Київський національний університет імені Тараса Шевченка

Курсова робота на тему

**«Словник з можливістю тестування»**

Виконав

студент 4 курсу, 3 групи

Цимбал Антон Миколайович

Науковий керівник:

Бородін Віктор Анатолійович

Київ 2016

**ЗМІСТ**

**Розділ 1.** Опис програми………………………………………………………….3

**Розділ 2.** Реалізація програми……………………………………………………6

**§2.1.** WPF та XAML………………………………………………………...6

**§2.2.** Шаблон проектування MVVM………………………………………6

**§2.3.** Binding (Зв’язування)………………………………………………...7

**§2.4.** ADO.NET Entity Framework………………………………………..10

**Розділ 3.** Висновок………………………………………………………………13

**Список використаних джерел**………………………………………………...14

Розділ 1. **Опис програми**

Дана програма надає можливість створювати власний словник іншомовних слів та їх перекладів. Оскільки програма не пов’язана з конкретною мовою, то можна створювати різноманітні словники (наприклад, англо-український, німецько-англійський та інші).

Використовуючи власноруч створений словник, можна проходити тестування за словами з нього. У даній програмі є два типи тестування, спільним для яких є те, що почергово відображаються всі *слова-запитання*, (слова з *мови\_1*, або з *мови\_2*, якщо маємо *мова\_1-мова\_2* словник) та пропонується відповісти на кожне з них. Користувач також може закінчити тестування раніше, ніж будуть відображені всі слова. Також є можливість пропустити слово.

Кожне додане в словник слово має також три показники: *Total*, *Correct*, *Incorrect* (кількість потрапляння слова в тесті, кількість правильних та кількість неправильних відповідей при потраплянні даного слова в тесті відповідно).

Користувач має можливість проходити один із двох типів тестування:

* **Так-Ні**. У цьому тестуванні *слово-запитання* обирається з *мови\_1*. На кожне слово користувач може відповісти двома варіантами: «Так, знаю переклад на *мову\_2»*, або «Ні, не знаю переклад на *мову\_2*». При будь-якій відповіді показник *Total* слова збільшується на одиницю. Якщо ж користувач обрав перший варіант відповіді, то показник *Correct* збільшується на одиницю, а якщо другий варіант відповіді – показник *Incorrect* збільшується на одиницю.

Перед початком даного тестування користувач повинен обрати спосіб сортування *слів-запитань*. Пропонується чотири види сортування: *ascending* (висхідний), *descending* (спадний), *simple random* (сортування випадковим чином), *clever selection* (на початку будуть з’являтися відсортовані випадковим чином слова, показник *Total* яких дорівнює 0. Потім будуть окремо сортуватися випадковим чином слова які мають показник *Incorrect* від найбільшого до найменшого).

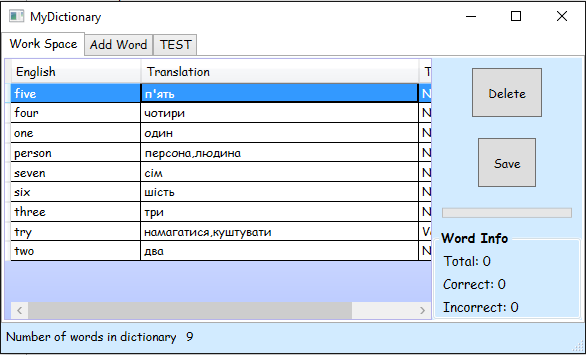
* **Транслятор**. У цьому тестуванні *слово-запитання* обирається або з *мови\_1*, або з *мови\_2* (за вибором користувача). На кожне слово користувач має написати переклад та підтвердити свою відповідь. Якщо відповідь правильна, то показник *Correct* збільшується на одиницю. У випадку неправильної відповіді – *Incorrect* збільшується на одиницю. Показник *Total* збільшується на одиницю в будь-якому разі. Сортування слів за замовчуванням відбувається методом *clever selection*.

Весь словник зберігається у локальній базі даних, тому можна один раз створити словник і змінювати його (додавати та видаляти слова), коли це буде необхідно.

Слід також зазначити, що немає можливості редагувати додані слова, тому треба уважно їх додавати. Також треба зауважити, що переклад для кожного слова має виглядати наступним чином: «переклад\_1,переклад\_2,переклад\_3,…». Тобто через кому та без пробілів між ними. Інакше програма може працювати не коректно.

