**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа бакалавриата «Программная инженерия»

|  |  |  |
| --- | --- | --- |
| СОГЛАСОВАНО  Научный руководитель,  Преподаватель факультета Компьютерных наук Департамента «Программная инженерия»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.К.Чуйкин  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»  профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **Агрегатор магазинов с элементами социальной сети**  **Клиент**  **Текст программы**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.04.01-01 12 01-1-ЛУ** | | |
|  |  | |
| Исполнитель  студент группы БПИ 194  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Е.В.Аникеев /  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. | |
|  | | |
|  | |  |

**Москва 2020**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| УТВЕРЖДЕН  RU.17701729.04.01-01 12 01-1-ЛУ |  | |  | |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **Агрегатор магазинов с элементами социальной сети**  **Клиент**  **Текст программы**  **RU.17701729.04.01-01 12 01-1**  **Листов 95** | | | | |
| Код вида документа в соответствии  с ГОСТ 19.101-77 | |  | | |
|  | | |
|  | | | | |
|  | | | |  |

**Москва 2020**

Содержание

[1 Текст программы 5](#_Toc40139941)

[1.1 Buyer.cs 5](#_Toc40139942)

[1.2 BuyerOrderView.cs 5](#_Toc40139943)

[1.3 BuyerPostView.cs 7](#_Toc40139944)

[1.4 CreatePostForm.cs 8](#_Toc40139945)

[1.5 IconPostForm.cs 9](#_Toc40139946)

[1.6 Post.cs 10](#_Toc40139947)

[1.7 RegistrationForm.cs 11](#_Toc40139948)

[1.8 Seller.cs 12](#_Toc40139949)

[1.10 User.cs 14](#_Toc40139950)

[1.11 AllPostsPage.xaml 15](#_Toc40139951)

[1.12 AllPostsPage.xaml.cs 17](#_Toc40139952)

[1.13 Authentification.xaml 22](#_Toc40139953)

[1.14 Authentification.xaml.cs 23](#_Toc40139954)

[1.15 BuyerMainPage.xaml 29](#_Toc40139955)

[1.16 BuyerMainPage.xaml.cs 31](#_Toc40139956)

[1.17 BuyerOrdersPage.xaml 38](#_Toc40139957)

[1.18 BuyerOrdersPage.xaml.cs 39](#_Toc40139958)

[1.19 CreatePostPage.xaml 43](#_Toc40139959)

[1.20 CreatePostPage.xaml.cs 44](#_Toc40139960)

[1.21 RegistrationPage.xaml 47](#_Toc40139961)

[1.22 RegistrationPage.xaml.cs 48](#_Toc40139962)

[1.23 SearchPage.xaml 55](#_Toc40139963)

[1.24 SearchPage.xaml.cs 56](#_Toc40139964)

[1.26 SellerMainPage.xaml.cs 61](#_Toc40139965)

[1.27 SellerOrdersPage.xaml 67](#_Toc40139966)

[1.29 WatchSellerPage.xaml 73](#_Toc40139967)

[1.30 WatchSellerPage.xaml.cs 75](#_Toc40139968)

[1.31 App.xaml 83](#_Toc40139969)

[1.32 App.xaml.cs 83](#_Toc40139970)

[1.33 ImageResourceExtension.cs 85](#_Toc40139971)

[1.35 MainTabbedPage.xaml.cs 86](#_Toc40139972)

[1.36 MainActivity.cs 88](#_Toc40139973)

[1.37 ApDelegate.cs 89](#_Toc40139974)

[1.38 Main.cs 90](#_Toc40139975)

[2 Лист изменений 91](#_Toc40139976)

