МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Факультет | | |  | | Информационных технологий | | |
| Кафедра | |  | | | Информационных систем и технологий | | |
| Специальность | | | | | 1-40 05 01-03 «Информационные системы и технологии» | |  |
|  |  |  |  |  | **ПОЯСНИТЕЛЬНАЯ ЗАПИСКА** | | |
|  |  |  |  |  | **КУРСОВОГО ПРОЕКТ:** | | |
| по дисциплине | | | |  | «Объектно-ориентированное программирование» | | |
| Тема | | | | | WPF приложение «Среда проверки знаний BrainC#» | | |
|  |  |  |  |  |  |  | |

|  |  |  |
| --- | --- | --- |
| Исполнитель | |  |
| студент 2 курса 1 группы | | Малмыга В. С. |
|  |  | (Ф.И.О.) |
| Руководитель | |  |
| ассистент | | Наркевич А. С. |
|  |  | (учен. степень, звание, должность, подпись, Ф.И.О.) |

|  |  |  |  |
| --- | --- | --- | --- |
| Курсовой проект защищен с оценкой | |  |  |
| Руководитель | |  | Наркевич А. С. |
|  | (подпись) |  |  |

Минск 2017

Содержание

[Введение 3](#_Toc483956224)

[1. Аналитическая часть 3](#_Toc483956225)

[1.1 Обзор темы проекта и анализ требований 3](#_Toc483956226)

[1.2 Анализ и проектирование архитектуры приложения 5](#_Toc483956227)

[2. Разработка проекта 7](#_Toc483956228)

[2.1 Основные компоненты паттерна 7](#_Toc483956229)

[2.2 Паттерн в деле 10](#_Toc483956230)

[2.3 Модель 12](#_Toc483956231)

[3. Дизайн 15](#_Toc483956232)

[4. Реализация основных требований приложения 17](#_Toc483956233)

[5. Тестирование 20](#_Toc483956234)

[Заключение 23](#_Toc483956235)

[Список используемых источников 24](#_Toc483956236)

# Введение

Данное приложение реализует функциональные возможности проверки знаний студентов по языку программирования С#. Подобного рода приложения должны реализовывать возможность объективной проверки знаний студентов с автоматическим выставлением оценки и хранением всех результатов в базе данных. Приложение должно быть расширяемым и изменяемым: должна быть реализована возможность редактирования существующих данных, а так же возможность добавление новых вопросов и заданий, так как технологии, как и язык C#, постоянно меняются и улучшаются. Среда проверки знаний должна быть максимально простой и одновременно реализовывать как можно больше функционала, который облегчит работу и студента, и преподавателя. Оценка выставляется автоматически, с просчетом всех данных внутри системы. Необходимым так же является наличие разделения ролей на студентов и преподавателей, причем возможности у данных ролей должны быть разные: преподаватель способен добавлять вопросы, проверять правильность ответов, а так же получать информацию о выполненных тестах группы или целого потока; студент же должен иметь минимальный набор возможностей: прохождение тестов, а так же просмотр выполненных заданий и результатов своих работ.

# Аналитическая часть

# 1.1 Обзор темы проекта и анализ требований

Так как основной частью разработки данного приложения будет тестовая среда, то стоит произвести четкий анализ правил составления тестов, а так же их проверки. Данная процедура необходима для того, чтобы знания студентов проверялись максимально объективно, а так же, чтобы оценка знаний была максимально точной.

Тесты можно классифицировать по различным признакам:

* по целям — информационные, диагностические, обучающие, мотивационные, аттестационные;
* по процедуре создания — стандартизованные, не стандартизованные;
* по способу формирования заданий — детерминированные, стохастические, динамические;
* по форме заданий — закрытого типа, открытого типа, установление соответствия, упорядочивание последовательности;
* по наличию обратной связи — традиционные и адаптивные

Важно составлять тесты таким образом, чтобы свести к минимуму вероятность их угадывания студентом наугад.

В среднем, многие исследователи полагают, что угадывающий наугад тесты студент пользуется следующим алгоритмом:

* предпочитать более развернутые, более обоснованные ответы, которые учитывают частные случаи, и избегать кратких, менее развернутых. Разработчик теста знаний может этим воспользоваться и сделать верный ответ кратким;
* если ответы образуют последовательность (например, на вопрос о дате Куликовской битвы: (1) 1200 год, (2) 1240 год, (3) 1300 год, (4) 1380 год), респондент будет стремиться дать средний ответ, предполагая, что правда в золотой середине. И ошибется, потому что Куликовская битва была в 1380 году, а автор задания позаботился о том, чтобы поставить верный ответ с краю;
* выбирать более наукообразные ответы, содержащие малоизвестные или иностранные слова;
* выбирать ответы, напоминающие по стилю написания что-то очень знакомое, ассоциирующееся с известной частью предметной области.

Для противодействия угадывания в своих тестах автор использует 6 вариантов правильного ответа, при этом, только один является верным. Таким образом, вероятность «угадать» правильный ответ составляет 1/6. Угадывая данный тест математически только каждый 1 студент из 28 430 288 029 929 700 000 студентов решит тест на 100%.

Именно данную вероятность и призван увеличить описанный ранее алгоритм, используемый студентами при угадывании тестов.

