Khmelnytskyi National University

Department of Computer Engineering and Information Systems

**Report**

Laboratory work №8

Discipline: “Object-oriented programming”

Topic: “GRAPHICAL USER INTERFACE DEVELOPMENT. APPLICATION

DEPLOYMENT”

Completed: 1st year student, group CEs-24-1 Maksim Lapko

Name, Surname

Checked: Viacheslav Boiko

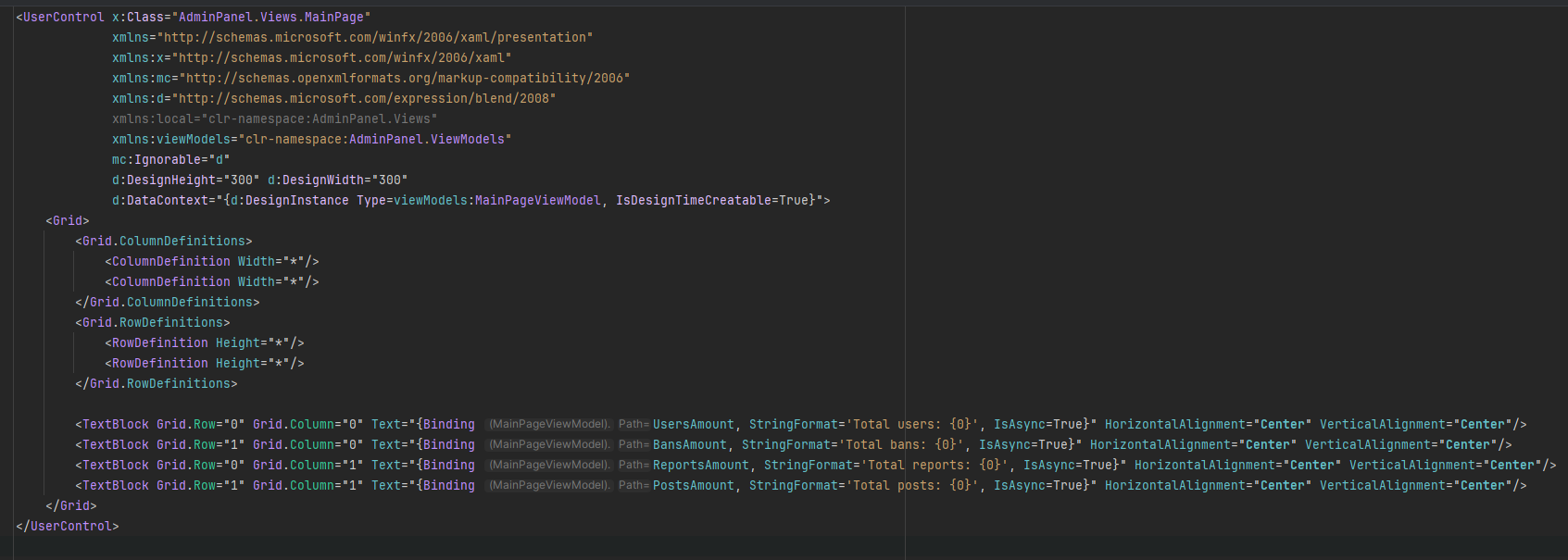
Name, Surname

Khmelnytskyi, 2024

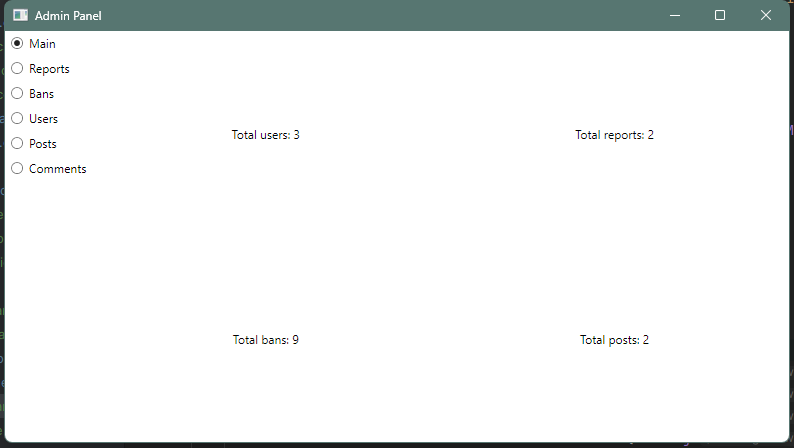
Purpose: Provide hands-on experience in designing, implementing, and refining user-friendly interfaces for software applications, and learn best practices for creating visually appealing and functional GUIs that enhance user interaction. Additionally, the lab focuses on deploying applications, ensuring that students understand the steps and tools required to distribute and maintain software effectively in real-world environments

