Разработка клиент-серверного приложения «Система тестирования» **ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ЯРОСЛАВСКОЙ ОБЛАСТИ**

государственное профессиональное образовательное учреждение   
Ярославской области

Рыбинский полиграфический колледж

**КУРСОВОЙ ПРОЕКТ**

|  |  |
| --- | --- |
| Разработка клиент-серверного приложения «Система тестирования» | |
|  | |
| по дисциплине | Технология разработки и защиты баз данных |
|  | |

Пояснительная записка

|  |
| --- |
| КП.0902.11.000000.00 ПЗ |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент группы | 4-ИС-2 |  |  |  | А. Р. Нужин |
|  | *(Код учебной группы)* |  | *(Подпись, дата)* |  | *(И.О.Фамилия)* |
| Руководитель | преподаватель |  |  |  | Е. А. Лобанова |
|  | *(Должность, звание)* |  | *(Подпись, дата)* |  | *(И.О.Фамилия)* |
| Нормоконтроль | преподаватель |  |  |  | Е. А. Лобанова |
|  | *(Должность, звание)* |  | *(Подпись, дата)* |  | *(И.О.Фамилия)* |

г. Рыбинск

2022

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc121487138)

[1 Исследовательский раздел 3](#_Toc121487139)

[2 Конструкторский раздел 3](#_Toc121487140)

[2.1 Проектирование информационной модели данных 3](#_Toc121487141)

[2.2 Проектирование серверной части приложения 3](#_Toc121487142)

[2.2.1 Разработка схемы базы данных 3](#_Toc121487143)

[2.2.2 Разработка сущностей базы данных 3](#_Toc121487144)

[2.3 Проектирование клиентской части приложения 3](#_Toc121487145)

[2.3.1 Разработка модулей схемы 3](#_Toc121487146)

[2.3.2 Разработка пользовательского интерфейса 3](#_Toc121487147)

[2.3.3 Организация доступа к объектам базы данных 3](#_Toc121487148)

[2.3.4 Разработка блох-схем алгоритмов процедур и функций 3](#_Toc121487149)

[2.4 Обеспечение коллективного доступа. Защита информации 3](#_Toc121487150)

[3 Технологическая часть 3](#_Toc121487151)

[3.1 Тестирование и отладка приложения 3](#_Toc121487152)

[3.2 Инструкция администратора базы данных 3](#_Toc121487153)

[3.3 Инструкция по эксплуатации приложения 3](#_Toc121487154)

[4 Раздел охраны труда 3](#_Toc121487155)

[Заключение 3](#_Toc121487156)

[Список используемых источников 3](#_Toc121487157)

[Приложение А 3](#_Toc121487158)

Введение

В рамках данного курсового проекта планируется разработка приложения, где будет реализовано прохождение тестов в отдельном приложении. Пользователь сможет создать, изменить, удалить, найти и проходить тесты. Данное приложение должно позволить реализовать удобное взаимодествие пользователя с тестами, удобнее всего реализовать это путем использования клиент-серверной архитектуры. Все данные будут храниться в базе данных на сервере, а клиент будет взаимодействовать с клиентской частью приложения.

Приложение позволит достичь автоматицазии процессов в прохождения тестов.

1 Исследовательский раздел

Процесс разработки программного обеспечения – набор правил, согласно которым построена разработка программного обеспечения. Приложение можно назвать клиент-серверным, если оно включает в себя клиент-серверную архитектуру. Разработку клиент-серверного приложения необходимо начинать с выбора архитектуры клиент-сервера.

Для разработки клиент/серверных систем имеется два подхода. Первый подход построение систем на основе двухзвенной архитектуры. Состоит из клиентской и серверной части. Как правило, серверная часть представляет собой сервер БД, на котором расположены общие данные. А клиентская часть представляет приложение, которое связывается с сервером БД, осуществляет к нему запросы и получает ответы. Такие системы используются в локальных сетях, т.к. нет затруднений с установкой клиентской части. Также системы с такой архитектурой более безопасны, т.к. могут использовать собственные протоколы передачи данных, не известные злоумышленникам. Поэтому многие крупные компании, которые располагаются не в едином месте и для соединения подразделений используют глобальную сеть Интернет, выбирают именно такую архитектуру построения клиент/серверных систем.

При разработке информационных систем, рассчитанных на широкую аудиторию, возникают проблемы с использованием двухзвенной архитектуры. Во-первых, пользователю необходимо иметь в наличии клиентскую часть, а, во-вторых, у неопытного пользователя, могут возникнуть проблемы с конфигурированием такой системы. Поэтому в последнее время, более часто разрабатывают приложения на базе трехзвенной архитектуры.

Второй подход построение систем на основе трехзвенной архитектуры. Серверная часть в этой архитектуре представляет собой сервер приложений и сервер БД. А в качестве клиента выступает web-браузер. Такая система очень проста для пользователя. Ему необходимо знать только адрес сервера приложения и наличие web-браузера на рабочем компьютере. Все данные представляются в виде html-разметки, с использование графики (jpeg, gif, flash) и JavaScript. Передача запросов от клиента к серверу приложений происходит по средствам CGI-интерфейса. Сервер приложений общается с сервером БД, используя другой интерфейс, зависящий от того, на основе каких средств строится конкретная информационная система. Недостатками такой архитектуры является использование общеизвестных протоколов и интерфейсов передачи данных. Злоумышленник может осуществить взлом системы, если она будет недостаточно хорошо проверять поступившие запросы от клиента.

При разработке клиент/серверных приложений необходимо учитывать:

- на каких пользователей будет рассчитана данная информационная система;

- какие требования предъявляются к безопасности.

Если информационная система должна быть общедоступной и рассчитана на широкую аудиторию, то необходимо использовать трехзвенную архитектуру.

Если информационная система используется внутри предприятия, доступ имеют к ней ограниченные пользователи и требуется создать максимально безопасную и защищенную систему, то следует отдать предпочтение двухзвенной архитектуре. Взято из материалов [1].

В рамках курсового проекта был выбран первый способ для разработки клиент/серверной системы на основе двухзвенной архитектуры.

Для реализации двухзвенной архитектуры была выбрана платформа WPF (Windows Presentetion Foundation).

При выборе WPF можно выделить такие преимущества как аппаратное ускорение через DirectX, что сильно влияет на производительность. Также можно отметить веб-подобную модель компоновки. Вместо того чтобы фиксировать элементы управления на месте с определенными координатами, WPF поддерживает гибкий поток, размещающий элементы управления на основе их содержимого. В результате получается пользовательский интерфейс, который может быть адаптирован для отображения высоко динамичного содержимого или к разным языкам.

Преимуществами WPF являются:

* **веб-подобная модель компоновки;**
* **богатая модель рисования;**
* **развитая текстовая модель;**
* **анимация;**
* **поддержка аудио и видео;**
* **стили и шаблоны;**
* **команды;**
* **декларативный пользовательский интерфейс;**
* **приложения на основе страниц.**

При выборе сред разработки были рассмотрены Visual Studio и Project Rider. Visual Studio – это удобная интегрированная среда разработки (IDE) от Microsoft, позволяющая быстро и эффективно создавать, и разрабатывать проект, выбрав для этого все необходимое. Среда использует платформы разработки программного обеспечения Microsoft: Windows API, Windows Forms, Windows Presentation Foundation, Windows Store и Microsoft Silverlight. Так же она принимает плагины, которые расширяют функциональные возможности практически на каждом уровне, включая добавление поддержки систем управления исходным кодом (таких как Subversion) и обеспечивает стандартный для Windows вид окон приложения. Единственным минусом можно считать сложность освоения данной среды разработки из-за её большого количества различных функций, спрятанных в подразделах меню. Взято из материалов [2].

Project Rider – это среда от JetBrains для работы с платформой .NET. Она обладает поддержкой полного цикла. Фирменная черта продуктов JetBrains, воплощенная и в Project Rider. С Project Rider появиться возможность организовать весь цикл создания программного обеспечения: от идеи до поддержки. Функциональность Project Rider позволяет подключить MSBuild и XBuild, работать с CLI-проектами и организовать отладку приложений .NET and Mono. Множество опций для быстрого создания кода улучшает производительность. Кроссплатформенность Project Rider работает с Windows, Linux и MacOS. Из минусов можно выделить её молодость. Часть функциональности еще в разработке, не все стартовые ошибки исправлены. Так же можно отметить её стоимость. Самая дешевая версия Project Rider обойдется в 139 долларов за первый год использования. Но есть триал-версия и специальные предложения для студентов и непрофильных организаций. Взято из материалов [3].

Из этих двух сред разработки был выбран Visual Studio, так как он обладает всем необходимым функционалом для реализации проекта, а также она является бесплатной и дольше находится на рынке труда.

В интернете существует аналог разрабатываемого приложения, у которого есть ряд ключевых отличий. Взято из материалов [4]. У аналога нет возможности регистрации пользователей, из-за чего может быть невозможным реализовать партнерский программы, и запоминание определенных данных пользователей для предотвращения повторного ими ввода данных. Можно отметить хороший функционал у аналога, например, в отличии от разрабатываемого приложения у аналога есть возможность создавтаь более «продвинутые» тесты, которые имеют различные виды создаваемых вопросов.

Приложение должно предоставить возможность пользователю выбрать интересующий его тест, сохранить его или же как-то изменить, создав отдельную, от основоного теста, копию или же и вовсе удалить его для себя. Пользователь может создать свой собственный тест, выложив его при этом в «общий доступ», предоставляя возможность пройти его и другим пользователям. Данные будут автоматически браться из базы данных. В приложении будет присутствовать отдельная админ панель, которая будет доступна для пользователей с определенным уровнем доступа. В админ панели, админ сможет редактировать базу данных.

Основными процессами будут выступать процессы создания списков тестов и самих тестов в целом. Эти списки будут необходимы для генерации интерфейса, и хранения определенного формата данных для дальнейшей работы с ними.

2 Конструкторский раздел

2.1 Проектирование информационной модели данных

Черная сфера представляет собой систему, внутреннее устройство которой не важно. В эту систему подаются входные данные, а на выходе из системы поступают выходные данные. Черная сфера представлена на рисунке 2.1.



Рисунок 2.1 – Модель «Черная сфера»

Представим наше приложение в виде черной сферы. В приложении будут присутствовать такие выходные данные, как пользователь. На выходе из приложения будет поступать тест. Пользователем будет выступать человек, совершающий какие-либо действия с тестами. Тестом будет результат, полученный после манипуляций пользователя. Черная сфера с перечисленными входными и выходными параметрами представлена на рисунке 2.2.



Рисунок 2.2 – Черная сфера с параметрами

В приложении существует 2 основных процесса отвечающие за создание и прохождение тестов, каждый тест хранит в себе ряд вопросов и ответов на них. Большая часть данных для заполнения тестов поступает из базы данных и формитируется под интерфейс.

2.2 Проектирование серверной части приложения

2.2.1 Разработка схемы базы данных

Для выявления всех возможных сущностей будущей базы и получения концептуальной модели данных будет проведено несколько серий нормализации.

На первом этапе нормализации можно представить модель как связь между пользователем и тестом. Первый этап нормализации представлен на рисунке 2.3.

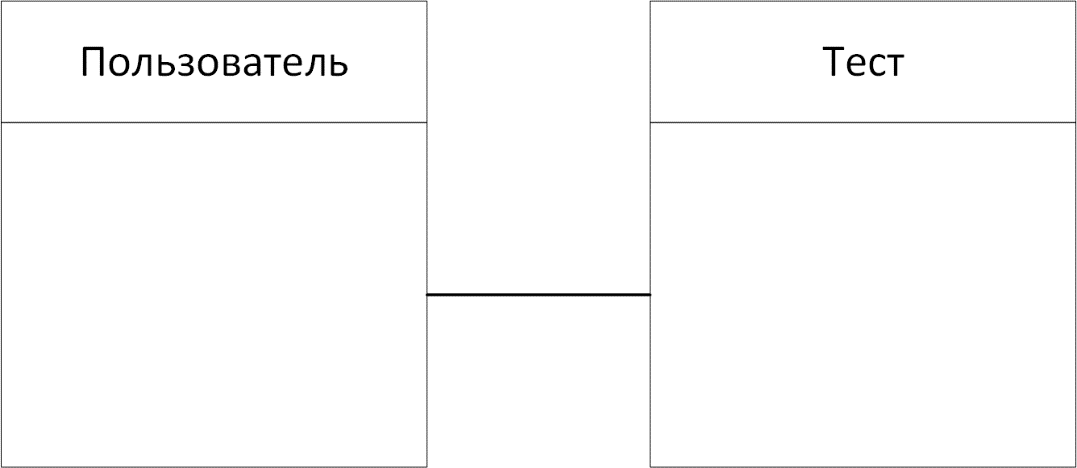


Рисунок 2.3 – Первый этап нормализации

Во втором этапе нормализации разобьём сущность теста на 3 сущности: Имя теста, Вопросы теста и Ответы теста. Сущность теста будет содержать в себе информацию об имени теста, вопросах теста и ответах теста. Второй этап нормализации на рисунке 2.4.