1. Текст программы
   1. Buyer.cs
      1. using System;
      2. using System.Collections.Generic;
      3. namespace ShopsAggregator.Models
      4. {
      5. /// <summary>
      6. /// Тип пользователя-покупателя.
      7. /// </summary>
      8. public class Buyer : User
      9. {
      10. /// <summary>
      11. /// Id понравившихся записей в базе данных.
      12. /// </summary>
      13. public List<Int32> LikedPosts { get; set; } = new List<Int32>();
      15. /// <summary>
      16. /// Id пользователей-продавцов, на которых подписан пользователь.
      17. /// </summary>
      18. public List<Int32> Subscribed { get; set; } = new List<Int32>();
      20. /// <summary>
      21. /// Id записей, которые добавли пользователи-продавцы, на которых подписан данный пользователь.
      22. /// </summary>
      23. public List<Int32> NewPosts { get; set; } = new List<Int32>();
      24. /// <summary>
      25. /// Пустой конструктор.
      26. /// </summary>
      27. public Buyer()
      28. {
      30. }
      31. }
      32. }
   2. BuyerOrderView.cs
      1. using System;
      2. namespace ShopsAggregator.Models
      3. {
      4. /// <summary>
      5. /// Модель данных для заказа пользователя-покупателя.
      6. /// </summary>
      7. public class BuyerOrderView
      8. {
      9. /// <summary>
      10. /// Id заказа.
      11. /// </summary>
      12. public Int32 OrderId { get; set; }
      13. /// <summary>
      14. /// Экземпляр пользователя-покупателя, который создал запись.
      15. /// </summary>
      16. public Seller OrderSeller { get; set; }
      17. /// <summary>
      18. /// Запись, которую опубликовал пользователь-продавец.
      19. /// </summary>
      20. public Post SellerPost { get; set; }
      21. /// <summary>
      22. /// Одобрил ли пользователь-продавец заказ.
      23. /// </summary>
      24. public Boolean IsSucceded { get; set; }
      25. /// <summary>
      26. /// Отклонил ли пользователь-продавец заказ.
      27. /// </summary>
      28. public Boolean IsCanceled { get; set; }
      30. /// <summary>
      31. /// Возвращает текст со статусом заказа.
      32. /// </summary>
      33. public String OrderStatus
      34. {
      35. get
      36. {
      37. if (this.IsSucceded)
      38. return "Заказ одобрен";
      39. if (this.IsCanceled)
      40. return "Заказ отклонен";
      41. return "Ожидание";
      42. }
      43. }
      44. }
      45. }
   3. BuyerPostView.cs
      1. using System;
      2. using System.Collections.Generic;
      3. namespace ShopsAggregator.Models
      4. {
      5. /// <summary>
      6. /// Модель данных записи для пользователя-покупателя.
      7. /// </summary>
      8. public class BuyerPostView
      9. {
      10. /// <summary>
      11. /// Id пользователя-покупателя.
      12. /// </summary>
      13. public Int32 BuyerId { get; set; }
      14. /// <summary>
      15. /// Имя пользователя-продавца, который опубликовал запись.
      16. /// </summary>
      17. public String Username { get; set; }
      18. /// <summary>
      19. /// Id записи.
      20. /// </summary>
      21. public Int32 PostId { get; set; }
      22. /// <summary>
      23. /// Информация о записи.
      24. /// </summary>
      25. public String Info { get; set; }
      27. /// <summary>
      28. /// Понравилась ли запись данному пользователю-покупателю.
      29. /// </summary>
      30. public Boolean IsUserLikePost => Likers.Contains(BuyerId);
      32. /// <summary>
      33. /// Пользователи-покупатели, которым понравилась запись.
      34. /// </summary>
      35. public List<Int32> Likers { get; set; } = new List<Int32>();
      36. /// <summary>
      37. /// Ссылка на фотографию к записи.
      38. /// </summary>
      39. public String IconPath => $"{App.BaseUrl}{PostId}photo.jpeg";
      40. /// <summary>
      41. /// Сслыка на иконку пользователя-продавца, опубликовавшего эту запись.
      42. /// </summary>
      43. public String SellerIconPath => $"{App.BaseUrl}{Username}icon.jpeg";
      45. /// <summary>
      46. /// Не ставил ли пользователь-покупатель лайк.
      47. /// </summary>
      48. public Boolean IsUserDontLikePost => !IsUserLikePost;
      50. /// <summary>
      51. /// Конструктор модели данных в зависимости от экземпляра записи.
      52. /// </summary>
      53. /// <param name="post">Экземпляр записи.</param>
      54. public BuyerPostView(Post post)
      55. {
      56. PostId = post.Id;
      57. Info = post.Info;
      58. Likers = post.Likers;
      59. }
      60. /// <summary>
      61. /// Пустой конструктор.
      62. /// </summary>
      63. public BuyerPostView()
      64. {
      66. }
      67. }
      68. }
   4. CreatePostForm.cs
      1. using System;
      2. using System.Collections.Generic;
      3. namespace ShopsAggregator.Models
      4. {
      5. /// <summary>
      6. /// Модель для создания записи, которая отправляется на сервер.
      7. /// </summary>
      8. public class CreatePostForm
      9. {
      10. /// <summary>
      11. /// Id пользователя-продавца, который создает запись.
      12. /// </summary>
      13. public Int32 CreatorId { get; set; }
      14. /// <summary>
      15. /// Информация о записи.
      16. /// </summary>
      17. public String Info { get; set; }
      18. /// <summary>
      19. /// Массив байтов картинки записи.
      20. /// </summary>
      21. public List<Int32> ImageBytes { get; set; } = new List<Int32>();
      23. /// <summary>
      24. /// Создает экземпляр с id пользователя-продавца.
      25. /// </summary>
      26. /// <param name="id">Id пользователя-продавца.</param>
      27. public CreatePostForm(Int32 id)
      28. {
      29. CreatorId = id;
      30. }
      31. }
      32. }
   5. IconPostForm.cs
      1. using System;
      2. using System.Collections.Generic;
      3. namespace ShopsAggregator.Models
      4. {
      5. /// <summary>
      6. /// Экземпляр типа установки иконки пользователю.
      7. /// </summary>
      8. public class IconPostForm
      9. {
      10. /// <summary>
      11. /// Байты фотографии, выбранной пользователем.
      12. /// </summary>
      13. public List<Int32> IconBytesArr { get; set; }
      15. /// <summary>
      16. /// Id пользователя, который изменяет иконку.
      17. /// </summary>
      18. public Int32 ToId { get; set; }
      20. /// <summary>
      21. /// Пустой конструктор.
      22. /// </summary>
      23. public IconPostForm()
      24. {
      26. }
      27. /// <summary>
      28. /// Конструктор, который создает экземпляр с id пользователя.
      29. /// </summary>
      30. /// <param name="toId">Id пользователя</param>
      31. public IconPostForm(Int32 toId)
      32. {
      33. this.ToId = toId;
      34. }
      35. }
      36. }
   6. Post.cs
      1. using System;
      2. using System.Collections.Generic;
      3. namespace ShopsAggregator.Models
      4. {
      5. /// <summary>
      6. /// Тип записи, которую создает пользователь-продавец
      7. /// </summary>
      8. public class Post
      9. {
      10. /// <summary>
      11. /// Строка информации о записи.
      12. /// </summary>
      13. private String \_info;
      14. /// <summary>
      15. /// Id записи.
      16. /// </summary>
      17. public Int32 Id { get; set; }
      19. /// <summary>
      20. /// Id создателя записи - пользователя-продавца.
      21. /// </summary>
      22. public Int32 CreatorId { get; set; }
      23. /// <summary>
      24. /// Устанавливает значение полю \_info и в зависимости от информации в этом поле возрвращает данные о записи.
      25. /// </summary>
      26. public String Info
      27. {
      28. get => String.IsNullOrEmpty(\_info) ? "Пользователь не оставил информацию о товаре" : \_info;
      29. set => \_info = value;
      30. }
      31. /// <summary>
      32. /// Возвращает ссылку на картинку к записи на сервере.
      33. /// </summary>
      34. public String ImagePath => App.BaseUrl + Id + "photo.jpeg";
      35. /// <summary>
      36. /// Id пользователей-покупателей, которым понравилась эта запись.
      37. /// </summary>
      38. public List<Int32> Likers { get; set; } = new List<Int32>();
      39. /// <summary>
      40. /// Количество пользователей-покупателей, которым понравилась эта запись.
      41. /// </summary>
      42. public Int32 LikesCount => Likers.Count;
      43. }
      44. }
   7. RegistrationForm.cs
      1. using System;
      2. namespace ShopsAggregator.Models
      3. {
      4. /// <summary>
      5. /// Форма для регистрации пользователя в сервисе.
      6. /// </summary>
      7. public class RegistrationForm
      8. {
      9. /// <summary>
      10. /// Имя пользователя.
      11. /// </summary>
      12. public String Username { get; set; }
      13. /// <summary>
      14. /// Email пользователя.
      15. /// </summary>
      16. public String Email { get; set; }
      17. /// <summary>
      18. /// Пароль пользователя.
      19. /// </summary>
      20. public String Password { get; set; }
      21. /// <summary>
      22. /// Является ли пользователь пользователем-продавцом.
      23. /// </summary>
      24. public Boolean IsSeller { get; set; }
      25. }
      26. }
   8. Seller.cs
      1. using System;
      2. using System.Collections.Generic;
      3. namespace ShopsAggregator.Models
      4. {
      5. /// <summary>
      6. /// Тип пользователя-продавца.
      7. /// </summary>
      8. public class Seller : User
      9. {
      10. /// <summary>
      11. /// Id записей, созданных пользователем.
      12. /// </summary>
      13. public List<Int32> Items { get; set; } = new List<Int32>();
      15. /// <summary>
      16. /// Id подписчиков, которые подписались на данного пользователя.
      17. /// </summary>
      18. public List<Int32> Subscribers { get; set; } = new List<Int32>();
      20. }
      21. }
   9. SellerOrderView.cs
      1. using System;
      2. namespace ShopsAggregator.Models
      3. {
      4. /// <summary>
      5. /// Модель заказа для пользователя-покупателя.
      6. /// </summary>
      7. public class SellerOrderView
      8. {
      9. /// <summary>
      10. /// Id заказа.
      11. /// </summary>
      12. public Int32 OrderId { get; set; }
      13. /// <summary>
      14. /// Экземпляр типа пользователя, который сделал заказ.
      15. /// </summary>
      16. public Buyer OrderBuyer { get; set; }
      17. /// <summary>
      18. /// Экземпляр типа записи, которую заказал пользователь.
      19. /// </summary>
      20. public Post CurrentPost { get; set; }
      21. /// <summary>
      22. /// Одобрил ли пользователь-продавец заказ.
      23. /// </summary>
      24. public Boolean IsSucceded { get; set; }
      25. /// <summary>
      26. /// Отклонил ли пользователь-продавец заказ.
      27. /// </summary>
      28. public Boolean IsCanceled { get; set; }
      29. /// <summary>
      30. /// Возвращает текст со статусом заказа.
      31. /// </summary>
      32. public String OrderStatus
      33. {
      34. get
      35. {
      36. if (this.IsSucceded)
      37. return "Заказ одобрен";
      38. if (this.IsCanceled)
      39. return "Заказ отклонен";
      40. return "Ожидание";
      41. }
      42. }
      43. }
      44. }
   10. User.cs
       1. using System;
       2. namespace ShopsAggregator.Models
       3. {
       4. /// <summary>
       5. /// Общий тип пользователей.
       6. /// </summary>
       7. public class User
       8. {
       9. /// <summary>
       10. /// Поле значения иконки пользователя.
       11. /// </summary>
       12. private String \_iconPath = "standarticon.jpeg";
       13. /// <summary>
       14. /// Id пользователя.
       15. /// </summary>
       16. public Int32 Id { get; set; }
       17. /// <summary>
       18. /// Имя пользователя.
       19. /// </summary>
       20. public String Username { get; set; }
       21. /// <summary>
       22. /// Email пользователя.
       23. /// </summary>
       24. public String Email { get; set; }
       25. /// <summary>
       26. /// Пароль пользователя.
       27. /// </summary>
       28. public String Password { get; set; }
       29. /// <summary>
       30. /// Информация о пользователе.
       31. /// </summary>
       32. public String Info { get; set; }
       33. /// <summary>
       34. /// Устанавливает имя иконки пользователя на сервер и возвращает ссылку на него.
       35. /// </summary>
       36. public String IconPath { get => $"{App.BaseUrl}{\_iconPath}"; set => \_iconPath = value; }
       38. /// <summary>
       39. /// Пустой конструктор.
       40. /// </summary>
       41. public User()
       42. {
       43. }
       44. }
       45. }
   11. AllPostsPage.xaml
       1. <?xml version="1.0" encoding="utf-8"?>
       2. <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
       3. xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       4. x:Class="ShopsAggregator.Views.AllPostsPage"
       5. xmlns:local="clr-namespace:ShopsAggregator;assembly=ShopsAggregator"
       6. Title="Товары">
       7. <ContentPage.Content>
       8. <StackLayout Margin="0,20,0,0">
       9. <ListView x:Name="Posts" HasUnevenRows="True" ItemTapped="Posts\_OnItemTapped" SeparatorVisibility="None">
       10. <ListView.ItemTemplate>
       11. <DataTemplate>
       12. <ViewCell>
       13. <StackLayout Padding="0,5" Margin="0,0,0,10">
       14. <Label Text="{Binding PostId}" IsVisible="False" />
       15. <FlexLayout JustifyContent="Start" Padding="5" AlignItems="Center">
       16. <FlexLayout.GestureRecognizers>
       17. <TapGestureRecognizer
       18. Tapped="OnPostHeaderTapped"/>
       19. </FlexLayout.GestureRecognizers>
       20. <Frame CornerRadius="15" IsClippedToBounds="True" HorizontalOptions="Center" BackgroundColor="Gray"
       21. WidthRequest="30" HeightRequest="30" VerticalOptions="Start" Padding="0" HasShadow="False">
       22. <Image x:Name="SellerIcon" Source="{Binding SellerIconPath}" VerticalOptions="FillAndExpand"
       23. HorizontalOptions="FillAndExpand" />
       24. </Frame>
       25. <Label Text="{Binding Username}" Padding="10,0"/>
       26. </FlexLayout>
       27. <StackLayout >
       28. <Image x:Name="PostImage" Source="{Binding IconPath}"/>
       29. <FlexLayout JustifyContent="SpaceBetween" Padding="20,0" HeightRequest="50" AlignItems="Center">
       30. <StackLayout Orientation="Horizontal">
       31. <Image Source="{local:ImageResource ShopsAggregator.images.heart.png}"
       32. IsVisible="{Binding IsUserDontLikePost}">
       33. <Image.GestureRecognizers>
       34. <TapGestureRecognizer
       35. Tapped="OnLikeImageTapped"
       36. NumberOfTapsRequired="1" />
       37. </Image.GestureRecognizers>
       38. </Image>
       39. <Image Source="{local:ImageResource ShopsAggregator.images.dislike.png}"
       40. IsVisible="{Binding IsUserLikePost}">
       41. <Image.GestureRecognizers>
       42. <TapGestureRecognizer
       43. Tapped="OnDislikeImageTapped"
       44. NumberOfTapsRequired="1" />
       45. </Image.GestureRecognizers>
       46. </Image>
       47. </StackLayout>
       48. <StackLayout>
       49. <Button x:Name="DeliverButton" Text="+" Margin="0" Padding="0" FontSize="30"
       50. HeightRequest="30" Clicked="DeliverButton\_OnClicked"/>
       51. <Button Text="Заказать" FontSize="13" Padding="0" Margin="0" HeightRequest="13"
       52. Clicked="DeliverButton\_OnClicked"/>
       53. </StackLayout>
       54. </FlexLayout>
       55. <Label Text="{Binding Info}" Padding="20,3"/>
       56. </StackLayout>
       57. </StackLayout>
       58. </ViewCell>
       59. </DataTemplate>
       60. </ListView.ItemTemplate>
       61. </ListView>
       62. </StackLayout>
       63. </ContentPage.Content>
       64. </ContentPage>
   12. AllPostsPage.xaml.cs
       1. using System;
       2. using System.Collections.Generic;
       3. using System.Linq;
       4. using System.Net;
       5. using System.Threading.Tasks;
       6. using Newtonsoft.Json;
       7. using RestSharp;
       8. using ShopsAggregator.Models;
       9. using Xamarin.Forms;
       10. using Xamarin.Forms.Xaml;
       11. namespace ShopsAggregator.Views
       12. {
       13. /// <summary>
       14. /// Код для страницы всех записей.
       15. /// </summary>
       16. [XamlCompilation(XamlCompilationOptions.Compile)]
       17. public partial class AllPostsPage : ContentPage
       18. {
       19. /// <summary>
       20. /// Экземпляр типа пользователя-покупателя.
       21. /// </summary>
       22. private Buyer \_buyer;
       23. /// <summary>
       24. /// Новые записи, доступные пользователю-покупателю.
       25. /// </summary>
       26. private List<BuyerPostView> newPosts = new List<BuyerPostView>();
       28. /// <summary>
       29. /// Конструктор страницы.
       30. /// </summary>
       31. /// <param name="buyer">Экземпляр типа пользователя-покупателя.</param>
       32. public AllPostsPage(Buyer buyer)
       33. {
       34. InitializeComponent();
       35. \_buyer = buyer;
       36. Posts.Refreshing += RefreshViewOnRefreshing;
       37. Posts.IsPullToRefreshEnabled = true;
       38. }
       39. /// <summary>
       40. /// Обновляет данные страницы по запросу пользователя.
       41. /// </summary>
       42. /// <param name="sender">Издатель события - RefreshView.</param>
       43. /// <param name="e">Аргументы события.</param>
       44. private void RefreshViewOnRefreshing(object sender, EventArgs e)
       45. {
       46. GetNewPostsAsync();
       47. }
       48. /// <summary>
       49. /// Получает данные о новых записях и вызывает базовое поведение метода.
       50. /// </summary>
       51. protected override void OnAppearing()
       52. {
       53. base.OnAppearing();
       54. GetNewPostsAsync();
       55. }
       56. /// <summary>
       57. /// Получает новые записи, доступные пользователю.
       58. /// </summary>
       59. private async void GetNewPostsAsync()
       60. {
       61. if (!App.IsConnected())
       62. {
       63. await DisplayAlert("Ошибка", "Осутствует подключение к интернету", "Поробовать снова");
       64. return;
       65. }
       66. Posts.IsRefreshing = true;
       67. RestClient client = new RestClient($"{App.BaseUrl}api/posts/getBuyerNewPost?buyerId={\_buyer.Id}");
       68. var request = new RestRequest(Method.GET);
       69. request.AddHeader("Content-Type", "application/text");
       70. var response = await client.ExecuteAsync(request);
       71. if (response.StatusCode == HttpStatusCode.BadRequest)
       72. {
       73. await DisplayAlert("Ошибка", "Не получилось получить новые записи", "Попробовать снова");
       74. Posts.IsRefreshing = false;
       75. return;
       76. }
       77. List<BuyerPostView> newPosts = new List<BuyerPostView>();
       78. try