Существует ряд эффективных рекомендаций к составлению компьютерных тестов:

* Все ответы к данному вопросу должны выглядеть правдоподобно, заставляя учащегося анализировать каждый вариант ответа и выявлять в нем неточность или ошибку.
* Там, где это возможно, стоит привести несколько истинных ответов, каждый из которых, являясь верным, в той или иной степени дополняет остальные правильные ответы.
* Правильное утверждение не должно быть полностью созвучно определению, данному в учебнике, чтобы в нем сразу не угадывался правильный ответ. Для этого же допускается приводить заведомо неверные ответы, созвучные приведенным в учебниках определениям.
* Вопросы по каждой теме стоит подбирать таким образом, чтобы они наиболее полно охватывали все разделы курса и позволяли контролировать как усвоение учащимися теоретических знаний, так и навыки применения этих знаний на практике.
* Выбор способа оформления заданий обусловлен как возможностями применяемых тестирующих программ (в случае применения для тестирования компьютерной техники), так и психологическими особенностями учащихся.
* Процесс создания вариантов тестов всегда должен включать опытную стадию, т.е. прежде чем предложить всему потоку, целесообразно дать тест для небольшой группы учащихся. Этот метод в сочетании с разбором ответов наиболее эффективно выявляет все ошибки, допущенные при составлении тестов.

Важно применить проанализированные требования к своей тестовой среде, чтобы приложение соответствовала объективной проверке знаний студентов.

# 1.2 Анализ и проектирование архитектуры приложения

Для полной картины стоит описать и изобразить основные концепции работы приложения.

Данное приложение будет содержать базу данных, которая будет хранить следующий данные:

* Вопросы, а так же всю информацию о них (уровень сложности, количество приносимых баллов, категория)
* Учетные записи
* Все необходимые данные о пользователях
* Результаты тестов
* Оценки

Учитывая все необходимые данные, схема базы данных может иметь следующий вид (Рис. 1):

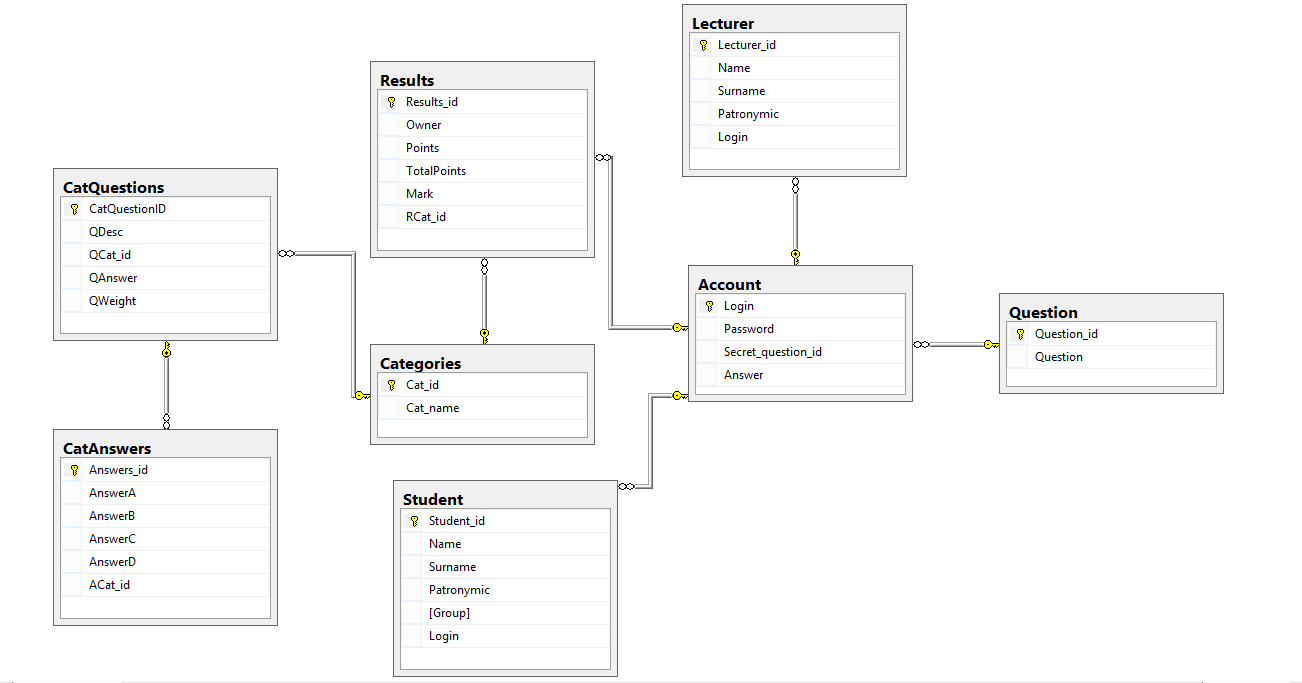


Рис. 1 – Диаграмма базы данных

Стоит описать каждую таблицу и ее содержимое в отдельности.

Таблица Account содержит себе информацию, которая будет необходима при авторизации пользователей в приложении: логин, пароль, секретный вопрос, а так же ответ на него. Очевидно, что наличие секретного вопроса в данной таблице, подразумевает создание отдельной таблицы для размещения перечня вопросов. Данная таблица имеет название Question с ID вопроса и его содержанием. Для более подробной информации о пользователях были созданы две таблицы для преподавателей (Lecturer), а так же для студентов (Student). Обе таблицы имеют схожий формат, с единственным отличием, что студент имеет группу. Важной частью базы данных является наличие таблицы Categories, которая содержи имя категории вопроса. Таблица CatQuestions содержит непосредственно вопрос, а так же важную информацию о нем: содержание, вес вопроса, правильный ответ, а так же категорию, к которой он относится. CatAnswers – таблица, содержащая перечень ответов на вопросы. Таблица результатов Results содержит логин человека, написавшего тест, количество набранных баллов, общее количество баллов, оценку, а так же категорию, по которой были получены данные результаты.