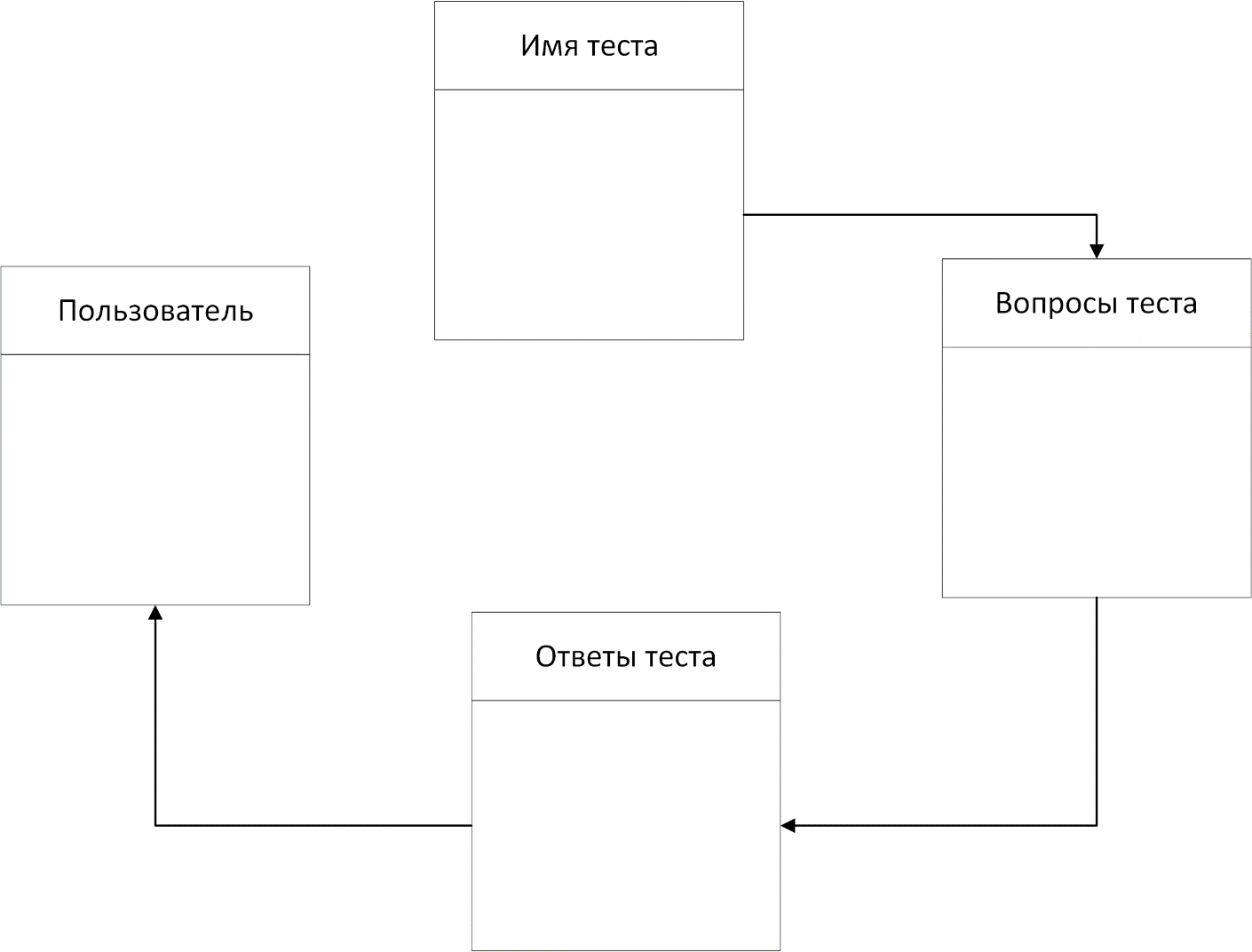


Рисунок 2.4 – Второй этап нормализации

В третьем заключительном этапе нормализации получим полную концептуальную схему добавив учётную запись пользователя, его статистику и результаты за конкретный тест. Третий этап нормализации представлена на рисунке 2.5.

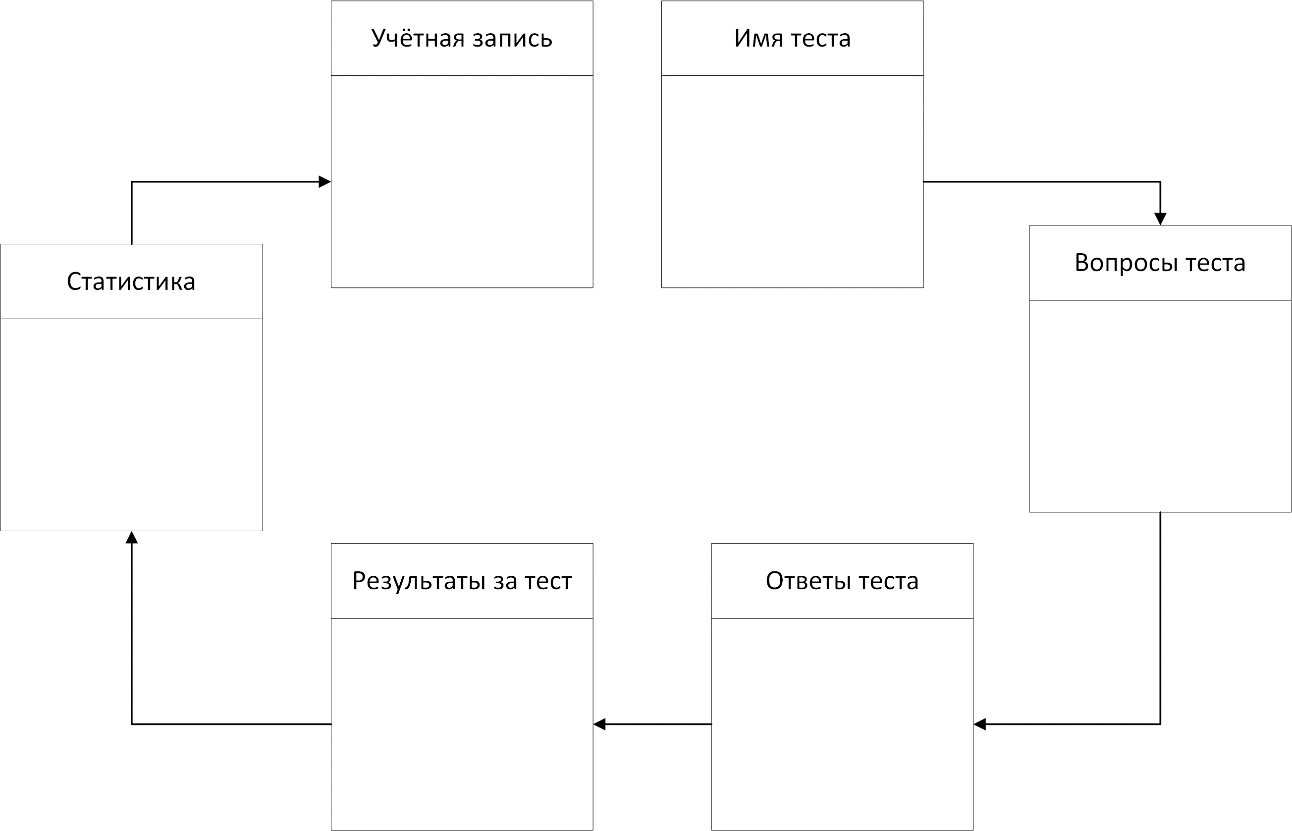


Рисунок 2.5 – Третий этап нормализации

Получим логическую модель данных с содержанием всех сущностей, связей и атрибутов данных. Логическая модель данных представлена на рисунке 2.6.

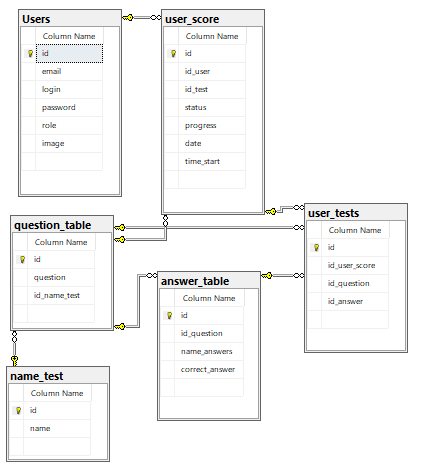


Рисунок 2.6 – Логическая модель данных

Получим физическую модель данных, включающая ассоциативные таблицы, которые иллюстрируют отношения между сущностями, а также первичные и внешние ключи для связи данных. Физическая модель данных представлена на рисунке 2.7.

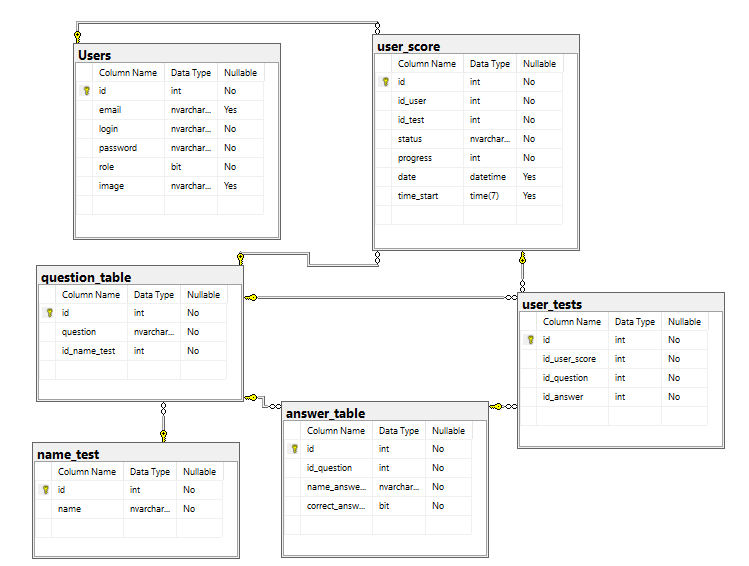


Рисунок 2.7 – Физическая модель данных

2.2.2 Разработка сущностей базы данных

В результатах раздела «Разработка схемы базы данных» получена схема базы данных, из которой следует необходимость присутствия определенных сущностей необходимых для полноценной работы приложения. Для удобства все сущности сведены в табличном виде. Сущности схемы базы данных представлены в таблице 2.1.

Таблица 2.1 – Сущности схемы базы данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя сущности | Назначение сущности | Типы данных | Перечисление наименований сущностей, которые подчиняются текущей сущности | Перечисление наименований сущностей, которым подчиняется текущая сущность |
| Users | Содержит данные пользователя приложения | int, nvarchar(100), bit | user\_score | - |
| user\_score | Содержит информацию об прохождении теста пользователем | int, nvarchar(100), datetime, time(7) | user\_tests | question\_table, Users |
| user\_tests | Содержит статистику пользователя | int | - | user\_score, question\_table, answer\_table |
| question\_table | Содержит информацию об вопросах теста | int, nvarchar(100) | user\_score, user\_tests, answer\_table | name\_test |
| name\_tests | Содержит информацию об названиях тестов | int, nvarchar(100) | question\_table | - |
| answer\_table | Содержит информацию об ответах теста | int, nvarchar(100), bit | user\_tests | question\_table |

2.3 Проектирование клиентской части приложения

2.3.1 Разработка модулей схемы

WPF предоставляет комплексный набор функций разработки приложений, которые включают в себя язык XAML, элементы управления, привязку к данным, макет, двумерную и трехмерную графику, анимацию, стили, шаблоны, документы, мультимедиа, текст и типографические функции. WPF является частью .NET, поэтому вы можете создавать приложения, включающие другие элементы .NET API. Взято из материалов [5].