**Task 1**

Explore your application and show the Event-driven programming within the UI implementation. Paste the code and a screenshot of the UI elements in the report.

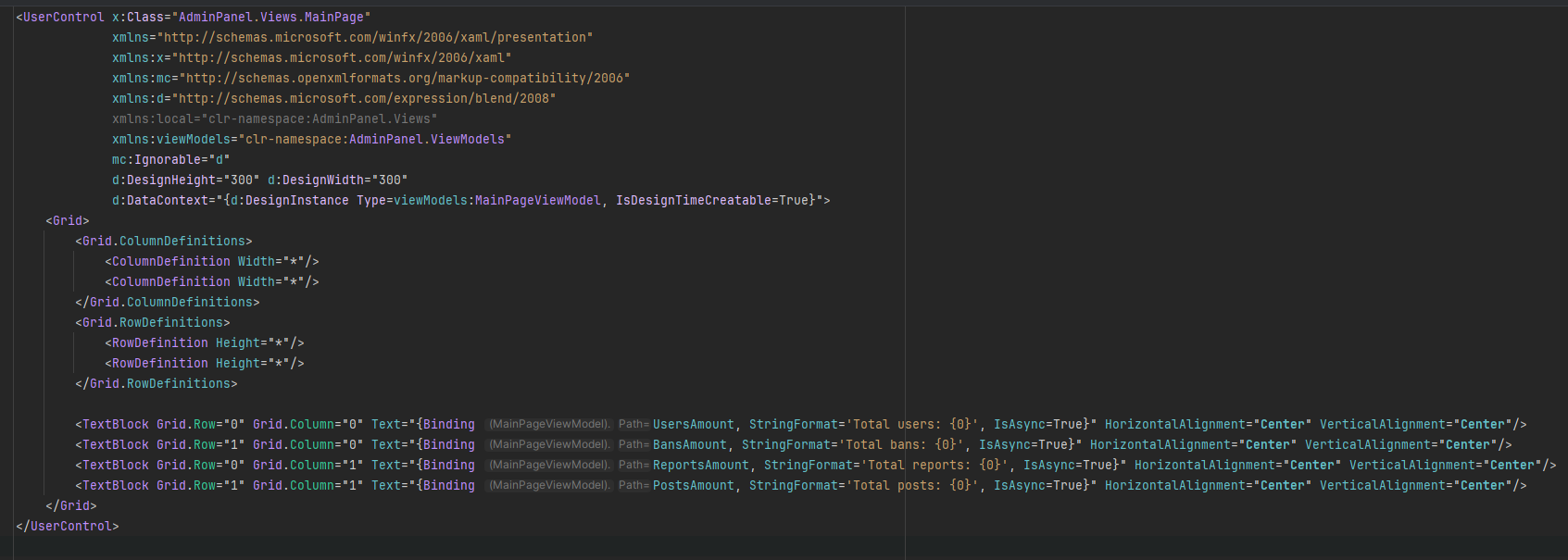


public class MainPageViewModel : ViewModelBase {  
 private readonly UserService \_userService;  
 private readonly BanService \_banService;  
 private readonly ReportService \_reportService;  
 private readonly PostService \_postService;  
  
  
 public MainPageViewModel(UserService userService, BanService banService, ReportService reportService, PostService postService) {  
 \_userService = userService;  
 \_banService = banService;  
 \_postService = postService;  
 \_reportService = reportService;  
 PeriodicUpdateAsync();  
 }  
   
 private async Task PeriodicUpdateAsync() {  
 while (true) {  
 await UpdateData();  
 await Task.Delay(TimeSpan.FromSeconds(5));  
 }  
 }  
   