Додаючи слово користувач має вказати три параметри: параметр *мова\_1*, параметр *мова\_2* та *тип слова*. У програмі є можливість обрати один із наступних *типів слова*: adjective, adverb, conjunction, determiner, interjection, noun, pronoun, phrase, preposition, verb, або ж залишити тип по замовчуванню – not given.

Розглянемо елементи основної вкладки у вікні програми, де відображається вміст бази даних:



Як бачимо, основну частину вкладки *Work Space* займає таблиця, пов’язана з базою даних. При виділенні слова з таблиці, в області *Word Info* відображаються параметри *Total*, *Correct* та *Incorrect*. Внизу вкладки відображається загальна кількість слів у словнику на момент роботи з програмою. Також тут користувач може видалити виділене в таблиці слово, натиснувши кнопку *Delete*.

Для додавання слова необхідно перейти на вкладку *Add Word*. Остання вкладка *TEST* надає можливість пройти вище описані тестування. Інтерфейс цих двох вкладок досить прозорий та зрозумілий для користувача, тому детальний розгляд їх вигляду можна пропустити.

Розділ 2. **Реалізація програми**

Програма реалізована на мові програмування *C#* на платформі *.NET Framework 4.0*. При розробці використовувалася графічна підсистема *WPF* (*Windows Presentation Foundation*). Для взаємодії з базою даних використовувалася технологія *ADO.NET* *Entity Framework*. Також слід зазначити, що програма побудована за шаблоном проектування *MVVM* (*Model-View-ViewModel*). Далі розглянемо всі використані технології детальніше.

§2.1.WPF та XAML

*WPF*(*Windows Presentation Foundation*) – графічна (презентаційна) підсистема в складі *.NET Framework*, що має пряме відношення до *XAML*.

*XAML* (*Extensible Application Markup Language*) є мовою розмітки, яку використовують для створення екземплярів об'єктів *.NET*. Хоча мова *XAML* – це технологія, що може бути застосовна до багатьох різних предметних областей, її головне призначення – конструювання інтерфейсів користувачів *WPF*. Інакше кажучи, документи *XAML* визначають розташування панелей, кнопок та інших елементів керування, що становлять вікна в програмі *WPF*.

§2.2.Шаблон проектування MVVM

*Model-View-ViewModel* (Модель-Представлення-Модель представлення) – це шаблон проектування, що застосовується під час проектування архітектури програми. *MVVM* полегшує відокремлення розробки графічного інтерфейсу від розробки бізнес логіки (бек-енд логіки), відомої як *Model* (можна також сказати, що це відокремлення *View* від *Model*). *ViewModel* являє собою частину, яка відповідає за перетворення даних для їх подальшої підтримки та використання. З цієї точки зору, *ViewModel* більше схожа на *Model*, ніж на *View* та оброблює більшість, якщо не всю, логіку відображення даних.

*MVVM* використовується для відокремлення *Model* та її *View*. Необхідністю цього є надання можливості змінювати їх незалежно одна від одної. Наприклад, розробник працює над логікою роботи з даними, а дизайнер – з користувацьким інтерфейсом.

Отже, шаблон *MVVM* ділиться на три частини:

* *Model (Модель)*, як і в класичному шаблоні *MVC* (*Model-View-Controller*), вона являє собою фундаментальні дані, що необхідні для роботи програми.
* *View (Представлення)*, як і в класичному шаблоні *MVC*, *View* – це графічний інтерфейс, тобто вікно, кнопки тощо.
* *ViewModel (Модель представлення)* з одного боку є абстракцією *View*, а з іншого – надає обгортку даних з *Model*, які мають зв'язуватись. Тобто вона містить *Model*, яка перетворена до *View*, а також містить у собі команди, якими може скористатися *View* для впливу на *Model*. Фактично *ViewModel* призначена для того, щоб здійснювати зв'язок між *Model* та вікном, відслідковувати зміни даних, що зроблені користувачем, а також відпрацьовувати логіку роботи *View*.

У цьому проекті *Model* – це база даних, *View* – *XAML* код, за допомогою якого відображається вікно програми у *WPF*, а *ViewModel* – клас *WindowViewModel*, де міститься логіка взаємодії користувача з програмою.