Представим клиентскую часть приложения в виде модульной схемы показывающая связь между окнами, классами и страницами при организации клиентской части приложения. Модульная схема клиентской части приложения представлена на рисунке 2.8.



Рисунок 2.8 – Модульная схема клиентской части приложения

В составе модульной схемы присутствует окно авторизации и регистрации пользователя, главное окно, для просмотра и выбора тестов, страница профиля, для взаимодействия с личным кабинетом.

2.3.2 Разработка пользовательского интерфейса

Пользовательский интерфейс – это совокупность информационной модели проблемной области, средств и способов взаимодействия пользователя с информационной моделью, а также компонентов, обеспечивающих формирование информационной модели в процессе работы программной системы.

**Графический пользовательский интерфейс – самый** популярный тип пользовательского интерфейса. Он представляет собой окно с различными элементами управления. Пользователи взаимодействуют с элементами с помощью клавиатуры, компьютерной мыши и голосовых команд.

**XAML** представляет собой язык разметки, используемый для создания экземпляров объектов .NET. Хотя язык XAML — это технология, которая может быть применима ко многим различным предметным областям, его главное назначение — конструирование пользовательских интерфейсов WPF. Другими словами, документы XAML определяют расположение панелей, кнопок и прочих элементов управления, составляющих окна в приложении WPF.

На этом этапе рассмотрим каждый блок модульной схемы поподробнее. При запуске приложения открывается окно авторизации. Данное окно содержит в себе название приложения, два текстовых поля для ввода логина и пароля, а также кнопка для входа в программу, кнопка регистрации пользователя и кнопка выхода из программы. Внешний вид страницы авторизации представлен на рисунке 2.8.

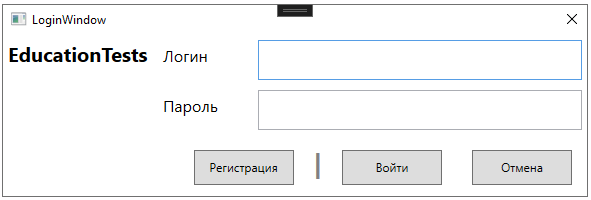


Рисунок 2.8 – Внешний вид окна авторизации

Окно регистрации содержит название приложения. Имеются пять текстовых поля для ввода логина, пароля и выбора фото профиля, а также две кнопки для создания аккаунта и кнопка отмены. Внешний вид страницы регистрации представлен на рисунке 2.9.

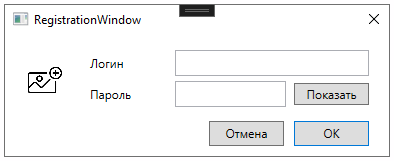


Рисунок 2.9 – Внешний вид окна регистрации

При авторизации под клиентской учетной записью в главном окне присутствует пять кнопок управления и область для отображения тестов. При нажатии на кнопки добавить, изменить и удалить над тестом выполняется одно из описанных действий. При нажатии кнопки «Пройти тест», с заранее выбранным тестом из списка, происходит переход на страницу прохождения тестов. На странице присутствуют название теста, данный вопрос и варианты ответа к нему, а также две кнопки «Принять» и «Дальше». При нажатии клавишей мыши по кнопке «Принять» происходит принятие варианта ответа на заданный вопрос и деактивация данной кнопки. Кнопка «Дальше» переносит к следующему вопросу. Кнопка «Вопросы» на главной странице вызовет окно для добавления в тест новых вопросов с вариантами ответа. При нажатии кнопки профиля откротея профиль авторизовнаного пользователя с его статистикой по тестам. Внешний вид главной страницы представлен на рисунке 2.10.

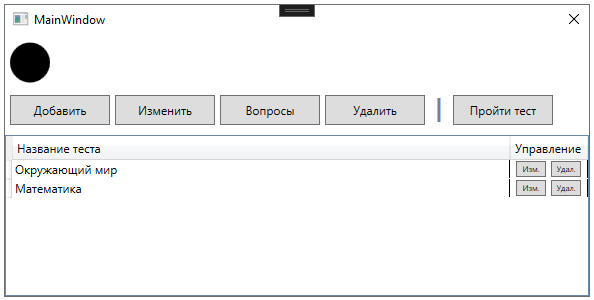


Рисунок 2.10 – Внешний вид главной страницы

При нажатии на кнопку профиля откроется страница личного кабинета. В личном кабинете отображается вся статистика по тестам. Результаты тестов можно удалить для повторного прохождения. Внешний вид страницы регистрации представлен на рисунке 2.11.

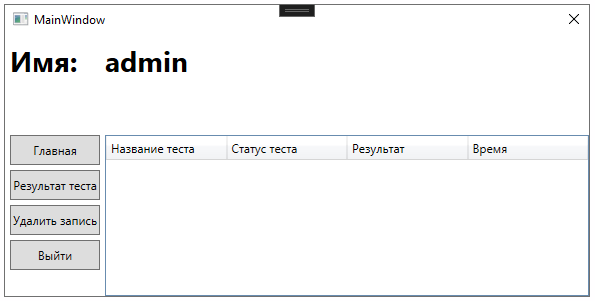


Рисунок 2.11 – Внешний вид страницы личного кабинета

2.3.3 Организация доступа к объектам базы данных

В WPF привязка (binding) является мощным инструментом программирования, без которого не обходится ни одно серьезное приложение.

Привязка подразумевает взаимодействие двух объектов: источника и приемника. Объект-приемник создает привязку к определенному свойству объекта-источника. В случае модификации объекта-источника, объект-приемник также будет модифицирован. Взято из материалов [6].

Возьмем для примера страницу с статистикой по тестам в профиле, где участвует DataGrid. DataGrid страницы с фильмами представлен на рисунке 2.17.

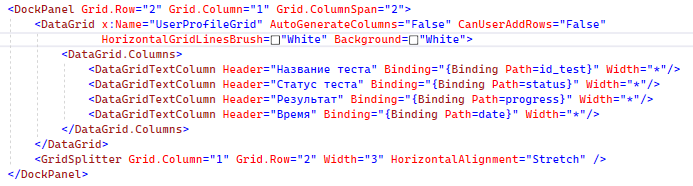


Рисунок 2.17 – DataGrid страницы со статистикой

У Binding в свойство Path мы записываем свойство объекта источника. Привязка объекта с данными к DataGrid представлена на рисунке 2.18.



Рисунок 2.18 – Привязка объекта с данными к DataGrid

Беря данные из базы данных, мы привязываем их с DataGrid. Сам DataGrid верстается под конкретное содержимое.

2.3.4 Разработка блох-схем алгоритмов процедур и функций

Основными функциональными задачами приложения являются процессы создания и прохождение тестов.

Метод CreateQuestion отвечает за сбор вопросов и ответов для теста. У метода нет ни входных, ни выходных данных, так как данные получаются в методе и в нём же передаются в элемент отображения. Блок-схема метода CreateQuestion представлена на рисунке 2.19.

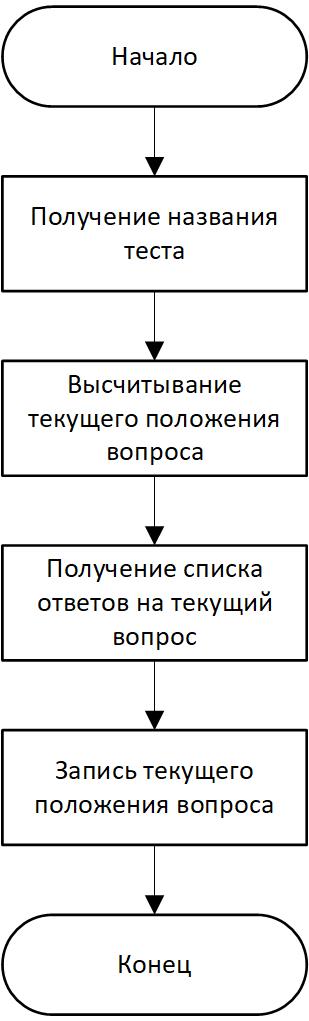


Рисунок 2.19 – Блок-схема метода CreateQuestion

Метод SelectAnswerButton\_Click отвечает за проверку выбранного пользователем ответа. У метода нет ни входных, ни выходных данных, так как данные получаются в методе и в нём же передаются в элемент отображения. Блок-схема метода SelectAnswerButton\_Click представлена на рисунке 2.20.

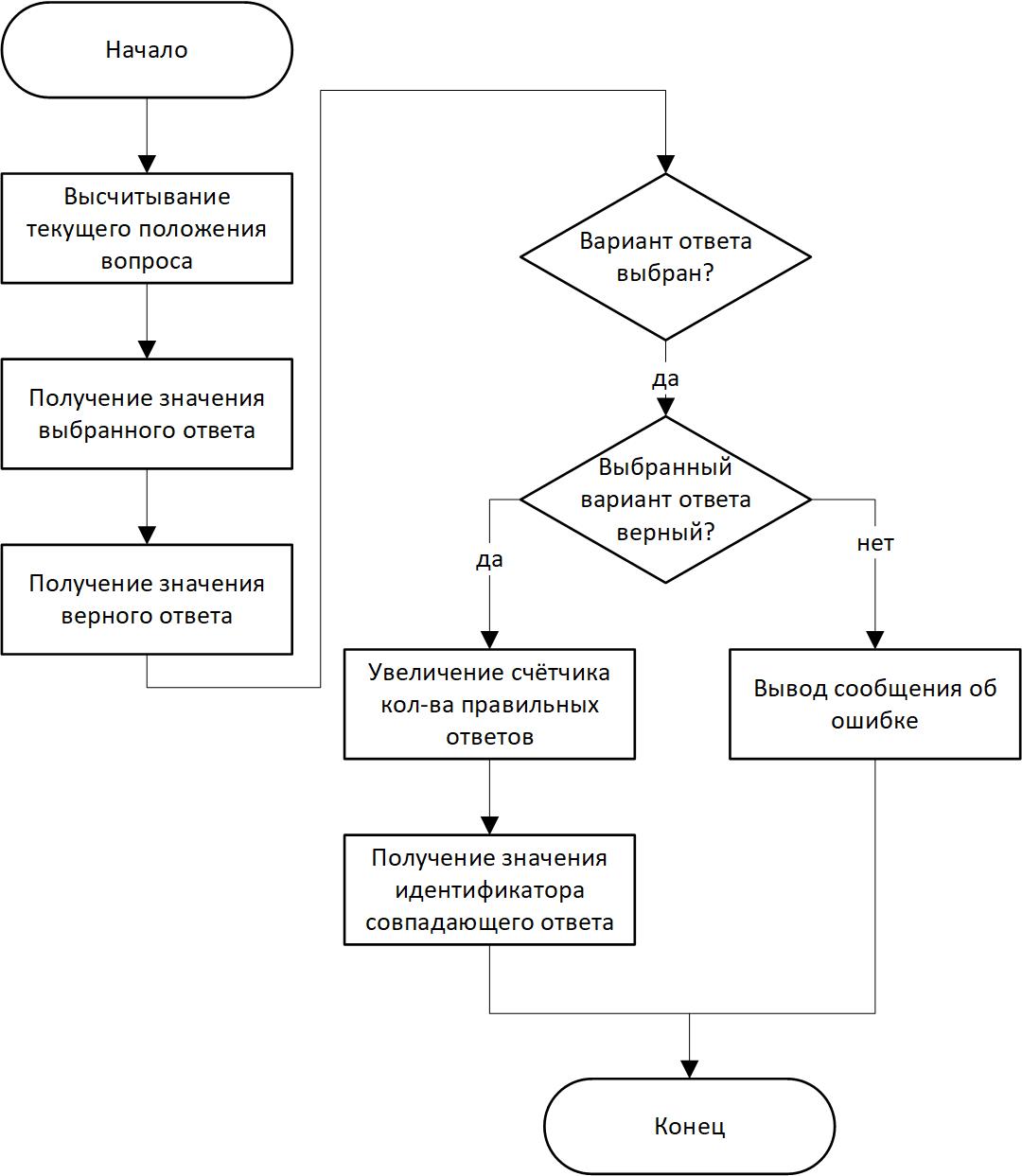


Рисунок 2.20 – Блок-схема метода SelectAnswerButton\_Click

Метод NextQuestionButton\_Click отвечает за проверку выбранного пользователем ответа. У метода нет ни входных, ни выходных данных, так как данные получаются в методе и в нём же передаются в элемент отображения. Блок-схема метода NextQuestionButton\_Click представлена на рисунке 2.21.