 private async Task UpdateData() {  
 await Task.Run(async () => {  
 var userStatistic = await \_userService.GetUsersStatisticAsync();  
 if (userStatistic != null) {  
 Application.Current.Dispatcher.Invoke(() => UsersAmount = userStatistic.Total);  
 }  
   
 var banStatistic = await \_banService.GetBansStatisticAsync();  
 if (banStatistic != null) {  
 Application.Current.Dispatcher.Invoke(() => BansAmount = banStatistic.Total);  
 }  
   
 var reportStatistic = await \_reportService.GetReportsStatisticAsync();  
 if (reportStatistic != null) {  
 Application.Current.Dispatcher.Invoke(() => ReportsAmount = reportStatistic.Total);  
 }  
  
 var postStatistic = await \_postService.GetPostsStatisticAsync();  
 if (postStatistic != null) {  
 Application.Current.Dispatcher.Invoke(() => PostsAmount = postStatistic.Total);  
 }  
 });  
 }  
  
 private uint \_usersAmount;  
 public uint UsersAmount {  
 get => \_usersAmount;  
 set {  
 if (\_usersAmount != value) {  
 \_usersAmount = value;  
 OnPropertyChanged(nameof(UsersAmount));  
 }  
 }  
 }  
  
 private uint \_bansAmount;  
 public uint BansAmount {  
 get => \_bansAmount;  
 set {  
 if (\_bansAmount != value) {  
 \_bansAmount = value;  
 OnPropertyChanged(nameof(BansAmount));  
 }  
 }  
 }  
   
 private uint \_postsAmount;  
 public uint PostsAmount {  
 get => \_postsAmount;  
 set {  
 if (\_postsAmount != value) {  
 \_postsAmount = value;  
 OnPropertyChanged(nameof(PostsAmount));  
 }  
 }  
 }  
  
 private uint \_reportsAmount;  
 public uint ReportsAmount {  
 get => \_reportsAmount;  
 set {  
 if (\_reportsAmount != value) {  
 \_reportsAmount = value;  
 OnPropertyChanged(nameof(ReportsAmount));  
 }  
 }  
 }  
  
}



**Task 2**

Find in your application places with frozen UI and fix them using the asynchronous programming. Provide the code in the report.

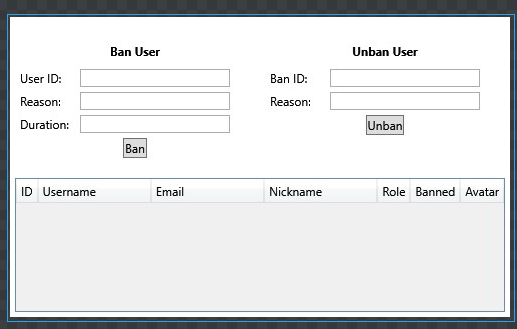


public class MainPageViewModel : ViewModelBase {  
 private readonly UserService \_userService;  
 private readonly BanService \_banService;  
 private readonly ReportService \_reportService;  
 private readonly PostService \_postService;  
  
  
 public MainPageViewModel(UserService userService, BanService banService, ReportService reportService, PostService postService) {  
 \_userService = userService;  
 \_banService = banService;  
 \_postService = postService;  
 \_reportService = reportService;  
 PeriodicUpdateAsync();  
 }  
   
 private async Task PeriodicUpdateAsync() {  
 while (true) {  
 await UpdateData();  
 await Task.Delay(TimeSpan.FromSeconds(5));  
 }  
 }  
   
 private async Task UpdateData() {  
 await Task.Run(async () => {  
 var userStatistic = await \_userService.GetUsersStatisticAsync();  
 if (userStatistic != null) {  
 Application.Current.Dispatcher.Invoke(() => UsersAmount = userStatistic.Total);  
 }  
   
 var banStatistic = await \_banService.GetBansStatisticAsync();  
 if (banStatistic != null) {  
 Application.Current.Dispatcher.Invoke(() => BansAmount = banStatistic.Total);  
 }  
   
 var reportStatistic = await \_reportService.GetReportsStatisticAsync();  
 if (reportStatistic != null) {  
 Application.Current.Dispatcher.Invoke(() => ReportsAmount = reportStatistic.Total);  
 }  
  