§2.3.Binding (Зв’язування)

Однією з причин використання саме *WPF* (а не, наприклад, *Windows Forms*) є можливість використання зв’язування даних (*Binding*). Це означає, що кожна характеристика об’єкта у *View* (*Button*, *TextBox*, *Label* тощо), яка може бути змінена в результаті роботи програми (зміна тексту, зміна видимості об’єкта та інше), має свою абстракцію у *ViewModel*, тобто пов’язана з деякою *властивістю* (*Property*). Розглянемо даний опис на прикладі одного з багатьох зв’язувань у даній програмі:

<**TextBlock** Grid.Row="2" Grid.Column="0" Grid.ColumnSpan="4" Text="SELECT APPROPRIATE TEST" **Visibility="{Binding SelectionOfTestVisibility}"** FontSize="22" TextAlignment="Center"/>

Це частина *XAML* коду, який міститься у *View*, де означено *TextBlock*, який відображає текст «SELECT APPROPRIATE TEST». Його атрибут видимості *Visibility* зв’язаний із властивістю *SelectionOfTestVisibility*, яка описана в класі *WindowViewModel* (тобто у *ViewModel*):

private string selectionOfTestVisibility;

public string SelectionOfTestVisibility

{

get { return selectionOfTestVisibility; }

set

{

selectionOfTestVisibility = value;

OnPropertyChanged("SelectionOfTestVisibility");

}

}

Важливу роль у цьому зв’язуванні відіграє метод *OnPropertyChanged()*, реалізований у класі *ObservableObject*, який в свою чергу реалізує інтерфейс *INotifyPropertyChanged*:

public class ObservableObject : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged(string propertyName)

{

if (PropertyChanged != null)

{

PropertyChanged(this,

new PropertyChangedEventArgs(propertyName));

}

}

}

Також зв’язування використовується для прив’язки властивостей до маніпуляції з об’єктами (наприклад, до натискання на кнопку):

<**Button** Grid.Row="4" **Command="{Binding Add}"** Content="Add Word" IsEnabled="{Binding IsButtonAddWordEnabled}" HorizontalAlignment="Center" VerticalAlignment="Center" Padding="10" Cursor="Hand"/>

Тут атрибут кнопки *Command*, який відповідає за подію після натискання на кнопку, зв’язаний із методом *AddToDictionary* у класі *WindowViewModel*:

private void AddToDictionary()

{

…

}

public DelegateCommand Add { get; set; }

Метод *AddToDictionary()* містить логіку додавання слова до бази даних.

У даному випадку властивість, з якою пов’язаний атрибут *Command* має тип *DelegateCommand*. Це клас, який реалізовує інтерфейс *ICommand*:

public class DelegateCommand : ICommand

{

private readonly Action action;

public DelegateCommand(Action action)

{

this.action = action;

}

public void Execute(object parameter)

{

action();

}

public bool CanExecute(object parameter)

{

return true;

}

public event EventHandler CanExecuteChanged;

}

Для остаточної реалізації даного зв'язування залишилося лише «підписатися» на метод *AddToDictionary()*:

Add = new DelegateCommand(AddToDictionary);

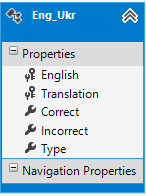
Це слід зробити в конструкторі класу *WindowViewModel*, тобто на самому початку роботи програми.

Очевидно, що використання *Binding* разом із шаблоном *MVVM* робить програму більш гнучкою та зручною для проведення змін. Наприклад, ми можемо замінити *View*, зовсім не змінюючи при цьому *ViewModel* (ми можемо просто зв’язати потрібний атрибут нового елемента *View* з уже описаною логікою у *ViewModel*).

§2.4.ADO.NET Entity Framework

У даному проекті використовується база даних для зберігання словника у вигляді однієї таблиці зі складовим ключем на двох полях: *мова\_1* та *мова\_2*. Тобто можна додавати до бази даних слова, які не мають, або мають лише один однаковий показник із двох зазначених, не порушуючи унікальності даних. Це зроблено для того, щоб можна було додавати слова з однаковим параметром *мова\_1*, але з різними параметрами перекладу (або навпаки). Логічно, що додавати слова з одночасно однаковими параметрами *мова\_1* та *мова\_2* просто немає сенсу, тому така функція є неможливою.

Наочно структура описаної бази даних виглядає наступним чином:



Поле *English* відповідає значенню *мова\_1* поле *Translation* – *мова\_2*, а поля *Correct*, *Incorrect* та *Type* були розглянуті раніше.

Для взаємодії з базою даних використовується технологія *ADO.NET Entity Framework*.