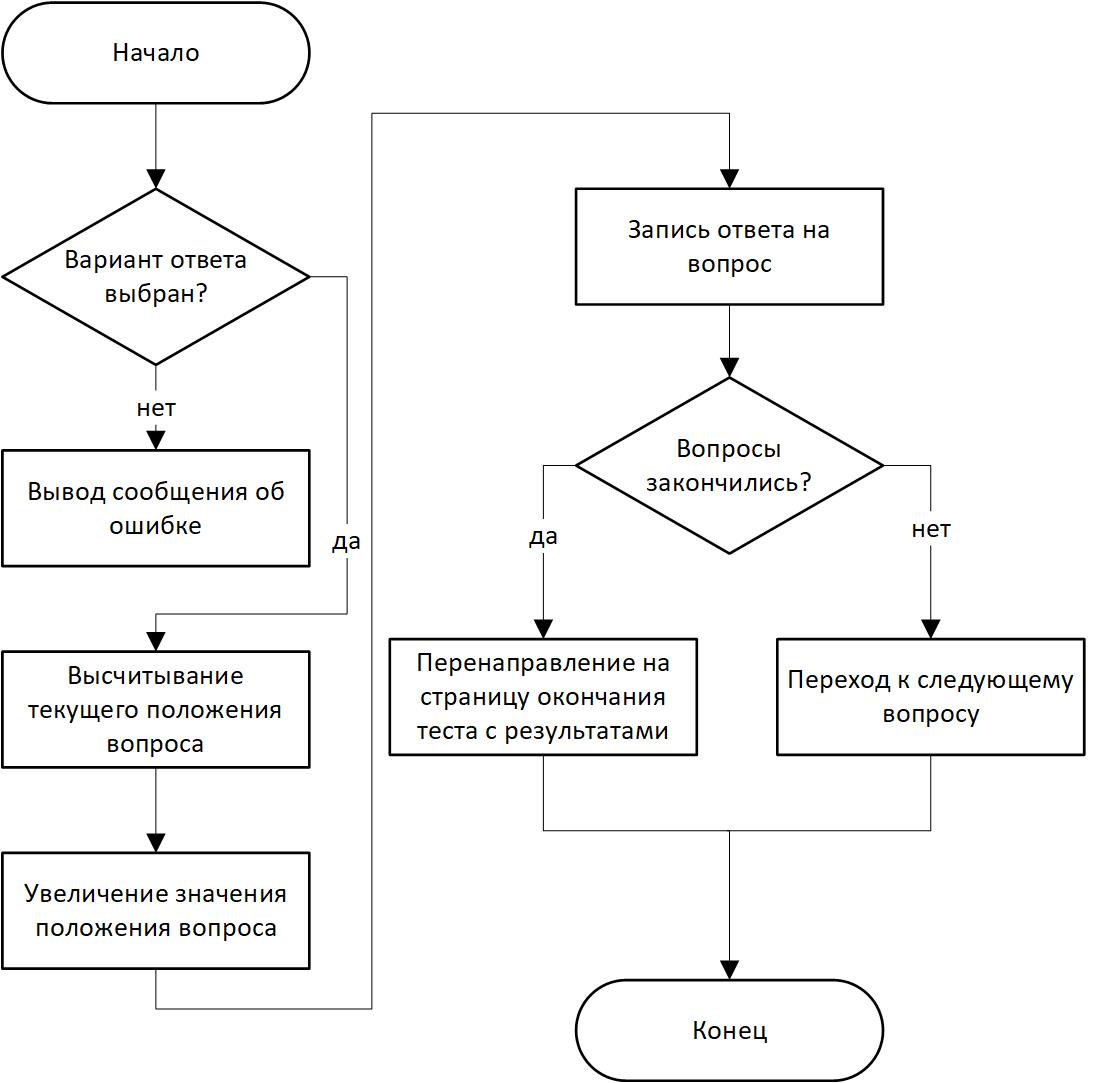


Рисунок 2.21 – Блок-схема метода NextQuestionButton\_Click

2.4 Обеспечение коллективного доступа. Защита информации

Основная идея ролевой модели контроля за доступом (Role-Based Access Control — RBAC) основана на максимальном приближении логики работы системы к реальному разделению функций персонала в организации.

Ролевой метод управления доступом контролирует доступ пользователей к информации на основе типов их активностей в системе. Применение данного метода подразумевает определение ролей в системе. Понятие роль можно определить, как совокупность действий и обязанностей, связанных с определенным видом деятельности. Таким образом, вместо того чтобы указывать все типы доступа для каждого пользователя к каждому объекту, достаточно указать тип доступа к объектам для роли. А пользователям, в свою очередь, указать их роли. Пользователь, «выполняющий» роль, имеет доступ, определенный для роли. Взято из материалов [7].

В системе доступно две роли, администратор и обычный пользователь. У администратора в отличии от обычного пользователя есть одно отличие – это возможность совершать над любыми тестами любые доступные действия, а то есть удаление или изменение тестов, независимо от того создал ли он их сам или нет.

Для авторизации пользователю необходимо ввести логин и пароль. В случае если пользователь не зарегистрирован, он сможет перейти на страницу регистрации из окна авторизации.

Для регистрации пользователю требуется ввести его логин и пароль. Все поля обладают своей валидацией и в случае некорректного ввода данных выводит сообщение с ошибкой. При желании пользователь может вернуться на страницу авторизации.

3 Технологическая часть

3.1 Тестирование и отладка приложения

Отладка — этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки, информация из работы. Взято из материалов [8]. В связи с тем, что почти невозможно составить реальную программу без ошибок, и почти невозможно для достаточно сложной программы быстро найти и устранить все имеющиеся в ней ошибки. Разумно уже при разработке программы на этапах алгоритмизации и программирования готовиться к обнаружению ошибок на стадии отладки принимать профилактические меры по их предупреждению, информация из работы. Взято из материалов [9].

Тестирование будет происходить через тест кейсы. Тест кейс – это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части. При передаче тестировщику тест-кейсов, он должен пройтись по всем его пунктам и выполнить описанные действия, которые должны привести к определенным результатам. информация из работы. Взято из материалов [10]. Тест кейс для функций представлен в таблице 3.1.

Таблица 3.1 – Тест-кейс для методов

|  |  |  |
| --- | --- | --- |
| Имя метода | Управляющее воздействие | Результат воздействия |
| AuthorizationCommit\_Click | Вызывается при нажатии кнопки «Войти» в окне авторизации | Отображение диалога ошибки если данные были введены некорректно |
| RegistrationButton\_Click | Вызывается при нажатии кнопки «Регистрация» в окне авторизации | Вывод окна регистрации |
| OkButton\_Click | Вызывается при нажатии кнопки «Регистрация» в окне регистрации | Отображение диалога ошибки если данные были введены некорректно |

Окончание таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Имя метода | Управляющее воздействие | Результат воздействия |
| TestQuestionButton\_Click | Вызывается при нажатии кнопки «Пройти тест» на главной странице | Отображение диалога ошибки если данные были введены некорректно |
| UserProfileEllipse | Вызывается при нажатии на элипс пользователя на главной странице | Отображение диалога ошибки если данные были введены некорректно |
| DeleteRecordButton\_Click | Вызывается при нажатии кнопки «Удалить запись» на странице пользователя | Отображение диалога ошибки если данные были введены некорректно |
| ShowResultButton\_Click | Вызывается при нажатии кнопки «Результат теста» на странице пользователя | Отображение диалога ошибки если данные были введены некорректно |
| EndTestButton\_Click | Вызывается при нажатии кнопки «Завершить» на странице результатов за прохождения теста | Переход на главную страницу |

3.2 Инструкция администратора базы данных

Перед началом работы с приложением необходимо установить и настроить SQL Server 2019. MS SQL Server это лидирующая РСУБД (Реляционная система управления базами данных) а также главный конкурент Oracle Database в корпоративном сегменте. В MSSQL чаще всего применяется для собственных разработок прикладного ПО и для 1С.

Для установки переходим на официальный сайт Microsoft и скачиваем бесплатную версию SQL Server 2019 для тестирования и разработки (Developer). Далее запускаем установщик и выбираем тип установки «Пользовательский», как на рисунке 3.1

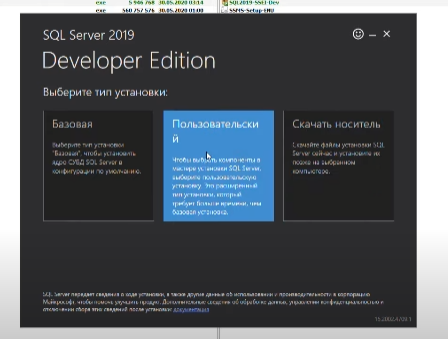


Рисунок 3.1 – «Пользовательский» тип установки

После выбора типа установки открывается следующее окно, где предлагается выбрать язык и место расположения носителя, можно выбрать стандартные настройки и нажать на кнопку «Установить». После чего начнется процесс загрузки. Изображение окна представлено на рисунке 3.2.

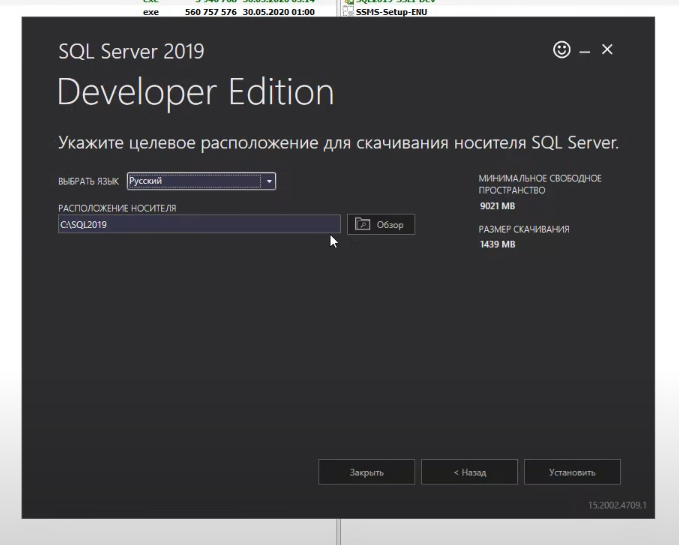


Рисунок 3.2 – Изображение окна

После установки откроется центр установки SQL server, где мы переходим в раздел установки «Новая установке изолированного экземпляра SQL Server или добавление компонентов к существующей установке», как показано на рисунке 3.3.

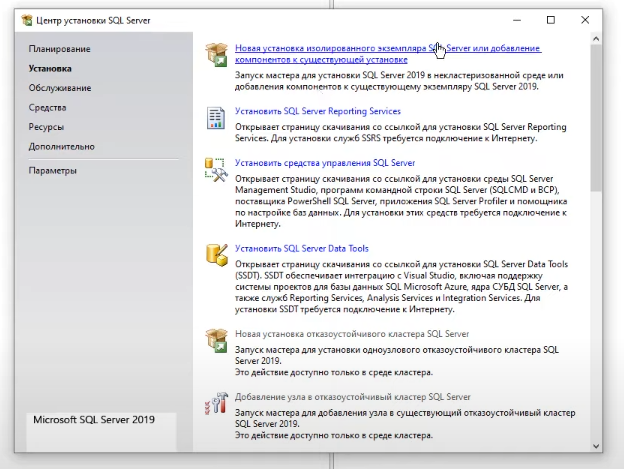


Рисунок 3.3 – Новая установке изолированного экземпляра SQL Server

После установки произойдет обновление продукта. Рисунок 3.4.