 var postStatistic = await \_postService.GetPostsStatisticAsync();  
 if (postStatistic != null) {  
 Application.Current.Dispatcher.Invoke(() => PostsAmount = postStatistic.Total);  
 }  
 });  
 }  
  
 private uint \_usersAmount;  
 public uint UsersAmount {  
 get => \_usersAmount;  
 set {  
 if (\_usersAmount != value) {  
 \_usersAmount = value;  
 OnPropertyChanged(nameof(UsersAmount));  
 }  
 }  
 }  
  
 private uint \_bansAmount;  
 public uint BansAmount {  
 get => \_bansAmount;  
 set {  
 if (\_bansAmount != value) {  
 \_bansAmount = value;  
 OnPropertyChanged(nameof(BansAmount));  
 }  
 }  
 }  
   
 private uint \_postsAmount;  
 public uint PostsAmount {  
 get => \_postsAmount;  
 set {  
 if (\_postsAmount != value) {  
 \_postsAmount = value;  
 OnPropertyChanged(nameof(PostsAmount));  
 }  
 }  
 }  
  
 private uint \_reportsAmount;  
 public uint ReportsAmount {  
 get => \_reportsAmount;  
 set {  
 if (\_reportsAmount != value) {  
 \_reportsAmount = value;  
 OnPropertyChanged(nameof(ReportsAmount));  
 }  
 }  
 }  
  
}

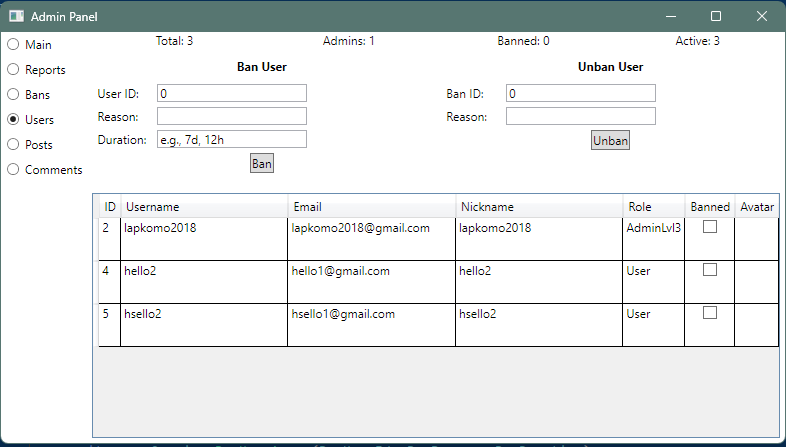
**Task 3**

Implement the two-way binding where it’s needed. Provide the code in the report.

