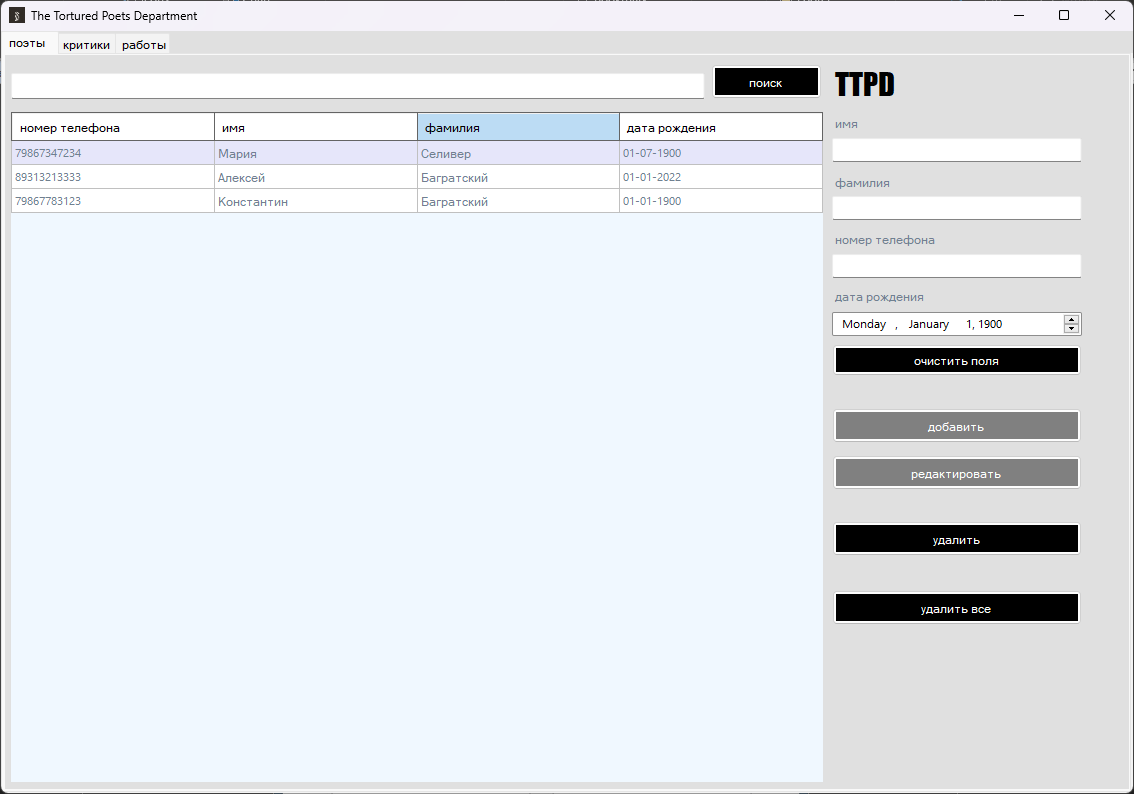


Рисунок А.6 – Данные отредактированы

При попытке добавить позицию с частично заполненными полями или же некорректными данными, на экран будут выведены надписи с предупреждениями, а неправильно заполненные поля будут подсвечены. Все пустые поля нужно заполнить, иначе информация о поэте не будет добавлена.

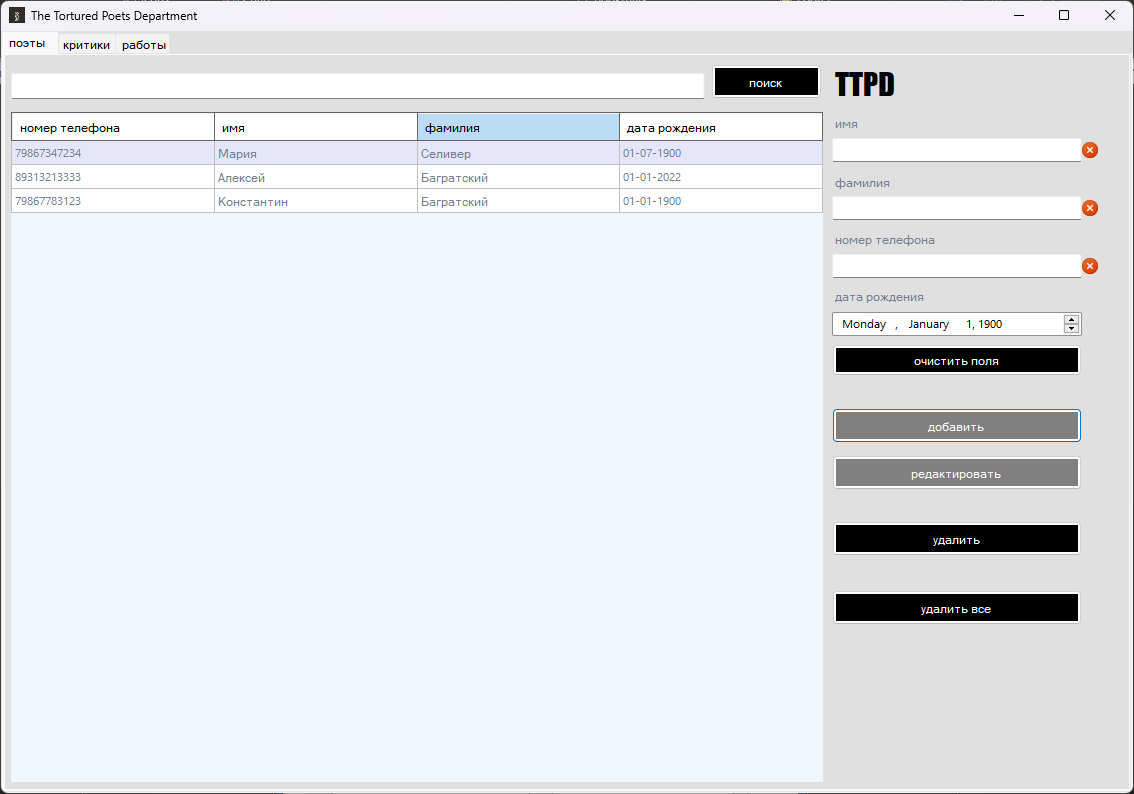


Рисунок А.7 – Добавление записи при наличии пустых полей

Если же в таблице добавились новые значения, то добавление лица прошло успешно и информация о нем добавлена в базу.

Для удаления позиции нужно выбрать строку в таблице, дважды щелкнув по ней и в форме отображения данных нажать на кнопку «Удалить». Все данные о выбранном лице удалятся из таблицы. При нажатии на кнопку «Удалить все» будут удалены данные о всех добавленых лицах.