Вся основная часть приложения будет реализована в виде WPF приложения, которая будет связана с базой данных через Entity Framework. Схематически общие положения данного приложения можно изобразить так (Рис. 2):

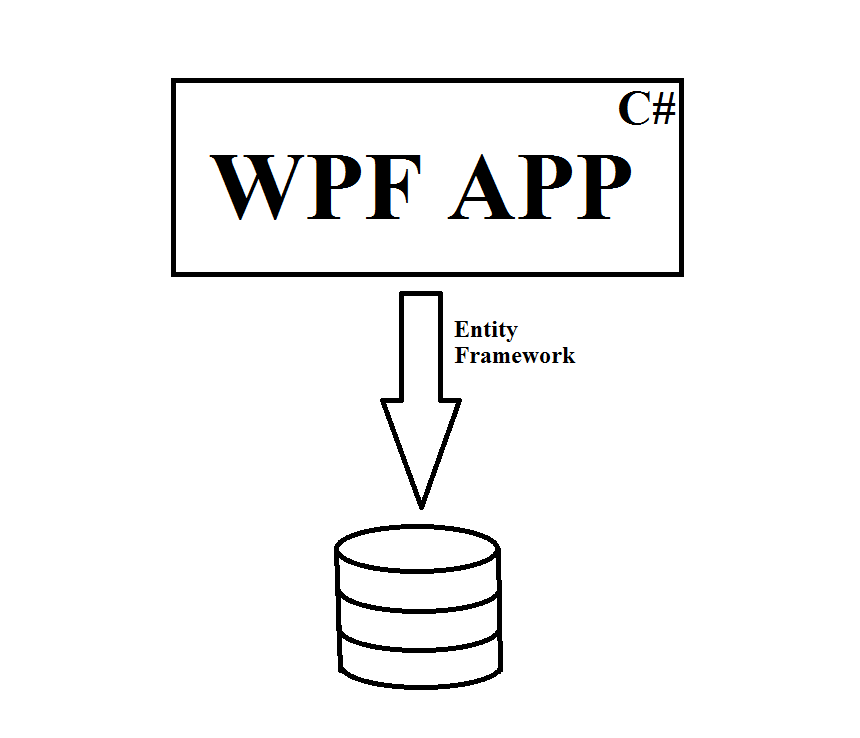


Рис. 2 – Схема работы приложения

Само WPF приложение будет реализовано с помощью паттерна MVVM(Model-View-ViewModel). Концепция данного паттерна подразумевает, что мы должны создать независимо View(Вид), который будет просто отвечать за вывод данных пользователю, а так же передачу информации в code behind. Независимая ViewModel, которая является связующим звеном между нашим View и моделью данных. ViewModel будет отвечать за события, а так же за посредничество между моделью и окнами. Так же существует Model, которая содержит всю информацию из базы данных, а так же бизнес логику проекта. Стоит отметить, что между тремя компонентами приложения существует строгое построение связей: View не должно ничего знать о модели или ViewModel, но должно быть связано с ViewModel при помощи Binding, ViewModel ничего не знает о View, но влияет на ее с помощью изменения своих свойств и прекрасно осведомлено о существовании Model, даже содержит в себе некоторые классы из модели, Model ничего не знает ни о View ни о ViewModel и лишь содержит в себе данные и методы обработки информации, а так же хранит информацию из базы данных. Схема данного паттерна может иметь вид (Рис. 3):

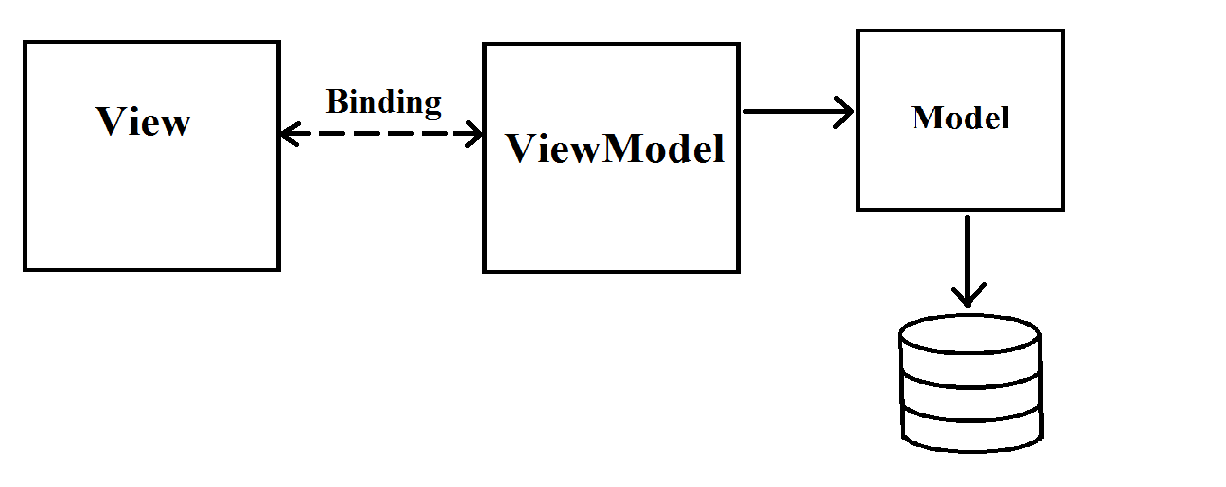


Рис. 3 – Схема паттерна MVVM

Стоит отметить, что данный паттерн желательно и даже нужно использовать по ряду причин:

* Позволяет легко расширять приложение
* Разбиение компонентов на отдельные части позволяет осуществлять разделение обязанностей по разработке приложения
* Так как Вид независим от code behind, то мы не зависим от дизайнера, который разрабатывает внешний вид приложения, а он не зависит от нас, что говорит о том, что приложение может разрабатываться максимально быстро и без помех
* Подобное разбиение компонентов позволяет вносить изменения в код и добавлять новую логику, практически не влияя на уже разработанную часть

Данный паттерн будет неэффективно использовать в случае небольших окон, но, в случае окон с большим количеством функционала, данный паттерн поможет решать сложные инженерные задачи.

