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.** Теоретичні відомості.

**§1.1.** Використані технології.

**§1.2.** Суфіксний автомат.

**§1.3.** Застосування при розв’язуванні задач.

**Розділ 2.** Реалізація на мові С++.

**§2.1.** Загальний опис задачі.

**§2.2.** Функція «main».

**§2.3.** Функція «programm».

**Розділ 3.** Висновки.

**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 додавати слова. 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.

**Реалізація програми**

Програма реалізована на мові програмування C# з використанням графічної підсистеми *WPF* (Windows Presentation Foundation) та з використанням технології *Entity Framework* для взаємодії з базою даних. Також слід зазначити, що програма побудована за шаблоном проектування *MVVM* (Model-View-ViewModel).

Шаблон проектування MVVM

*Model-View-ViewModel* (Модель-Представлення-Модель представлення) – це шаблон проектування, що застосовується під час проектування архітектури програм. *MVVM* полегшує відокремлення розробки графічного інтерфейсу від розробки бізнес логіки (бек-енд логіки), відомої як модель (можна також сказати, що це відокремлення представлення від моделі). Модель представлення являє собою частину, яка відповідає за перетворення даних для їх подальшої підтримки та використання. З цієї точки зору, модель представлення більше схожа на модель, ніж на представлення та оброблює більшість, якщо не всю, логіку відображення даних. Модель представлення може також реалізовувати патерн *медіатор*, організовуючи доступ до бек-енд логіки навколо множини правил використання, які підтримуються представленням.

*MVVM* використовується для відокремлення моделі та її відображення. Необхідністю цього є надання можливості змінювати їх незалежно одну від одної. Наприклад, розробник працює над логікою роботи з даними, а дизайнер відповідно із користувацьким інтерфейсом.

Шаблон *MVVM* ділиться на три частини:

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

У цьому проекті *Модель* – 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.

Binding

Однією з причин використання саме *WPF* (а не, наприклад, *Windows Forms*) є використання зв’язування даних (*Binding*). Це означає, що кожна характеристика об’єкта у *Представленні* (*Button*, *TextBox*, *Label*,…), яка може бути змінена в результаті роботи програми (зміна тексту, зміна видимості об’єкта,…), 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 *TextBlock*, який відображає текст «SELECT APPROPRIATE TEST». Його атрибут видимості зв’язаний із властивістю *SelectionOfTestVisibility*, яка описана в класі *WindowViewModel*:

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 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 методів 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 *Command*, який відповідає за натискання на кнопку, зв’язаний із методом *Add* у класі *WindowViewModel*:

private void AddToDictionary()

{

…

}

public DelegateCommand Add { get; set; }

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

Аналогічно до зв’язування з властивостями, потрібно реалізувати спеціальний 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;

}

Як бачимо, у зв’язуванні ми використовуємо цей клас *DelegateCommand*.

Для реалізації зв'язування залишилося лише «підписатися» на метод *AddToDictionary*:

Add = new DelegateCommand(AddToDictionary);

Це слід зробити в конструкторі класу *WindowViewModel*.