<UserControl x:Class="AdminPanel.Views.UsersPage"  
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  
 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  
 xmlns:local="clr-namespace:AdminPanel.Views"  
 xmlns:viewModels="clr-namespace:AdminPanel.ViewModels"  
 mc:Ignorable="d"  
 d:DesignHeight="300" d:DesignWidth="500"  
 d:DataContext="{d:DesignInstance Type=viewModels:UsersPageViewModel, IsDesignTimeCreatable=True}">  
 <Grid>  
 <Grid.RowDefinitions>  
 <RowDefinition Height="Auto"/>  
 <RowDefinition Height="Auto"/>  
 <RowDefinition Height="\*"/>  
 </Grid.RowDefinitions>  
 <Grid Grid.Row="0">  
 <Grid.ColumnDefinitions>  
 <ColumnDefinition Width="\*"/>  
 <ColumnDefinition Width="\*"/>  
 <ColumnDefinition Width="\*"/>  
 <ColumnDefinition Width="\*"/>  
 </Grid.ColumnDefinitions>  
 <TextBlock Grid.Column="0" HorizontalAlignment="**Center**" VerticalAlignment="**Center**" Text="{Binding UsersStatistic.Total, StringFormat='Total: {0}', IsAsync=True}"/>  
 <TextBlock Grid.Column="1" HorizontalAlignment="**Center**" VerticalAlignment="**Center**" Text="{Binding UsersStatistic.Admin, StringFormat='Admins: {0}', IsAsync=True}"/>  
 <TextBlock Grid.Column="2" HorizontalAlignment="**Center**" VerticalAlignment="**Center**" Text="{Binding UsersStatistic.Banned, StringFormat='Banned: {0}', IsAsync=True}"/>  
 <TextBlock Grid.Column="3" HorizontalAlignment="**Center**" VerticalAlignment="**Center**" Text="{Binding UsersStatistic.Active, StringFormat='Active: {0}', IsAsync=True}"/>  
 </Grid>  
 <Grid Grid.Row="1">  
 <Grid.ColumnDefinitions>  
 <ColumnDefinition Width="\*"/>  
 <ColumnDefinition Width="\*"/>  
 </Grid.ColumnDefinitions>  
 *<!-- Ban Form -->*  
 <StackPanel Grid.Column="0" Margin="10">  
 <TextBlock Text="Ban User" FontWeight="Bold" HorizontalAlignment="**Center**" Margin="0,0,0,10"/>  
 <StackPanel Orientation="**Horizontal**" Margin="0,0,0,5">  
 <TextBlock Text="User ID:" VerticalAlignment="**Center**" Width="60"/>  
 <TextBox Width="150" Text="{Binding BanUserId, UpdateSourceTrigger=**PropertyChanged**}"/>  
 </StackPanel>  
 <StackPanel Orientation="**Horizontal**" Margin="0,0,0,5">  
 <TextBlock Text="Reason:" VerticalAlignment="**Center**" Width="60"/>  
 <TextBox Width="150" Text="{Binding BanReason, UpdateSourceTrigger=**PropertyChanged**}"/>  
 </StackPanel>  
 <StackPanel Orientation="**Horizontal**" Margin="0,0,0,5">  
 <TextBlock Text="Duration:" VerticalAlignment="**Center**" Width="60"/>  
 <TextBox Width="150" Text="{Binding BanDuration, UpdateSourceTrigger=**PropertyChanged**}"/>  
 </StackPanel>  
 <Button Content="Ban" HorizontalAlignment="**Center**" Command="{Binding BanUserCommand}"/>  
 </StackPanel>  
   
 *<!-- Unban Form -->*  
 <StackPanel Grid.Column="1" Margin="10">  
 <TextBlock Text="Unban User" FontWeight="Bold" HorizontalAlignment="**Center**" Margin="0,0,0,10"/>  
 <StackPanel Orientation="**Horizontal**" Margin="0,0,0,5">  
 <TextBlock Text="Ban ID:" VerticalAlignment="**Center**" Width="60"/>  
 <TextBox Width="150" Text="{Binding UnbanBanId, UpdateSourceTrigger=**PropertyChanged**}"/>  
 </StackPanel>  
 <StackPanel Orientation="**Horizontal**" Margin="0,0,0,5">  
 <TextBlock Text="Reason:" VerticalAlignment="**Center**" Width="60"/>  
 <TextBox Width="150" Text="{Binding UnbanReason, UpdateSourceTrigger=**PropertyChanged**}"/>  
 </StackPanel>  
 <Button Content="Unban" HorizontalAlignment="**Center**" Command="{Binding UnbanUserCommand}"/>  
 </StackPanel>  
 </Grid>  
 <DataGrid Grid.Row="2"  
 ItemsSource="{Binding Users}"  
 AutoGenerateColumns="False"  
 IsReadOnly="True"  
 SelectionMode="**Single**"  
 Margin="5 10 5 5">  
 <DataGrid.Columns>  
 <DataGridTextColumn Header="ID" Binding="{Binding Id}" Width="Auto"/>  
 <DataGridTextColumn Header="Username" Binding="{Binding Username}" Width="\*"/>  
 <DataGridTextColumn Header="Email" Binding="{Binding Email}" Width="\*"/>  
 <DataGridTextColumn Header="Nickname" Binding="{Binding Nickname}" Width="\*"/>  
 <DataGridTextColumn Header="Role" Binding="{Binding Role}" Width="Auto"/>  
 <DataGridCheckBoxColumn Header="Banned" Binding="{Binding IsBanned}" Width="Auto"/>  
 <DataGridTemplateColumn Header="Avatar" Width="Auto">  
 <DataGridTemplateColumn.CellTemplate>  
 <DataTemplate>  
 <Image Source="{Binding Avatar}" Width="40" Height="40"/>  
 </DataTemplate>  
 </DataGridTemplateColumn.CellTemplate>  
 </DataGridTemplateColumn>  
 </DataGrid.Columns>  
 </DataGrid>  
 </Grid>  
</UserControl>



public class UsersPageViewModel: ViewModelBase {  
 private readonly UserService \_userService;  
 private const int **UpdateIntervalMilliseconds** = 10000;  
 private readonly CancellationTokenSource \_cancellationTokenSource = new ();  
   
 public UsersPageViewModel(UserService userService) {  
 \_userService = userService;  
 BanUserCommand = new RelayCommand(async () => await BanUserAsync());  
 UnbanUserCommand = new RelayCommand(async () => await UnbanUserAsync());  
   