       79. {
       80. newPosts = JsonConvert.DeserializeObject<List<BuyerPostView>>(response.Content);
       81. }
       82. catch (Exception e)
       83. {
       84. await DisplayAlert("Ошибка", "Не получилось получить новые записи", "Попробовать снова");
       85. Posts.IsRefreshing = false;
       86. return;
       87. }
       88. foreach (BuyerPostView buyerPostView in newPosts)
       89. {
       90. buyerPostView.BuyerId = \_buyer.Id;
       91. }
       92. this.newPosts = newPosts;
       93. Posts.ItemsSource = this.newPosts;
       94. Posts.IsRefreshing = false;
       95. }
       96. /// <summary>
       97. /// Событие нажатия пользователем на картинку "понравилась запись(лайк)".
       98. /// </summary>
       99. /// <param name="sender">Издатель события - картинка "понравилась запись"</param>
       100. /// <param name="e">Аргументы события.</param>
       101. private void OnLikeImageTapped(object sender, EventArgs e)
       102. {
       103. BuyerPostView post = (BuyerPostView) (((Image) sender).ParentView.BindingContext);
       104. Task.Run(() =>
       105. {
       106. BuyerPostView currentPost =
       107. (from newPost in newPosts where newPost.PostId == post.PostId select newPost).FirstOrDefault();
       108. if (currentPost == null)
       109. return;
       110. currentPost.Likers.Add(\_buyer.Id);
       111. Posts.ItemsSource = newPosts;
       112. });
       113. SendLikeTapped("addLike", post);
       114. }
       115. /// <summary>
       116. /// Событие нажатия пользователем на картинку "убрать лайк".
       117. /// </summary>
       118. /// <param name="sender">Издатель события - картинка "убрать лайк".</param>
       119. /// <param name="e">Аргументы события.</param>
       120. private void OnDislikeImageTapped(object sender, EventArgs e)
       121. {
       122. BuyerPostView post = (BuyerPostView) (((Image) sender).ParentView.BindingContext);
       123. Task.Run(() =>
       124. {
       125. BuyerPostView currentPost =
       126. (from newPost in newPosts where newPost.PostId == post.PostId select newPost).FirstOrDefault();
       127. if (currentPost == null)
       128. return;
       129. currentPost.Likers.Remove(\_buyer.Id);
       130. });
       131. SendLikeTapped("removeLike", post);
       132. }
       134. /// <summary>
       135. /// Отправляет на сервер PUT запрос, чтобы добавить в или убрать картинку из понравившееся у пользователя.
       136. /// </summary>
       137. /// <param name="param">Параметр с Url строки, в зависимости от которого будет добавляться или убираться лайк.</param>
       138. /// <param name="post">Запись, которая понравилась/разонравилась пользователю.</param>
       139. private async void SendLikeTapped(String param, BuyerPostView post)
       140. {
       141. if (!App.IsConnected())
       142. {
       143. await DisplayAlert("Ошибка", "Осутствует подключение к интернету", "Поробовать снова");
       144. return;
       145. }
       146. Posts.ItemsSource = null;
       147. Posts.ItemsSource = newPosts;
       148. RestClient client = new RestClient($"{App.BaseUrl}api/posts/{param}?likerId={\_buyer.Id}&postId={post.PostId}");
       149. var request = new RestRequest(Method.PUT);
       150. request.AddHeader("Content-Type", "application/text");
       151. var response = await client.ExecuteAsync(request);
       152. }
       153. /// <summary>
       154. /// Отменяет событие нажатия на элемент из ListView.
       155. /// </summary>
       156. /// <param name="sender">Издатель события - ListView Posts.</param>
       157. /// <param name="e">Аргументы события.</param>
       158. private void Posts\_OnItemTapped(object sender, ItemTappedEventArgs e)
       159. {
       160. if (sender is ListView listView)
       161. {
       162. listView.SelectedItem = null;
       163. }
       164. }
       165. /// <summary>
       166. /// Обрабатывает событие нажатия на иконку или имя пользователя-продавца, который опубликовал запись.
       167. /// </summary>
       168. /// <param name="sender">Издатель события - FlexLayout.</param>
       169. /// <param name="e">Аргументы события.</param>
       170. private async void OnPostHeaderTapped(object sender, EventArgs e)
       171. {
       172. if (sender is FlexLayout flexLayout)
       173. {
       174. await Navigation.PushAsync(new WatchSellerPage((flexLayout.Children[1] as Label).Text,\_buyer));
       175. }
       176. }
       177. /// <summary>
       178. /// Событие добавления пользователем записи к заказам.
       179. /// </summary>
       180. /// <param name="sender">Издатель события - Button</param>
       181. /// <param name="e">Аргументы события.</param>
       182. private async void DeliverButton\_OnClicked(object sender, EventArgs e)
       183. {
       184. if (!App.IsConnected())
       185. {
       186. await DisplayAlert("Ошибка", "Нет подключения к интернету", "Попробовать снова");
       187. return;
       188. }
       189. if (String.IsNullOrEmpty(\_buyer.Info))
       190. {
       191. await DisplayAlert("Заполните поле информации об аккаунте!",
       192. "Так пользователь продавец будет знать как с вами связаться", "Хорошо");
       193. return;
       194. }
       195. if (sender is Button button)
       196. {
       197. BuyerPostView view = (BuyerPostView) button.ParentView.BindingContext;
       198. if (view == null)
       199. return;
       200. RestClient client = new RestClient($"{App.BaseUrl}api/orders/addOrder?buyerId={\_buyer.Id}&" +
       201. $"postId={view.PostId}");
       202. var request = new RestRequest(Method.POST);
       203. request.AddHeader("Content-Type", "application/text");
       204. var response = await client.ExecuteAsync(request);
       205. }
       206. }
       207. }
       208. }
   13. Authentification.xaml
       1. <?xml version="1.0" encoding="utf-8" ?>
       2. <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
       3. xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       4. xmlns:local="clr-namespace:ShopsAggregator"
       5. xmlns:customControls="clr-namespace:ShopsAggregator.CustomControls;assembly=ShopsAggregator"
       6. x:Class="ShopsAggregator.Views.Authentification">
       7. <ContentPage.Content>
       8. <StackLayout VerticalOptions="CenterAndExpand" HorizontalOptions="CenterAndExpand" Padding="20,0">
       9. <FlexLayout Direction="Column" JustifyContent="SpaceAround">
       10. <StackLayout>
       11. <Label Text="ShopYou" TextColor="{StaticResource purple}" HorizontalTextAlignment="Center" FontSize="44"
       12. Margin="0,0,0,100" FontFamily="SchlangeBold"/>
       14. <customControls:CustomEditor x:Name="username" Placeholder="Enter your buyername or email" Margin="0,0,0,3"/>
       15. <Entry x:Name="password" IsPassword="true" Placeholder="Enter your password" Margin="0,0,0,7"/>
       16. <Button Text="Войти как покупатель" x:Name="buyerSignInButton" Clicked="OnBuyerSignInButtonClicked"
       17. Style="{StaticResource buttonStyle}" CornerRadius="5"/>
       18. <Button Text="Войти как магазин" x:Name="sellerSignInButton" Clicked="OnSellerSignInButtonClicked"
       19. Style="{StaticResource buttonStyle}" CornerRadius="5"/>
       20. <Label Text="или" Margin="0,10" HorizontalTextAlignment="Center" TextColor="{StaticResource purple}"/>
       21. </StackLayout>
       22. <Button Text="Зарегистрироваться" Clicked="OnRegistrationButtonClick"/>
       23. <Label x:Name="StatusOfSignIn" FontSize="Small" TextColor="Red" HorizontalTextAlignment="Center" Text="заглушка" IsVisible="False"/>
       24. </FlexLayout>
       25. </StackLayout>
       26. </ContentPage.Content>
       27. </ContentPage>
   14. Authentification.xaml.cs
       1. using System;
       2. using System.Net;
       3. using Newtonsoft.Json;
       4. using RestSharp;
       5. using ShopsAggregator.Models;
       6. using Xamarin.Forms;
       7. namespace ShopsAggregator.Views
       8. {
       9. /// <summary>
       10. /// Код страницы авторизации пользователя.
       11. /// </summary>
       12. public partial class Authentification : ContentPage
       13. {
       14. /// <summary>
       15. /// Текст сообщения о том, что поле имени пользователя пустое.
       16. /// </summary>
       17. private const String IncorrectUsernameInputErrorMessage = "Имя ползователя не может быть пустым";
       18. /// <summary>
       19. /// Текст сообщения о том, что поля пароля пустое.
       20. /// </summary>
       21. private const String IncorrectPasswordInputErrorMessage = "Необходимо ввести пароль";
       22. /// <summary>
       23. /// Заголовок сообщения о неудачном входе.
       24. /// </summary>
       25. private const String BadSignInTryAlertTitle = "Неудачная попытка входа";
       26. /// <summary>
       27. /// Текст сообщения о неудачном входе.
       28. /// </summary>
       29. private const String BadSignInTryAlertCancelText = "Попробовать снова";
       31. /// <summary>
       32. /// Конструктор страницы.
       33. /// </summary>
       34. public Authentification()
       35. {
       36. InitializeComponent();
       37. NavigationPage.SetHasNavigationBar(this, false);
       38. }
       39. /// <summary>
       40. /// Устанавливает цвет границ кнопок авторизации и вызывает базовое поведение метода.
       41. /// </summary>
       42. protected override void OnAppearing()
       43. {
       44. base.OnAppearing();
       45. buyerSignInButton.BorderColor = sellerSignInButton.BorderColor = Color.FromRgba(12, 12, 12, 1);
       46. }
       47. /// <summary>
       48. /// Событие попытки входа в аккаунт пользователя-покупателя.
       49. /// </summary>
       50. /// <param name="sender">Издатель события - Button buyerSignInButton.</param>
       51. /// <param name="e">Аргументы события.</param>
       52. private void OnBuyerSignInButtonClicked(object sender, EventArgs e)
       53. {
       54. if (!CheckIsBoxesAndConnectionCorrect())
       55. return;
       56. Buyer user = null;
       57. try
       58. {
       59. user = TrySignIn<Buyer>(username.Text, password.Text, "authBuyer");
       60. }
       61. catch (Exception exception)
       62. {
       63. DisplayAlert(BadSignInTryAlertTitle, "Ошибка сервера", BadSignInTryAlertCancelText);
       64. return;
       65. }
       66. if (user == null)
       67. {
       68. DisplayAlert(BadSignInTryAlertTitle, "Проверьте введеные данные", BadSignInTryAlertCancelText);
       69. return;
       70. }
       72. var mainTabbedPage = new MainTabbedPage(user);
       73. NavigationPage.SetHasNavigationBar(mainTabbedPage, true);
       74. NavigationPage.SetHasBackButton(mainTabbedPage, false);
       75. Navigation.PushAsync(mainTabbedPage);
       76. }
       77. /// <summary>
       78. /// Событие попытки входа в аккаунт пользователя-продавца.
       79. /// </summary>
       80. /// <param name="sender">Издатель события - Button sellerSignInButton.</param>
       81. /// <param name="e">Аргументы события.</param>
       82. private void OnSellerSignInButtonClicked(object sender, EventArgs e)
       83. {
       84. if (!CheckIsBoxesAndConnectionCorrect())
       85. return;
       86. Seller user = null;
       87. try
       88. {
       89. user = TrySignIn<Seller>(username.Text, password.Text, "authSeller");
       90. }
       91. catch (Exception)
       92. {
       93. DisplayAlert(BadSignInTryAlertTitle, "Ошибка сервера", BadSignInTryAlertCancelText);
       94. return;
       95. }
       96. if(user == null)
       97. {
       98. DisplayAlert(BadSignInTryAlertTitle, "Проверьте введеные данные", BadSignInTryAlertCancelText);
       99. return;
       100. }
       101. var mainTabbedPage = new MainTabbedPage(user);
       102. NavigationPage.SetHasNavigationBar(mainTabbedPage, true);
       103. NavigationPage.SetHasBackButton(mainTabbedPage, false);
       104. Navigation.PushAsync(mainTabbedPage);
       105. }
       106. /// <summary>
       107. /// Проверяет есть ли соединение с интернетом и все ли поля заполнены верно.
       108. /// </summary>
       109. /// <returns>Результат проверки.</returns>
       110. private Boolean CheckIsBoxesAndConnectionCorrect()
       111. {
       112. if (!App.IsConnected())
       113. {
       114. DisplayAlert("Нет подключения к интернету", "Проверьте подключение к интернету", "Попробовать снова");
       115. return false;
       116. }
       117. if (String.IsNullOrWhiteSpace(username.Text))
       118. {
       119. MessageAboutIncorrectBox(username, IncorrectUsernameInputErrorMessage);
       120. return false;
       121. }
       122. if (String.IsNullOrEmpty(password.Text))
       123. {
       124. MessageAboutIncorrectBox(password, IncorrectPasswordInputErrorMessage);
       125. return false;
       126. }
       127. return true;
       128. }
       129. /// <summary>
       130. /// Отправляет запрос на сервер для входа в аккаунт.
       131. /// </summary>
       132. /// <param name="login">Логин, под которым пользователь хочет войти в аккаунт.</param>
       133. /// <param name="password">Пароль, под которым пользователь хочет войти в аккаунт.</param>
       134. /// <param name="authType">Строка, под каким типом хочет войти пользователь.</param>
       135. /// <typeparam name="T">Ожидаемый тип возврата с сервера.</typeparam>
       136. /// <returns>Результат запроса в виде экземпляра типа Т.</returns>
       137. private T TrySignIn<T>(String login, String password, String authType) where T : class
       138. {
       140. var client = new RestClient($"{App.ServerUrl}{authType}?login={login}&password={password}");
       141. client.Timeout = 10000;
       142. var request = new RestRequest(Method.GET);
       143. IRestResponse response = client.Execute(request);
       144. User user;
       145. if (response.StatusCode == HttpStatusCode.BadRequest)
       146. {
       147. return null;
       148. }
       149. return JsonConvert.DeserializeObject<T>(response.Content);
       150. }
       152. /// <summary>
       153. /// Вызывает анимацию встряски View элемента и выводит сообщение о том что поле заполнено неверно.
       154. /// </summary>
       155. /// <param name="view">View элемент, который заполнен неверно.</param>
       156. /// <param name="errorMsg">Сообщение об ошибки, котороые необходимо вывести.</param>
       157. private void MessageAboutIncorrectBox(Entry view, String errorMsg)
       158. {
       159. App.Shake(view);
       160. StatusOfSignIn.Text = errorMsg;
       161. StatusOfSignIn.IsVisible = true;
       162. view.TextChanged += OnEntryTextChanged;
       163. }
       164. /// <summary>
       165. /// Убирает сообщение о статусе входа при измененнии полей входа.
       166. /// </summary>
       167. /// <param name="sender">Издатель события - Entry.</param>
       168. /// <param name="e">Аргументы события.</param>
       169. private void OnEntryTextChanged(object sender, TextChangedEventArgs e)
       170. {
       171. if (sender is Entry entry)
       172. {
       173. StatusOfSignIn.IsVisible = false;
       174. entry.TextChanged -= OnEntryTextChanged;
       175. }
       176. }
       178. /// <summary>
       179. /// Перенаправляет пользователя на страницу регистрации.
       180. /// </summary>
       181. /// <param name="sender">Издатель события - Button.</param>
       182. /// <param name="e">Аргументы события.</param>
       183. private async void OnRegistrationButtonClick(object sender, EventArgs e)
       184. {
       185. await Navigation.PushAsync(new RegistrationPage());
       186. }
       187. }
       188. }
   15. BuyerMainPage.xaml
       1. <?xml version="1.0" encoding="utf-8"?>
       2. <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
       3. xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       4. xmlns:shopsAggregator="clr-namespace:ShopsAggregator;assembly=ShopsAggregator"
       5. x:Class="ShopsAggregator.Views.BuyerMainPage"
       6. Title="Аккаунт">
       7. <ContentPage.ToolbarItems>
       8. <ToolbarItem Text="Выйти" Clicked="OnExitToolbarItemTapped" />
       9. </ContentPage.ToolbarItems>
       10. <ContentPage.Content>
       11. <RefreshView Refreshing="RefreshView\_OnRefreshing" x:Name="RefreshView">
       12. <ScrollView>
       13. <StackLayout Margin="0,10,0,0" Padding="50,0">
       14. <Frame CornerRadius="100" IsClippedToBounds="True" HorizontalOptions="Center" BackgroundColor="Gray"
       15. WidthRequest="200" HeightRequest="200" VerticalOptions="Start" Padding="0" HasShadow="True" Margin="0,20">
       16. <Frame.GestureRecognizers>
       17. <TapGestureRecognizer Tapped="GetPhoto"/>
       18. </Frame.GestureRecognizers>
       19. <StackLayout HorizontalOptions="Center" VerticalOptions="Center">
       20. <Image x:Name="Icon" HorizontalOptions="FillAndExpand" VerticalOptions="FillAndExpand"
       21. Source="{Binding IconPath}"/>
       22. <Label x:Name="ImagePickerText" Text="Нажмите чтобы выбрать фотографию" TextColor="{StaticResource cloudWhite}"
       23. FontSize="Small" HorizontalTextAlignment="Center" VerticalTextAlignment="Center"/>
       24. </StackLayout>
       25. </Frame>
       26. <StackLayout x:Name="UserInfoStack" HorizontalOptions="FillAndExpand">
       27. <Label Text="{Binding Username}" FontSize="Large" HorizontalTextAlignment="Center" />
       28. <Label Text="{Binding Email}" TextColor="{StaticResource grey}" FontSize="Small" HorizontalTextAlignment="Center"/>
       29. <Editor x:Name="InfoEditor" Text="{Binding Info}" Placeholder="Введите адрес и другую ифнорацию, чтобы продавец смог связаться с вами"
       30. Unfocused="VisualElement\_OnUnfocused"/>
       31. </StackLayout>
       32. <StackLayout HorizontalOptions="Center">
       33. <Label x:Name="SubscribedCounter" Text="0" FontSize="Large" HorizontalTextAlignment="Center"/>
       34. <Label x:Name="SubscribedLabel" Text="Подписан" FontSize="Medium" TextColor="{StaticResource grey}"
       35. HorizontalTextAlignment="Center"/>
       36. </StackLayout>
       37. <Label FontSize="Title" HorizontalTextAlignment="Start">Понравившиеся товары</Label>
       38. <ListView x:Name="Posts" CachingStrategy="RecycleElementAndDataTemplate" HasUnevenRows="True"
       39. SeparatorVisibility="None" ItemTapped="OnPostsItemTapped">
       40. <ListView.ItemTemplate>
       41. <DataTemplate>
       42. <ViewCell>
       43. <StackLayout Padding="0,5" Margin="0,0,0,10">
       44. <Label Text="{Binding PostId}" IsVisible="False" />
       45. <FlexLayout JustifyContent="Start" Padding="5" AlignItems="Center">
       46. <FlexLayout.GestureRecognizers>
       47. <TapGestureRecognizer
       48. Tapped="OnPostHeaderTapped"/>
       49. </FlexLayout.GestureRecognizers>
       50. <Frame CornerRadius="15" IsClippedToBounds="True" HorizontalOptions="Center" BackgroundColor="Gray"