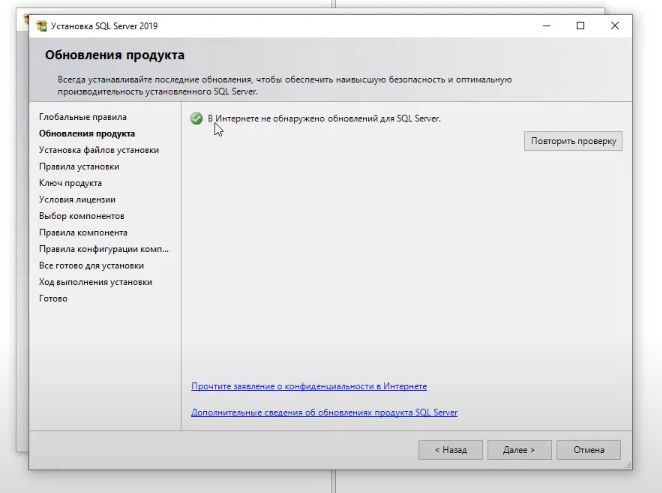


Рисунок 3.4 – Обновление продукта

После обновления пропускаем все пункты до ключа продукта и выбираем версию «Developer» как на рисунке 3.5.

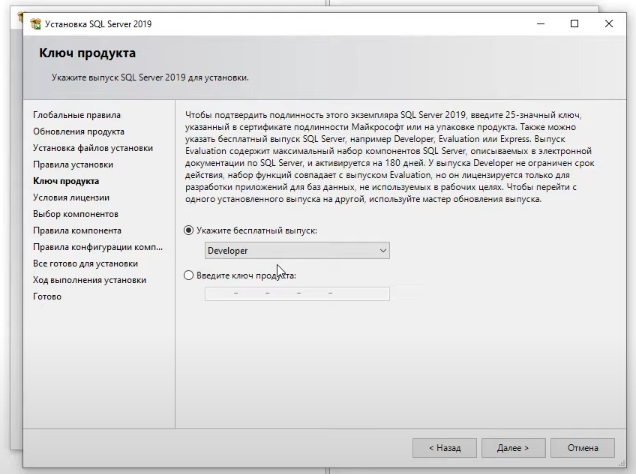


Рисунок 3.5 – Версия «Developer»

В пункте «Условия лицензии» принимаем условия и переходим в раздел «Выбор компонентов», где установим базовый набор компонентов: «Служба ядра СУБД» и «Полнотекстовой и семантический поиск», как представлено на рисунке 3.6.

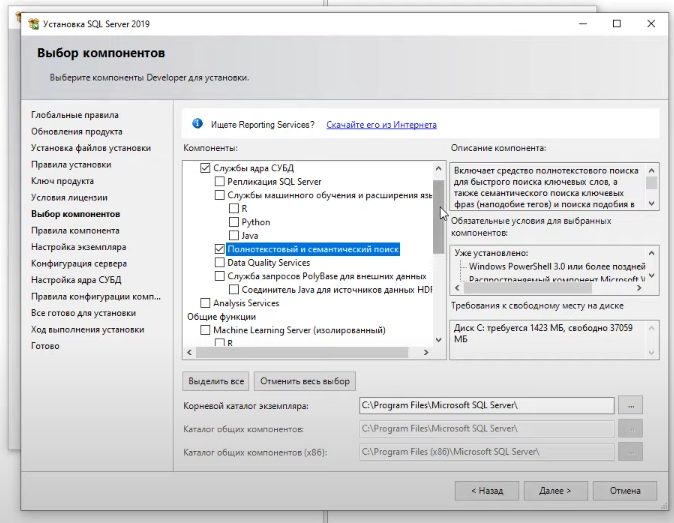


Рисунок 3.6 – Выбор компонентов

В разделе правила компонента всё оставляем по умолчанию и переходим в раздел «Конфигурация сервера» где можно настроить работу служб SQL Server. Задать тип запуска какой-либо службы. Поставить ее на автозапуск, вручную, или отключить. Так же можем зайти в меню "Параметры сортировки" — это настройки таблицы кодировок. Выполнять сортировку, как учитывать верхний и нижний регистр, как реагировать на символы, и т.п. Настройки можно оставить по умолчанию как на рисунке 3.7.

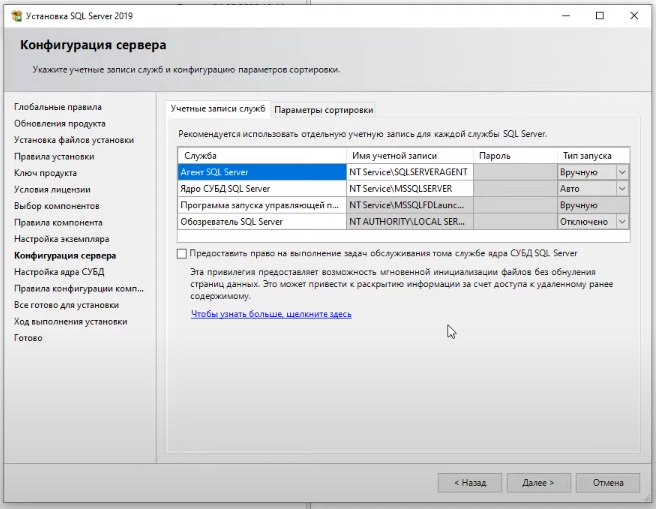


Рисунок 3.7 – Конфигурация сервера

В разделе «Настройка ядра СУБД» нам предлагают выбор режим входа под учетными записями Windows, либо смешанный режим, т.е. возможность входа под учетной записью Windows и под учетной записью SQL Server, если выбрать смешанную, то вам предложат создать учетную запись SQL Server. Оставляем режим аутентификации Windows и выбираем пользователя как показано на рисунке 3.8.

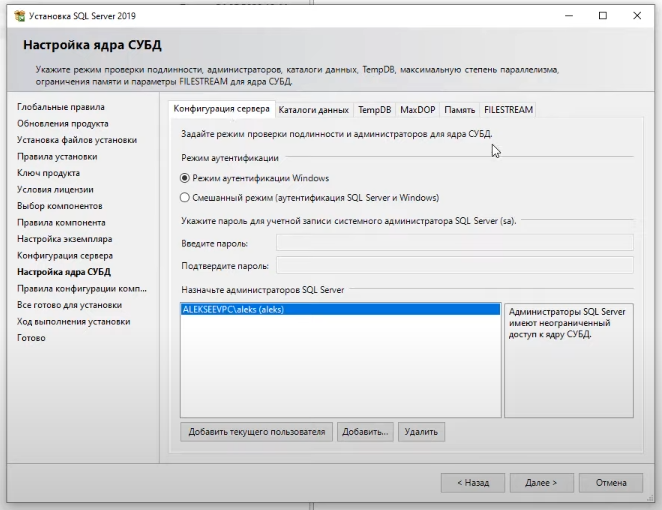


Рисунок 3.8 – Настройка ядра СУБД

В разделе «Все готово для установки» можно сверится с выбранными настройками» и начать установку. Окно с разделом представлено на рисунке 3.9.

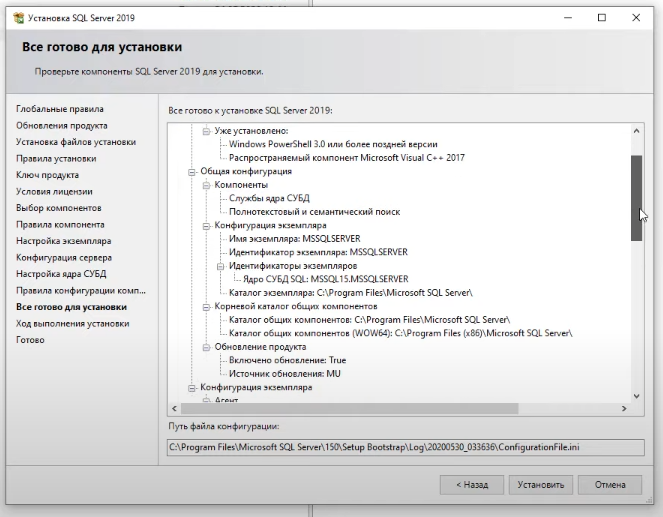


Рисунок 3.9 – Все готово для установки

Установка завершена и можно закрыть окно. Финальный экран представлен на рисунке 3.10.

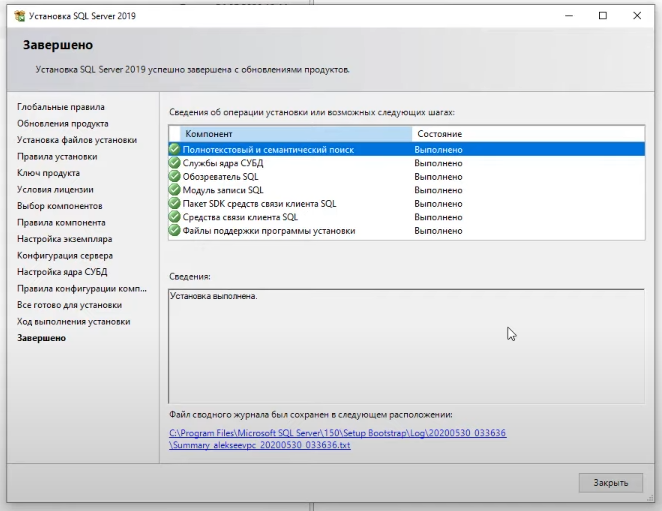
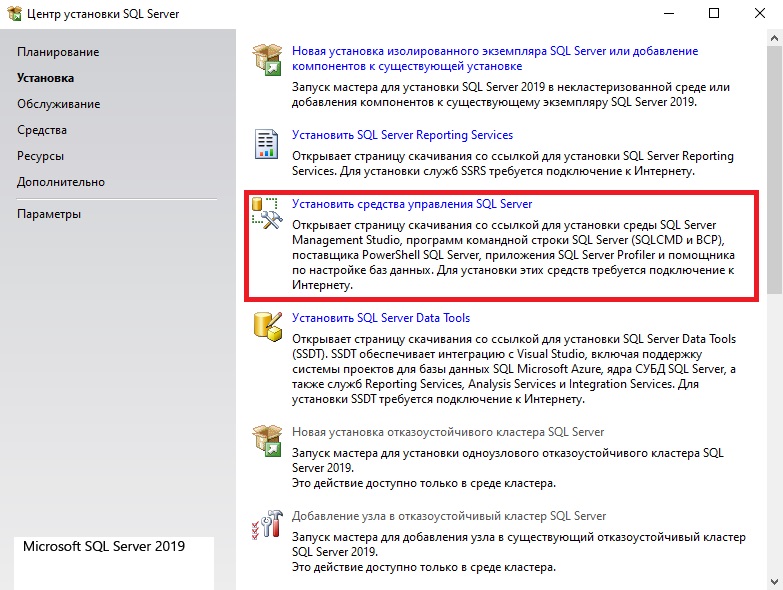


Рисунок 3.10 – Финальный экран

Если все прошло успешно, закрываем окно. После того, как установка SQL Server 2019 завершена, нам нужно установить приложение, с помощью которого мы будем подключаться к серверу баз данных. Это приложение SQL Server Management Studio (SSMS).

Заходим снова в центр установки SQL Server и наживаем "Установить средства управления SQL Server", как показано на рисунке 3.11.