# Разработка проекта

# 2.1 Основные компоненты паттерна

Первое, что необходимо сделать для реализации данного паттерна, это определить необходимые классы, функционал который позволит осуществлять необходимые действия над компонентами нашего приложения, а так же code behind. Первый и наиболее важным компонентом является реализация интерфейса INotifyPropertyChanged в наших классах ViewModel. В проекте будет содержаться большое количество ViewModel классов, следовательно, реализовывать данный интерфейс во всех данных классах является глупым решением. Более эффективным будет реализовать наследование от какого либо одного класса, который будет задавать поведение всем своим наследникам. Назовем данный класс ObservableObject и реализуем задуманное (Листинг 1):

public class ObservableObject : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged(string propertyName)

{

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs(propertyName));

}

}

Листинг 1 – ObservableObject

Все создаваемые ViewModel классы будут наследоваться от данного. Вторым важным аспектом является реализация паттерна Command, который позволяет осуществлять привязку команд в коде с нашими элементами управления в окнах. Для реализации этого класса будем использовать имя RelayCommand. Так же стоит создать обобщенный класс RelayCommand<T>, в случае, если требуется передача параметра в функцию. Таким образом класс может выглядеть так (Листинг 2):

public class RelayCommand<T> : ICommand

{

readonly Action<T> \_execute;

readonly Predicate<T> \_canExecute;

public RelayCommand(Action<T> execute, Predicate<T> canExecute)

{

if (execute == null)

throw new NullReferenceException("Null Execute Command Reference!");

\_execute = execute;

\_canExecute = canExecute;

}

public RelayCommand(Action<T> execute) : this(execute, null) { }

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

public bool CanExecute(object param)

{

return \_canExecute == null ? true : \_canExecute((T)param);

}

public void Execute(object param)

{

\_execute.Invoke((T)param);

}

}

public class RelayCommand : ICommand

{

readonly Action \_execute;

readonly Func<bool> \_canExecute;

public RelayCommand(Action execute, Func<bool> canExecute)

{

if (execute == null)

throw new NullReferenceException("Null Execute Command Reference!");

\_execute = execute;

\_canExecute = canExecute;

}

public RelayCommand(Action execute) : this(execute, null) { }

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

public bool CanExecute(object param)

{

if (\_canExecute != null)

{

return \_canExecute();

}

return true;

}

public void Execute(object param)

{

\_execute();

}

}

Листинг 2 – RelayCommand

Теперь любое свойство в нашем проекте может быть связано с свойством класса посредством привязки (Листинг 3):

<Button Grid.ColumnSpan="2" Content="Начать тест" HorizontalAlignment="Left" Margin="129,416,0,0" VerticalAlignment="Top" Width="144" Height="55" Command="{Binding StartTestCommand}" FontSize="18"/>

<Label Grid.ColumnSpan="2" Content="{Binding Mark}" HorizontalAlignment="Left" Margin="473,277,0,0" VerticalAlignment="Top" Height="26" Width="26" FontSize="18"/>

Листинг 3 –Binding

Так же для более удобного размещения элементов стоит разбить проект на каталоги, чтобы поиск необходимых классов и компонентов производился быстрее. Пример разделения проекта на каталоги для реализации паттерна MVVM приведен на рисунке 4:

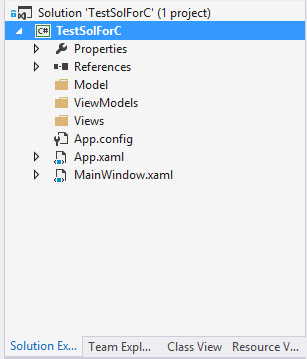


Рис. 5 – структура проекта MVVM

Очевидно, что в папке Model будут располагаться классы модели, в папке ViewModels – классы ViewModel, а во Views – все окна и иные компоненты интерфейса пользователя.

# 2.2 Паттерн в деле

Для большего понимания стоит реализовать данный паттерн на конкретном примере. В курсовом проекте для соответствия дизайну был разработан свой простой MessageBox, который реализует паттерн MVVM.

