Федеральное государственное образовательное бюджетное учреждение высшего образования

«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ

РОССИЙСКОЙ ФЕДЕРАЦИИ»

**Колледж информатики и программирования**

ПРАКТИЧЕСКАЯ РАБОТА №

по МДК.01.01 Разработка программных модулей

по теме:

Выполнил:

студент группы

Проверил:

преподаватель Малинин М.М.

16,2

<Window x:Class="CaesarCipherApp.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="Шифр Цезаря" Height="300" Width="400">

<Grid Margin="10">

<Label Content="Введите текст:" HorizontalAlignment="Left" VerticalAlignment="Top"/>

<TextBox x:Name="InputTextBox" HorizontalAlignment="Stretch" VerticalAlignment="Top" Margin="0,25,0,0"/>

<Button Content="Зашифровать" HorizontalAlignment="Center" VerticalAlignment="Top" Margin="0,60,0,0" Click="EncryptButton\_Click"/>

<Label Content="Зашифрованный текст:" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="0,100,0,0"/>

<TextBox x:Name="OutputTextBox" IsReadOnly="True" HorizontalAlignment="Stretch" VerticalAlignment="Top" Margin="0,125,0,0"/>

</Grid>

</Window>

using System.Windows;

namespace CaesarCipherApp

{

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

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

{

string input = InputTextBox.Text;

string encrypted = Encrypt(input, 3); // Сдвиг на 3 символа

OutputTextBox.Text = encrypted;

}

private string Encrypt(string input, int shift)

{

char[] buffer = input.ToCharArray();

for (int i = 0; i < buffer.Length; i++)

{

char letter = buffer[i];

// Шифрование только для букв

if (char.IsLetter(letter))

{

char offset = char.IsUpper(letter) ? 'A' : 'a';

letter = (char)((((letter + shift) - offset) % 26) + offset);

}

buffer[i] = letter;

}

return new string(buffer);

}

}

}

1

public class Note

{

public string Title { get; set; }

public string Content { get; set; }

}

using System.Collections.ObjectModel;

using System.ComponentModel;

using System.Windows.Input;

public class NotesViewModel : INotifyPropertyChanged

{

public ObservableCollection<Note> Notes { get; set; } = new ObservableCollection<Note>();

private Note \_selectedNote;

public Note SelectedNote

{

get => \_selectedNote;

set

{

\_selectedNote = value;

OnPropertyChanged(nameof(SelectedNote));

OnPropertyChanged(nameof(Title));

OnPropertyChanged(nameof(Content));

}

}

public string Title

{

get => SelectedNote?.Title;

set

{

if (SelectedNote != null)

{

SelectedNote.Title = value;

OnPropertyChanged(nameof(Title));

}

}

}

public string Content

{

get => SelectedNote?.Content;

set

{

if (SelectedNote != null)

{

SelectedNote.Content = value;

OnPropertyChanged(nameof(Content));

}

}

}

public ICommand AddNoteCommand { get; set; }

public ICommand DeleteNoteCommand { get; set; }

public NotesViewModel()

{

AddNoteCommand = new RelayCommand(AddNote);

DeleteNoteCommand = new RelayCommand(DeleteNote, CanDelete);

}

private void AddNote()

{

var newNote = new Note { Title = "Новая заметка", Content = "Содержимое..." };

Notes.Add(newNote);

SelectedNote = newNote; // Автоматически выбираем новую заметку

}

private void DeleteNote()

{

if (SelectedNote != null)

{

Notes.Remove(SelectedNote);

SelectedNote = null; // Сбрасываем выбранную заметку

}

}

private bool CanDelete() => SelectedNote != null;

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged(string propertyName) =>

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

using System;

using System.Windows.Input;

public class RelayCommand : ICommand

{

private readonly Action \_execute;

private readonly Func<bool> \_canExecute;

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

{

\_execute = execute ?? throw new ArgumentNullException(nameof(execute));

\_canExecute = canExecute;

}

public bool CanExecute(object parameter) => \_canExecute == null || \_canExecute();

public void Execute(object parameter) => \_execute();

public event EventHandler CanExecuteChanged

{

add => CommandManager.RequerySuggested += value;

remove => CommandManager.RequerySuggested -= value;

}

}

<Window x:Class="NotesApp.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="Заметки" Height="400" Width="600">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="200"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<ListBox ItemsSource="{Binding Notes}"

SelectedItem="{Binding SelectedNote}"

DisplayMemberPath="Title"

Margin="10"/>

<StackPanel Grid.Column="1" Margin="10">

<TextBox Text="{Binding Title, UpdateSourceTrigger=PropertyChanged}"

PlaceholderText="Заголовок" Margin="0,0,0,10"/>

<TextBox Text="{Binding Content, UpdateSourceTrigger=PropertyChanged}"

PlaceholderText="Содержимое"

AcceptsReturn="True" Height="200" Margin="0,0,0,10"/>

<StackPanel Orientation="Horizontal">

<Button Content="Добавить" Command="{Binding AddNoteCommand}" Margin="0,0,10,0"/>

<Button Content="Удалить" Command="{Binding DeleteNoteCommand}"/>

</StackPanel>

</StackPanel>

</Grid>

</Window>

using System.Windows;

namespace NotesApp

{

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

DataContext = new NotesViewModel();

}

}

}

Москва

2024