 Task.Run(BackgroundUpdateAsync, \_cancellationTokenSource.Token);  
 }  
   
 private async Task BackgroundUpdateAsync() {  
 while (!\_cancellationTokenSource.Token.IsCancellationRequested) {  
 try {  
 await UpdateData();  
 await Task.Delay(**UpdateIntervalMilliseconds**);  
 }  
 catch (TaskCanceledException) {  
 Console.WriteLine("Loop canceled by token");  
 }  
 catch (Exception ex) {  
 Console.WriteLine($"Background users update error: {ex.Message}");  
 }   
 }  
 }  
   
 private async Task UpdateData() {  
 var userStatistic = await \_userService.GetUsersStatisticAsync();  
 if (userStatistic != null) {  
 this.UsersStatistic = userStatistic;  
 }  
   
 var users = await \_userService.GetUsersAsync();  
 if (users != null) {  
 this.Users = users;  
 }  
 }  
   
 private UsersStatistic \_usersStatistic = new ();  
 public UsersStatistic UsersStatistic {  
 get => \_usersStatistic;  
 set {  
 \_usersStatistic = value;  
 OnPropertyChanged(nameof(UsersStatistic));  
 }  
 }  
   
 private List<User> \_users = new();  
  
 public List<User> Users {  
 get => \_users;  
 set {  
 \_users = value;  
 OnPropertyChanged(nameof(Users));  
 }  
 }  
   
 private uint \_banUserId;  
 public uint BanUserId {  
 get => \_banUserId;  
 set {  
 \_banUserId = value;  
 OnPropertyChanged(nameof(BanUserId));  
 }  
 }  
  
 private string \_banReason = string.Empty;  
 public string BanReason {  
 get => \_banReason;  
 set {  
 \_banReason = value;  
 OnPropertyChanged(nameof(BanReason));  
 }  
 }  
   
 private string \_banDuration = "e.g., 7d, 12h";  
 public string BanDuration {  
 get => \_banDuration;  
 set {  
 \_banDuration = value;  
 OnPropertyChanged(nameof(BanDuration));  
 }  
 }  
  
 private uint \_unbanBanId;  
 public uint UnbanBanId {  
 get => \_unbanBanId;  
 set {  
 \_unbanBanId = value;  
 OnPropertyChanged(nameof(UnbanBanId));  
 }  
 }  
  
 private string \_unbanReason = string.Empty;  
 public string UnbanReason {  
 get => \_unbanReason;  
 set {  
 \_unbanReason = value;  
 OnPropertyChanged(nameof(UnbanReason));  
 }  
 }  
   
 public ICommand BanUserCommand { get; }  
 public ICommand UnbanUserCommand { get; }  
  
 private async Task BanUserAsync() {  
 if (BanUserId > 0 && !string.IsNullOrWhiteSpace(BanReason) && !string.IsNullOrWhiteSpace(BanDuration) && BanDuration != "e.g., 7d, 12h") {  
 await \_userService.BanUserAsync(BanUserId, BanReason, BanDuration);  
 await UpdateData();  
 }  
 }  
  
 private async Task UnbanUserAsync() {  
 if (UnbanBanId > 0 && !string.IsNullOrWhiteSpace(UnbanReason)) {  
 await \_userService.UnbanUserAsync(UnbanBanId, UnbanReason);  
 await UpdateData();  
 }  
 }  
}



**Task 4**

Find places in your application where unsubscription events are needed. Implement the proper unsubscribe to avoid potential memory leaks. Provide the code in the report.

No need.

**Conclusions**

In completing this laboratory work, I gained practical experience in designing, implementing, and refining user-friendly interfaces for software applications.