Рисунок 3.11 – Установка SQL Server Management Studio

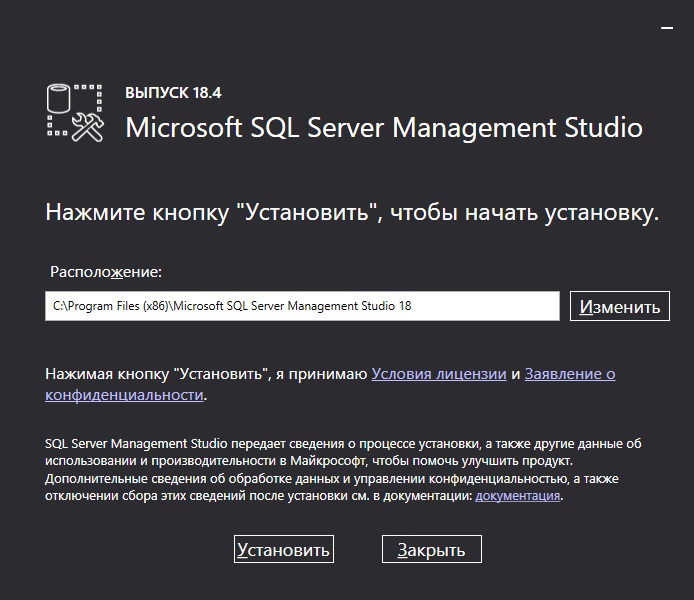
При нажатии у нас откроется [сайт Microsoft](https://docs.microsoft.com/ru-ru/sql/ssms/download-sql-server-management-studio-ssms?redirectedfrom=MSDN&view=sql-server-ver15) и нам нужно будет скачать SSMS. Нажимаем "Установить", как показано на рисунке 3.12.

Рисунок 3.12 – Установка SQL Server Management Studio

Реляционная схема базы данных представлена на рисунке 3.13.

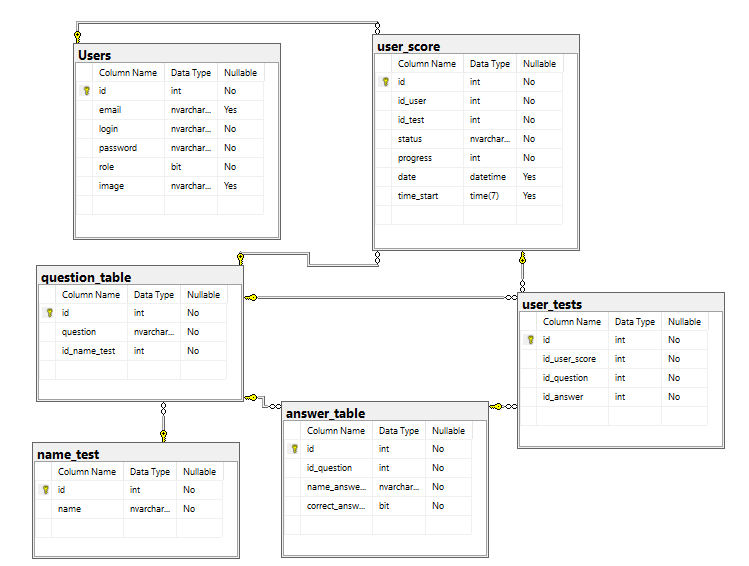


Рисунок 3.13 – Реляционная схема базы данных

Схема базы данных состоит из шести, описывающих сущности, отношений:

* Users – Учётные записи пользователей;
* user\_score – Статистика пользователя;
* user\_tests – Результаты за тест;
* name\_test – Имя теста;
* question\_table – Вопросы теста;
* answer\_table – Ответы теста;

3.3 Инструкция по эксплуатации приложения

Приложение предназначено для удобного прохождения, создания, редактирования и удаления тестов. Пользователь может установить приложения, после чего пройти регистрацию и в любой момент воспользоваться импроходя все или редактурия и изменяя свои тесты. Пользователь не сможет увидеть тесты после того, как администратор удалит их.

Чтобы начать работать с приложение достаточно его запустить. После запуска, приложение готово к работе. Окно с авторизованным пользователем представлено на рисунке 3.14.

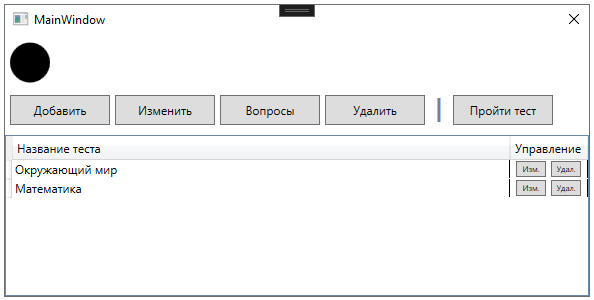


Рисунок 3.14 – Окно с авторизованным пользователем

Чтобы добавить в тест вопросы нужно на панели инструментов нажать на кнопку «Вопросы». Вести имя вопроса и ответы к нему, не забыв пометить верный галочкой. Панель добавления новых вопросов представлена на рисунке 3.15.

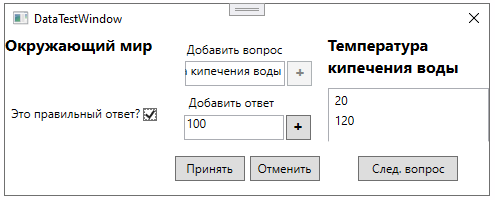


Рисунок 3.16 - Панель добавления новых вопросов

Чтобы добавить, изменить или удалить тест пользователя, нужно на панели инструментов нажать на кнопку добавления, изменения или удаления. Вести или изменить имя теста. Панель добавления нового теста представлено на рисунке 3.17.

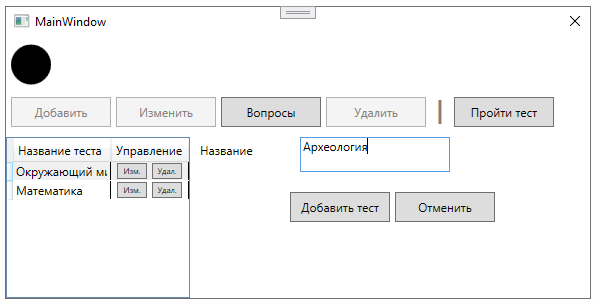


Рисунок 3.17 – Панель добавления нового теста

Для прохождения теста необходимо выбрать его из списка предложенных тестов на главном экране и нажать на кнопку «Пройти тест». После этого пользователь будет перенесён на страницу прохождения, на которой кнопка «Выбрать» будет отвечать за выбор правильного ответа, а кнопка «Дальше» за переход к следующему вопросу. Страница прохождения теста представлена на рисунке 3.18.

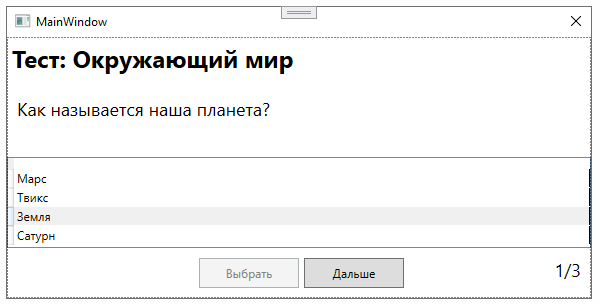


Рисунок 3.18 – Страница прохождения теста

Также клиент может посмотреть информацию об пройденных тестах в личном кабинете. При нажатии на кнопку профиль откроется страница с личным кабинетом, где можно узреть необходимое. Страница с личным кабинетом представлена на рисунке 3.19

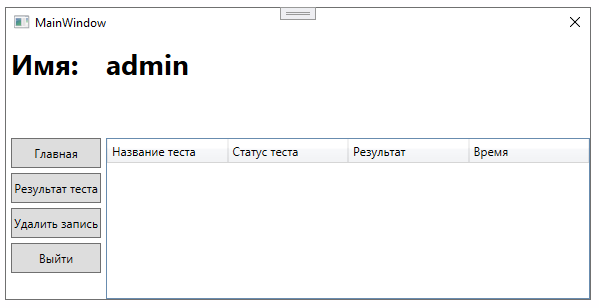


Рисунок 3.19 - Страница с личным кабинетом

4 Раздел охраны труда

Охрана труда – это целая система законодательных и нормативно-правовых актов, технических, гигиенических, лечебно-профилактических мероприятий и средств, которые обеспечивают безопасность, сохранение здоровья и работоспособности человека в процессе труда. В наши дни труд стал более интенсивным и требует огромных затрат умственной, эмоциональной и физической нагрузок.

На рабочем месте программист осуществляет трудовую деятельность и проводит большую часть рабочего времени. Правильная организация рабочего места программиста повышает производительность труда от 8 до 20%. Следуя рекомендациям ГОСТ 12.2.032-78, необходимо организовать рабочее место таким образом, чтобы взаимное расположение всех его элементов соответствовало физическим и психологическим требованиям. Главные элементы рабочего места программиста – это письменный стол и кресло. Рабочее место организуется в соответствии с ГОСТ 12.2.032-78, информация из работы [11].

Площадь рабочего места с компьютером с жидкокристаллическим или плазменным экраном должна быть не менее 4,5 кв. м, а расстояние между столами с мониторами (от тыла одного монитора до экрана другого) не менее 2 м. Монитор должен располагаться на расстоянии 50-70 см от глаз программиста. Параметры рабочего стола сотрудника: возможность регулировки высоты рабочего стола, или точная высота — 72,5 см, ширина — 80, 100, 120 или 140 см, глубина рабочего стола 80 или 100 см, высота и ширина пространства под столешницей (для ног) – не менее 50 см, глубина на уровне колен не менее 45 см, а на уровне вытянутых ног не менее 65 см.

Правильное освещение рабочего места – это очень важный момент в трудовой деятельности человека, влияющий на эффективность труда, при этом такой момент предупреждает травматизм и профессиональные заболевания. При недостаточном освещении приходится напрягать зрение, при этом ослабляется внимание и это приводит к наступлению преждевременной утомленности. Слишком яркое освещение тоже плохо, так как оно вызывает ослепление, раздражение и резь в глазах. При искусственном освещении, источниками света служат два вида ламп: лампы накаливания и люминесцентные.

Известно, что шум ухудшает условия труда и оказывает вредное воздействие на организм человека. Согласно ГОСТ 12.1.003-88 «Шум для помещений расчетчиков и программистов, уровни шума не должны превышать соответственно: 71, 61, 54, 49, 45, 42, 40, 38 дБ», информация из работы [12].

При работе компьютерной техники выделяется много тепла, что может привести к пожароопасной ситуации. Источниками зажигания так же могут служить приборы, применяемые для технического обслуживания, устройства электропитания, кондиционеры воздуха. Серьёзную опасность представляют различные электроизоляционные материалы, используемые для защиты от механических воздействий отдельных радиодеталей. В связи с этим, участки, на которых используется компьютерная техника, по пожарной опасности относятся к категории пожароопасных “В”. При пожаре люди должны покинуть помещение в течение минимального времени. В помещениях с компьютерной техникой, недопустимо применение воды и пены ввиду опасности повреждения или полного выхода из строя дорогостоящего электронного оборудования. Для тушения пожаров необходимо применять углекислотные и порошковые огнетушители, которые обладают высокой скоростью тушения, большим временем действия, возможностью тушения электроустановок, высокой эффективностью борьбы с огнем. Воду разрешено применять только во вспомогательных помещениях, информация из работы [13].

Заключение

По итогу работы было разработано приложение, позволяющее пользователю проходить созданные другими пользователями тесты. Для удобства пользователь также может зарегистрироваться в приложения для того, чтобы просматривать статистику по пройденным тестам и создавать собственные тесты. Пользователи с правами администратора имеют возможность модерировать список созданных тестов.

Главным достоинством можно выделить простоту в использовании приложения. Все дейсвтия выполняются на интуитовно понятном уровне.

Приложение можно использовать для создания любых несложных тестов.

Разработанное приложение можно доработать, добавив возможность создания более сложных и продвинутых тестов. А также добавив возможность просматривать результаты тестов других пользователей.