Для начала создадим сам View с именем MBView и снабдим всеми необходимыми элементами управления, а именно TextBlock и Button (Рис. 6):

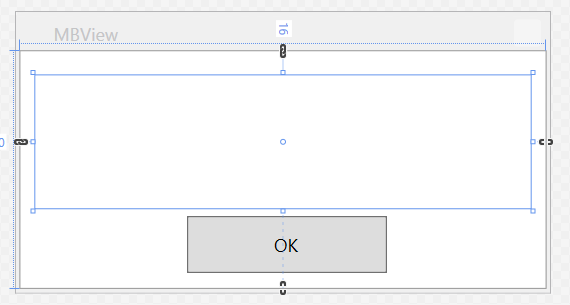


Рис. 6 – Окно MessageBox

Анализируя данное окно, можно выделить свойство типа string, на которое будет ссылаться наш текст TextBlock, а так же команду в классе ViewModel, к которой будет привязана кнопка. Дадим данной ViewModel имя MBViewModel и добавим в нее код, представленный в листинге 4:

public class MBViewModel : ObservableObject

{

private string \_message;

public string Message

{

get { return \_message; }

set

{

\_message = value;

OnPropertyChanged("Message");

}

}

public MBViewModel(string message)

{

Message = message;

CancelCommand = new RelayCommand(CancelExecute);

}

public RelayCommand CancelCommand { get; set; }

private void CancelExecute()

{

ViewsContainer.MessageWin.Close();

}

}

Листинг 4 – Код MBViewModel

Первоначально, стоит отметить, что данный класс наследуется от класса ObservableObject. Как было сказано ранее, это сделано для того, чтобы наше окно могло реагировать на изменения в код и наоборот, чтобы наши переменные в коде могли изменять свое значение в ответ на изменение состояния элементов управления. Далее мы декларируем наше поле \_message и связанное с ним свойство Message. Реализация свойства почти ничем не отличается от стандартного, за исключением того, что оно имеет метод OnPropertyChanged(param), которое дает системе информацию о том, что свойство, переданное в параметре, изменилось. Далее стоит обратить внимание на автоматическое свойство CancelCommand типа RelayCommand. Здесь мы объявляем наш объект класса RelayCommand, который будет хранить метод для обработки нажатия на кнопки. Метод CancelExecute и есть метод обработки, который будет закрывать текущее окно. Теперь, чтобы CancelCommand хранил метод, мы должны выделить для данного поля память и передать в конструктор метод-обработчик CancelExecute. Данную процедуру мы производим в конструкторе. В конструктор CancelCommand мы так же могли передать метод, возвращающий булево значение, который говорил о возможности или, напротив, невозможности нажатия на кнопку, чтобы уберечь пользователя от ошибок. Но данных действий недостаточно: теперь важны шагом является передачи экземпляра данной ViewModel в DataContext окна (Листинг 5):

public partial class MBView : Window

{

public MBView(string message)

{

InitializeComponent();

DataContext = new MBViewModel(message);

}

}

Листинг 5 – Класс окна

Последним действием является совершить привязку нашего TextBlock и Button к нашим данным из ViewModel (Листинг 6):

<TextBlock HorizontalAlignment="Left" Margin="10,16,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Height="112" Width="330" FontSize="18" TextAlignment="Center" Text="{Binding Message}"/>

<Button Content="OK" HorizontalAlignment="Left" Height="38" Margin="111,133,0,0" VerticalAlignment="Top" Width="133" Command="{Binding CancelCommand}"/>

Листинг 6 – Binding в MBView

Таким образом, мы получили полностью функционирующее окно, которое работает по паттерну MVVM. Теперь у нас появилась возможность изменять свойства объекта, не обращаясь к самим элементам управления. Все остальные окна будут реализованы по такому же принципу.

# 2.3 Модель

Для полноты данных нам не хватает информации из базы данных. Для ее подключения будет использоваться концепция Database First и Entity Framework, для упрощенной работы. Чтобы импортировать данные из базы данных необходимо создать объект Model.edmx, следуя стандартной процедуре подключения к базе данных. В результате Entity Framework сформирует нужную модель, основываясь на базе данных (Рис. 7):

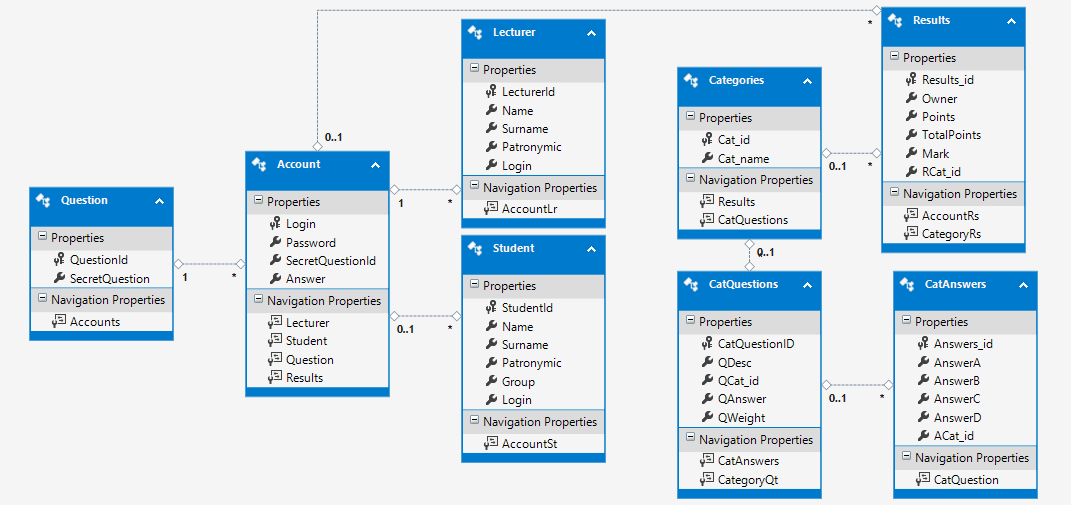


Рис. 7 – Модель базы данных

На основе данной модели так же будут сформированы необходимые классы, которые мы позже сможем использовать в нашем коде (Рис. 8):