       51. WidthRequest="30" HeightRequest="30" VerticalOptions="Start" Padding="0" HasShadow="False">
       52. <Image x:Name="SellerIcon" Source="{Binding SellerIconPath}" VerticalOptions="FillAndExpand"
       53. HorizontalOptions="FillAndExpand" />
       54. </Frame>
       55. <Label Text="{Binding Username}" Padding="10,0"/>
       56. </FlexLayout>
       57. <StackLayout >
       58. <Image x:Name="PostImage" Source="{Binding IconPath}"/>
       59. <FlexLayout JustifyContent="Start" HeightRequest="50" AlignItems="Center">
       60. <StackLayout Orientation="Horizontal">
       61. <Image Source="{shopsAggregator:ImageResource ShopsAggregator.images.dislike.png}" >
       62. <Image.GestureRecognizers>
       63. <TapGestureRecognizer Tapped="SendDislikeImageTapped"/>
       64. </Image.GestureRecognizers>
       65. </Image>
       66. <Label HorizontalTextAlignment="Center" Text="{Binding LikesCount}" FontSize="Small"/>
       67. </StackLayout>
       68. </FlexLayout>
       69. <Label Text="{Binding Info}" Padding="0,3"/>
       70. </StackLayout>
       71. </StackLayout>
       72. </ViewCell>
       73. </DataTemplate>
       74. </ListView.ItemTemplate>
       75. </ListView>
       76. </StackLayout>
       77. </ScrollView>
       78. </RefreshView>
       79. </ContentPage.Content>
       80. </ContentPage>
   16. BuyerMainPage.xaml.cs
       1. using System;
       2. using System.Collections.Generic;
       3. using System.IO;
       4. using System.Linq;
       5. using System.Net;
       6. using Newtonsoft.Json;
       7. using Plugin.Media;
       8. using Plugin.Media.Abstractions;
       9. using RestSharp;
       10. using ShopsAggregator.Models;
       11. using Xamarin.Forms;
       12. using Xamarin.Forms.Xaml;
       13. namespace ShopsAggregator.Views
       14. {
       15. /// <summary>
       16. /// Код страницы пользователя-покупателя.
       17. /// </summary>
       18. [XamlCompilation(XamlCompilationOptions.Compile)]
       19. public partial class BuyerMainPage : ContentPage
       20. {
       21. /// <summary>
       22. /// Ссылка на иконку пользователя.
       23. /// </summary>
       24. private String iconPath;
       25. /// <summary>
       26. /// Экземпляр типа пользователя-покупателя.
       27. /// </summary>
       28. private Buyer \_buyer;
       29. /// <summary>
       30. /// Заголовок сообщения об ошибке подключения к серверу.
       31. /// </summary>
       32. private const String FailedConnectionToServerAlertTitle = "Ошибка подключения к серверу";
       33. /// <summary>
       34. /// Список записей, которые понравились пользователю.
       35. /// </summary>
       36. private List<BuyerPostView> likedPosts = new List<BuyerPostView>();
       37. /// <summary>
       38. /// Конструктор страницы.
       39. /// </summary>
       40. /// <param name="buyer">Экземпляр типа пользователя-покупателя.</param>
       41. public BuyerMainPage(Buyer buyer)
       42. {
       43. InitializeComponent();
       44. \_buyer = buyer;
       45. this.BindingContext = \_buyer;
       46. }
       47. /// <summary>
       48. /// Устанавливает ширину ListView Posts, получает понравившиеся записи и вызывает базовое поведение метода.
       49. /// </summary>
       50. protected override void OnAppearing()
       51. {
       52. Posts.WidthRequest = App.Current.MainPage.Width;
       53. SubscribedCounter.Text = \_buyer.Subscribed.Count.ToString();
       54. GetBuyerLikedPosts();
       55. base.OnAppearing();
       56. }
       57. /// <summary>
       58. /// Получает с сервера записи, которые понравились пользователю.
       59. /// </summary>
       60. private async void GetBuyerLikedPosts()
       61. {
       62. if (!App.IsConnected())
       63. {
       64. await DisplayAlert("Нет подключения к интернету", "Проверьте подключение к интернету", "Попробовать снова");
       65. return;
       66. }
       67. RefreshView.IsRefreshing = true;
       68. var client = new RestClient($"{App.BaseUrl}api/posts/getBuyerLikedPosts?buyerId={\_buyer.Id}");
       69. client.Timeout = -1;
       70. var request = new RestRequest(Method.GET);
       71. request.AddHeader("Content-Type", "application/text");
       72. IRestResponse response = await client.ExecuteAsync(request);
       73. if (response.StatusCode == HttpStatusCode.BadRequest)
       74. {
       75. await DisplayAlert("Ошибка", "Не получилось удалось понравившиеся записи", "Попробовать снова");
       76. return;
       77. }
       78. try
       79. {
       80. likedPosts = JsonConvert.DeserializeObject<List<BuyerPostView>>(response.Content);
       81. }
       82. catch (Exception e)
       83. {
       84. await DisplayAlert("Ошибка", "Не получилось удалось понравившиеся записи", "Попробовать снова");
       85. return;
       86. }
       87. Posts.ItemsSource = likedPosts;
       88. RefreshView.IsRefreshing = false;
       89. }
       91. /// <summary>
       92. /// Отправляет запрос с обновлением фотографии пользователя.
       93. /// </summary>
       94. /// <param name="form">Форма с информацией о пользователе и его новой фотграфией.</param>
       95. private async void SendUpdateUserPut(IconPostForm form)
       96. {
       97. String json = JsonConvert.SerializeObject(form);
       98. var client = new RestClient($"{App.ServerUrl}addBuyerIcon");
       99. client.Timeout = -1;
       100. var request = new RestRequest(Method.PUT);
       101. request.AddHeader("Content-Type", "application/json");
       102. request.AddParameter("application/json", json, ParameterType.RequestBody);
       103. IRestResponse response = await client.ExecuteAsync(request);
       104. if (response.StatusCode == HttpStatusCode.RequestEntityTooLarge)
       105. {
       106. await DisplayAlert(FailedConnectionToServerAlertTitle, response.Content, "Попробовать снова");
       107. return;
       108. }
       109. if (response.StatusCode == HttpStatusCode.BadRequest)
       110. {
       111. await DisplayAlert(FailedConnectionToServerAlertTitle, response.Content, "Попробовать снова");
       112. return;
       113. }
       114. \_buyer.IconPath = \_buyer.Username + "icon.jpeg";
       115. }
       117. /// <summary>
       118. /// При нажатии пользователя на иконку предоставляет пользователю выбрать фотографию.
       119. /// </summary>
       120. /// <param name="sender">Издатель события - Frame.</param>
       121. /// <param name="e">Аргументы события.</param>
       122. private async void GetPhoto(object sender, EventArgs e)
       123. {
       124. if (CrossMedia.Current.IsPickPhotoSupported)
       125. {
       126. MediaFile photo = await CrossMedia.Current.PickPhotoAsync();
       127. iconPath = photo.Path;
       128. Icon.Source = ImageSource.FromFile(iconPath);
       129. IconPostForm form = new IconPostForm(\_buyer.Id);
       130. GetPhotoBytes(iconPath, form);
       131. if (!App.IsConnected())
       132. {
       133. return;
       134. }
       135. SendUpdateUserPut(form);
       136. }
       137. }
       139. /// <summary>
       140. /// Получает байты фотографии, которую выбрал пользователь.
       141. /// </summary>
       142. /// <param name="path">Путь к файлу с указаной фотографией.</param>
       143. /// <param name="form">Модель, в которую записываются байты фотографии.</param>
       144. private void GetPhotoBytes(String path, IconPostForm form)
       145. {
       146. List<Int32> bytes = new List<Int32>();
       147. using (FileStream fs = new FileStream(path, FileMode.Open))
       148. {
       149. while(fs.Position != fs.Length)
       150. bytes.Add(fs.ReadByte());
       151. }
       152. form.IconBytesArr = bytes.ToList();
       153. }
       154. /// <summary>
       155. /// Обработчик события нажатия на кнопку выхода.
       156. /// </summary>
       157. /// <param name="sender">Издатель события - Button.</param>
       158. /// <param name="e">Аргументы события.</param>
       159. private async void OnExitToolbarItemTapped(object sender, EventArgs e)
       160. {
       161. await Navigation.PopToRootAsync();
       162. }
       164. /// <summary>
       165. /// Обновляет информацию о пользователе.
       166. /// </summary>
       167. /// <param name="sender">Издатель события - RefreshView.</param>
       168. /// <param name="e">Аргументы события.</param>
       169. private void RefreshView\_OnRefreshing(object sender, EventArgs e)
       170. {
       171. this.BindingContext = \_buyer;
       172. GetBuyerLikedPosts();
       173. }
       174. /// <summary>
       175. /// Открывает страницу пользователя, который опубликовал запись.
       176. /// </summary>
       177. /// <param name="sender">Издатель события - FlexLayout.</param>
       178. /// <param name="e">Аргументы события.</param>
       179. private async void OnPostHeaderTapped(object sender, EventArgs e)
       180. {
       181. if (sender is FlexLayout flexLayout)
       182. {
       183. await Navigation.PushAsync(new WatchSellerPage((flexLayout.Children[1] as Label).Text,\_buyer));
       184. }
       185. }
       186. /// <summary>
       187. /// Отправляет запрос на сервер, чтобы убрать запись из понравившегося.
       188. /// </summary>
       189. /// <param name="sender">Издатель события - Image.</param>
       190. /// <param name="e">Аргументы события.</param>
       191. private async void SendDislikeImageTapped(object sender, EventArgs e)
       192. {
       193. if (!App.IsConnected())
       194. {
       195. await DisplayAlert("Нет подключения к интернету", "Проверьте подключение к интернету", "Попробовать снова");
       196. return;
       197. }
       198. if (sender is Image image)
       199. {
       200. BuyerPostView bindingPost = (BuyerPostView) (image.ParentView.BindingContext);
       201. BuyerPostView post =
       202. (from selectionPost in likedPosts
       203. where selectionPost.PostId == bindingPost.PostId
       204. select selectionPost).FirstOrDefault();
       205. if (post == null)
       206. return;
       207. likedPosts.Remove(post);
       208. RestClient client = new RestClient($"{App.BaseUrl}api/posts/removeLike?likerId={\_buyer.Id}&postId={post.PostId}");
       209. var request = new RestRequest(Method.PUT);
       210. request.AddHeader("Content-Type", "application/text");
       211. var response = await client.ExecuteAsync(request);
       212. Posts.ItemsSource = null;
       213. Posts.ItemsSource = likedPosts;
       214. }
       215. }
       216. /// <summary>
       217. /// Отправляет запрос, чтобы изменить информацию о пользователе.
       218. /// </summary>
       219. /// <param name="sender">Издатель события - Editor.</param>
       220. /// <param name="e">Аргументы события.</param>
       221. private async void VisualElement\_OnUnfocused(object sender, FocusEventArgs e)
       222. {
       223. if (!App.IsConnected() || String.IsNullOrWhiteSpace(\_buyer.Info))
       224. {
       225. return;
       226. }
       227. RestClient client = new RestClient($"{App.BaseUrl}api/users/setBuyerInfo?buyerId={\_buyer.Id}&info={\_buyer.Info}");
       228. var request = new RestRequest(Method.PUT);
       229. request.AddHeader("Content-Type", "application/text");
       230. var response = await client.ExecuteAsync(request);
       231. }
       233. /// <summary>
       234. /// Отменяет нажатие на элемент ListView.
       235. /// </summary>
       236. /// <param name="sender">Издатель события - ListView.</param>
       237. /// <param name="e">Аргументы события.</param>
       238. private void OnPostsItemTapped(object sender, ItemTappedEventArgs e)
       239. {
       240. if (sender is ListView listView)
       241. {
       242. listView.SelectedItem = null;
       243. }
       244. }
       245. }
       246. }
   17. BuyerOrdersPage.xaml
       1. <?xml version="1.0" encoding="utf-8"?>
       2. <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
       3. xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       4. x:Class="ShopsAggregator.Views.BuyerOrdersPage"
       5. Title="Мои заказы">
       6. <ContentPage.Content>
       7. <StackLayout>
       8. <ListView x:Name="Orders" SeparatorVisibility="None" IsPullToRefreshEnabled="True" Refreshing="OnOrdersUpdate"
       9. HasUnevenRows="True" Footer="Это все заказы" ItemTapped="OnOrdersItemTapped">
       10. <ListView.ItemTemplate>
       11. <DataTemplate>
       12. <ViewCell>
       13. <StackLayout Padding="0,5" Margin="0,0,0,10">
       14. <Label Text="{Binding OrderId}" IsVisible="False" />
       15. <Label Text="{Binding SellerPost.Postd}" IsVisible="False" />
       16. <FlexLayout JustifyContent="Start" Padding="5" AlignItems="Center">
       17. <FlexLayout.GestureRecognizers>
       18. <TapGestureRecognizer
       19. Tapped="OnPostHeaderTapped"/>
       20. </FlexLayout.GestureRecognizers>
       21. <Frame CornerRadius="15" IsClippedToBounds="True" HorizontalOptions="Center" BackgroundColor="Gray"
       22. WidthRequest="30" HeightRequest="30" VerticalOptions="Start" Padding="0" HasShadow="False">
       23. <Image x:Name="SellerIcon" Source="{Binding OrderSeller.IconPath}" VerticalOptions="FillAndExpand"
       24. HorizontalOptions="FillAndExpand" />
       25. </Frame>
       26. <Label Text="{Binding OrderSeller.Username}" Padding="10,0"/>
       27. </FlexLayout>
       28. <StackLayout >
       29. <Image x:Name="PostImage" Source="{Binding SellerPost.ImagePath}"/>
       30. <Label Text="{Binding SellerPost.Info}" Padding="20,3"/>
       31. </StackLayout>
       32. <StackLayout Orientation="Horizontal" Padding="20,5">
       33. <Label Text="Статус заказа: " FontSize="Header" />
       34. <Label Text="{Binding OrderStatus}" FontSize="Header" TextColor="{StaticResource purple}" />
       35. </StackLayout>
       36. <Button Text="Удалить заказ" Clicked="DeleteButtonClickedAsync" />
       37. </StackLayout>
       38. </ViewCell>
       39. </DataTemplate>
       40. </ListView.ItemTemplate>
       41. </ListView>
       42. </StackLayout>
       43. </ContentPage.Content>
       44. </ContentPage>
   18. BuyerOrdersPage.xaml.cs
       1. using System;
       2. using System.Collections.Generic;
       3. using System.Linq;
       4. using System.Net;
       5. using Newtonsoft.Json;
       6. using RestSharp;
       7. using ShopsAggregator.Models;
       8. using Xamarin.Forms;
       9. using Xamarin.Forms.Xaml;
       10. namespace ShopsAggregator.Views
       11. {
       12. /// <summary>
       13. /// Код страницы заказов пользователя.
       14. /// </summary>
       15. [XamlCompilation(XamlCompilationOptions.Compile)]
       16. public partial class BuyerOrdersPage : ContentPage
       17. {
       18. /// <summary>
       19. /// Экземпляр типа пользователя.
       20. /// </summary>
       21. private Buyer \_buyer;
       22. /// <summary>
       23. /// Список заказов пользователя.
       24. /// </summary>
       25. private List<BuyerOrderView> buyerOrders = new List<BuyerOrderView>();
       26. /// <summary>
       27. /// Конструктор страницы.
       28. /// </summary>
       29. /// <param name="buyer">Экземпляр типа пользователя-покупателя.</param>
       30. public BuyerOrdersPage(Buyer buyer)
       31. {
       32. InitializeComponent();
       33. \_buyer = buyer;
       34. }
       35. /// <summary>
       36. /// Получает заказы пользователя и вызывает базовое поведение метода.
       37. /// </summary>
       38. protected override void OnAppearing()
       39. {
       40. UpdateOrdersAsync();
       41. base.OnAppearing();
       42. }
       43. /// <summary>
       44. /// Открывает страницу пользователя, который опубликовал запись.
       45. /// </summary>
       46. /// <param name="sender">Издатель события - FlexLayout.</param>
       47. /// <param name="e">Аргументы события.</param>
       48. private async void OnPostHeaderTapped(object sender, EventArgs e)
       49. {
       50. if (sender is FlexLayout flexLayout)
       51. {
       52. await Navigation.PushAsync(new WatchSellerPage((flexLayout.Children[1] as Label).Text,\_buyer));
       53. }
       54. }
       55. /// <summary>
       56. /// Событие обновления заказов.
       57. /// </summary>
       58. /// <param name="sender">Издатель события - ListView.</param>
       59. /// <param name="e">Аргументы события.</param>
       60. private void OnOrdersUpdate(object sender, EventArgs e)
       61. {
       62. UpdateOrdersAsync();
       63. }
       64. /// <summary>
       65. /// Отправляет на сервер запрос, чтобы получить заказы пользователя.
       66. /// </summary>
       67. private async void UpdateOrdersAsync()
       68. {
       69. Orders.IsRefreshing = true;
       70. if (!App.IsConnected())
       71. {
       72. await DisplayAlert("Ошибка", "Осутствует подключение к интернету", "Поробовать снова");
       73. return;
       74. }
       75. RestClient client = new RestClient($"{App.BaseUrl}api/orders/getBuyerOrders?buyerId={\_buyer.Id}");
       76. var request = new RestRequest(Method.GET);
       77. request.AddHeader("Content-Type", "application/text");
       78. var response = await client.ExecuteAsync(request);
       79. if (response.StatusCode == HttpStatusCode.BadRequest)
       80. {
       81. await DisplayAlert("Ошибка", "Не удалось получить данные", "Поробовать снова");
       82. return;
       83. }
       84. try
       85. {
       86. buyerOrders = JsonConvert.DeserializeObject<List<BuyerOrderView>>(response.Content);
       87. }
       88. catch (Exception)
       89. {
       90. await DisplayAlert("Ошибка", "Ошибка в обработке данных", "Поробовать снова");
       91. return;
       92. }
       93. Orders.ItemsSource = buyerOrders;
       94. Orders.IsRefreshing = false;
       95. }
       96. /// <summary>
       97. /// Отменяет нажатие на элемент ListView.
       98. /// </summary>
       99. /// <param name="sender">Издатель события - ListView.</param>
       100. /// <param name="e">Аргументы события.</param>
       101. private void OnOrdersItemTapped(object sender, ItemTappedEventArgs e)
       102. {
       103. if (sender is ListView listView)
       104. {
       105. listView.SelectedItem = null;
       106. }
       107. }
       108. /// <summary>
       109. /// Обработчик события удаления заказа.
       110. /// </summary>
       111. /// <param name="sender">Издатель события - Button.</param>
       112. /// <param name="e">Аргументы события.</param>
       113. private async void DeleteButtonClickedAsync(object sender, EventArgs e)
       114. {
       115. if (!App.IsConnected())
       116. {
       117. await DisplayAlert("Ошибка", "Осутствует подключение к интернету", "Поробовать снова");
       118. return;
       119. }
       120. if (sender is Button button)