Список используемых источников

1. <https://kazedu.com/referat/133091/5>
2. <https://ruprogi.ru/software/visual-studio>
3. <https://gb.ru/posts/c_sharp_ides>
4. <https://anketolog.ru/tests.html>
5. <https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/overview/?view=netdesktop-6.0>
6. <https://metanit.com/sharp/wpf/11.php>
7. <https://steptosleep.ru/ролевая-модель/#:~:text=Основная%20идея%20ролевой%20модели%20контроля,типов%20их%20активностей%20в%20системе>
8. <https://studopedia.ru/22_29871_neobhodimost-otladki-programmnogo-produkta.html>
9. <https://infopedia.su/4x1ec5.html>
10. <https://sergeygavaga.gitbooks.io/kurs-lektsii-testirovanie-programnogo-obespecheni/content/lektsiya-4-ch3.html>
11. <https://www.retail.ru/rbc/pressreleases/tsentr-povysheniya-kvalifikatsii-lider-organizatsiya-rabochego-mesta-ofisnogo-rabotnika/>
12. <https://xn--d1aux.xn--p1ai/opisanie-rabochego-mesta-programmista-na-predpriyatii/>
13. <https://studopedia.ru/8_107307_osveshchenie-pomeshcheniy-vichislitelnih-tsentrov.html>

Приложение А

Программный код окна EndTestPage.xaml.cs

using System.Linq;

using System.Windows;

using System.Windows.Controls;

namespace EducationTests.Pages

{

/// <summary>

/// Логика взаимодействия для EndTestPage.xaml

/// </summary>

public partial class EndTestPage : Page

{

public EndTestPage(int result, int countQuestion, string testName, int userId, string timeAll, int userTestsId)

{

InitializeComponent();

Database = new Base.EducationTestsEntities();

\_testName = testName;

\_testId = Database.name\_test.SingleOrDefault(t => t.name == \_testName).id;

\_result = result;

\_countQuestion = countQuestion;

\_userId = userId;

\_timeAll = timeAll;

\_userTestsId = userTestsId;

NumTimeTextBlock.Text = \_timeAll;

NumResultTextBlock.Text = $"{\_result} из {\_countQuestion}";

}

private string \_testName;

private readonly int \_testId;

private readonly int \_result;

private readonly int \_countQuestion;

private readonly int \_userId;

private readonly string \_timeAll;

private readonly int \_userTestsId;

private Base.EducationTestsEntities Database;

private void EndTestButton\_Click(object sender, RoutedEventArgs e)

{

Base.user\_score user\_Score = new Base.user\_score { id\_user = \_userId, id\_test = \_testId, id\_user\_test = \_userTestsId, status = "Завершено", progress = $"{\_result}/{\_countQuestion}", date = \_timeAll };

Database.user\_score.Add(user\_Score);

Database.SaveChanges();

MainPage page = new MainPage(\_testId, \_userId, \_userTestsId);

this.NavigationService.Navigate(page);

}

private void RepeatTestButton\_Click(object sender, RoutedEventArgs e)

{

PassTestPage page = new PassTestPage(\_testName, \_userId);

this.NavigationService.Navigate(page);

}

}

}

Программный код страницы MainPage.xaml.cs

using EducationTests.Windows;

using System;

using System.Collections.ObjectModel;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

namespace EducationTests.Pages

{

/// <summary>

/// Interaction logic for MainPage.xaml

/// </summary>

public partial class MainPage : Page

{

public MainPage(int testId, int userId, int userTestsId)

{

Database = new Base.EducationTestsEntities();

InitializeComponent();

DataContext = this;

MainGrid.ItemsSource = Database.name\_test.ToList();

UpdateGrid(null);

TestDlgLoad(false, "");

DataContext = this;

\_testId = testId;

\_userId = userId;

\_userTestsId = userTestsId;

}

private int DlgMode;

private Base.name\_test SelectedTest;

private ObservableCollection<Base.name\_test> NameTest;

private readonly int \_testId;

private readonly int \_userId;

private readonly int \_userTestsId;

private Base.EducationTestsEntities Database;

public void TestDlgLoad(bool b, string DlgModeContent)

{

if (b == true)

{

TestColumnChange.Width = new GridLength(250);

TestAddCommit.Content = DlgModeContent;

TestAdd.IsEnabled = false;

TestEdit.IsEnabled = false;

TestDellete.IsEnabled = false;

StartTest.IsEnabled = false;

TestQuestion.IsEnabled = false;

}

else

{

TestColumnChange.Width = new GridLength(0);

MainGrid.IsHitTestVisible = true;

TestAdd.IsEnabled = true;

TestEdit.IsEnabled = true;

TestDellete.IsEnabled = true;

StartTest.IsEnabled = true;

TestQuestion.IsEnabled = true;

DlgMode = -1;

}

}

public void UpdateGrid(Base.name\_test nameTest)

{

if ((nameTest == null) && (MainGrid.ItemsSource != null))

{

nameTest = (Base.name\_test)MainGrid.SelectedItem;

}

NameTest = new ObservableCollection<Base.name\_test>(Database.name\_test);

MainGrid.ItemsSource = NameTest;

MainGrid.SelectedItem = nameTest;

}

private void TestAdd\_Click(object sender, RoutedEventArgs e)

{

TestDlgLoad(true, "Добавить тест");

DataContext = null;

DlgMode = 0;

}

private void TestEdit\_Click(object sender, RoutedEventArgs e)

{

if (MainGrid.SelectedItem != null)

{

TestDlgLoad(true, "Изменить тест");

SelectedTest = (Base.name\_test)MainGrid.SelectedItem;

TestTextName.Text = SelectedTest.name;

}

else

{

MessageBox.Show("Не выбрано ниодной строки!", "Сообщение", MessageBoxButton.OK);

}

}

private void TestDelete\_Click(object sender, RoutedEventArgs e)

{

if (MessageBox.Show("Удалить запись?", "Внимание", MessageBoxButton.OKCancel, MessageBoxImage.Warning) == MessageBoxResult.OK)

{

try

{

Base.name\_test DeletingBook = (Base.name\_test)MainGrid.SelectedItem;

if (MainGrid.SelectedIndex < MainGrid.Items.Count - 1)

{

MainGrid.SelectedIndex++;

}

else

{

if (MainGrid.SelectedIndex > 0)

{

MainGrid.SelectedIndex--;

}

}

Base.name\_test SelectingTest = (Base.name\_test)MainGrid.SelectedItem;

Database.name\_test.Remove(DeletingBook);

Database.SaveChanges();

UpdateGrid(SelectingTest);

}

catch

{

MessageBox.Show("Невозможно удалить запись, так как " +

"она используется в других справочниках базы " +

"данных.","Предупреждение", MessageBoxButton.OK, MessageBoxImage.Warning, MessageBoxResult.None);

}

}

}

private void TestAddCommit\_Click(object sender, RoutedEventArgs e)

{

StringBuilder errors = new StringBuilder();

if (string.IsNullOrEmpty(TestTextName.Text))

errors.AppendLine("Укажите имя теста");

if (errors.Length > 0)

{

MessageBox.Show(errors.ToString());

return;

}

if (DlgMode == 0)

{

var NewTest = new Base.name\_test();

NewTest.name = TestTextName.Text;

Database.name\_test.Add(NewTest);

SelectedTest = NewTest;

}

else

{

var EditTest = new Base.name\_test();

EditTest = Database.name\_test.First(t => t.name == SelectedTest.name);

EditTest.name = TestTextName.Text;

}

try

{

Database.SaveChanges();

TestDlgLoad(false, "");

UpdateGrid(SelectedTest);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message.ToString());

}

}

private void TestAddRollback\_Click(object sender, RoutedEventArgs e)

{

TestDlgLoad(false, "");

UpdateGrid(SelectedTest);

}

private void StartTest\_Click(object sender, RoutedEventArgs e)

{

try

{

if (MainGrid.SelectedIndex != -1)

{

Base.name\_test value = (Base.name\_test)MainGrid.SelectedItem;

int valueName = Database.name\_test.SingleOrDefault(t => t.name == value.name).id;

int valueTest = 0;

try

{ valueTest = Database.user\_tests.First(t => t.id\_test == valueName).id; }

catch { }

if (valueTest > 0)

{

MessageBox.Show("Данный тест уже пройден", "Инфо", MessageBoxButton.OK, MessageBoxImage.Question);

return;

}

PassTestPage page = new PassTestPage(value.name, \_userId);

this.NavigationService.Navigate(page);

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message.ToString());

}

}

private void UserProfileEllipse\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

try

{

UserProfilePage page = new UserProfilePage(\_userId, \_testId);

this.NavigationService.Navigate(page);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message.ToString());

}

}

private void TestQuestionButton\_Click(object sender, RoutedEventArgs e)

{

Base.name\_test value = null;

try { value = (Base.name\_test)MainGrid.SelectedItem; } catch { }

if (value == null)

{

MessageBox.Show("Выберите тест", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Question);

return;

}

DataTestWindow window = new DataTestWindow(value.name);

window.Show();

}

}

}

Программный код страницы PassTestPage.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Threading;

namespace EducationTests.Pages

{

/// <summary>

/// Логика взаимодействия для PassTestPage.xaml

/// </summary>

public partial class PassTestPage : Page

{

public PassTestPage(string testName, int userId)

{

InitializeComponent();

Database = new Base.EducationTestsEntities();

\_testName = testName;

\_testId = Database.name\_test.SingleOrDefault(t => t.name == \_testName).id;

\_idQuestion = 1;

\_userScore = 0;

questionList = Database.question\_table.Where(t => t.id\_name\_test == \_testId).ToList();

TestNameLabel.Content += Database.name\_test.SingleOrDefault(t => t.id == \_testId).name;

\_userId = userId;

CreateQuestion();

DispatcherTimerSample();

}

private string \_testName;

protected int \_testId;

protected int \_userScore;

protected int \_idQuestion;

protected List<Base.question\_table> questionList;

private readonly int \_userId;

DispatcherTimer timer;

private int \_second;

private int \_minute;

private string \_timeAll;

private Base.EducationTestsEntities Database;

public void DispatcherTimerSample()

{

\_second = 0;

\_minute = 0;

timer = new DispatcherTimer();

timer.Interval = TimeSpan.FromSeconds(1);

timer.Tick += Timer\_Tick;

timer.Start();

}

private void Timer\_Tick(object sender, EventArgs e)

{

if (\_second > 58)

{

++\_minute;

\_second = 0;

}

else ++\_second;

if (\_second > 9)

{

if (\_minute > 9) \_timeAll = $"{\_minute}:{\_second}";

else \_timeAll = $"0{\_minute}:{\_second}";

}

else

{

if (\_minute > 9) \_timeAll = $"{\_minute}:0{\_second}";

else \_timeAll = $"0{\_minute}:0{\_second}";

}

TimerLabel.Content = \_timeAll;

}

private void CreateQuestion()

{

QuestionNameTextBlock.Text = questionList[\_idQuestion - 1].question;

int id = questionList[\_idQuestion - 1].id;

AnswerGrid.ItemsSource = Database.answer\_table.Where(t => t.id\_question == id).ToList();

NumQuestionTextBLock.Text = $"{\_idQuestion}/{questionList.Count}";

}

private int \_selectValue;

private void SelectAnswerButton\_Click(object sender, RoutedEventArgs e)

{

int id = questionList[\_idQuestion - 1].id;

Base.answer\_table value = (Base.answer\_table)AnswerGrid.SelectedItem;

var correctValue = Database.answer\_table.SingleOrDefault(t => t.correct\_answer == true && t.id\_question == id).name\_answers;

if (value != null)

{

if (value.name\_answers == correctValue) ++\_userScore;

\_selectValue = Database.answer\_table.First(t => t.name\_answers == value.name\_answers).id;

}

else MessageBox.Show("Выберите вариант", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Question);

if (value != null) SelectAnswerButton.IsEnabled = false;

}

private void NextQuestionButton\_Click(object sender, RoutedEventArgs e)

{

if (SelectAnswerButton.IsEnabled == true)

{

MessageBox.Show("Выберите вариант", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Question);

return;

}

SelectAnswerButton.IsEnabled = true;

int id = questionList[\_idQuestion - 1].id;

++\_idQuestion;

Base.user\_tests user\_Tests = new Base.user\_tests { id\_user = \_userId, id\_test = \_testId, id\_question = id, id\_answer = \_selectValue};

Database.user\_tests.Add(user\_Tests);

if (questionList.Count < \_idQuestion)

{

timer.Stop();

Database.SaveChanges();

EndTestPage page = new EndTestPage(\_userScore, questionList.Count, \_testName, \_userId, \_timeAll, user\_Tests.id);

this.NavigationService.Navigate(page);

}

else CreateQuestion();

}

private void ExitTestButton\_Click(object sender, RoutedEventArgs e)

{

MainPage page = new MainPage(0, \_userId, 0);

this.NavigationService.Navigate(page);

}

}

}

Программный код страницы UserProfilePage.xaml.cs

using EducationTests.Base;

using EducationTests.Windows;

using System.Collections.ObjectModel;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

namespace EducationTests.Pages

{

/// <summary>

/// Логика взаимодействия для UserProfilePage.xaml

/// </summary>

public partial class UserProfilePage : Page

{

public UserProfilePage(int userId, int testId)