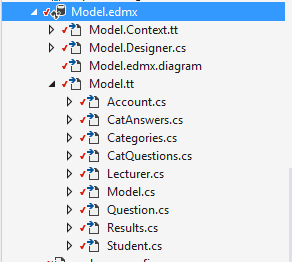


Рис. 8 – Классы модели

Наибольший интерес для нас здесь представляет класс ModelContex, который является хранилищем всех классов нашей базы данных (Листинг 7):

public partial class UsersContex : DbContext

{

public UsersContex()

: base("name=UsersContex")

{

}

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

throw new UnintentionalCodeFirstException();

}

public virtual DbSet<Account> Account { get; set; }

public virtual DbSet<Lecturer> Lecturer { get; set; }

public virtual DbSet<Question> Question { get; set; }

public virtual DbSet<Student> Student { get; set; }

public virtual DbSet<Categories> Categories { get; set; }

public virtual DbSet<Results> Results { get; set; }

public virtual DbSet<CatAnswers> CatAnswers { get; set; }

public virtual DbSet<CatQuestions> CatQuestions { get; set; }

}

Листинг 7 – Класс UserContex

Все классы и компоненты для разработки проекта собраны. Теперь остается просто развивать проект, следуя концепции паттерна. В итоге получим следующую иерархию классов и компонентов (Рис. 8):

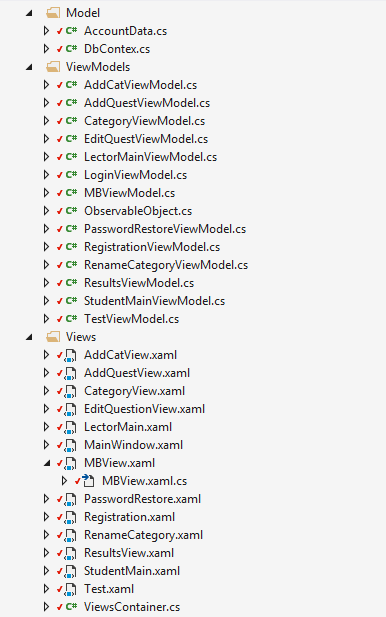


Рис. 8 – Финальная структура проекта

# Дизайн

Для данного приложения важными компонентами является удобность чтения, а так же простота элементов управления, следовательно, интерфейс не должен содержать лишних элементов управления и иметь простой и доступный дизайн. Цвета так же не должны быть слишком яркими и отвлекающими, так как они будут нарушать концентрацию студентов. В соответствии с требованиями, для данного приложения хорошо подойдет следующий набор цветов (Рис. 9):

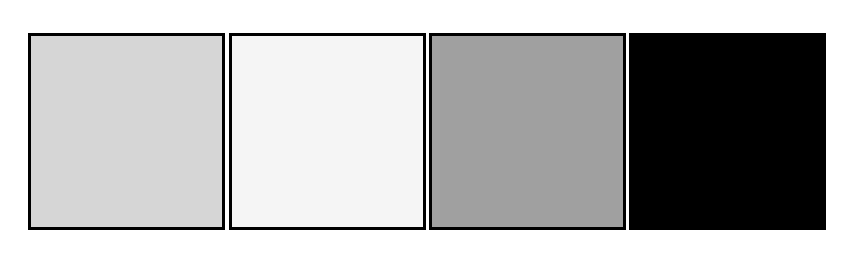


Рис. 9 – Набор цветов приложения

Помимо обычной цветовой раскраски, хочется так же придать окнам какой-то интересный интерактивный вид, а так же изменить форму кнопок и других элементов управления. К примеру, стоит изменить острые углы кнопок и других элементов управления на закругленные. На рисунке 10 приведен набор элементов управления, к которым были применены стилистические преобразования.

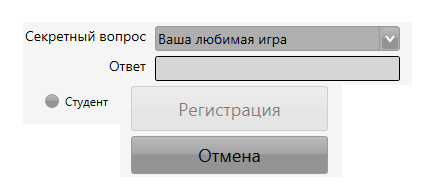


Рис. 10 – Стилизованные элементы управления

Стоит так же отметить, что подобную стилистику имеет и такой сложный элемент управления как ListBox (Рис. 11):

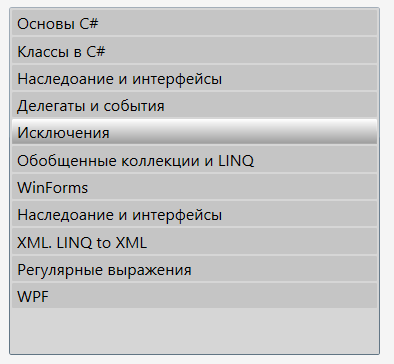


Рис. 11 - ListBox с примененными стилями

Теперь стоит добавить реакцию элемента управления при его наведении. Для данного приложения применим реакцию, изображенную на рисунке 12:

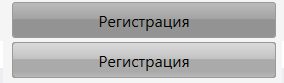


Рис. 12 – Реакция кнопки при наведении

Применив все вышеперечисленные принципы, получим полное стилизованное окно (Рис. 13):

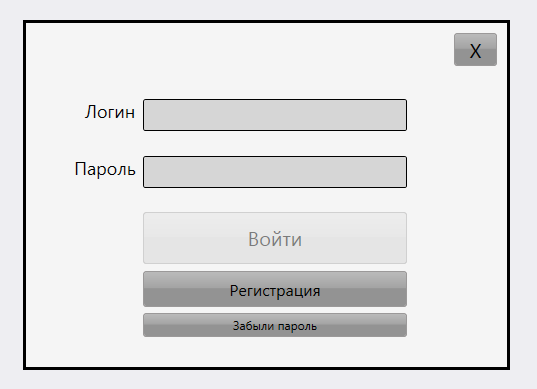


Рис. 13 – Стилизованное окно авторизации

Особую важность стоит придать шрифту: он должен быть читаемым, кажадя буква или цифра должна быть различима пользователем. Из большого выбора допустимых стилей был выбран шрифт Segoe UI:

Аа Ёё Лл Сс Чч Ээ Бб Жж Мм Тт Шш Юю Вв Зз Нн Уу Щщ Яя Гг Ии Оо Фф Ъъ Дд Йй Пп Хх Ыы Ее Кк Рр Цц Ьь

**Аа Ёё Лл Сс Чч Ээ Бб Жж Мм Тт Шш Юю Вв Зз Нн Уу Щщ Яя Гг Ии Оо Фф Ъъ Дд Йй Пп Хх Ыы Ее Кк Рр Цц Ьь**

*Аа Ёё Лл Сс Чч Ээ Бб Жж Мм Тт Шш Юю Вв Зз Нн Уу Щщ Яя Гг Ии Оо Фф Ъъ Дд Йй Пп Хх Ыы Ее Кк Рр Цц Ьь*

Аа Ёё Лл Сс Чч Ээ Бб Жж Мм Тт Шш Юю Вв Зз Нн Уу Щщ Яя Гг Ии Оо Фф Ъъ Дд Йй Пп Хх Ыы Ее Кк Рр Цц Ьь

Данный шрифт является вполне различимым и приятен для восприятия пользователем.

# Реализация основных требований приложения

Так как данное приложение призвано осуществлять объективную оценку студентам, оно должно ограничивать действия студентов, которые могут попытаться обмануть систему и воспользоваться ее слабостями. Первое, что необходимо сделать, это убрать возможность покинуть приложение в любой точке, например во время прохождения тестирования. К примеру, если окно прохождения тестирования будет иметь стандартную кнопку Windows «Закрыть», то студент может запомнить перечень вопросов, закрыть приложение и запустить тест снова, а результата предыдущего не будет записан в базу данных. Чтобы этого избежать во всех окнах была удалена кнопка «Закрыть» (Рис. 14).



Рис. 14 – Модифицированное окно

Теперь студент не сможет закрыть окно прохождения тестирования до тех пор, пока не пройдет тест, либо пока не завершит тест преднамеренно.

Студент может так же осуществить несколько попыток прохождения тесты, корректируя ответы на те вопросы, в которых он вероятней всего допустил ошибку. Таким образом студент будет запускать один и тот же тест до тех пор, пока не получит желаемый результат за прохождения тестирования. Чтобы этого избежать, в окно учетной записи преподавателя было введено специальное окно с выводом результатов написанных студентом тестов. Данное окно содержит перечень написанных текстов, категорию теста, а так же полный результат, причем написанные тесты расположены в порядке, в котором они были пройдены, следовательно попытки обмануть преподавателя усекаются (Рис. 15):

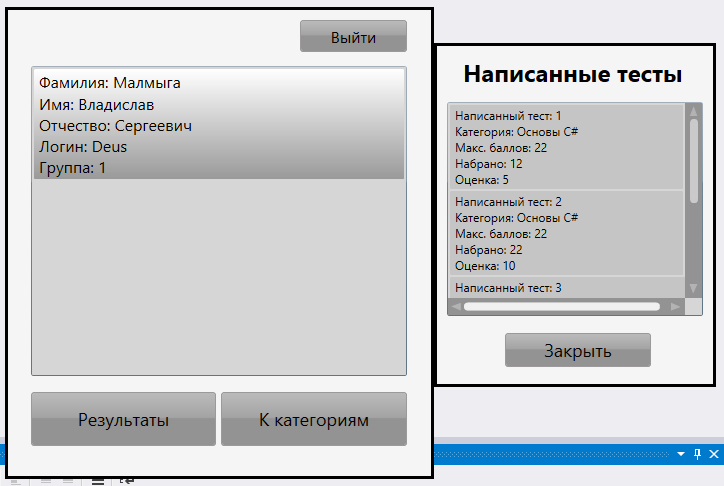


Рис. 15 – Окно результатов

Как видно из окна результатов, студент написал один и тот же тест 2 раза, изменяя свой результат прохождения тестов.

Третьим способом получить хороший результат нечестным путем является попытка создать учетную запись с правами преподавателя. Регистрация открыта для всех, и, следовательно, каждый может выбрать рол, которую будет иметь его четная запись. Чтобы избежать ситуации, при которой студент может создавать учетную запись преподавателя, стоит ввести дополнительное поле для ввода пароля, который будет известен только преподавателям и не должен быть ведом студентам (Рисю 15):

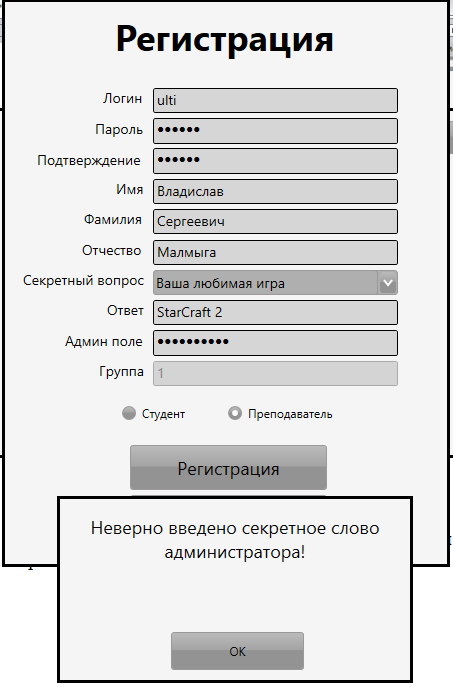


Рис. 15 – Неверный ввод в поле администратора

Таким образом, реализовав все вышеперечисленные функции приложения, мы добились того, что любые попытки студента обмануть систему сведены к минимуму, а, следовательно, можно достигнуть более объективного оценивания знаний студентов.