       121. {
       122. BuyerOrderView order = (BuyerOrderView) (button.ParentView.BindingContext);
       123. if (order == null)
       124. return;
       125. RestClient client = new RestClient($"{App.BaseUrl}api/orders/deleteOrder?orderId={order.OrderId}");
       126. var request = new RestRequest(Method.DELETE);
       127. request.AddHeader("Content-Type", "application/text");
       128. var response = await client.ExecuteAsync(request);
       129. if (response.StatusCode == HttpStatusCode.BadRequest)
       130. {
       131. await DisplayAlert("Ошибка", "Не удалось удалить заказ", "Поробовать снова");
       132. return;
       133. }
       134. BuyerOrderView orderFromList =
       135. (from orderList in buyerOrders where orderList.OrderId == order.OrderId select orderList)
       136. .FirstOrDefault();
       137. buyerOrders.Remove(orderFromList);
       138. }
       139. }
       140. }
       141. }
   19. CreatePostPage.xaml
       1. <?xml version="1.0" encoding="utf-8"?>
       2. <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
       3. xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       4. x:Class="ShopsAggregator.Views.CreatePostPage"
       5. Title="Добавить товар">
       6. <ContentPage.Content>
       7. <ScrollView>
       8. <StackLayout HorizontalOptions="Center">
       9. <Label FontSize="Large" Text="Добавить новый товар" HorizontalTextAlignment="Center" />
       10. <Image x:Name="PostImage" />
       11. <Button x:Name="GetPostPhoto" Text="Выбрать фотографию" Clicked="OnGetPostPhotoClicked" />
       12. <Entry Placeholder="Введите информацию о товаре" Text="{Binding Info}"/>
       13. <Button Text="Добавить" Style="{StaticResource buttonStyle}" Clicked="CreatePostButtonClicked" />
       14. </StackLayout>
       15. </ScrollView>
       16. </ContentPage.Content>
       17. </ContentPage>
   20. CreatePostPage.xaml.cs
       1. using System;
       2. using System.Collections.Generic;
       3. using System.IO;
       4. using System.Net;
       5. using Newtonsoft.Json;
       6. using RestSharp;
       7. using ShopsAggregator.Models;
       8. using ShopsAggregator.Services;
       9. using Xamarin.Forms;
       10. using Xamarin.Forms.Xaml;
       11. namespace ShopsAggregator.Views
       12. {
       13. /// <summary>
       14. /// Код страницы создания записи.
       15. /// </summary>
       16. [XamlCompilation(XamlCompilationOptions.Compile)]
       17. public partial class CreatePostPage : ContentPage
       18. {
       19. /// <summary>
       20. /// Экземпляр типа пользователя-покупателя.
       21. /// </summary>
       22. private Seller seller;
       23. /// <summary>
       24. /// Формы создания записи.
       25. /// </summary>
       26. private CreatePostForm form;
       27. /// <summary>
       28. /// Конструктор страницы.
       29. /// </summary>
       30. /// <param name="seller">Экземпляр типа пользователя-покупателя.</param>
       31. public CreatePostPage(Seller seller)
       32. {
       33. InitializeComponent();
       34. this.seller = seller;
       35. form = new CreatePostForm(seller.Id);
       36. this.BindingContext = form;
       37. }
       38. /// <summary>
       39. /// Позволяет пользователю выбрать фотографию после нажатия на кнопку.
       40. /// </summary>
       41. /// <param name="sender">Издатель события - Button.</param>
       42. /// <param name="e">Аргументы события.</param>
       43. private async void OnGetPostPhotoClicked(object sender, EventArgs e)
       44. {
       45. try
       46. {
       47. String imagePath = String.Empty;
       48. await CrossMedia.Current.Initialize();
       49. if (CrossMedia.Current.IsPickPhotoSupported)
       50. {
       51. MediaFile photo = await CrossMedia.Current.PickPhotoAsync();
       52. imagePath = photo.Path;
       53. PostImage.Source = ImageSource.FromFile(imagePath);
       54. GetImageBytesFromPath(imagePath, form);
       55. if (!App.IsConnected())
       56. {
       57. return;
       58. }
       59. GetPostPhoto.Text = "Изменить фотографию";
       60. }
       61. }
       62. catch (Exception)
       63. {
       64. await DisplayAlert("Ошибка", "Что-то пошло не так", "Попробовать снова");
       65. }
       66. }
       67. /// <summary>
       68. /// Получает байты изображения, выбранного пользователем.
       69. /// </summary>
       70. /// <param name="path">Путь к фотографии.</param>
       71. /// <param name="form">Форма создания записи.</param>
       72. private void GetImageBytesFromPath(String path, CreatePostForm form)
       73. {
       74. List<Int32> bytes = new List<Int32>();
       75. using (FileStream fs = new FileStream(path, FileMode.Open, FileAccess.Read))
       76. {
       77. while(fs.Position != fs.Length)
       78. bytes.Add(fs.ReadByte());
       79. }
       80. form.ImageBytes= bytes.ToList();
       81. }
       82. /// <summary>
       83. /// Обработчик события нажатия на кнопку создания записи.
       84. /// </summary>
       85. /// <param name="sender">Издатель события - Button.</param>
       86. /// <param name="e">Аргументы события.</param>
       87. private async void CreatePostButtonClicked(object sender, EventArgs e)
       88. {
       89. if (form.ImageBytes.Count == 0)
       90. {
       91. await DisplayAlert("Ошибка", "Выберите фотографию", "Выбрать");
       92. return;
       93. }
       94. if (!App.IsConnected())
       95. {
       96. await DisplayAlert("Нет подключения к интернету", "Проверьте подключение к интернету", "Попробовать снова");
       97. return;
       98. }
       100. if (form.CreatorId == 0)
       101. {
       102. await DisplayAlert("Ошибка в данных пользователя", "Зайдите снова, чтобы ее устранить", "Ок");
       103. return;
       104. }
       105. var client = new RestClient(App.BaseUrl+"api/posts/create");
       106. client.Timeout = -1;
       107. var request = new RestRequest(Method.POST);
       108. request.AddHeader("Content-Type", "application/json");
       109. String json = JsonConvert.SerializeObject(form);
       110. request.AddParameter("application/json", json, ParameterType.RequestBody);
       111. IRestResponse response = await client.ExecuteAsync(request);
       112. if (response.StatusCode == HttpStatusCode.BadRequest)
       113. {
       114. await DisplayAlert("Ошибка", "Не удалось добавить запись", "Попробовать снова");
       115. return;
       116. }
       117. await DisplayAlert("Запись успешно добавлена", "", "Хорошо");
       118. }
       119. }
       120. }
   21. RegistrationPage.xaml
       1. <?xml version="1.0" encoding="utf-8"?>
       2. <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
       3. xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       4. x:Class="ShopsAggregator.Views.RegistrationPage">
       5. <ContentPage.Content>
       6. <StackLayout Margin="0,40,0,0" Padding="30,0"
       7. VerticalOptions="FillAndExpand">
       8. <StackLayout x:Name="Layout" VerticalOptions="Center" HorizontalOptions="CenterAndExpand">
       9. <Label FontAttributes="Bold" FontSize="34" Text="Регистрация" Margin="0,0,0,15"/>
       10. <Label FontSize="Medium" Margin="0,10,0,0">Введите свой активный email</Label>
       11. <Entry x:Name="EmailEntry" Keyboard="Email" IsPassword="False"
       12. Placeholder="example@example.com" Text="{Binding Email}"/>
       13. <Label FontSize="Medium" Margin="0,10,0,0">Пользователи будут видеть вас под этим именем</Label>
       14. <Entry x:Name="UsernameEntry" Keyboard="Text" IsPassword="False"
       15. Placeholder="nickname" Text="{Binding Username}"/>
       16. <Label FontSize="Medium" Margin="0,10,0,0">Придумайте надежный пароль</Label>
       17. <Entry x:Name="PasswordEntry" IsPassword="True" Placeholder="########"
       18. Text="{Binding Password}"/>
       19. <Label FontSize="Medium" Margin="0,10,0,0">Введите пароль повторно</Label>
       20. <Entry x:Name="PasswordAgainEntry" IsPassword="True" Placeholder="########" Text=""/>
       21. <Label FontSize="Medium" Margin="0,10,0,0">Хотите быть продавцом?</Label>
       22. <Switch x:Name="IsSeller" IsToggled="{Binding IsSeller}"/>
       23. <StackLayout HorizontalOptions="StartAndExpand" Orientation="Horizontal" Margin="0,10">
       24. <CheckBox x:Name="CheckBox" HorizontalOptions="Start" CheckedChanged="CheckBoxChanged" Color="{StaticResource purple}"/>
       25. <Label x:Name="CheckBoxText" HorizontalOptions="FillAndExpand" Text="Согласен с политикой конфиденциальности и условиями пользования" FontSize="Medium"/>
       26. </StackLayout>
       27. <Button FontSize="Medium" Style="{StaticResource buttonStyle}" Text="Зарегистрироваться" Clicked="OnRegistrationButtonCLicked"/>
       28. <Label x:Name="ErrorRegistration" FontSize="Medium" TextColor="{StaticResource errorColor}" HorizontalTextAlignment="Center" IsVisible="False"/>
       29. </StackLayout>
       30. </StackLayout>
       31. </ContentPage.Content>
       32. </ContentPage>
   22. RegistrationPage.xaml.cs
       1. using System;
       2. using System.Globalization;
       3. using System.Net;
       4. using System.Text.RegularExpressions;
       5. using Newtonsoft.Json;
       6. using RestSharp;
       7. using ShopsAggregator.Models;
       8. using Xamarin.Forms;
       9. using Xamarin.Forms.Xaml;
       10. namespace ShopsAggregator.Views
       11. {
       12. /// <summary>
       13. /// Код страницы регистрации пользователя.
       14. /// </summary>
       15. [XamlCompilation(XamlCompilationOptions.Compile)]
       16. public partial class RegistrationPage : ContentPage
       17. {
       18. /// <summary>
       19. /// Текст сообщения о неверном Email.
       20. /// </summary>
       21. private const String IncorrectEmailMessage = "Email должен иметь вид example@example.com";
       22. /// <summary>
       23. /// Текст сообщения о неверном Username.
       24. /// </summary>
       25. private const String IncorrectUsernameMessage = "Имя пользователя может содержать только латинские символы или цифры";
       26. /// <summary>
       27. /// Текст сообщения о неверном пароле.
       28. /// </summary>
       29. private const String IncorrectPasswordMessage=
       30. "Пароль должен содержать не менее 8 символов и состоять из заглавных и строчных латинских букв и цифр";
       31. /// <summary>
       32. /// Текст сообщения о неверном повторном пароле.
       33. /// </summary>
       34. private const String IncorrectPasswordAgainMessage = "Введеные пароли не совпадают";
       35. /// <summary>
       36. /// Заголовок сообщения о неудачном создании аккаунта.
       37. /// </summary>
       38. private const String BadAccountCreateAlertTitle = "Не удалось создать акканут";
       39. /// <summary>
       40. /// Текст кнопки сообщения о неудачном создании аккаунта.
       41. /// </summary>
       42. private const String BadAccountCreateAlertCancelText = "Попробовать снова";
       43. /// <summary>
       44. /// Текст сообщения о неудачном создании аккаунта.
       45. /// </summary>
       46. private const String BadRequestAlertMessage = "Ошибка подключения к серверу";
       47. /// <summary>
       48. /// Текст кнопки сообщения об удачном создании аккаунта.
       49. /// </summary>
       50. private const String SuccessAccountCreateTitle = "Поздравляю!!";
       51. /// <summary>
       52. /// Текст сообщения об удачном создании аккаунта.
       53. /// </summary>
       54. private const String SuccessAccountCreateMessage = "Аккаунт успешно создан";
       55. /// <summary>
       56. /// Заголовок сообщения об удачном создании аккаунта.
       57. /// </summary>
       58. private const String SuccessAccountCreateCancel = "Ура";
       59. /// <summary>
       60. /// Модель регистрации пользователя.
       61. /// </summary>
       62. private RegistrationForm form = new RegistrationForm();
       63. /// <summary>
       64. /// Конструктор страницы.
       65. /// </summary>
       66. public RegistrationPage()
       67. {
       68. InitializeComponent();
       69. CheckBox.CheckedChanged += (sender, args) => CheckBox.Color = (Color)App.Current.Resources["purple"];
       70. this.BindingContext = form;
       71. }
       72. /// <summary>
       73. /// Проверяет все ли поля заполнены верно и вызывает метод отправки формы на сервер.
       74. /// </summary>
       75. /// <param name="sender">Издатель события - Button.</param>
       76. /// <param name="e">Аргументы события.</param>
       77. private async void OnRegistrationButtonCLicked(object sender, EventArgs e)
       78. {
       79. if (!App.IsConnected())
       80. {
       81. await DisplayAlert("Ошибка", "Осутствует подключение к интернету", "Поробовать снова");
       82. return;
       83. }
       84. if (!IsBoxesCorrect())
       85. {
       86. return;
       87. }
       88. Boolean result = false;
       89. try
       90. {
       91. result = SendRegistrationPost(form);
       92. }
       93. catch (Exception)
       94. {
       95. await DisplayAlert(BadAccountCreateAlertTitle,BadRequestAlertMessage, BadAccountCreateAlertCancelText);
       96. }
       97. if (!result)
       98. return;
       99. await DisplayAlert(SuccessAccountCreateTitle, SuccessAccountCreateMessage, SuccessAccountCreateCancel);
       100. await Navigation.PopAsync();
       101. }
       102. /// <summary>
       103. /// Отправляет на сервер запрос создания аккаунта.
       104. /// </summary>
       105. /// <param name="user">Модель регистрацции пользователя, которая отправляется на сервер.</param>
       106. /// <returns>Результат отправки запроса на сервер.</returns>
       107. private Boolean SendRegistrationPost(RegistrationForm user)
       108. {
       109. String json = JsonConvert.SerializeObject(user);
       110. var client = new RestClient($"{App.ServerUrl}reg");
       111. client.Timeout = -1;
       112. var request = new RestRequest(Method.POST);
       113. request.AddHeader("Content-Type", "application/json");
       114. request.AddParameter("application/json", json, ParameterType.RequestBody);
       115. IRestResponse response = client.Execute(request);
       116. if (response.StatusCode == HttpStatusCode.BadRequest)
       117. {
       118. DisplayAlert(BadAccountCreateAlertTitle, response.Content, BadAccountCreateAlertCancelText);
       119. return false;
       120. }
       121. return true;
       122. }
       123. /// <summary>
       124. /// Изменяет цвет CheckBox и его текста при изменении полей ввода данных.
       125. /// </summary>
       126. /// <param name="sender">Издатель события - Entry.</param>
       127. /// <param name="e">Аргументы события.</param>
       128. private void CheckBoxChanged(object sender, EventArgs e)
       129. {
       130. CheckBox.Color = CheckBoxText.TextColor = Color.Default;
       131. }
       132. /// <summary>
       133. /// Проверяет все ли поля для заполнения заполнены верно.
       134. /// </summary>
       135. /// <returns>Результат проверки.</returns>
       136. private Boolean IsBoxesCorrect()
       137. {
       138. if (!IsTextboxCorrect(EmailEntry, ValidateEmail))
       139. {
       140. MessageAboutIncorrectBox(EmailEntry, IncorrectEmailMessage);
       141. return false;
       142. }
       143. if (!IsTextboxCorrect(UsernameEntry,
       144. (Char symb) => { return symb >= 'A' && symb <= 'z' || Char.IsDigit(symb); }))
       145. {
       146. MessageAboutIncorrectBox(UsernameEntry, IncorrectUsernameMessage);
       147. return false;
       148. }
       149. if (!IsTextboxCorrect(PasswordEntry, ValidatePassword))
       150. {
       151. MessageAboutIncorrectBox(PasswordEntry, IncorrectPasswordMessage);
       152. return false;
       153. }
       154. if (!IsTextboxCorrect(PasswordAgainEntry, (String pwd) => pwd == PasswordEntry.Text))
       155. {
       156. MessageAboutIncorrectBox(PasswordAgainEntry, IncorrectPasswordAgainMessage);
       157. App.Shake(PasswordEntry);
       158. return false;
       159. }
       160. if (!CheckBox.IsChecked)
       161. {
       162. CheckBoxText.TextColor = CheckBox.Color = Color.Red;
       163. App.Shake(CheckBoxText);
       164. App.Shake(CheckBox);
       166. CheckBox.CheckedChanged += (sender, e) => CheckBoxText.TextColor = Color.Default;
       167. return false;
       168. }
       169. return true;
       170. }
       171. /// <summary>
       172. /// Встряхивает неверное поле с данными и выводит соответствующее сообщение.
       173. /// </summary>
       174. /// <param name="view">Невернре поле данных.</param>
       175. /// <param name="errorMsg">Сообщени об ошибке.</param>
       176. private void MessageAboutIncorrectBox(Entry view, String errorMsg)
       177. {
       178. App.Shake(view);
       179. ErrorRegistration.Text = errorMsg;
       180. ErrorRegistration.IsVisible = true;
       181. view.TextChanged += OnEntryTextChanged;
       182. }
       183. /// <summary>
       184. /// Событие об изменении Entry.
       185. /// </summary>
       186. /// <param name="sender">Издатель события - Entry.</param>
       187. /// <param name="e">Аргументы события.</param>
       188. private void OnEntryTextChanged(object sender, TextChangedEventArgs e)
       189. {
       190. if (sender is Entry entry)
       191. {
       192. ErrorRegistration.IsVisible = false;
       193. entry.TextChanged -= OnEntryTextChanged;
       194. }
       195. }
       197. /// <summary>
       198. /// Проверяет введенные данные.
       199. /// </summary>
       200. /// <param name="box">Entry для проверки.</param>
       201. /// <param name="func">Метод проверки данных.</param>
       202. /// <returns>Результат проверки.</returns>
       203. private Boolean IsTextboxCorrect(Entry box, Func<Char, Boolean> func)
       204. {
       205. if (String.IsNullOrWhiteSpace(box.Text))
       206. return false;
       207. Char[] symbols = box.Text.ToCharArray();
       208. return Array.TrueForAll(symbols, symb => func(symb));
       209. }
       211. /// <summary>
       212. /// Проверяет введенные данные.
       213. /// </summary>
       214. /// <param name="box">Entry для проверки.</param>
       215. /// <param name="func">Метод проверки данных.</param>
       216. /// <returns>Результат проверки.</returns>
       217. private Boolean IsTextboxCorrect(Entry box, Func<String, Boolean> func)
       218. {
       219. if (String.IsNullOrWhiteSpace(box.Text))
       220. return false;
       221. return func(box.Text);
       222. }
       223. /// <summary>
       224. /// Проверяет, является ли введеные email верным.
       225. /// </summary>
       226. /// <param name="email">Строка с email.</param>
       227. /// <returns>Результат проверки.</returns>
       228. private Boolean ValidateEmail(String email)