*ADO.NET* (*ActiveX Data Objects .NET*) – це набір бібліотек, що постачається з *Microsoft .NET Framework* і призначений для взаємодії з різними сховищами даних з *.NET*-програм. Бібліотеки *ADO.NET* включають класи для приєднання до джерела даних, виконання запитів і обробки їхніх результатів. Крім того, *ADO.NET* можна використовувати в якості надійного, ієрархічно організованого, відокремленого кешу даних для автономної роботи з даними.

*ADO.NET Entity Framework* – об'єктно-орієнтована технологія доступу до даних, є object-relational mapping (ORM) рішенням для *.NET Framework* від Microsoft. Надає можливість взаємодії з об'єктами як за допомогою *LINQ* у вигляді *LINQ to Entities*, так і з використанням *Entity SQL*.

У даному проекті використано *LINQ*(*Language Integrated Query*) – проект Microsoft по додаванню синтаксису мови запитів, що нагадує *SQL*, в мови програмування платформи *.NET Framework*.

Використовуючи описані технології, генерується клас на основі бази даних (кожний клас – це таблиця у базі даних). Взаємодіючи з об’єктами даного класу відбувається взаємодія з відповідними записами бази даних. Таким чином зміна у базі даних відображається на даних у програмі та навпаки.

Клас, створений на основі наведеної вище таблиці має вигляд:

public partial class Eng\_Ukr

{

public string English { get; set; }

public string Translation { get; set; }

public int Correct { get; set; }

public int Incorrect { get; set; }

public string Type { get; set; }

}

Таким чином, створюючи колекцію *ObservableCollection* із об’єктів класу *Eng\_Ukr*, можна змінювати записи таблиці бази даних.

private DictionaryDBEntities dataContext = new DictionaryDBEntities();

public static ObservableCollection<Eng\_Ukr> Dictionary { get; set; }

Dictionary = new ObservableCollection<Eng\_Ukr>(dataContext.Eng\_Ukr);

У даному випадку *dataContext* містить всі дані, отримані з бази даних, а *Dictionary* пов’язується з конкретною таблицею *dataContext.Eng\_Ukr*, що й було описано вище. Також *Dictionary* пов’язаний із таблицею у *View*.

Як зазначалося вище, для вибірки з бази даних використовується *LINQ*. Прикладом його застосування в програмі може слугувати вибірка слів, у яких показник *Total* (тобто *Correct + Incorrect*) рівний нулю. Це означає, що слово ще жодного разу не було задіяне в тестах. Дана вибірка використовується при сортуванні *CleverSelection* і має вигляд:

var selectedWordsWithNullInTotal = (from item in words

where item.Correct + item.Incorrect == 0

select item);

Слід зауважити, що тут *words* – це колекція *ObservableCollection<Eng\_Ukr>*, яка містить усі слова зі словника.

Бачимо, що синтаксис дуже схожий на *SQL* запит і означає, що з колекції *words* у вибірку потрапляють такі слова *item* (тобто такі об’єкти класу *Eng\_Ukr*), які задовольняють умові *Correct+Incorrect = 0*.

Результат вибірки записується у так звану неявно типізовану змінну з іменем *selectedWordsWithNullInTotal*. Така змінна з неявним типом має строгу типізацію, як якщо б тип був заданий явно, тільки тип визначає компілятор.

Розділ 3. **Висновок**

Програма надає можливість користувачеві самостійно складати словник саме з тих слів, які він бажає вивчити, а також проходити по ним тести, вдосконалюючи свої знання.

Користувач може видаляти внесені раніше слова, або додавати нові. Також є можливість відслідковувати свою успішність по кожному слову окремо, завдяки тому, що програма запам’ятовує кількість правильних та неправильних відповідей на кожне запитання.

Інтерфейс програми є досить простим, що забезпечує зручність у користуванні.

З повним кодом програми можна ознайомитися за посиланням:

https://github.com/fantotsy/Dictionary\_Project.git

**Список використаних джерел**

1. Документація від Microsoft­\_https://msdn.microsoft.com/ru-ru/default.aspx\_17.05.2016;
2. Паттерн MVVM­\_ http://professorweb.ru/my/WPF/documents\_WPF/level36/36\_5.php

\_17.05.2016;

1. Everything you wanted to know about databinding in WPF\_ http://blog.scottlogic.com/2012/04/05/everything-you-wanted-to-know-about-databinding-in-wpf-silverlight-and-wp7-part-one.html\_17.05.2016;
2. Использование Entity Framework совместно с WPF\_ http://ferrum-box.blogspot.com/2012/03/entity-framework-wpf.html\_17.05.2016.