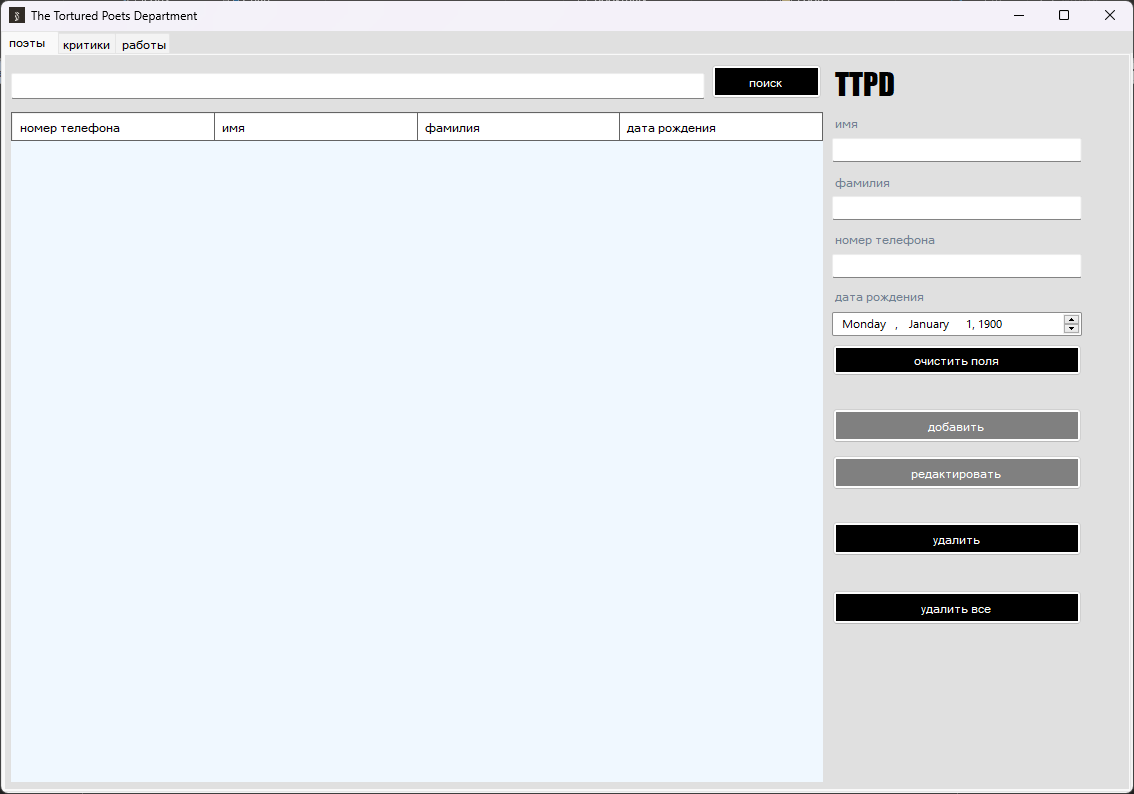


Рисунок А.8 – Все поэты удалены

Для сортировки по значению поля нужного пользователю необходимо вернуться в меню и выбрать нужное поле и нажать на него. После нажатия на поле, таблица сразу поменяет отображение позиций соответственно сортировке.

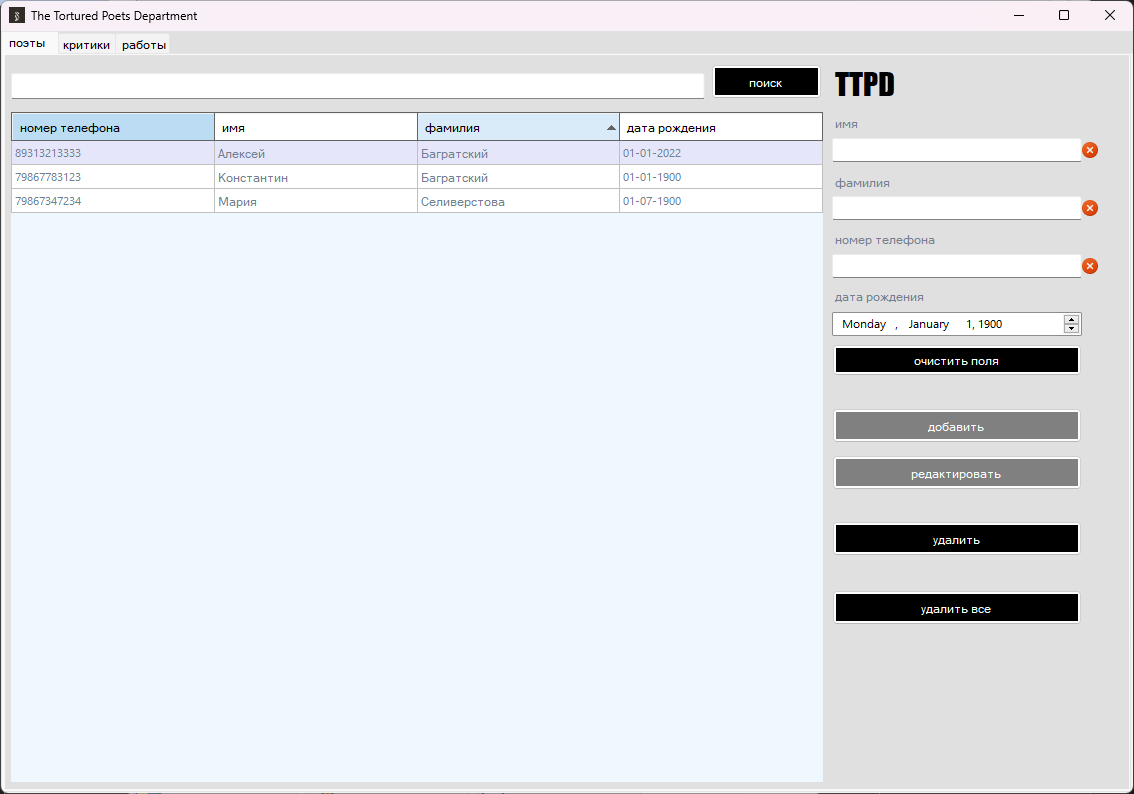


Рисунок А.9 – Сортировка по фамилии А до Я

Вместе с сортировкой или отдельно можно использовать поиск.