       229. {
       230. if (string.IsNullOrWhiteSpace(email))
       231. return false;
       232. try
       233. {
       234. email = Regex.Replace(email, @"(@)(.+)$", DomainMapper,
       235. RegexOptions.None, TimeSpan.FromMilliseconds(200));
       237. string DomainMapper(Match match)
       238. {
       240. var idn = new IdnMapping();
       242. var domainName = idn.GetAscii(match.Groups[2].Value);
       243. return match.Groups[1].Value + domainName;
       244. }
       245. }
       246. catch (RegexMatchTimeoutException e)
       247. {
       248. return false;
       249. }
       250. catch (ArgumentException e)
       251. {
       252. return false;
       253. }
       254. try
       255. {
       256. return Regex.IsMatch(email,
       257. @"^(?("")("".+?(?<!\\)""@)|(([0-9a-z]((\.(?!\.))|[-!#\$%&'\\*\+/=\?\^`\{\}\|~\w])\*)(?<=[0-9a-z])@))" +
       258. @"(?(\[)(\[(\d{1,3}\.){3}\d{1,3}\])|(([0-9a-z][-0-9a-z]\*[0-9a-z]\*\.)+[a-z0-9][\-a-z0-9]{0,22}[a-z0-9]))$",
       259. RegexOptions.IgnoreCase, TimeSpan.FromMilliseconds(250));
       260. }
       261. catch (RegexMatchTimeoutException)
       262. {
       263. return false;
       264. }
       265. }
       266. /// <summary>
       267. /// Проверяет является ли введеный пароль допустимым.
       268. /// </summary>
       269. /// <param name="input">Введеный пароль.</param>
       270. /// <returns>Результат проверки.</returns>
       271. private Boolean ValidatePassword(String input)
       272. {
       273. var hasNumber = new Regex(@"[0-9]+");
       274. var hasUpperChar = new Regex(@"[A-Z]+");
       275. var hasMinimum8Chars = new Regex(@".{8,}");
       276. return hasNumber.IsMatch(input) && hasUpperChar.IsMatch(input) && hasMinimum8Chars.IsMatch(input);
       277. }
       279. }
       280. }
   23. SearchPage.xaml
       1. <?xml version="1.0" encoding="utf-8"?>
       2. <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
       3. xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       4. x:Class="ShopsAggregator.Views.SearchPage"
       5. Title="Поиск">
       6. <ContentPage.Content>
       7. <StackLayout>
       8. <SearchBar Placeholder="Введите имя пользователя" SearchButtonPressed="OnSearchButtonPressed"/>
       9. <ListView x:Name="UsersListView" ItemSelected="OnSearchItemSelected">
       10. <ListView.ItemTemplate>
       11. <DataTemplate>
       12. <TextCell Text="{Binding .}" />
       13. </DataTemplate>
       14. </ListView.ItemTemplate>
       15. </ListView>
       16. </StackLayout>
       17. </ContentPage.Content>
       18. </ContentPage>
   24. SearchPage.xaml.cs
       1. using System;
       2. using System.Collections.Generic;
       3. using System.Threading.Tasks;
       4. using Newtonsoft.Json;
       5. using RestSharp;
       6. using ShopsAggregator.Models;
       7. using Xamarin.Forms;
       8. using Xamarin.Forms.Xaml;
       9. namespace ShopsAggregator.Views
       10. {
       11. /// <summary>
       12. /// Код страницы поиска пользователей-продавцов.
       13. /// </summary>
       14. [XamlCompilation(XamlCompilationOptions.Compile)]
       15. public partial class SearchPage : ContentPage
       16. {
       17. /// <summary>
       18. /// Заголовок сообщения о неудачном запросе к серверу.
       19. /// </summary>
       20. private const String connectionAlertTitle = "Невозможно сделать запрос";
       21. /// <summary>
       22. /// Текст сообщения об отсутствии интернета.
       23. /// </summary>
       24. private String connectionAlertContent = "Отсутствует подключение к интернету";
       25. /// <summary>
       26. /// Текст кнопки сообщения.
       27. /// </summary>
       28. private const String alertCancel = "Попробовать снова";
       29. /// <summary>
       30. /// Заголовок сообщения о неудачном запросе к серверу.
       31. /// </summary>
       32. private const String getResponseAlertTitle = "Ошибка запроса";
       33. /// <summary>
       34. /// Текст сообщения о неудачном получении данных.
       35. /// </summary>
       36. private const String getResponseAlertContent = "Не удалось получить данные";
       37. /// <summary>
       38. /// Список найденных пользователей.
       39. /// </summary>
       40. private List<String> searchedSellers;
       41. /// <summary>
       42. /// Экземпляр типа пользователя-покупателя.
       43. /// </summary>
       44. private Buyer \_buyer;
       45. /// <summary>
       46. /// Конструктор страницы.
       47. /// </summary>
       48. /// <param name="buyer">Экземпляр типа пользователя-покупателя.</param>
       49. public SearchPage(Buyer buyer)
       50. {
       51. InitializeComponent();
       52. \_buyer = buyer;
       53. }
       54. /// <summary>
       55. /// Обработчик события поиска пользователя по имени.
       56. /// </summary>
       57. /// <param name="sender">Издатель события - SearchBar.</param>
       58. /// <param name="e">Аргументы события.</param>
       59. private async void OnSearchButtonPressed(object sender, EventArgs e)
       60. {
       61. if (!App.IsConnected())
       62. {
       63. await DisplayAlert(connectionAlertTitle, connectionAlertContent, alertCancel);
       64. return;
       65. }
       66. if (sender is SearchBar searchBar)
       67. {
       68. if (String.IsNullOrWhiteSpace(searchBar.Text))
       69. return;
       70. Search(searchBar.Text);
       71. }
       72. }
       73. /// <summary>
       74. /// Вызывает метод с запросом на сервер, получает результат работы сервера и обрабатывает их.
       75. /// </summary>
       76. /// <param name="searchText">Текст запроса пользователя.</param>
       77. private async void Search(String searchText)
       78. {
       79. String responseContent = String.Empty;
       80. try
       81. {
       82. responseContent = await TryGetSearchResult(searchText);
       83. }
       84. catch (Exception)
       85. {
       86. await DisplayAlert(getResponseAlertTitle, getResponseAlertContent, alertCancel);
       87. return;
       88. }
       89. List<String> sellers;
       90. try
       91. {
       92. sellers = JsonConvert.DeserializeObject<List<String>>(responseContent);
       93. }
       94. catch (Exception)
       95. {
       96. await DisplayAlert(getResponseAlertTitle, getResponseAlertContent, alertCancel);
       97. return;
       98. }
       99. searchedSellers = sellers;
       100. UsersListView.ItemsSource = sellers;
       101. }
       102. /// <summary>
       103. /// Отправляет запрос на сервер для поиска пользователя.
       104. /// </summary>
       105. /// <param name="searchLine">Строка поиска пользователя.</param>
       106. /// <returns>Результат запроса.</returns>
       107. private async Task<String> TryGetSearchResult(String searchLine)
       108. {
       109. var client = new RestClient($"{App.BaseUrl}api/search?q={searchLine}");
       110. client.Timeout = -1;
       111. var request = new RestRequest(Method.GET);
       112. IRestResponse response = await client.ExecuteAsync(request);
       113. return response.Content;
       114. }
       115. /// <summary>
       116. /// Обработчик события нажатия на элемент списка найденных пользователей.
       117. /// </summary>
       118. /// <param name="sender">Издатель события - ListView.</param>
       119. /// <param name="e">Аргументы события.</param>
       120. private async void OnSearchItemSelected(object sender, SelectedItemChangedEventArgs e)
       121. {
       122. if (sender is ListView list)
       123. {
       124. if (searchedSellers == null)
       125. return;
       126. Int32 index = e.SelectedItemIndex;
       127. if (index < 0 || index >= searchedSellers.Count)
       128. return;
       129. list.SelectedItem = null;
       130. await Navigation.PushAsync(new WatchSellerPage(searchedSellers[index], \_buyer));
       131. }
       132. }
       133. }
       134. }
   25. SellerMainPage.xaml
       1. <?xml version="1.0" encoding="utf-8"?>
       2. <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
       3. xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       4. x:Class="ShopsAggregator.Views.SellerMainPage"
       5. Title="Аккаунт">
       6. <ContentPage.ToolbarItems>
       7. <ToolbarItem Text="Выйти" Clicked="OnExitToolbarItemTapped" />
       8. </ContentPage.ToolbarItems>
       9. <ContentPage.Content>
       10. <RefreshView x:Name="RefreshView">
       11. <ScrollView x:Name="ScrollView">
       12. <StackLayout Margin="0,10,0,0">
       13. <StackLayout x:Name="InfoStack" Padding="50,0">
       14. <Frame CornerRadius="100" IsClippedToBounds="True" HorizontalOptions="Center" BackgroundColor="Gray"
       15. WidthRequest="200" HeightRequest="200" VerticalOptions="Start" Padding="0" HasShadow="True" Margin="0,20">
       16. <Frame.GestureRecognizers>
       17. <TapGestureRecognizer Tapped="GetPhoto"/>
       18. </Frame.GestureRecognizers>
       19. <StackLayout HorizontalOptions="Center" VerticalOptions="Center">
       20. <Image x:Name="Icon" HorizontalOptions="FillAndExpand" VerticalOptions="FillAndExpand" Aspect="AspectFill"
       21. Source="{Binding IconPath}"/>
       22. <Label x:Name="ImagePickerText" Text="Нажмите чтобы выбрать фотографию" TextColor="{StaticResource cloudWhite}"
       23. FontSize="Small" HorizontalTextAlignment="Center" VerticalTextAlignment="Center"/>
       24. </StackLayout>
       25. </Frame>
       26. <StackLayout x:Name="UserInfoStack">
       27. <Label Text="{Binding Username}" FontSize="Large" HorizontalTextAlignment="Center" />
       28. <Label Text="{Binding Email}" TextColor="{StaticResource grey}" FontSize="Small" HorizontalTextAlignment="Center"/>
       29. <Editor x:Name="InfoEditor" Text="{Binding Info}" Placeholder="Расскажите покупателям о себе"
       30. Unfocused="VisualElement\_OnUnfocused"/>
       31. </StackLayout>
       32. <StackLayout HorizontalOptions="FillAndExpand">
       33. <Label x:Name="SubscribersCounter" FontSize="Large" HorizontalTextAlignment="Center"/>
       34. <Label Text="Подписчики" FontSize="Medium" TextColor="{StaticResource grey}" HorizontalTextAlignment="Center"/>
       35. </StackLayout>
       36. </StackLayout>
       37. <Label FontSize="Title" HorizontalTextAlignment="Center" Padding="0,15">Мои товары</Label>
       38. <ListView x:Name="Posts" CachingStrategy="RecycleElementAndDataTemplate" HasUnevenRows="True"
       39. ItemSelected="Posts\_OnItemSelected" SeparatorVisibility="None">
       40. <ListView.ItemTemplate>
       41. <DataTemplate>
       42. <ViewCell>
       43. <StackLayout x:Name="StackLayout" Margin="0,0,0,10">
       44. <Image x:Name="PostImage" Source="{Binding ImagePath}"/>
       45. <Label Text="{Binding Info}" Padding="30, 0" />
       46. <StackLayout Orientation="Horizontal">
       47. <Label Text="Понравилось: " FontSize="Body"/>
       48. <Label Text="{Binding LikesCount}" FontSize="Body" TextColor="{StaticResource purple}"/>
       49. </StackLayout>
       50. </StackLayout>
       51. </ViewCell>
       52. </DataTemplate>
       53. </ListView.ItemTemplate>
       54. </ListView>
       55. </StackLayout>
       56. </ScrollView>
       57. </RefreshView>
       58. </ContentPage.Content>
       59. </ContentPage>
   26. SellerMainPage.xaml.cs
       1. using System;
       2. using System.Collections.Generic;
       3. using System.IO;
       4. using System.Linq;
       5. using System.Net;
       6. using Newtonsoft.Json;
       7. using Plugin.Media;
       8. using Plugin.Media.Abstractions;
       9. using RestSharp;
       10. using ShopsAggregator.Models;
       11. using Xamarin.Forms;
       12. using Xamarin.Forms.Xaml;
       13. namespace ShopsAggregator.Views
       14. {
       15. /// <summary>
       16. /// Код страницы пользователя-продавца.
       17. /// </summary>
       18. [XamlCompilation(XamlCompilationOptions.Compile)]
       19. public partial class SellerMainPage : ContentPage
       20. {
       21. /// <summary>
       22. /// Заголовок сообщения об отсутствии подключения.
       23. /// </summary>
       24. private const String FailedConnectionToServerAlertTitle = "Ошибка подключения к серверу";
       25. /// <summary>
       26. /// Экземпляр типа пользователя-продавца.
       27. /// </summary>
       28. private Seller \_seller;
       29. /// <summary>
       30. /// Список записей пользователя.
       31. /// </summary>
       32. private List<Post> sellerPosts = new List<Post>();
       33. /// <summary>
       34. /// Конструктор страницы.
       35. /// </summary>
       36. /// <param name="seller">Экземпляр типа пользователя-продавца.</param>
       37. public SellerMainPage(Seller seller)
       38. {
       39. InitializeComponent();
       40. this.\_seller = seller;
       41. this.BindingContext = seller;
       42. RefreshView.Refreshing += PostsOnRefreshing;
       43. Posts.WidthRequest = App.Current.MainPage.Width;
       44. }
       46. /// <summary>
       47. /// Обработчик события обновления данных.
       48. /// </summary>
       49. /// <param name="sender">Издатель события - RefreshView.</param>
       50. /// <param name="e">Аргументы события.</param>
       51. private void PostsOnRefreshing(object sender, EventArgs e)
       52. {
       53. GetSellerPosts(\_seller.Username);
       54. }
       55. /// <summary>
       56. /// Получает записи пользователя и вызывает базовое поведение метода.
       57. /// </summary>
       58. protected override void OnAppearing()
       59. {
       60. SubscribersCounter.Text = \_seller.Subscribers.Count.ToString();
       61. GetSellerPosts(\_seller.Username);
       62. base.OnAppearing();
       63. }
       65. /// <summary>
       66. /// Получает записи пользователя с сервера.
       67. /// </summary>
       68. /// <param name="sellerName">Имя пользователя.</param>
       69. private async void GetSellerPosts(String sellerName)
       70. {
       71. if (!App.IsConnected())
       72. {
       73. await DisplayAlert("Нет подключения к интернету", "Проверьте подключение к интернету", "Попробовать снова");
       74. return;
       75. }
       76. RefreshView.IsRefreshing = true;
       77. var client = new RestClient($"{App.BaseUrl}api/posts/getSellerPosts?sellerName={sellerName}");
       78. client.Timeout = -1;
       79. var request = new RestRequest(Method.GET);
       80. request.AddHeader("Content-Type", "application/text");
       81. IRestResponse response = await client.ExecuteAsync(request);
       82. if (response.StatusCode == HttpStatusCode.BadRequest)
       83. {
       84. return;
       85. }
       86. try
       87. {
       88. sellerPosts = JsonConvert.DeserializeObject<List<Post>>(response.Content);
       89. }
       90. catch(Exception e){}
       91. Posts.ItemsSource = sellerPosts;
       92. RefreshView.IsRefreshing = false;
       93. }
       94. /// <summary>
       95. /// Обработчик нажатия на кнопку выхода.
       96. /// </summary>
       97. /// <param name="sender">Издатель события - Button.</param>
       98. /// <param name="e">Аргументы события.</param>
       99. private void OnExitToolbarItemTapped(object sender, EventArgs e)
       100. {
       101. Navigation.PopAsync();
       102. }
       103. /// <summary>
       104. /// Отправляет запрос на сервер для обновления фотографии пользователя.
       105. /// </summary>
       106. /// <param name="form">Модель формы с информацией о фотографии.</param>
       107. private async void SendUpdateUserPut(IconPostForm form)
       108. {
       109. if (!App.IsConnected())
       110. {
       111. return;
       112. }
       113. String json = JsonConvert.SerializeObject(form);
       114. var client = new RestClient($"{App.ServerUrl}addSellerIcon");
       115. client.Timeout = -1;
       116. var request = new RestRequest(Method.PUT);
       117. request.AddHeader("Content-Type", "application/json");
       118. request.AddParameter("application/json", json, ParameterType.RequestBody);
       119. IRestResponse response = await client.ExecuteAsync(request);
       120. if (response.StatusCode == HttpStatusCode.RequestEntityTooLarge)
       121. {
       122. await DisplayAlert(FailedConnectionToServerAlertTitle, response.Content, "Попробовать снова");
       123. return;
       124. }
       125. if (response.StatusCode == HttpStatusCode.BadRequest)
       126. {
       127. await DisplayAlert(FailedConnectionToServerAlertTitle, response.Content, "Попробовать снова");
       128. return;
       129. }
       130. \_seller.IconPath = \_seller.Username + "icon.jpeg";
       131. }
       133. /// <summary>
       134. /// Обработчик события получения пользователем фотографии.
       135. /// </summary>
       136. /// <param name="sender">Издатель события - Frame.</param>
       137. /// <param name="e">Аргументы события.</param>
       138. private async void GetPhoto(object sender, EventArgs e)
       139. {
       140. String iconPath = String.Empty;
       141. await CrossMedia.Current.Initialize();
       142. if (CrossMedia.Current.IsPickPhotoSupported && CrossMedia.IsSupported)
       143. {
       144. MediaFile photo = await CrossMedia.Current.PickPhotoAsync();
       145. if (photo == null)
       146. return;
       147. iconPath = photo.Path;
       148. Icon.Source = ImageSource.FromStream(() => photo.GetStream());
       149. IconPostForm form = new IconPostForm(\_seller.Id);
       150. GetPhotoBytes(iconPath, form);
       151. if (!App.IsConnected())
       152. {
       153. return;
       154. }
       155. SendUpdateUserPut(form);
       156. } }
       158. /// <summary>
       159. /// Получает байты фотографии.
       160. /// </summary>
       161. /// <param name="path">Путь к файлу с фотографией.</param>
       162. /// <param name="form">Форма для запроса, в которую записываются байты фотографии.</param>
       163. private void GetPhotoBytes(String path, IconPostForm form)
       164. {
       165. List<Int32> bytes = new List<Int32>();
       166. using (FileStream fs = new FileStream(path, FileMode.Open))
       167. {
       168. while(fs.Position != fs.Length)
       169. bytes.Add(fs.ReadByte());
       170. }
       171. form.IconBytesArr = bytes.ToList();
       172. }
       173. /// <summary>
       174. /// Отменяет событие нажатия на элемент из ListView.
       175. /// </summary>
       176. /// <param name="sender">Издатель события - ListView.</param>
       177. /// <param name="e">Аргументы события.</param>
       178. private void Posts\_OnItemSelected(object sender, SelectedItemChangedEventArgs e)
       179. {
       180. if (sender is ListView listView)
       181. {
       182. listView.SelectedItem = null;
       183. }
       184. }
       186. /// <summary>
       187. /// Отправляет запрос, чтобы изменить информацию о пользователе.