# Тестирование

Очевидно, что нет таких пользователей, которые выдержат постоянное свертывание приложения из-за того, что программист плохо организовал работу приложения, в котором неверное нажатие клавиши или ввод могут вызвать исключение, которое закроет приложение и прервет работу. Для избегания таких казусов стоит предусмотреть все возможные исключительные ситуации, при которых приложение может вылететь.

В данном приложении существует несколько механизмов предотвращения ошибок. Первое и самое сильное ограничение – это ограничение вводимого типа данных, который предоставляет нам сам паттерн и не требует от программиста даже реализации самостоятельного механизма проверки вводимых данных. На рисунке 16 представлена попытка ввести в поле «Группа» не целочисленного значения, в результате чего данное окно выделяется, а так же не позволяет пользователю покинуть его до тех пор, пока он н введет валидные данные.

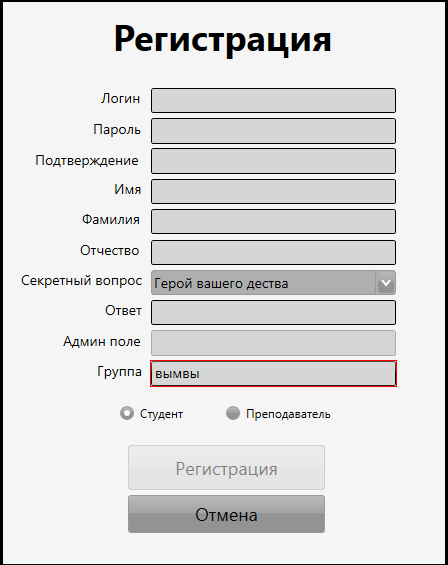


Рис. 16 – Ввод неверного типа данных

Второй механизм, который предотвращает вылет приложения – это стандартный механизм исключений. Он является наиболее простым и доступным, так как любая ошибка может быть выловлена и обработана, при этом приложение не вылетит, а пользователю может быть выдано сообщение о его неправильных действиях. На рисунке 17 представлена ситуация при которой пользователь пытается нажать кнопку «Начать тест», не выбрав категорию. На самом деле в коде, данное действие передает null значение, которое не может быть обработано, а следовательно вызывает исключение, которое может закрыть приложение.

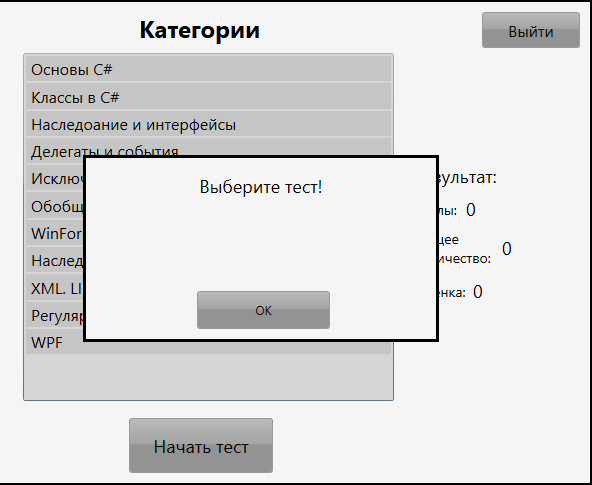


Рис. 17 – Исключительная ситуация

В листинге 8 представлена обработка данной ситуации.

private void StartTest()

{

try

{

int cat = 0;

List<Categories> catlist = DbContex.GetDbListCategories();

foreach (var s in catlist)

{

if (s.Cat\_name == Selected)

{

cat = s.Cat\_id;

if(s.CatQuestions.ToList().Count==0)

{

ViewsContainer.Show("Текущая категория не содержит вопросов!");

return;

}

}

}

ViewsContainer.TestWin = new Test(cat, \_login);

ViewsContainer.TestWin.ShowDialog();

\_results = DbContex.FindResults(\_login, \_password);

ChangeResults(\_selected);

}

catch

{

ViewsContainer.Show("Выберите тест!");

}

}

Листинг 8 – Обработка исключительной ситуации

# Заключение

Разработка приложений в области, неведомой программисту, является невероятно сложной задачей. Проектирование усложняется трудоемкими задачами, которые ставятся перед программистом в процессе разработки. Использования различного рода API интерфейсов позволяют значительно упростить подобные задачи. WPF сыграл большую роль в процессе разработки, упростив проектирование и развитие проекта. Паттерн MVVM так же помог реализовать сложные задумки и решить непростые задачи, путем высокого уровня абстракции.

# Список используемых источников

1. Тесты как эффективный инструмент проверки знаний – Режим доступа: <http://web.snauka.ru/issues/2014/02/31055>
2. Open Source took center stage – Режим доступа: [https://open.microsoft.com/#Ms-PL](https://open.microsoft.com/%23Ms-PL)
3. Using WPF styles tutorial – Режим доступа: <http://www.wpf-tutorial.com/styles/using-styles/>
4. WPF ToolKit – Режим доступа: <http://wpf.codeplex.com/>