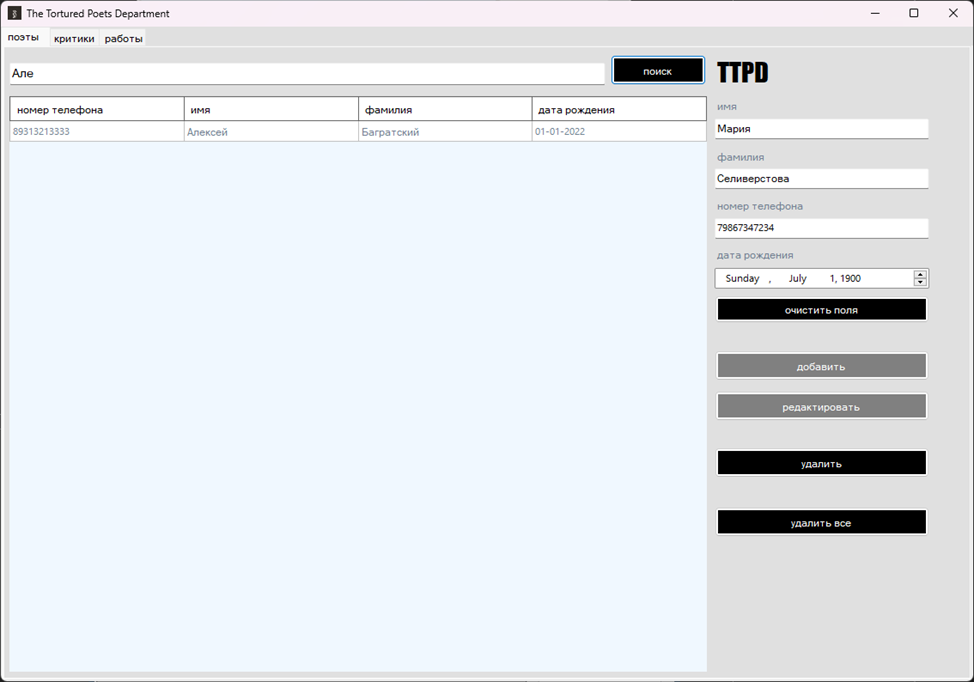


Рисунок А.10 – Отображение найденного поэта

# **Приложение Б - Исходные тексты программы**

**Файл Program.cs**

using System;

using System.Windows.Forms;

namespace CourseWork

{

internal static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new WelcomeForm());

}

}

}

**Файл DataBase.cs**

using System;

using System.Collections.Generic;

using System.Data.SQLite;

using System.Data;

using System.Linq;

namespace CourseWork.Source.DataBaseRelated

{

internal class DataBase

{

public SQLiteConnection Connection { set; get; }

public void ExecuteCommandNQ(SQLiteCommand command)

{

try

{

Connection.Open();

command.Connection = Connection;

command.ExecuteNonQuery();

Connection.Close();

}

catch (Exception ex)

{

if (Connection.State == ConnectionState.Open) Connection.Close();

throw ex.InnerException;

}

}

public object ExecuteCommandScalar(SQLiteCommand command)

{

try

{

Connection.Open();

command.Connection = Connection;

object scalar = command.ExecuteScalar();

Connection.Close();

return scalar;

}

catch (Exception ex)

{

if (Connection.State == ConnectionState.Open) Connection.Close();

throw ex.InnerException;

}

}

public DataTable GetDataTable(SQLiteCommand command)

{

try

{

Connection.Open();

command.Connection = Connection;

SQLiteDataAdapter dataAdapter = new SQLiteDataAdapter(command);

SQLiteCommandBuilder commandBuilder = new SQLiteCommandBuilder(dataAdapter);

DataTable dataTable = new DataTable();

dataAdapter.Fill(dataTable);

dataAdapter.Dispose();

Connection.Close();

return dataTable;

}

catch (Exception ex)

{

if (Connection.State == ConnectionState.Open) Connection.Close();

throw ex.InnerException;

}

}

public Dictionary<string, string> GetRowAsDictionary(SQLiteCommand command)

{

//method for getting a row from sql data base as a dictionary

try

{

Connection.Open();

command.Connection = Connection;

SQLiteDataReader reader = command.ExecuteReader();

if (reader.FieldCount == 0) {

throw new Exception("ошибка получения данных");

}

reader.Read();

Dictionary<string, string> dictionary = Enumerable.Range(0, reader.FieldCount)

.ToDictionary(reader.GetName, reader.GetString);

reader.Close();

reader.DisposeAsync();

Connection.Close();

return dictionary;

}

catch (Exception ex)

{

if (Connection.State == ConnectionState.Open) Connection.Close();

throw ex.InnerException;

}

}

public List<string> GetColumnValuesAsList(SQLiteCommand command)

{

//method for getting values from one column in sql data base

try

{

Connection.Open();

command.Connection = Connection;

SQLiteDataReader reader = command.ExecuteReader();

List<string> list = new List<string>();

while (reader.Read())

{

string item = reader.GetString(0);

list.Add(item);

}

reader.Close();

reader.DisposeAsync();

Connection.Close();

return list;

}

catch (Exception ex)

{

if (Connection.State == ConnectionState.Open) Connection.Close();

throw ex.InnerException;

}

}

}

}

**Файл DataBaseService.cs**

using CourseWork.Source.DataBaseRelated;

using System.Data.SQLite;

using System.IO;

using System.Text.RegularExpressions;

namespace CourseWork.Source.Services

{

internal partial class DataBaseService

{

private static DataBaseService \_instance;

private string \_dataBasesPath = string.Empty;

private DataBase \_dataBase;

private DataBaseService() { }

public static DataBaseService GetInstance()

{

if (\_instance == null)

{

\_instance = new DataBaseService();

\_instance.\_dataBasesPath = "DataBases";

\_instance.\_dataBase = new DataBase();

}

return \_instance;

}

public void SetNewDataBase(string dataBaseName)

{

string dataBasePath = Path.Combine(\_dataBasesPath, $"{dataBaseName}.db");

SQLiteConnection.CreateFile(dataBasePath);

SetConnectionToDataBase(dataBasePath);

CreateDataTables();

SetDataBaseToServices();

}

public void SetExistingDataBase(string dataBaseName)

{

string dataBasePath = Path.Combine(\_dataBasesPath, $"{dataBaseName}.db");

SetConnectionToDataBase(dataBasePath);

SetDataBaseToServices();

}

private void SetConnectionToDataBase(string dataBasePath)

{

string connectionString = $"Data Source={dataBasePath}; Version=3;";

\_dataBase.Connection = new SQLiteConnection(connectionString);

}

private void SetDataBaseToServices()

{

PoetService.DB = \_dataBase;

CriticService.DB = \_dataBase;

PoemService.DB = \_dataBase;

}

private void CreateDataTables()