       188. /// </summary>
       189. /// <param name="sender">Издатель события - Editor.</param>
       190. /// <param name="e">Аргументы события.</param>
       191. private async void VisualElement\_OnUnfocused(object sender, FocusEventArgs e)
       192. {
       193. if (!App.IsConnected() || String.IsNullOrWhiteSpace(\_seller.Info))
       194. {
       195. return;
       196. }
       197. RestClient client = new RestClient($"{App.BaseUrl}api/users/setSellerInfo?sellerId={\_seller.Id}&info={\_seller.Info}");
       198. var request = new RestRequest(Method.PUT);
       199. request.AddHeader("Content-Type", "application/json");
       200. var response = await client.ExecuteAsync(request);
       201. }
       202. }
       203. }
   27. SellerOrdersPage.xaml
       1. <?xml version="1.0" encoding="utf-8"?>
       2. <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
       3. xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       4. x:Class="ShopsAggregator.Views.SellerOrdersPage"
       5. Title="Заказы пользователей">
       6. <ContentPage.Content>
       7. <StackLayout>
       8. <ListView x:Name="Orders" SeparatorVisibility="None" IsPullToRefreshEnabled="True" Refreshing="OnOrdersUpdate"
       9. HasUnevenRows="True" ItemTapped="OnOrdersItemTapped">
       10. <ListView.ItemTemplate>
       11. <DataTemplate>
       12. <ViewCell>
       13. <StackLayout Padding="0,5" Margin="0,0,0,10">
       14. <Label Text="{Binding OrderId}" IsVisible="False" />
       15. <StackLayout Margin="0,0,0,10">
       16. <Image x:Name="PostImage" Source="{Binding CurrentPost.ImagePath}"/>
       17. <Label Text="{Binding CurrentPost.Info}" Padding="20,3"/>
       18. </StackLayout>
       19. <StackLayout Orientation="Horizontal" Margin="0,0,0,10" Padding="20,5">
       20. <Label Text="Заказчик:" FontSize="Header"/>
       21. <StackLayout>
       22. <StackLayout Orientation="Horizontal">
       23. <Label Text="Имя пользователя: " TextColor="{StaticResource purple}" FontSize="Body"/>
       24. <Label Text="{Binding OrderBuyer.Username}" FontSize="Body" />
       25. </StackLayout>
       26. <StackLayout Orientation="Horizontal">
       27. <Label Text="Email: " TextColor="{StaticResource purple}" FontSize="Body"/>
       28. <Label Text="{Binding OrderBuyer.Email}" FontSize="Body" />
       29. </StackLayout>
       30. <StackLayout Orientation="Horizontal">
       31. <Label Text="Информация о пользователе: " TextColor="{StaticResource purple}" FontSize="Body"/>
       32. <Label Text="{Binding OrderBuyer.Info}" FontSize="Body" />
       33. </StackLayout>
       34. </StackLayout>
       35. </StackLayout>
       36. <StackLayout Orientation="Horizontal" Padding="20,5">
       37. <Label Text="Статус заказа: " FontSize="Header" />
       38. <Label Text="{Binding OrderStatus}" FontSize="Header" TextColor="{StaticResource purple}" />
       39. </StackLayout>
       40. <FlexLayout JustifyContent="SpaceAround">
       41. <Button BackgroundColor="{StaticResource successColor}" TextColor="{StaticResource cloudWhite}"
       42. Text="Подтвердить заказ" Padding="10,0" Clicked="SetOrderSuccess"/>
       43. <Button BackgroundColor="{StaticResource errorColor}" Text="Отклонить заказ"
       44. TextColor="{StaticResource cloudWhite}" Padding="10,0" Clicked="SetOrderCanceledStatus"/>
       45. </FlexLayout>
       46. </StackLayout>
       47. </ViewCell>
       48. </DataTemplate>
       49. </ListView.ItemTemplate>
       50. </ListView>
       51. </StackLayout>
       52. </ContentPage.Content>
       53. </ContentPage>
   28. SellerOrdersPage.xaml.cs
       1. using System;
       2. using System.Collections.Generic;
       3. using System.Net;
       4. using Newtonsoft.Json;
       5. using RestSharp;
       6. using ShopsAggregator.Models;
       7. using Xamarin.Forms;
       8. using Xamarin.Forms.Xaml;
       9. namespace ShopsAggregator.Views
       10. {
       11. /// <summary>
       12. /// Код страницы заказы пользователя-продавца.
       13. /// </summary>
       14. [XamlCompilation(XamlCompilationOptions.Compile)]
       15. public partial class SellerOrdersPage : ContentPage
       16. {
       17. /// <summary>
       18. /// Экземпляр типа пользователя-продавца.
       19. /// </summary>
       20. private readonly Seller \_seller;
       21. /// <summary>
       22. /// Список заказов пользователя-продавца.
       23. /// </summary>
       24. private List<SellerOrderView> sellerOrders = new List<SellerOrderView>();
       25. /// <summary>
       26. /// Конструктор страницы.
       27. /// </summary>
       28. /// <param name="seller">Экземпляр типа пользователя-продавца.</param>
       29. public SellerOrdersPage(Seller seller)
       30. {
       31. \_seller = seller;
       32. InitializeComponent();
       33. }
       35. /// <summary>
       36. /// Вызывает метод получения заказов пользователя и вызывает базовое поведение метода.
       37. /// </summary>
       38. protected override void OnAppearing()
       39. {
       40. UpdateOrdersAsync();
       41. base.OnAppearing();
       42. }
       44. /// <summary>
       45. /// Обработчик события обновления заказов.
       46. /// </summary>
       47. /// <param name="sender">Издатель события - ListView.</param>
       48. /// <param name="e">Аргументы события.</param>
       49. private void OnOrdersUpdate(object sender, EventArgs e)
       50. {
       51. UpdateOrdersAsync();
       52. }
       53. /// <summary>
       54. /// Обновляет заказы пользователя.
       55. /// </summary>
       56. private async void UpdateOrdersAsync()
       57. {
       58. Orders.IsRefreshing = true;
       59. if (!App.IsConnected())
       60. {
       61. await DisplayAlert("Ошибка", "Осутствует подключение к интернету", "Поробовать снова");
       62. return;
       63. }
       64. RestClient client = new RestClient($"{App.BaseUrl}api/orders/getSellerOrders?sellerId={\_seller.Id}");
       65. var request = new RestRequest(Method.GET);
       66. request.AddHeader("Content-Type", "application/text");
       67. var response = await client.ExecuteAsync(request);
       68. if (response.StatusCode == HttpStatusCode.BadRequest)
       69. {
       70. await DisplayAlert("Ошибка", "Не удалось получить данные", "Поробовать снова");
       71. return;
       72. }
       73. try
       74. {
       75. sellerOrders = JsonConvert.DeserializeObject<List<SellerOrderView>>(response.Content);
       76. }
       77. catch (Exception)
       78. {
       79. await DisplayAlert("Ошибка", "Ошибка в обработке данных", "Поробовать снова");
       80. return;
       81. }
       82. Orders.ItemsSource = sellerOrders;
       83. Orders.IsRefreshing = false;
       84. }
       85. /// <summary>
       86. /// Отменяет событие нажатия на элемент из ListView.
       87. /// </summary>
       88. /// <param name="sender">Издатель события - ListView.</param>
       89. /// <param name="e">Аргументы события.</param>
       90. private void OnOrdersItemTapped(object sender, ItemTappedEventArgs e)
       91. {
       92. if (sender is ListView listView)
       93. {
       94. listView.SelectedItem = null;
       95. }
       96. }
       98. /// <summary>
       99. /// Ставит статус заказа одобреным и отправляет запрос на сервер.
       100. /// </summary>
       101. /// <param name="sender">Издатель события - Button.</param>
       102. /// <param name="e">Аргументы события.</param>
       103. private void SetOrderSuccess(object sender, EventArgs e)
       104. {
       105. if (sender is Button button)
       106. {
       107. Int32 orderId = ((SellerOrderView) (button.ParentView.BindingContext)).OrderId;
       108. SendOrderStatusAsync("setOrderSuccess", orderId);
       109. foreach (SellerOrderView sellerOrderView in sellerOrders)
       110. {
       111. if (sellerOrderView.OrderId == orderId)
       112. sellerOrderView.IsSucceded = true;
       113. }
       114. }
       115. }
       116. /// <summary>
       117. /// Ставит статус заказа отклоненым и отправляет запрос на сервер.
       118. /// </summary>
       119. /// <param name="sender">Издатель события - Button.</param>
       120. /// <param name="e">Аргументы события.</param>
       121. private void SetOrderCanceledStatus(object sender, EventArgs e)
       122. {
       123. if (sender is Button button)
       124. {
       125. Int32 orderId = ((SellerOrderView) (button.ParentView.BindingContext)).OrderId;
       126. SendOrderStatusAsync("setOrderCanceled", orderId);
       127. foreach (SellerOrderView sellerOrderView in sellerOrders)
       128. {
       129. if (sellerOrderView.OrderId == orderId)
       130. sellerOrderView.IsCanceled = true;
       131. }
       132. }
       133. }
       134. /// <summary>
       135. /// Отправляет запрос на сервер чтобы обновить статус заказа.
       136. /// </summary>
       137. /// <param name="reqStatus">Строка запроса на сервер.</param>
       138. /// <param name="orderId">Id заказа.</param>
       139. private async void SendOrderStatusAsync(String reqStatus, Int32 orderId)
       140. {
       141. Orders.IsRefreshing = true;
       142. if (!App.IsConnected())
       143. {
       144. await DisplayAlert("Ошибка", "Осутствует подключение к интернету", "Поробовать снова");
       145. return;
       146. }
       147. RestClient client = new RestClient($"{App.BaseUrl}api/orders/{reqStatus}?orderId={orderId}");
       148. var request = new RestRequest(Method.PUT);
       149. request.AddHeader("Content-Type", "application/text");
       150. var response = await client.ExecuteAsync(request);
       151. if (response.StatusCode == HttpStatusCode.BadRequest)
       152. {
       153. await DisplayAlert("Ошибка", "Не удалось изменить статус заказа", "Поробовать снова");
       154. return;
       155. }
       156. Orders.ItemsSource = sellerOrders;
       157. Orders.IsRefreshing = false;
       158. }
       159. }
       160. }
   29. WatchSellerPage.xaml
       1. <?xml version="1.0" encoding="utf-8"?>
       2. <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
       3. xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       4. xmlns:local="clr-namespace:ShopsAggregator;assembly=ShopsAggregator"
       5. x:Class="ShopsAggregator.Views.WatchSellerPage">
       6. <ContentPage.Content>
       7. <StackLayout>
       8. <RefreshView x:Name="RefreshView">
       9. <ScrollView>
       10. <StackLayout Margin="0,10,0,0">
       11. <StackLayout x:Name="InfoStack" Padding="50,0">
       12. <Frame CornerRadius="100" IsClippedToBounds="True" HorizontalOptions="Center" BackgroundColor="Gray"
       13. WidthRequest="200" HeightRequest="200" VerticalOptions="Start" Padding="0" HasShadow="True" Margin="0,20">
       14. <Image x:Name="Icon" HorizontalOptions="FillAndExpand" VerticalOptions="FillAndExpand" Aspect="AspectFill"
       15. Source="{Binding IconPath}"/>
       16. </Frame>
       17. <StackLayout x:Name="UserInfoStack" HorizontalOptions="FillAndExpand">
       18. <Label Text="{Binding Username}" FontSize="Large" HorizontalTextAlignment="Center" />
       19. <Label Text="{Binding Email}" TextColor="{StaticResource grey}" FontSize="Small" HorizontalTextAlignment="Center"/>
       20. <Label Text="{Binding Info}" />
       21. <Button x:Name="SubscribeSatusButton" HorizontalOptions="Center" Style="{StaticResource buttonStyle}"
       22. Padding="30,5"/>
       23. </StackLayout>
       24. <Label FontSize="Title" HorizontalTextAlignment="Center" Padding="0,15">Товары пользователя</Label>
       25. <ListView x:Name="Posts" CachingStrategy="RecycleElementAndDataTemplate" HasUnevenRows="True"
       26. ItemSelected="Posts\_OnItemSelected" SeparatorVisibility="None">
       27. <ListView.ItemTemplate>
       28. <DataTemplate>
       29. <ViewCell>
       30. <StackLayout Margin="0,0,0,10">
       31. <Label Text="{Binding PostId}" IsVisible="False" />
       32. <Image x:Name="PostImage" Source="{Binding IconPath}"/>
       33. <FlexLayout JustifyContent="Start" Padding="20,0" HeightRequest="50" AlignItems="Center">
       34. <StackLayout Orientation="Horizontal">
       35. <Image Source="{local:ImageResource ShopsAggregator.images.heart.png}"
       36. IsVisible="{Binding IsUserDontLikePost}">
       37. <Image.GestureRecognizers>
       38. <TapGestureRecognizer
       39. Tapped="OnLikeImageTapped"
       40. NumberOfTapsRequired="1" />
       41. </Image.GestureRecognizers>
       42. </Image>
       43. <Label HorizontalTextAlignment="Center" Text="{Binding LikesCount}" FontSize="Small"/>
       44. </StackLayout>
       45. <Image Source="{local:ImageResource ShopsAggregator.images.dislike.png}"
       46. IsVisible="{Binding IsUserLikePost}">
       47. <Image.GestureRecognizers>
       48. <TapGestureRecognizer
       49. Tapped="OnDislikeImageTapped"
       50. NumberOfTapsRequired="1" />
       51. </Image.GestureRecognizers>
       52. </Image>
       53. </FlexLayout>
       54. <Label Text="{Binding Info}" Padding="20,3"/>
       55. </StackLayout>
       56. </ViewCell>
       57. </DataTemplate>
       58. </ListView.ItemTemplate>
       59. </ListView>
       60. </StackLayout>
       61. </StackLayout>
       62. </ScrollView>
       63. </RefreshView>
       64. </StackLayout>
       65. </ContentPage.Content>
       66. </ContentPage>
   30. WatchSellerPage.xaml.cs
       1. using System;
       2. using System.Collections.Generic;
       3. using System.Linq;
       4. using System.Net;
       5. using System.Threading.Tasks;
       6. using Newtonsoft.Json;
       7. using RestSharp;
       8. using ShopsAggregator.Models;
       9. using Xamarin.Forms;
       10. using Xamarin.Forms.Xaml;
       11. namespace ShopsAggregator.Views
       12. {
       13. /// <summary>
       14. /// Код страницы просмотра страницы пользователя-продавца пользователем-покупателем.
       15. /// </summary>
       16. [XamlCompilation(XamlCompilationOptions.Compile)]
       17. public partial class WatchSellerPage : ContentPage
       18. {
       19. /// <summary>
       20. /// Экземпляр типа пользователя-продавца.
       21. /// </summary>
       22. private Seller \_seller;
       23. /// <summary>
       24. /// Имя пользователя-продавца.
       25. /// </summary>
       26. private String \_sellerName;
       27. /// <summary>
       28. /// Список записей пользователя-продавца.
       29. /// </summary>
       30. private List<BuyerPostView> \_sellerPosts = new List<BuyerPostView>();
       31. /// <summary>
       32. /// Экземпляр типа пользователя-покупателя.
       33. /// </summary>
       34. private Buyer \_buyer;
       35. /// <summary>
       36. /// Конструктор страницы. Получает экземпляр пользователя-продавца с сервера.
       37. /// </summary>
       38. /// <param name="sellerName">Имя пользователя-продавца.</param>
       39. /// <param name="buyer">Экземпляр типа пользователя-покупателя.</param>
       40. public WatchSellerPage(String sellerName, Buyer buyer)
       41. {
       42. InitializeComponent();
       43. \_buyer = buyer;
       44. GetSellerByNameAsync(sellerName);
       45. \_sellerName = sellerName;
       46. RefreshView.Refreshing += RefreshViewOnRefreshing;
       47. }
       48. /// <summary>
       49. /// Устанавливает кнопки статус подписки.
       50. /// </summary>
       51. private void SetSubscribeButton()
       52. {
       53. SubscribeSatusButton.BackgroundColor = (Color) App.Current.Resources["purple"];
       54. SubscribeSatusButton.TextColor = (Color) App.Current.Resources["buttonTextColor"];
       55. SubscribeSatusButton.Text = "+Подписаться";
       56. SubscribeSatusButton.Clicked -= UnsubscribeFromSeller;
       57. SubscribeSatusButton.Clicked += SubscribeToSeller;
       58. }
       59. /// <summary>
       60. /// Устанавливает кнопки статус отписки.
       61. /// </summary>
       62. private void SetUnsubscribeButton()
       63. {
       64. SubscribeSatusButton.BackgroundColor = (Color) App.Current.Resources["cloudWhite"];
       65. SubscribeSatusButton.TextColor = (Color) App.Current.Resources["grey"];
       66. SubscribeSatusButton.Text = "Отписаться";
       67. SubscribeSatusButton.Clicked += UnsubscribeFromSeller;
       68. SubscribeSatusButton.Clicked -= SubscribeToSeller;
       69. }
       71. /// <summary>
       72. /// Обработчик собтыия подписки на пользователя-продавца.
       73. /// </summary>
       74. /// <param name="sender">Издатель события - Button.</param>
       75. /// <param name="e">Аргументы события.</param>
       76. private async void SubscribeToSeller(object sender, EventArgs e)
       77. {
       78. if (!App.IsConnected())
       79. {
       80. await DisplayAlert("Нет подключения к интернету", "Проверьте подключение к интернету", "Попробовать снова");
       81. return;
       82. }
       83. SetUnsubscribeButton();
       84. RestClient client = new RestClient($"{App.BaseUrl}api/sub/addSub?buyerId={\_buyer.Id}&sellerName={\_seller.Username}");
       85. var request = new RestRequest(Method.PUT);
       86. request.AddHeader("Content-Type", "application/text");
       87. var response = await client.ExecuteAsync(request);
       88. if (response.StatusCode == HttpStatusCode.BadRequest)
       89. {
       90. SetSubscribeButton();
       91. await DisplayAlert("Ошибка подписки", $"Не удалось подписаться на пользователя ${\_seller.Username}", "Жаль");
       92. }
       93. }
       94. /// <summary>
       95. /// Обработчик собтыия отписки от пользователя-продавца.
       96. /// </summary>
       97. /// <param name="sender">Издатель события - Button.</param>
       98. /// <param name="e">Аргументы события.</param>
       99. private async void UnsubscribeFromSeller(object sender, EventArgs e)
       100. {
       101. if (!App.IsConnected())
       102. {
       103. await DisplayAlert("Нет подключения к интернету", "Проверьте подключение к интернету", "Попробовать снова");
       104. return;
       105. }
       106. SetSubscribeButton();
       107. RestClient client = new RestClient($"{App.BaseUrl}api/sub/rmSub?buyerId={\_buyer.Id}&sellerName={\_seller.Username}");
       108. var request = new RestRequest(Method.PUT);
       109. request.AddHeader("Content-Type", "application/text");
       110. var response = await client.ExecuteAsync(request);
       111. if (response.StatusCode == HttpStatusCode.BadRequest)
       112. {
       113. SetUnsubscribeButton();
       114. await DisplayAlert("Ошибка отписки", $"Не удалось отписаться на пользователя ${\_seller.Username}", "Жаль");
       115. }
       116. }
       117. /// <summary>
       118. /// Обработчик события обновления информации о пользователе-продавце.
       119. /// </summary>
       120. /// <param name="sender">Издатель события - RefreshView.</param>