{

InitializeComponent();

Database = new Base.EducationTestsEntities();

\_userId = userId;

\_testId = testId;

UserProfileGrid.ItemsSource = Database.user\_score.ToList();

UpdateGrid(null);

UserNameTextBlock.Text = Database.Users.SingleOrDefault(t => t.id == \_userId).login;

}

private readonly int \_userId;

private readonly int \_testId;

private ObservableCollection<Base.user\_score> UserScore;

private Base.EducationTestsEntities Database;

public void UpdateGrid(Base.user\_score userScore)

{

if ((userScore == null) && (UserProfileGrid.ItemsSource != null))

{

userScore = (Base.user\_score)UserProfileGrid.SelectedItem;

}

UserScore = new ObservableCollection<Base.user\_score>(Database.user\_score.Where(t => t.id\_user == \_userId));

UserProfileGrid.ItemsSource = UserScore;

UserProfileGrid.SelectedItem = userScore;

}

private void ReturnMainButton\_Click(object sender, RoutedEventArgs e)

{

MainPage page = new MainPage(0, \_userId, \_testId);

this.NavigationService.Navigate(page);

}

private void ExitAccountButton\_Click(object sender, RoutedEventArgs e)

{

LoginWindow window = new LoginWindow();

Window.GetWindow(this).Close();

window.Show();

}

private void DeleteRecordButton\_Click(object sender, RoutedEventArgs e)

{

if (MessageBox.Show("Удалить запись?", "Внимание", MessageBoxButton.OKCancel, MessageBoxImage.Warning) == MessageBoxResult.OK)

{

try

{

Base.user\_score DeletingScore = (Base.user\_score)UserProfileGrid.SelectedItem;

ObservableCollection<Base.user\_tests> DeletingUserTests = new ObservableCollection<user\_tests>(Database.user\_tests.Where(t => t.id\_user == \_userId && t.id\_test == \_testId));

if (UserProfileGrid.SelectedIndex < UserProfileGrid.Items.Count - 1)

{

UserProfileGrid.SelectedIndex++;

}

else

{

if (UserProfileGrid.SelectedIndex > 0)

{

UserProfileGrid.SelectedIndex--;

}

}

Base.user\_score SelectingTest = (Base.user\_score)UserProfileGrid.SelectedItem;

Database.user\_score.Remove(DeletingScore);

Database.SaveChanges();

for (int i = 0; i < DeletingUserTests.Count; ++i)

{

Database.user\_tests.Remove(DeletingUserTests[i]);

}

Database.SaveChanges();

UpdateGrid(SelectingTest);

}

catch

{

MessageBox.Show("Невозможно удалить запись, так как " +

"она используется в других справочниках базы " +

"данных.", "Предупреждение", MessageBoxButton.OK, MessageBoxImage.Warning, MessageBoxResult.None);

}

}

}

private void ShowResultButton\_Click(object sender, RoutedEventArgs e)

{

Base.user\_score value = null;

try { value = (Base.user\_score)UserProfileGrid.SelectedItem; } catch { }

if (value == null)

{

MessageBox.Show("Выберите тест", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Question);

return;

}

UserTestResult window = new UserTestResult(value.id\_test, \_userId);

window.Show();

}

}

}

Программный код окна DataTestWindow.xaml.cs

using System;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Input;

namespace EducationTests.Windows

{

/// <summary>

/// Логика взаимодействия для DataTestWindow.xaml

/// </summary>

public partial class DataTestWindow : Window

{

public DataTestWindow(string testName)

{

InitializeComponent();

Database = new Base.EducationTestsEntities();

\_testName = testName;

TestNameTextBlock.Text = \_testName;

\_testId = Database.name\_test.SingleOrDefault(t => t.name == \_testName).id;

}

private int DlgMode;

private Base.EducationTestsEntities Database;

private string \_testName;

private string \_questionName;

private string \_answerName;

private int \_questionId;

private Base.answer\_table \_answer;

private int \_testId;

private void AddQuestionButton\_Click(object sender, RoutedEventArgs e)

{

\_questionName = TextQuestion.Text;

if (\_questionName == "")

{

MessageBox.Show("Введите вопрос.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Question);

return;

}

QuestionNameTextBlock.Text = \_questionName;

var QuestionTest = new Base.question\_table();

QuestionTest.id\_name\_test = \_testId;

QuestionTest.question = \_questionName;

Database.question\_table.Add(QuestionTest);

Database.SaveChanges();

\_questionId = Database.question\_table.SingleOrDefault(t => t.question == \_questionName).id;

AddQuestionButton.IsEnabled = false;

}

private void AddAnswerButton\_Click(object sender, RoutedEventArgs e)

{

if (\_questionName == null)

{

MessageBox.Show("Введите вопрос.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Question);

return;

}

\_answerName = TextAnswer.Text;

if (\_answerName == "")

{

MessageBox.Show("Введите ответ.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Question);

return;

}

AnswerList.Items.Add(\_answerName);

Base.answer\_table answer = new Base.answer\_table();

answer.name\_answers = \_answerName;

answer.id\_question = \_questionId;

answer.correct\_answer = true ? CorrectCheck.IsChecked == true : false;

CorrectCheck.IsChecked = false;

TextAnswer.Clear();

Database.answer\_table.Add(answer);

\_answer = answer;

}

private void AddCommitButton\_Click(object sender, RoutedEventArgs e)

{

StringBuilder errors = new StringBuilder();

if (string.IsNullOrEmpty(TextQuestion.Text))

errors.AppendLine("Укажите имя вопроса.");

if (string.IsNullOrEmpty(AnswerList.Items.ToString()))

errors.AppendLine("Укажите ответы.");

if (errors.Length > 0)

{

MessageBox.Show(errors.ToString());

return;

}

try

{

AddCommitButton.IsEnabled = false;

Database.SaveChanges();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message.ToString());

}

}

private void NextQuestionButton\_Click(object sender, RoutedEventArgs e)

{

if (QuestionNameTextBlock.Text == "")

{

MessageBox.Show("Введите вопрос.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Question);

return;

}

if (AnswerList.Items.Count < 3)

{

MessageBox.Show("Малое количество ответов.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Question);

return;

}

TextQuestion.Clear();

TextAnswer.Clear();

AnswerList.Items.Clear();

QuestionNameTextBlock.Text = "";

AddQuestionButton.IsEnabled = true;

AddCommitButton.IsEnabled = true;

}

private void CancelButton\_Click(object sender, RoutedEventArgs e)

{

Base.question\_table RemoveQuestion = Database.question\_table.SingleOrDefault(t => t.id == \_questionId);

if (\_answer != null)

Database.answer\_table.Remove(\_answer);

if (RemoveQuestion != null)

Database.question\_table.Remove(RemoveQuestion);

Database.SaveChanges();

Close();

}

private void Window\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e) => DragMove();

}

}

Программный код окна LoginWindow.xaml.cs

using System.Linq;

using System.Windows;

using System.Windows.Input;

namespace EducationTests.Windows

{

/// <summary>

/// Interaction logic for LoginWindow.xaml

/// </summary>

public partial class LoginWindow : Window

{

public LoginWindow()

{

InitializeComponent();

try

{

DataBase = new Base.EducationTestsEntities();

}

catch

{

MessageBox.Show("Не удалось подключиться к базе данных. Проверьте настройки подключения приложения.",

"Предупреждение", MessageBoxButton.OK, MessageBoxImage.Warning);

Close();

}

}

private Base.EducationTestsEntities DataBase;

private void Window\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e) => DragMove();

private void AuthorizationCommit\_Click(object sender, RoutedEventArgs e)

{

Base.Users User = DataBase.Users.SingleOrDefault(U => U.login == LoginText.Text && U.password == PasswordText.Text);

if (User != null)

{

int userId = DataBase.Users.SingleOrDefault(U => U.login == LoginText.Text).id;

MainWindow window = new MainWindow(userId);

window.Show();

Close();

}

else

{

MessageBox.Show("Неверно указан логин и/или пароль!", "Предупреждение", MessageBoxButton.OK, MessageBoxImage.Warning);

};

}

private void RegistrationButton\_Click(object sender, RoutedEventArgs e)

{

RegistrationWindow window = new RegistrationWindow(DataBase);

window.ShowDialog();

}

}

}

Программный код окна MainWindow.xaml.cs

using System.Windows;

using System.Windows.Input;

namespace EducationTests

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow(int \_userId)

{

InitializeComponent();

MainFrame.Navigate(new Pages.MainPage(0, \_userId, 0));

}

private void Window\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e) => DragMove();

}

}

Программный код окна RegistrationWindow.xaml.cs

using Microsoft.Win32;

using System;

using System.Windows;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

namespace EducationTests.Windows

{

/// <summary>

/// Interaction logic for RegistrationWindow.xaml

/// </summary>

public partial class RegistrationWindow : Window

{

public RegistrationWindow(Base.EducationTestsEntities database)

{

InitializeComponent();

Database = database;

}

private Base.EducationTestsEntities Database;

private void OkButton\_Click(object sender, RoutedEventArgs e)

{

Base.Users User = new Base.Users();

User.login = LoginTextBox.Text;

User.password = PasswordPasswordBox.Password != "" ? PasswordPasswordBox.Password : PasswordTextBox.Text;

User.role = false;

Database.Users.Add(User);

Database.SaveChanges();

Close();

}

private void CancelButton\_Click(object sender, RoutedEventArgs e) => Close();

private void PasswordButton\_Click(object sender, RoutedEventArgs e)

{

String Password = PasswordPasswordBox.Password;

Visibility Visibility = PasswordPasswordBox.Visibility;

double Width = PasswordPasswordBox.ActualWidth;

PasswordButton.Content = Visibility == Visibility.Visible ? "Скрыть" : "Показать";

PasswordPasswordBox.Password = PasswordTextBox.Text;

PasswordPasswordBox.Visibility = PasswordTextBox.Visibility;

PasswordPasswordBox.Width = PasswordTextBox.Width;

PasswordTextBox.Text = Password;

PasswordTextBox.Visibility = Visibility;

PasswordTextBox.Width = Width;

}

private void Window\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

try { DragMove(); } catch { }

}

private void Ellipse\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

OpenFileDialog openDialog = new OpenFileDialog();

openDialog.Filter = "Image files (\*.BMP, \*.JPG, \*.JPEG, \*.GIF, \*.TIF, \*.PNG, \*.ICO, \*.EMF, \*.WMF)|\*.bmp;\*.jpg;\*.jpeg;\*.gif; \*.tif; \*.png; \*.ico; \*.emf; \*.wmf";

if (openDialog.ShowDialog() == true)

{

ImageSource image = new BitmapImage(new Uri(openDialog.FileName));

imageEllipse.ImageSource = new BitmapImage(new Uri(openDialog.FileName));

}

}

}

}

Программный код окна UserTestResult.xaml.cs

using System.Collections.ObjectModel;

using System.Linq;

using System.Windows;

namespace EducationTests.Windows

{

/// <summary>

/// Логика взаимодействия для UserTestResult.xaml

/// </summary>

public partial class UserTestResult : Window

{

public UserTestResult(int testId, int userId)

{

InitializeComponent();

Database = new Base.EducationTestsEntities();

\_testId = testId;

\_userId = userId;

TestNameTextBlock.Text = Database.name\_test.SingleOrDefault(t => t.id == \_testId).name;

UserResultGrid.ItemsSource = Database.user\_tests.ToList();

UpdateGrid(null);

}

private Base.EducationTestsEntities Database;

private ObservableCollection<Base.user\_tests> UserScore;

private int \_testId;

private int \_userId;

public void UpdateGrid(Base.user\_tests userScore)

{

if ((userScore == null) && (UserResultGrid.ItemsSource != null))

{

userScore = (Base.user\_tests)UserResultGrid.SelectedItem;

}

UserScore = new ObservableCollection<Base.user\_tests>(Database.user\_tests.Where(t => t.id\_user == \_userId && t.id\_test == \_testId));

UserResultGrid.ItemsSource = UserScore;

UserResultGrid.SelectedItem = userScore;

}

}

}