{

SQLiteCommand command = new SQLiteCommand(

@"CREATE TABLE poets (

phone\_number VARCHAR(12) NOT NULL PRIMARY KEY,

first\_name VARCHAR(20) NOT NULL,

last\_name VARCHAR(20) NOT NULL,

date\_of\_birth DATE NOT NULL

);

CREATE TABLE critics (

phone\_number VARCHAR(12) NOT NULL PRIMARY KEY,

first\_name VARCHAR(20) NOT NULL,

last\_name VARCHAR(20) NOT NULL,

date\_of\_birth DATE NOT NULL

);

CREATE TABLE poems (

poet\_phone\_number VARCHAR(12) NOT NULL,

critic\_phone\_number VARCHAR(12) NOT NULL,

uploaded\_date DATE NOT NULL,

uploaded\_time TIME NOT NULL,

text\_data VARCHAR NOT NULL,

FOREIGN KEY (poet\_phone\_number) REFERENCES poets(phone\_number),

FOREIGN KEY (critic\_phone\_number) REFERENCES critics(phone\_number)

);");

\_dataBase.ExecuteCommandNQ(command);

}

public bool CheckDataBaseExists(string dataBaseName)

{

return File.Exists(Path.Combine(\_dataBasesPath, $"{dataBaseName}.db"));

}

[GeneratedRegex("^[A-Za-zА]{3,15}$")]

public static partial Regex RegexDataBaseName();

public static string GetRequirementsForDataBaseName()

{

return "Некорректное имя для базы данных. Имя должно состоять из латинских букв и быть длиной от 3 до 15 символов";

}

}

}

**Файл Person.cs**

using System;

namespace CourseWork.Source.Entities

{

internal class Person

{

//primary key in database

public string PhoneNumber { get; set; } = string.Empty;

public string FirstName { get; set; } = string.Empty;

public string LastName { get; set; } = string.Empty;

public DateTime DateOfBirth { get; set; } = DateTime.MinValue;

public Roles Role { get; set; }

//roles of persons, matches to DBtables

public enum Roles

{

Poet,

Critic,

}

}

}

**Файл Poem.cs**

using System;

namespace CourseWork.Source.Entities

{

internal class Poem

{

public string PoetPhoneNumber { get; set; } = string.Empty;

public string CriticPhoneNumber { get; set; } = string.Empty;

public DateTime Uploaded { get; set; } = DateTime.MinValue;

public string TextData { get; set; } = string.Empty;

}

}

**Файл WelcomeForm.cs**

using CourseWork.Source.Services;

using System;

using System.IO;

using System.Reflection;

using System.Windows.Forms;

namespace CourseWork

{

public partial class WelcomeForm : Form

{

DataBaseService dataBaseService = null;

public WelcomeForm()

{

InitializeComponent();

}

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

{

try

{

SetUpDB();

MainForm mainForm = new MainForm();

mainForm.Closed += (s, args) => this.Close();

mainForm.Show();

Hide();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void SetUpDB()

{

string nameDB = NameDB.Text;

dataBaseService = DataBaseService.GetInstance();

if (CreateEmptyDB.Checked)

{

if (!DataBaseService.RegexDataBaseName().IsMatch(nameDB))

{

throw new Exception(DataBaseService.GetRequirementsForDataBaseName());

}

dataBaseService.SetNewDataBase(nameDB);

}

if (UseExistingDB.Checked)

{

if (!dataBaseService.CheckDataBaseExists(nameDB))

{

throw new Exception("Базы данных с таким именем не существует.");

}

dataBaseService.SetExistingDataBase(nameDB);

}

}

}

}

**Файл MainForm.cs**

using CourseWork.Source.Entities;

using CourseWork.Source.Services;

using System.Windows.Forms;

using System;

using System.Text.RegularExpressions;

using System.Collections.Generic;

using System.Data;

namespace CourseWork

{

public partial class MainForm : Form

{

private CriticService \_criticService;

private PoetService \_poetService;

private PoemService \_poemService;

private DataGridViewRow \_choosenPoet;

private DataGridViewRow \_choosenCritic;

public MainForm()

{

InitializeComponent();

\_criticService = new CriticService();

\_poetService = new PoetService();

\_poemService = new PoemService();

LoadPoetsDataGrid();

LoadCriticDataGrid();

LoadPoemDataGrid();

FormatDataGrids();

}

#region DataGrid Methods

private void FormatDataGrids()

{

var SetAutoSizeToDataGrid = (DataGridView dataGrid) =>

{

foreach (DataGridViewColumn column in dataGrid.Columns)

{

column.AutoSizeMode = DataGridViewAutoSizeColumnMode.Fill;

}

};

SetAutoSizeToDataGrid(PoetsDataGrid);

SetAutoSizeToDataGrid(CriticsDataGrid);

SetAutoSizeToDataGrid(PoemDataGrid);

PoemDataGrid.Columns[PoemService.ColumnMapping["UploadedDate"]].DefaultCellStyle.Format = "dd-MM-yyyy";

PoemDataGrid.Columns[PoemService.ColumnMapping["UploadedTime"]].DefaultCellStyle.Format = "HH:mm";

PoetsDataGrid.Columns[PoetService.ColumnMapping["DateOfBirth"]].DefaultCellStyle.Format = "dd-MM-yyyy";

CriticsDataGrid.Columns[CriticService.ColumnMapping["DateOfBirth"]].DefaultCellStyle.Format = "dd-MM-yyyy";

PoetsDataGrid.Columns[PoetService.ColumnMapping["PhoneNumber"]].HeaderText = "номер телефона";

PoetsDataGrid.Columns[PoetService.ColumnMapping["FirstName"]].HeaderText = "имя";

PoetsDataGrid.Columns[PoetService.ColumnMapping["LastName"]].HeaderText = "фамилия";

PoetsDataGrid.Columns[PoetService.ColumnMapping["DateOfBirth"]].HeaderText = "дата рождения";

CriticsDataGrid.Columns[CriticService.ColumnMapping["PhoneNumber"]].HeaderText = "номер телефона";

CriticsDataGrid.Columns[CriticService.ColumnMapping["FirstName"]].HeaderText = "имя";

CriticsDataGrid.Columns[CriticService.ColumnMapping["LastName"]].HeaderText = "фамилия";

CriticsDataGrid.Columns[CriticService.ColumnMapping["DateOfBirth"]].HeaderText = "дата рождения";

PoemDataGrid.Columns[PoemService.ColumnMapping["Poet"]].HeaderText = "номер поэта";

PoemDataGrid.Columns[PoemService.ColumnMapping["Critic"]].HeaderText = "номер критика";

PoemDataGrid.Columns[PoemService.ColumnMapping["UploadedDate"]].HeaderText = "дата загрузки";

PoemDataGrid.Columns[PoemService.ColumnMapping["UploadedTime"]].HeaderText = "время загрузки";

PoemDataGrid.Columns[PoemService.ColumnMapping["TextData"]].HeaderText = "текст работы";

}

private DataGridViewRow GetSelectedRowFromDataGrid(DataGridView dataGrid)

{

if (dataGrid.SelectedRows.Count == 0 || dataGrid.SelectedRows is null) return null;

return dataGrid.SelectedRows[0];

}

private void ReplaceRowInDataGrid(DataGridView dataGrid, int rowToReplaceIndex, Dictionary<string, string> columnValuePairs)

{

var dataTable = (DataTable)dataGrid.DataSource;

var row = dataTable.NewRow();

foreach (KeyValuePair<string, string> cvp in columnValuePairs)

{

row[cvp.Key] = cvp.Value;

}

dataTable.Rows.RemoveAt(rowToReplaceIndex);

dataTable.Rows.InsertAt(row, rowToReplaceIndex);

dataGrid.Refresh();

}

private void AddRowToDataGrid(DataGridView dataGrid, Dictionary<string, string> columnValuePairs)

{

var dataTable = (DataTable)dataGrid.DataSource;

var row = dataTable.NewRow();

foreach (KeyValuePair<string, string> cvp in columnValuePairs)

{

row[cvp.Key] = cvp.Value;

}

dataTable.Rows.Add(row);

dataGrid.Refresh();

}

private void ShowRowsWithKeyword(DataGridView dataGridView, string keyword)

{

CurrencyManager currencyManager = (CurrencyManager)BindingContext[dataGridView.DataSource];

currencyManager.SuspendBinding();

foreach (DataGridViewRow row in dataGridView.Rows)

{

bool keywordFound = false;

foreach (DataGridViewCell cell in row.Cells)

{

if (cell.Value != null && cell.Value.ToString().Contains(keyword))

{

keywordFound = true;

break;

}

}

row.Visible = keywordFound;

}

dataGridView.Refresh();

currencyManager.ResumeBinding();

}

#endregion

#region Form Methods

private bool CheckInput(Control input, Func<Regex> RegexGen, Func<string> RequirementsGen)

{

if (!RegexGen().IsMatch(input.Text))

{

errorProvider.SetError(input, RequirementsGen());

return false;

}

else

{

errorProvider.SetError(input, "");

return true;

}

}

private void TabControl\_SelectedIndexChanged(object sender, EventArgs e)

{

if (TabControl.SelectedTab == PoetPage) { }

if (TabControl.SelectedTab == CriticPage) { }

if (TabControl.SelectedTab == PoemPage)

{

ChoosenPoetLabel.Text = \_choosenPoet != null ?

$"поэт: {\_choosenPoet.Cells[PoetService.ColumnMapping["FirstName"]].Value} {\_choosenPoet.Cells[PoetService.ColumnMapping["LastName"]].Value}" :

"поэт не выбран";

ChoosenCriticLabel.Text = \_choosenCritic != null ?

$"критик: {\_choosenCritic.Cells[CriticService.ColumnMapping["FirstName"]].Value} {\_choosenCritic.Cells[CriticService.ColumnMapping["LastName"]].Value}" :

"критик не выбран";

}

}

#endregion

#region Poet Page

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

{

if (\_choosenPoet == null)

{

MessageBox.Show("Выберите поэта для удаления.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

string phoneNumber = \_choosenPoet.Cells[PoetService.ColumnMapping["PhoneNumber"]].Value.ToString();

try

{

\_poetService.Delete(phoneNumber);

PoetsDataGrid.Rows.Remove(\_choosenPoet);

\_choosenPoet = null;

ClearPoetsInputs();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

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

{

bool isFirstNameValid = CheckInput(PoetFirstNameInput, PoetService.RegexFirstName, PoetService.GetRequirementsForFirstName);

bool isLastNameValid = CheckInput(PoetLastNameInput, PoetService.RegexLastName, PoetService.GetRequirementsForLastName);

bool isPhoneNumberValid = CheckInput(PoetPhoneNumberInput, PoetService.RegexPhoneNumber, PoetService.GetRequirementsForPhoneNumber);

if (!isFirstNameValid || !isLastNameValid || !isPhoneNumberValid) return;

Person poet = new Person()

{

Role = Person.Roles.Poet,

FirstName = PoetFirstNameInput.Text,

LastName = PoetLastNameInput.Text,

PhoneNumber = PoetPhoneNumberInput.Text,

DateOfBirth = PoetDateOfBirthInput.Value

};

try

{

\_poetService.Save(poet);

AddRowToDataGrid(PoetsDataGrid, \_poetService.LoadFields(poet.PhoneNumber));

ClearPoetsInputs();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

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

{

if (\_choosenPoet == null)

{

MessageBox.Show("Выберите поэта для редактирования.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

bool isFirstNameValid = CheckInput(PoetFirstNameInput, PoetService.RegexFirstName, PoetService.GetRequirementsForFirstName);

bool isLastNameValid = CheckInput(PoetLastNameInput, PoetService.RegexLastName, PoetService.GetRequirementsForLastName);

if (!isFirstNameValid || !isLastNameValid) return;

string phoneNumber = \_choosenPoet.Cells[PoetService.ColumnMapping["PhoneNumber"]].Value.ToString();

try

{

Person poet = \_poetService.GetByPhoneNumber(phoneNumber);

poet.FirstName = PoetFirstNameInput.Text;

poet.LastName = PoetLastNameInput.Text;

poet.DateOfBirth = PoetDateOfBirthInput.Value;

\_poetService.Update(poet);

ReplaceRowInDataGrid(PoetsDataGrid, \_choosenPoet.Index, \_poetService.LoadFields(poet.PhoneNumber));

ClearPoetsInputs();

\_choosenPoet = null;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void DataGridPoets\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{

ClearPoetsInputs();

\_choosenPoet = GetSelectedRowFromDataGrid(PoetsDataGrid);

PoetPhoneNumberInput.Text = \_choosenPoet.Cells[PoetService.ColumnMapping["PhoneNumber"]].Value.ToString();

PoetFirstNameInput.Text = \_choosenPoet.Cells[PoetService.ColumnMapping["FirstName"]].Value.ToString();

PoetLastNameInput.Text = \_choosenPoet.Cells[PoetService.ColumnMapping["LastName"]].Value.ToString();

PoetDateOfBirthInput.Value = DateTime.Parse(

\_choosenPoet.Cells[PoetService.ColumnMapping["DateOfBirth"]].Value.ToString()

);

PoetPhoneNumberInput.ReadOnly = true;

}

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

{

ShowRowsWithKeyword(PoetsDataGrid, PoetSearchInput.Text);

}

private void ClearPoetsInputs()

{

PoetFirstNameInput.Text = "";

PoetLastNameInput.Text = "";

PoetPhoneNumberInput.Text = "";

PoetDateOfBirthInput.Value = PoetDateOfBirthInput.MinDate;

PoetPhoneNumberInput.ReadOnly = false;

errorProvider.SetError(PoetFirstNameInput, "");

errorProvider.SetError(PoetLastNameInput, "");

errorProvider.SetError(PoetPhoneNumberInput, "");

}

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

{

ClearPoetsInputs();

}

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

{

try

{

\_poetService.DeleteAll();

LoadPoetsDataGrid();

ClearPoetsInputs();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void LoadPoetsDataGrid()

{

PoetsDataGrid.DataSource = \_poetService.GetDataTableOfAll();

}

#endregion

#region Critic Page

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

{

if (\_choosenCritic == null)

{

MessageBox.Show("Выберите критика для удаления.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

string phoneNumber = \_choosenCritic.Cells[CriticService.ColumnMapping["PhoneNumber"]].Value.ToString();

try

{

\_criticService.Delete(phoneNumber);

CriticsDataGrid.Rows.Remove(\_choosenCritic);

\_choosenCritic = null;

ClearCriticsInputs();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

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

{

bool isFirstNameValid = CheckInput(CriticFirstNameInput, CriticService.RegexFirstName, CriticService.GetRequirementsForFirstName);

bool isLastNameValid = CheckInput(CriticLastNameInput, CriticService.RegexLastName, CriticService.GetRequirementsForLastName);

bool isPhoneNumberValid = CheckInput(CriticPhoneNumberInput, CriticService.RegexPhoneNumber, CriticService.GetRequirementsForPhoneNumber);

if (!isFirstNameValid || !isLastNameValid || !isPhoneNumberValid) return;

Person critic = new Person()

{

Role = Person.Roles.Critic,

FirstName = CriticFirstNameInput.Text,

LastName = CriticLastNameInput.Text,

PhoneNumber = CriticPhoneNumberInput.Text,

DateOfBirth = CriticDateOfBirthInput.Value

};

try

{

\_criticService.Save(critic);

AddRowToDataGrid(CriticsDataGrid, \_criticService.LoadFields(critic.PhoneNumber));

ClearCriticsInputs();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

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

{

if (\_choosenCritic == null)

{

MessageBox.Show("Выберите критика для редактирования.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

bool isFirstNameValid = CheckInput(CriticFirstNameInput, CriticService.RegexFirstName, CriticService.GetRequirementsForFirstName);

bool isLastNameValid = CheckInput(CriticLastNameInput, CriticService.RegexLastName, CriticService.GetRequirementsForLastName);

if (!isFirstNameValid || !isLastNameValid) return;

string phoneNumber = \_choosenCritic.Cells[CriticService.ColumnMapping["PhoneNumber"]].Value.ToString();

try

{

Person critic = \_criticService.GetByPhoneNumber(phoneNumber);

critic.FirstName = CriticFirstNameInput.Text;

critic.LastName = CriticLastNameInput.Text;

critic.DateOfBirth = CriticDateOfBirthInput.Value;

\_criticService.Update(critic);

ReplaceRowInDataGrid(CriticsDataGrid, \_choosenCritic.Index, \_criticService.LoadFields(critic.PhoneNumber));

ClearCriticsInputs();

\_choosenCritic = null;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void CriticDataGrid\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{

ClearCriticsInputs();

\_choosenCritic = GetSelectedRowFromDataGrid(CriticsDataGrid);

CriticPhoneNumberInput.Text = \_choosenCritic.Cells[CriticService.ColumnMapping["PhoneNumber"]].Value.ToString();

CriticFirstNameInput.Text = \_choosenCritic.Cells[CriticService.ColumnMapping["FirstName"]].Value.ToString();

CriticLastNameInput.Text = \_choosenCritic.Cells[CriticService.ColumnMapping["LastName"]].Value.ToString();

CriticDateOfBirthInput.Value = DateTime.Parse(

\_choosenCritic.Cells[CriticService.ColumnMapping["DateOfBirth"]].Value.ToString()

);

CriticPhoneNumberInput.ReadOnly = true;

}

private void ClearCriticsInputs()

{

CriticFirstNameInput.Text = "";

CriticLastNameInput.Text = "";

CriticPhoneNumberInput.Text = "";

CriticDateOfBirthInput.Value = CriticDateOfBirthInput.MinDate;

CriticPhoneNumberInput.ReadOnly = false;

errorProvider.SetError(CriticFirstNameInput, "");

errorProvider.SetError(CriticLastNameInput, "");

errorProvider.SetError(CriticPhoneNumberInput, "");

}

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

{

ClearCriticsInputs();

}

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

{

ShowRowsWithKeyword(CriticsDataGrid, CriticSearchInput.Text);

}

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

{

try

{

\_criticService.DeleteAll();

LoadCriticDataGrid();

ClearCriticsInputs();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void LoadCriticDataGrid()

{

CriticsDataGrid.DataSource = \_criticService.GetDataTableOfAll();

}

#endregion

#region Poem Page

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

{

if (\_choosenPoet == null)

{

MessageBox.Show("Выберите поэта для добавления работы.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (\_choosenCritic == null)

{

MessageBox.Show("Выберите критика для добавления работы.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (PoemTextData.Text.Length == 0)

{

MessageBox.Show("Введите текст работы.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

var poem = new Poem()

{

PoetPhoneNumber = \_choosenPoet.Cells[PoetService.ColumnMapping["PhoneNumber"]].Value.ToString(),

CriticPhoneNumber = \_choosenCritic.Cells[CriticService.ColumnMapping["PhoneNumber"]].Value.ToString(),

Uploaded = DateTime.Now,

TextData = PoemTextData.Text,

};

try

{

\_poemService.Save(poem);

PoemTextData.Text = "";

LoadPoemDataGrid();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void LoadPoemDataGrid()

{

PoemDataGrid.DataSource = \_poemService.GetDataTableOfAll();

}

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

{

try

{

\_poemService.DeleteAll();

LoadPoemDataGrid();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

#endregion

private void PoemDataGrid\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{

var \_choosenPoem = GetSelectedRowFromDataGrid(PoemDataGrid);

PoemTextData.Text = \_choosenPoem.Cells[PoemService.ColumnMapping["TextData"]].Value.ToString();

}

}

}

**Файл CriticService.cs**

using CourseWork.Source.Entities;

using CourseWork.Source.DataBaseRelated;

using System;

using System.Collections.Generic;

using System.Data.SQLite;

using System.Data;

using System.Text.RegularExpressions;

namespace CourseWork.Source.Services

{

internal partial class CriticService

{

public static DataBase DB { get; set; }

private const string TableName = "critics";

public static readonly Dictionary<string, string> ColumnMapping = new Dictionary<string, string>()

{

{ "FirstName", "first\_name" },

{ "LastName", "last\_name" },

{ "PhoneNumber", "phone\_number" },

{ "DateOfBirth", "date\_of\_birth"}

};

#region CRUD Operations

public Person GetByPhoneNumber(string phoneNumber)

{

var fields = LoadFields(phoneNumber);

return new Person()

{

FirstName = fields[ColumnMapping["FirstName"]],

LastName = fields[ColumnMapping["LastName"]],

PhoneNumber = fields[ColumnMapping["PhoneNumber"]],

DateOfBirth = DateTime.Parse(fields[ColumnMapping["DateOfBirth"]]),

Role = Person.Roles.Critic

};

}

public DataTable GetDataTableOfAll()

{

var command = new SQLiteCommand($"SELECT \* FROM `{TableName}`");

return DB.GetDataTable(command);

}

public void Save(Person employee)

{

if (Exists(employee.PhoneNumber))

{

throw new Exception("Добавление невозможно: номер телефона уже используется.");

}

var command = new SQLiteCommand(

$"INSERT INTO `{TableName}`" +

$"({ColumnMapping["PhoneNumber"]}, {ColumnMapping["FirstName"]}, " +

$"{ColumnMapping["LastName"]}, {ColumnMapping["DateOfBirth"]})" +

$"VALUES (@phoneNumber, @firstName, @lastName, @dateOfBirth);");

command.Parameters.AddWithValue("@phoneNumber", employee.PhoneNumber);

command.Parameters.AddWithValue("@firstName", employee.FirstName);

command.Parameters.AddWithValue("@lastName", employee.LastName);

command.Parameters.AddWithValue("@dateOfBirth", GetFormatDateOfBirth(employee.DateOfBirth));

DB.ExecuteCommandNQ(command);

}

public void Delete(string phoneNumber)

{

var command = new SQLiteCommand($"DELETE FROM `{TableName}` " +

$"WHERE {ColumnMapping["PhoneNumber"]} = @phoneNumber;");

command.Parameters.AddWithValue("@phoneNumber", phoneNumber);

DB.ExecuteCommandNQ(command);

}

public void Update(Person employee)

{

var command = new SQLiteCommand($"UPDATE " +

$"`{TableName}` " +

$"SET {ColumnMapping["FirstName"]}=@firstName, " +

$"{ColumnMapping["LastName"]}=@lastName, {ColumnMapping["DateOfBirth"]}=@dateOfBirth " +

$"WHERE {ColumnMapping["PhoneNumber"]}=@phoneNumber;");

command.Parameters.AddWithValue("@phoneNumber", employee.PhoneNumber);

command.Parameters.AddWithValue("@firstName", employee.FirstName);

command.Parameters.AddWithValue("@lastName", employee.LastName);

command.Parameters.AddWithValue("@dateOfBirth", employee.DateOfBirth);

DB.ExecuteCommandNQ(command);

}

public void DeleteAll()

{

var command = new SQLiteCommand($"DELETE FROM `{TableName}`;");

DB.ExecuteCommandNQ(command);

}

#endregion

#region Helper Methods

public string GetFormatDateOfBirth(DateTime dateOfBirth)

{

return dateOfBirth.ToString("yyyy-MM-dd");

}

public Dictionary<string, string> LoadFields(string phoneNumber)

{

var command = new SQLiteCommand(

$"SELECT \* FROM `{TableName}` " +

$"WHERE {ColumnMapping["PhoneNumber"]}=@phoneNumber;");

command.Parameters.AddWithValue("@phoneNumber", phoneNumber);

return DB.GetRowAsDictionary(command);

}

private bool Exists(string phoneNumber)

{

var command = new SQLiteCommand(

$"SELECT COUNT(\*) FROM `{TableName}` " +

$"WHERE {ColumnMapping["PhoneNumber"]}=@phoneNumber;");

command.Parameters.AddWithValue("@phoneNumber", phoneNumber);

return Convert.ToInt32(DB.ExecuteCommandScalar(command)) != 0;

}

#endregion

#region Validation Methods

[GeneratedRegex("^[A-Za-zА-ЯЁа-яё]{3,15}$")]

public static partial Regex RegexFirstName();

public static string GetRequirementsForFirstName()

{

return "имя должно состояьть из латинских или кириллических букв и быть длиной от 3 до 15 символов";

}

[GeneratedRegex("^[A-Za-zА-ЯЁа-яё]{3,15}$")]

public static partial Regex RegexLastName();

public static string GetRequirementsForLastName()

{

return "фамилия должна состояьть из латинских или кириллических букв и быть длиной от 3 до 15 символов";

}

[GeneratedRegex("^\\+?\\d{1,3}\\s?\\(?\\d{3}\\)?[-.\\s]?\\d{3}[-.\\s]?\\d{4}$")]

public static partial Regex RegexPhoneNumber();

public static string GetRequirementsForPhoneNumber()

{

return "номер телефона должен быть корректным";

}

#endregion

}

}

**Файл PoemService.cs**

using CourseWork.Source.DataBaseRelated;

using CourseWork.Source.Entities;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SQLite;

using System.Linq;

namespace CourseWork.Source.Services

{

internal class PoemService

{

public static DataBase DB { get; set; }

private const string TableName = "poems";

public static readonly Dictionary<string, string> ColumnMapping = new Dictionary<string, string>()

{

{ "Poet", "poet\_phone\_number" },

{ "Critic", "critic\_phone\_number" },

{ "UploadedDate", "uploaded\_date" },

{ "UploadedTime", "uploaded\_time"},

{ "TextData", "text\_data"}

};

#region CRUD operations

public void Save(Poem poem)

{

if (Exists(poem))

{

throw new Exception("Добавление невозможно: работа у данного автора уже заявлена.");

}

var command = new SQLiteCommand($"INSERT INTO `{TableName}` (" +

$"{ColumnMapping["Poet"]}, {ColumnMapping["Critic"]}, {ColumnMapping["UploadedDate"]}, {ColumnMapping["UploadedTime"]}, {ColumnMapping["TextData"]}) " +

$"VALUES(@poet, @critic, @date, @time, @textData);");

command.Parameters.AddWithValue("@poet", poem.PoetPhoneNumber);

command.Parameters.AddWithValue("@critic", poem.CriticPhoneNumber);

command.Parameters.AddWithValue("@date", GetFormatDate(poem.Uploaded));

command.Parameters.AddWithValue("@time", GetFormatTime(poem.Uploaded));

command.Parameters.AddWithValue("@textData", poem.TextData);

DB.ExecuteCommandNQ(command);

}

public void DeleteAll()

{

var command = new SQLiteCommand($"DELETE FROM `{TableName}`;");

DB.ExecuteCommandNQ(command);

}

public DataTable GetDataTableOfAll()

{

var command = new SQLiteCommand($"SELECT \* FROM `{TableName}`");

return DB.GetDataTable(command);

}

#endregion

#region Helper Methods

public bool Exists(Poem poem)

{

var command = new SQLiteCommand($"SELECT COUNT(\*) FROM `{TableName}` " +

$"WHERE {ColumnMapping["Poet"]}=@poet;");

command.Parameters.AddWithValue("@poet", poem.PoetPhoneNumber);

return Convert.ToInt32(DB.ExecuteCommandScalar(command)) != 0;

}

public string GetFormatDate(DateTime date)

{

return date.ToString("yyyy-MM-dd");

}

public string GetFormatTime(DateTime time)

{

return time.ToString("HH:mm");

}

#endregion

}

}

**Файл PoetService.cs**

using CourseWork.Source.DataBaseRelated;

using CourseWork.Source.Entities;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SQLite;

using System.Text.RegularExpressions;

namespace CourseWork.Source.Services

{

internal partial class PoetService

{

public static DataBase DB { get; set; }

private const string TableName = "poets";

public static readonly Dictionary<string, string> ColumnMapping = new Dictionary<string, string>()

{

{ "FirstName", "first\_name" },

{ "LastName", "last\_name" },

{ "PhoneNumber", "phone\_number" },

{ "DateOfBirth", "date\_of\_birth"}

};

#region CRUD Operations

public Person GetByPhoneNumber(string phoneNumber)

{

var fields = LoadFields(phoneNumber);

return new Person()

{

FirstName = fields[ColumnMapping["FirstName"]],

LastName = fields[ColumnMapping["LastName"]],

PhoneNumber = fields[ColumnMapping["PhoneNumber"]],

DateOfBirth = DateTime.Parse(fields[ColumnMapping["DateOfBirth"]]),

Role = Person.Roles.Poet

};

}

public DataTable GetDataTableOfAll()

{

var command = new SQLiteCommand($"SELECT \* FROM `{TableName}`");

return DB.GetDataTable(command);

}

public void Save(Person poet)

{

if (Exists(poet.PhoneNumber))

{

throw new Exception("Добавление невозможно: номер телефона уже используется.");

}

var command = new SQLiteCommand(

$"INSERT INTO `{TableName}`" +

$"({ColumnMapping["PhoneNumber"]}, {ColumnMapping["FirstName"]}, " +

$"{ColumnMapping["LastName"]}, {ColumnMapping["DateOfBirth"]})" +

$"VALUES (@phoneNumber, @firstName, @lastName, @dateOfBirth);");

command.Parameters.AddWithValue("@phoneNumber", poet.PhoneNumber);

command.Parameters.AddWithValue("@firstName", poet.FirstName);

command.Parameters.AddWithValue("@lastName", poet.LastName);

command.Parameters.AddWithValue("@dateOfBirth", GetFormatDateOfBirth(poet.DateOfBirth));

DB.ExecuteCommandNQ(command);

}

public void Delete(string phoneNumber)

{

var command = new SQLiteCommand($"DELETE FROM `{TableName}` " +

$"WHERE {ColumnMapping["PhoneNumber"]} = @phoneNumber;");

command.Parameters.AddWithValue("@phoneNumber", phoneNumber);

DB.ExecuteCommandNQ(command);

}

public void Update(Person poet)

{

var command = new SQLiteCommand($"UPDATE " +

$"`{TableName}` " +

$"SET {ColumnMapping["FirstName"]}=@firstName, " +

$"{ColumnMapping["LastName"]}=@lastName, {ColumnMapping["DateOfBirth"]}=@dateOfBirth " +

$"WHERE {ColumnMapping["PhoneNumber"]}=@phoneNumber;");

command.Parameters.AddWithValue("@phoneNumber", poet.PhoneNumber);

command.Parameters.AddWithValue("@firstName", poet.FirstName);

command.Parameters.AddWithValue("@lastName", poet.LastName);

command.Parameters.AddWithValue("@dateOfBirth", GetFormatDateOfBirth(poet.DateOfBirth));

DB.ExecuteCommandNQ(command);

}

public void DeleteAll()

{

var command = new SQLiteCommand($"DELETE FROM `{TableName}`;");

DB.ExecuteCommandNQ(command);

}

#endregion

#region Helper Methods

public string GetFormatDateOfBirth(DateTime dateOfBirth)

{

return dateOfBirth.ToString("yyyy-MM-dd");

}

public Dictionary<string, string> LoadFields(string phoneNumber)

{

var command = new SQLiteCommand(

$"SELECT \* FROM `{TableName}` " +

$"WHERE {ColumnMapping["PhoneNumber"]}=@phoneNumber;");

command.Parameters.AddWithValue("@phoneNumber", phoneNumber);

return DB.GetRowAsDictionary(command);

}

private bool Exists(string phoneNumber)

{

var command = new SQLiteCommand(

$"SELECT COUNT(\*) FROM `{TableName}` " +

$"WHERE {ColumnMapping["PhoneNumber"]}=@phoneNumber;");

command.Parameters.AddWithValue("@phoneNumber", phoneNumber);

return Convert.ToInt32(DB.ExecuteCommandScalar(command)) != 0;

}

#endregion

#region Validation Methods

[GeneratedRegex("^[A-Za-zА-ЯЁа-яё]{3,15}$")]

public static partial Regex RegexFirstName();

public static string GetRequirementsForFirstName()

{

return "имя должно состоять из латинских или кириллических букв и быть длиной от 3 до 15 символов";

}

[GeneratedRegex("^[A-Za-zА-ЯЁа-яё]{3,15}$")]

public static partial Regex RegexLastName();

public static string GetRequirementsForLastName()

{

return "фамилия должна состоять из латинских или кириллических букв и быть длиной от 3 до 15 символов";

}

[GeneratedRegex("^\\+?\\d{1,3}\\s?\\(?\\d{3}\\)?[-.\\s]?\\d{3}[-.\\s]?\\d{4}$")]

public static partial Regex RegexPhoneNumber();

public static string GetRequirementsForPhoneNumber()

{

return "номер телефона должен быть корректным";

}

#endregion

}

}