       121. /// <param name="e">Аргументы события.</param>
       122. private void RefreshViewOnRefreshing(object sender, EventArgs e)
       123. {
       124. GetSellerByNameAsync(\_sellerName);
       125. }
       127. /// <summary>
       128. /// Получает с сервера пользователя-продавца по имени.
       129. /// </summary>
       130. /// <param name="sellerName">Имя пользователя-продавца.</param>
       131. private async void GetSellerByNameAsync(String sellerName)
       132. {
       133. if (!App.IsConnected())
       134. {
       135. await DisplayAlert("Нет подключения к интернету", "Проверьте подключение к интернету", "Попробовать снова");
       136. return;
       137. }
       138. RefreshView.IsRefreshing = true;
       139. RestClient client =new RestClient($"{App.BaseUrl}api/search/getSeller?sellerName={sellerName}");
       140. var request = new RestRequest(Method.GET);
       141. request.AddHeader("Content-Type", "application/text");
       142. var response = await client.ExecuteAsync(request);
       143. if (response.StatusCode == HttpStatusCode.BadRequest)
       144. {
       145. await DisplayAlert("Ошибка", "Не получилось получить данные о пользователе", "Поробовать снова");
       146. await Navigation.PopAsync();
       147. return;
       148. }
       149. try
       150. {
       151. \_seller = JsonConvert.DeserializeObject<Seller>(response.Content);
       152. }
       153. catch (Exception e)
       154. {
       155. await DisplayAlert("Ошибка", "Не получилось получить данные о пользователе", "Поробовать снова");
       156. await Navigation.PopAsync();
       157. return;
       158. }
       159. this.BindingContext = \_seller;
       160. GetSellerPostsAsync(\_seller.Username);
       161. if (\_seller.Subscribers == null || \_seller.Subscribers.Contains(\_buyer.Id))
       162. {
       163. SetUnsubscribeButton();
       164. }
       165. else
       166. {
       167. SetSubscribeButton();
       168. }
       169. RefreshView.IsRefreshing = false;
       170. }
       171. /// <summary>
       172. /// Получает записи пользователя-продавца с сервера.
       173. /// </summary>
       174. /// <param name="sellerName">Имя пользователя-продавца.</param>
       175. private async void GetSellerPostsAsync(String sellerName)
       176. {
       177. if (!App.IsConnected())
       178. {
       179. await DisplayAlert("Нет подключения к интернету", "Проверьте подключение к интернету", "Попробовать снова");
       180. return;
       181. }
       182. RefreshView.IsRefreshing = true;
       183. var client = new RestClient($"{App.BaseUrl}api/posts/getSellerPosts?sellerName={sellerName}");
       184. client.Timeout = -1;
       185. var request = new RestRequest(Method.GET);
       186. request.AddHeader("Content-Type", "application/text");
       187. IRestResponse response = await client.ExecuteAsync(request);
       188. if (response.StatusCode == HttpStatusCode.BadRequest)
       189. {
       190. await DisplayAlert("Ошибка", "Не получилось получить данные о пользователе", "Поробовать снова");
       191. return;
       192. }
       193. List<Post> posts = new List<Post>();
       194. try
       195. {
       196. posts = JsonConvert.DeserializeObject<List<Post>>(response.Content);
       197. }
       198. catch (Exception e)
       199. {
       200. await DisplayAlert("Ошибка", "Не получилось получить данные о пользователе", "Поробовать снова");
       201. }
       202. foreach (Post post in posts)
       203. {
       204. BuyerPostView postView = new BuyerPostView(post);
       205. postView.BuyerId = \_buyer.Id;
       206. \_sellerPosts.Add(postView);
       207. }
       208. Posts.ItemsSource = \_sellerPosts;
       209. RefreshView.IsRefreshing = false;
       210. }
       211. /// <summary>
       212. /// Отменяет событие нажатия на элемент из ListView.
       213. /// </summary>
       214. /// <param name="sender">Издатель события - ListView.</param>
       215. /// <param name="e">Аргументы события.</param>
       216. private void Posts\_OnItemSelected(object sender, SelectedItemChangedEventArgs e)
       217. {
       218. if (sender is ListView listView)
       219. {
       220. listView.SelectedItem = null;
       221. }
       222. }
       223. /// <summary>
       224. /// Событие нажатия пользователем на картинку "понравилась запись(лайк)".
       225. /// </summary>
       226. /// <param name="sender">Издатель события - картинка "понравилась запись"</param>
       227. /// <param name="e">Аргументы события.</param>
       228. private void OnLikeImageTapped(object sender, EventArgs e)
       229. {
       230. if (sender is Image image)
       231. {
       232. BuyerPostView post = (BuyerPostView) image.ParentView.BindingContext;
       233. Task.Run(() =>
       234. {
       235. BuyerPostView currentPost =
       236. (from newPost in \_sellerPosts where newPost.PostId == post.PostId select newPost).FirstOrDefault();
       237. if (currentPost == null)
       238. return;
       239. currentPost.Likers.Add(\_buyer.Id);
       240. });
       241. SendLikeTapped("addLike", post);
       242. }
       243. }
       244. /// <summary>
       245. /// Событие нажатия пользователем на картинку "убрать лайк".
       246. /// </summary>
       247. /// <param name="sender">Издатель события - картинка "убрать лайк".</param>
       248. /// <param name="e">Аргументы события.</param>
       249. private void OnDislikeImageTapped(object sender, EventArgs e)
       250. {
       251. BuyerPostView post = (BuyerPostView) (((Image) sender).ParentView.BindingContext);
       252. Task.Run(() =>
       253. {
       254. BuyerPostView currentPost =
       255. (from newPost in \_sellerPosts where newPost.PostId == post.PostId select newPost).FirstOrDefault();
       256. if (currentPost == null)
       257. return;
       258. currentPost.Likers.Remove(\_buyer.Id);
       259. });
       260. SendLikeTapped("removeLike", post);
       261. }
       262. /// <summary>
       263. /// Отправляет на сервер PUT запрос, чтобы добавить в или убрать картинку из понравившееся у пользователя.
       264. /// </summary>
       265. /// <param name="param">Параметр с Url строки, в зависимости от которого будет добавляться или убираться лайк.</param>
       266. /// <param name="post">Запись, которая понравилась/разонравилась пользователю.</param>
       267. private async void SendLikeTapped(string param, BuyerPostView post)
       268. {
       269. if (!App.IsConnected())
       270. {
       271. await DisplayAlert("Нет подключения к интернету", "Проверьте подключение к интернету", "Попробовать снова");
       272. return;
       273. }
       274. RestClient client = new RestClient($"{App.BaseUrl}api/posts/{param}?likerId={\_buyer.Id}&postId={post.PostId}");
       275. var request = new RestRequest(Method.PUT);
       276. request.AddHeader("Content-Type", "application/text");
       277. var response = await client.ExecuteAsync(request);
       278. Posts.ItemsSource = null;
       279. Posts.ItemsSource = \_sellerPosts;
       280. }
       281. }
       282. }
   31. App.xaml
       1. <?xml version="1.0" encoding="utf-8" ?>
       2. <Application xmlns="http://xamarin.com/schemas/2014/forms"
       3. xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       4. x:Class="ShopsAggregator.App">
       5. <Application.Resources>
       6. <ResourceDictionary>
       7. <Color x:Key="purple">#9b59b6</Color>
       8. <Color x:Key="grey">#95a5a6</Color>
       9. <Color x:Key="cloudWhite">#ecf0f1</Color>
       10. <Color x:Key="buttonTextColor">#ecf0f1</Color>
       11. <Color x:Key="errorColor">#e74c3c</Color>
       12. <Color x:Key="successColor">#2ecc71</Color>
       13. <x:String x:Key="appName">ShopYou</x:String>
       14. <Style x:Key="buttonStyle" TargetType="Button">
       15. <Setter Property="BackgroundColor" Value="{StaticResource purple}"/>
       16. <Setter Property="TextColor" Value="{StaticResource buttonTextColor}"></Setter>
       17. </Style>
       18. </ResourceDictionary>
       19. </Application.Resources>
       20. </Application>
   32. App.xaml.cs
       1. using System;
       2. using System.Globalization;
       3. using System.IO;
       4. using System.Threading;
       5. using Newtonsoft.Json;
       6. using ShopsAggregator.Models;
       7. using ShopsAggregator.Services;
       8. using ShopsAggregator.Views;
       9. using Xamarin.Forms;
       10. using Xamarin.Forms.Xaml;
       11. using Xamarin.Essentials;
       12. [assembly: XamlCompilation(XamlCompilationOptions.Compile)]
       13. [assembly: ExportFont("SchlangeBold.ttf", Alias = "SchlangeBold")]
       14. namespace ShopsAggregator
       15. {
       16. /// <summary>
       17. /// Основная информация о приложении хранится в этом классе.
       18. /// </summary>
       19. public partial class App : Application
       20. {
       21. /// <summary>
       22. /// Ссылка сервера на контроллер users.
       23. /// </summary>
       24. public static String ServerUrl = "http://62.113.116.228/api/users/";
       25. /// <summary>
       26. /// Базовый адрес сервера.
       27. /// </summary>
       28. public static String BaseUrl = "http://62.113.116.228/";
       29. /// <summary>
       30. /// Конструктор. Устанавливает язык приложения и интерфейса.
       31. /// </summary>
       32. public App()
       33. {
       34. Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
       35. Thread.CurrentThread.CurrentUICulture = CultureInfo.CurrentCulture;
       36. InitializeComponent();
       38. var navPage = new NavigationPage(new Authentification());
       39. MainPage = navPage;
       40. }
       41. /// <summary>
       42. /// Анимирует встряхивание элемента.
       43. /// </summary>
       44. /// <param name="element">View элемент.</param>
       45. public static async void Shake(View element)
       46. {
       47. UInt32 timer = 50;
       48. await element.TranslateTo(-15, 0, timer);
       49. await element.TranslateTo(15, 0, timer);
       50. await element.TranslateTo(-10, 0, timer);
       51. await element.TranslateTo(10, 0, timer);
       52. await element.TranslateTo(-5, 0, timer);
       53. await element.TranslateTo(5, 0, timer);
       54. element.TranslationX = 0;
       55. }
       56. /// <summary>
       57. /// Проверяет, есть ли у приложения подключение к интернету.
       58. /// </summary>
       59. /// <returns>Результат проверки.</returns>
       60. public static Boolean IsConnected()
       61. {
       62. var current = Connectivity.NetworkAccess;
       63. return current != NetworkAccess.None;
       64. }
       65. }
       66. }
   33. ImageResourceExtension.cs
       1. using System;
       2. using Xamarin.Forms;
       3. using Xamarin.Forms.Xaml;
       4. namespace ShopsAggregator
       5. {
       6. /// <summary>
       7. /// Расширение функционала Image view.
       8. /// </summary>
       9. [ContentProperty("Source")]
       10. public class ImageResourceExtension : IMarkupExtension
       11. {
       12. /// <summary>
       13. /// Путь к файлу с изображением.
       14. /// </summary>
       15. public string Source { get; set; }
       16. /// <summary>
       17. /// Получает значени из view.
       18. /// </summary>
       19. /// <param name="serviceProvider">Механизм извлечения данных.</param>
       20. /// <returns>Расположение изображения.</returns>
       21. public object ProvideValue(IServiceProvider serviceProvider)
       22. {
       23. if (Source == null)
       24. {
       25. return null;
       26. }
       27. var imageSource = ImageSource.FromResource(Source);
       29. return imageSource;
       30. }
       31. }
       32. }
   34. MainTabbedPage.xaml
       1. <?xml version="1.0" encoding="utf-8"?>
       2. <TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
       3. xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       4. x:Class="ShopsAggregator.MainTabbedPAge">
       6. </TabbedPage>
   35. MainTabbedPage.xaml.cs
       1. using ShopsAggregator.Models;
       2. using ShopsAggregator.Views;
       3. using Xamarin.Forms;
       4. using Xamarin.Forms.Xaml;
       5. namespace ShopsAggregator
       6. {
       7. /// <summary>
       8. /// Код для страницы с множеством вкладок.
       9. /// </summary>
       10. [XamlCompilation(XamlCompilationOptions.Compile)]
       11. public partial class MainTabbedPage : TabbedPage
       12. {
       13. /// <summary>
       14. /// Общий конструктор. Устанавливает заголовок страницы.
       15. /// </summary>
       16. public MainTabbedPage()
       17. {
       18. Title = "ShopYou";
       19. }
       20. /// <summary>
       21. /// Добавляет страницы если пользователем является пользователь-покупатель.
       22. /// </summary>
       23. /// <param name="buyer">Экземпляр типа пользователя-покупателя.</param>
       24. public MainTabbedPage(Buyer buyer) : this()
       25. {
       26. var userSettingsPage = new BuyerMainPage(buyer);
       27. userSettingsPage.IconImageSource = ImageSource.FromResource("ShopsAggregator.images.accounticon.png");
       28. var searchPage = new SearchPage(buyer);
       29. searchPage.IconImageSource = ImageSource.FromResource("ShopsAggregator.images.searchicon.png");
       30. var postsPage = new AllPostsPage(buyer);
       31. postsPage.IconImageSource = ImageSource.FromResource("ShopsAggregator.images.gallery.png");
       32. var ordersPage = new BuyerOrdersPage(buyer);
       33. ordersPage.IconImageSource = ImageSource.FromResource("ShopsAggregator.images.orders.png");
       34. Children.Add(postsPage);
       35. Children.Add(searchPage);
       36. Children.Add(ordersPage);
       37. Children.Add(userSettingsPage);
       38. }
       39. /// <summary>
       40. /// Добавляет страницы если пользователем является пользователь-продавец.
       41. /// </summary>
       42. /// <param name="seller">Экземпляр типа пользователя-продавца.</param>
       43. public MainTabbedPage(Seller seller) : this()
       44. {
       45. var sellerMainPage = new SellerMainPage(seller);
       46. sellerMainPage.IconImageSource = ImageSource.FromResource("ShopsAggregator.images.accounticon.png");
       47. var createPostPage = new CreatePostPage(seller);
       48. createPostPage.IconImageSource = ImageSource.FromResource("ShopsAggregator.images.plus.png");
       49. var ordersPage = new SellerOrdersPage(seller);
       50. ordersPage.IconImageSource = ImageSource.FromResource("ShopsAggregator.images.orders.png");
       51. Children.Add(ordersPage);
       52. Children.Add(createPostPage);
       53. Children.Add(sellerMainPage);
       54. }
       55. }
       56. }
   36. MainActivity.cs
       1. using System.IO;
       2. using System.Threading.Tasks;
       3. using Android.App;
       4. using Android.Content;
       5. using Android.Content.PM;
       6. using Android.OS;
       7. namespace ShopsAggregator.Android
       8. {
       9. [Activity(Label = "ShopYou", Theme = "@style/MainTheme", MainLauncher = true,
       10. ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
       11. public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsAppCompatActivity
       12. {
       13. protected override void OnCreate(Bundle savedInstanceState)
       14. {
       15. TabLayoutResource = Resource.Layout.Tabbar;
       16. ToolbarResource = Resource.Layout.Toolbar;
       17. base.OnCreate(savedInstanceState);
       18. global::Xamarin.Forms.Forms.Init(this, savedInstanceState);
       19. LoadApplication(new App());
       20. Instance = this;
       21. }
       23. internal static MainActivity Instance { get; private set; }
       24. // ...
       25. // Field, property, and method for Picture Picker
       26. public static readonly int PickImageId = 1000;
       27. public TaskCompletionSource<Stream> PickImageTaskCompletionSource { set; get; }
       28. protected override void OnActivityResult(int requestCode, Result resultCode, Intent intent)
       29. {
       30. base.OnActivityResult(requestCode, resultCode, intent);
       31. if (requestCode == PickImageId)
       32. {
       33. if ((resultCode == Result.Ok) && (intent != null))
       34. {
       35. global::Android.Net.Uri uri = intent.Data;
       36. Stream stream = ContentResolver.OpenInputStream(uri);
       37. // Set the Stream as the completion of the Task
       38. PickImageTaskCompletionSource.SetResult(stream);
       39. }
       40. else
       41. {
       42. PickImageTaskCompletionSource.SetResult(null);
       43. }
       44. }
       45. }
       46. }
       47. }
   37. ApDelegate.cs
       1. using Foundation;
       2. using UIKit;
       3. namespace ShopsAggregator.iOS
       4. {
       5. // The UIApplicationDelegate for the application. This class is responsible for launching the
       6. // User Interface of the application, as well as listening (and optionally responding) to
       7. // application events from iOS.
       8. [Register("ApDelegate")]
       9. public partial class AppDelegate : global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
       10. {
       11. //
       12. // This method is invoked when the application has loaded and is ready to run. In this
       13. // method you should instantiate the window, load the UI into it and then make the window
       14. // visible.
       15. //
       16. // You have 17 seconds to return from this method, or iOS will terminate your application.
       17. //
       18. public override bool FinishedLaunching(UIApplication app, NSDictionary options)
       19. {
       20. global::Xamarin.Forms.Forms.Init();
       21. LoadApplication(new App());
       22. return base.FinishedLaunching(app, options);
       23. }
       24. }
       25. }
   38. Main.cs
       1. using UIKit;
       2. namespace ShopsAggregator.iOS
       3. {
       4. public class Application
       5. {
       6. // This is the main entry point of the application.
       7. static void Main(string[] args)
       8. {
       9. // if you want to use a different Application Delegate class from "AppDelegate"
       10. // you can specify it here.
       11. UIApplication.Main(args, null, "AppDelegate");
       12. }
       13. }
       14. }
2. Список литературы
   1. ГОСТ 19.401-78. ЕСПД. Текст программы. Требования к содержанию и оформлению. – М.: ИПК Издательство стандартов, 2001.
   2. Документация по C# [Электронный ресурс]\\ - URL: <https://docs.microsoft.com/ru-ru/dotnet/csharp/>
3. Лист изменений

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | №  документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
| Изменен  ных | Заменен  ных | новых | Аннулиро  ванных |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |