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

Агрегатор магазинов с элементами социальной сети
Клиент

Текст программы

RU.17701729.04.01-01 12 01-1

Листов 109

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Москва 2020

Содержание

1 Текст программы

1.1 Buyer.cs

```
using System;
using System.Collections.Generic;

namespace ShopsAggregator.Models
{
    /// <summary>
    /// Тип пользователя-покупателя.
    /// </summary>
    public class Buyer : User
    {
        /// <summary>
        /// Id понравившихся записей в базе данных.
        /// </summary>
        public List<Int32> LikedPosts { get; set; } = new
List<Int32>();

        /// <summary>
        /// Id пользователей-продавцов, на которых подписан
пользователь.
        /// </summary>
        public List<Int32> Subscribed { get; set; } = new
List<Int32>();

        /// <summary>
        /// Id записей, которые добавли пользователи-продавцы,
на которых подписан данный пользователь.
        /// </summary>
        public List<Int32> NewPosts { get; set; } = new
List<Int32>();

        /// <summary>
        /// Пустой конструктор.
        /// </summary>
        public Buyer()
        {

        }
```

```
        }
    }
```

1.2 BuyerOrderView.cs

```
    using System;

    namespace ShopsAggregator.Models
    {
        /// <summary>
        /// Модель данных для заказа пользователя-покупателя.
        /// </summary>
        public class BuyerOrderView
        {
            /// <summary>
            /// Id заказа.
            /// </summary>
            public Int32 OrderId { get; set; }
            /// <summary>
            /// Экземпляр пользователя-покупателя, который создал
    запись.
            /// </summary>
            public Seller OrderSeller { get; set; }
            /// <summary>
            /// Запись, которую опубликовал пользователь-продавец.
            /// </summary>
            public Post SellerPost { get; set; }
            /// <summary>
            /// Одобрил ли пользователь-продавец заказ.
            /// </summary>
            public Boolean IsSucceded { get; set; }
            /// <summary>
            /// Отклонил ли пользователь-продавец заказ.
            /// </summary>
            public Boolean IsCanceled { get; set; }

            /// <summary>
            /// Возвращает текст со статусом заказа.
            /// </summary>
            public String OrderStatus
            {
                get
```

```
                {
                    if (this.IsSucceded)
                        return "Заказ одобрен";
                    if (this.IsCanceled)
                        return "Заказ отклонен";
                    return "Ожидание";
                }
            }
        }
    }
```

1.3 BuyerPostView.cs

```
    using System;
    using System.Collections.Generic;

    namespace ShopsAggregator.Models
    {
        /// <summary>
        /// Модель данных записи для пользователя-покупателя.
        /// </summary>
        public class BuyerPostView
        {
            /// <summary>
            /// Id пользователя-покупателя.
            /// </summary>
            public Int32 BuyerId { get; set; }
            /// <summary>
            /// Имя пользователя-продавца, который опубликовал
    запись.
            /// </summary>
            public String Username { get; set; }
            /// <summary>
            /// Id записи.
            /// </summary>
            public Int32 PostId { get; set; }
            /// <summary>
            /// Информация о записи.
            /// </summary>
            public String Info { get; set; }

            /// <summary>
```

```
        /// Понравилась ли запись данному пользователю-
покупателю.
        /// </summary>
        public Boolean IsUserLikePost =>
Likers.Contains(BuyerId);

        /// <summary>
        /// Пользователи-покупатели, которым понравилась
запись.
        /// </summary>
        public List<Int32> Likers { get; set; } = new
List<Int32>();

        /// <summary>
        /// Ссылка на фотографию к записи.
        /// </summary>
        public String IconPath =>
$"{App.BaseUrl}{PostId}photo.jpeg";
        /// <summary>
        /// Сслыка на иконку пользователя-продавца,
опубликовавшего эту запись.
        /// </summary>
        public String SellerIconPath =>
$"{App.BaseUrl}{Username}icon.jpeg";

        /// <summary>
        /// Не ставил ли пользователь-покупатель лайк.
        /// </summary>
        public Boolean IsUserDontLikePost => !IsUserLikePost;

        /// <summary>
        /// Конструктор модели данных в зависимости от
экземпляра записи.
        /// </summary>
        /// <param name="post">Экземпляр записи.</param>
        public BuyerPostView(Post post)
        {
            PostId = post.Id;
            Info = post.Info;
            Likers = post.Likers;
```

```
            }

            /// <summary>
            /// Пустой конструктор.
            /// </summary>
            public BuyerPostView()
            {

            }
        }
    }
```

1.4 CreatePostForm.cs

```
    using System;
    using System.Collections.Generic;

    namespace ShopsAggregator.Models
    {
        /// <summary>
        /// Модель для создания записи, которая отправляется на
    сервер.
        /// </summary>
        public class CreatePostForm
        {
            /// <summary>
            /// Id пользователя-продавца, который создает запись.
            /// </summary>
            public Int32 CreatorId { get; set; }
            /// <summary>
            /// Информация о записи.
            /// </summary>
            public String Info { get; set; }
            /// <summary>
            /// Массив байтов картинки записи.
            /// </summary>
            public List<Int32> ImageBytes { get; set; } = new
    List<Int32>();

            /// <summary>
            /// Создает экземпляр с id пользователя-продавца.
            /// </summary>
```

```
            /// <param name="id">Id пользователя-продавца.</param>
            public CreatePostForm(Int32 id)
            {
                CreatorId = id;
            }
        }
    }
```

1.5IconPostForm.cs

```
    using System;
    using System.Collections.Generic;

    namespace ShopsAggregator.Models
    {
        /// <summary>
        /// Экземпляр типа установки иконки пользователю.
        /// </summary>
        public class IconPostForm
        {
            /// <summary>
            /// Байты фотографии, выбранной пользователем.
            /// </summary>
            public List<Int32> IconBytesArr { get; set; }

            /// <summary>
            /// Id пользователя, который изменяет иконку.
            /// </summary>
            public Int32 ToId { get; set; }

            /// <summary>
            /// Пустой конструктор.
            /// </summary>
            public IconPostForm()
            {

            }

            /// <summary>
            /// Конструктор, который создает экземпляр с id
    пользователя.
            /// </summary>
```

```
            /// <param name="toId">Id пользователя</param>
            public IconPostForm(Int32 toId)
            {
                this.ToId = toId;
            }
        }
    }
```

1.6Post.cs

```
    using System;
    using System.Collections.Generic;

    namespace ShopsAggregator.Models
    {
        /// <summary>
        /// Тип записи, которую создает пользователь-продавец
        /// </summary>
        public class Post
        {
            /// <summary>
            /// Строка информации о записи.
            /// </summary>
            private String _info;
            /// <summary>
            /// Id записи.
            /// </summary>
            public Int32 Id { get; set; }

            /// <summary>
            /// Id создателя записи - пользователя-продавца.
            /// </summary>
            public Int32 CreatorId { get; set; }

            /// <summary>
            /// Устанавливает значение полю _info и в зависимости
    от информации в этом поле возрвращает данные о записи.
            /// </summary>
            public String Info
            {
                get => String.IsNullOrEmpty(_info) ? "Пользователь
    не оставил информацию о товаре" : _info;
```

```
            set => _info = value;
        }

        /// <summary>
        /// Возвращает ссылку на картинку к записи на сервере.
        /// </summary>
        public String ImagePath => App.BaseUrl + Id +
"photo.jpeg";

        /// <summary>
        /// Id пользователей-покупателей, которым понравилась
эта запись.
        /// </summary>
        public List<Int32> Likers { get; set; } = new
List<Int32>();

        /// <summary>
        /// Количество пользователей-покупателей, которым
понравилась эта запись.
        /// </summary>
        public Int32 LikesCount => Likers.Count;
    }
}
```

1.7RegistrationForm.cs

```
using System;

namespace ShopsAggregator.Models
{
    /// <summary>
    /// Форма для регистрации пользователя в сервисе.
    /// </summary>
    public class RegistrationForm
    {
        /// <summary>
        /// Имя пользователя.
        /// </summary>
        public String Username { get; set; }
        /// <summary>
        /// Email пользователя.
        /// </summary>
```

```
            public String Email { get; set; }
            /// <summary>
            /// Пароль пользователя.
            /// </summary>
            public String Password { get; set; }
            /// <summary>
            /// Является ли пользователь пользователем-продавцом.
            /// </summary>
            public Boolean IsSeller { get; set; }
        }
    }
```

1.8 Seller.cs

```
    using System;
    using System.Collections.Generic;

    namespace ShopsAggregator.Models
    {
        /// <summary>
        /// Тип пользователя-продавца.
        /// </summary>
        public class Seller : User
        {
            /// <summary>
            /// Id записей, созданных пользователем.
            /// </summary>
            public List<Int32> Items { get; set; } = new
    List<Int32>();

            /// <summary>
            /// Id подписчиков, которые подписались на данного
    пользователя.
            /// </summary>
            public List<Int32> Subscribers { get; set; } = new
    List<Int32>();

        }
    }
```

1.9 SellerOrderView.cs

```
    using System;
```

Изм.	Лист	№ док.	Подп.	Дата
RU.17701729.04.01-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам инв. №	Инв. № дубл.	Подп. и дата

```
namespace ShopsAggregator.Models
{
    /// <summary>
    /// Модель заказа для пользователя-покупателя.
    /// </summary>
    public class SellerOrderView
    {
        /// <summary>
        /// Id заказа.
        /// </summary>
        public Int32 OrderId { get; set; }
        /// <summary>
        /// Экземпляр типа пользователя, который сделал заказ.
        /// </summary>
        public Buyer OrderBuyer { get; set; }
        /// <summary>
        /// Экземпляр типа записи, которую заказал
пользователь.
        /// </summary>
        public Post CurrentPost { get; set; }
        /// <summary>
        /// Одобрил ли пользователь-продавец заказ.
        /// </summary>
        public Boolean IsSucceded { get; set; }
        /// <summary>
        /// Отклонил ли пользователь-продавец заказ.
        /// </summary>
        public Boolean IsCanceled { get; set; }
        /// <summary>
        /// Возвращает текст со статусом заказа.
        /// </summary>
        public String OrderStatus
        {
            get
            {
                if (this.IsSucceded)
                    return "Заказ одобрен";
                if (this.IsCanceled)
                    return "Заказ отклонен";
```

```
                    return "Ожидание";
                }
            }
        }
    }
```

1.10 User.cs

```
using System;

namespace ShopsAggregator.Models
{
    /// <summary>
    /// Общий тип пользователей.
    /// </summary>
    public class User
    {
        /// <summary>
        /// Поле значения иконки пользователя.
        /// </summary>
        private String _iconPath = "standarticon.jpeg";
        /// <summary>
        /// Id пользователя.
        /// </summary>
        public Int32 Id { get; set; }
        /// <summary>
        /// Имя пользователя.
        /// </summary>
        public String Username { get; set; }
        /// <summary>
        /// Email пользователя.
        /// </summary>
        public String Email { get; set; }
        /// <summary>
        /// Пароль пользователя.
        /// </summary>
        public String Password { get; set; }
        /// <summary>
        /// Информация о пользователе.
        /// </summary>
        public String Info { get; set; }
        /// <summary>
```

```
        /// Устанавливает имя иконки пользователя на сервер и
возвращает ссылку на него.
        /// </summary>
        public String IconPath { get =>
$"{App.BaseUrl}{_iconPath}"; set => _iconPath = value; }


        /// <summary>
        /// Пустой конструктор.
        /// </summary>
        public User()
        {

        }
    }
}
```

1.11 AllPostsPage.xaml

```
<?xml version="1.0" encoding="utf-8"?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="ShopsAggregator.Views.AllPostsPage"
             xmlns:local="clr-
namespace:ShopsAggregator;assembly=ShopsAggregator"
             Title="Товары">
    <ContentPage.Content>
        <StackLayout Margin="0,20,0,0">
            <ListView x:Name="Posts" HasUnevenRows="True"
ItemTapped="Posts_OnItemTapped" SeparatorVisibility="None">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <StackLayout Padding="0,5"
Margin="0,0,0,10">
                                <Label Text="{Binding PostId}"
IsVisible="False" />
                                <FlexLayout
JustifyContent="Start" Padding="5" AlignItems="Center">
```

```
<FlexLayout.GestureRecognizers>
                                        <TapGestureRecognizer
Tapped="OnPostHeaderTapped"/>
</FlexLayout.GestureRecognizers>
                                    <Frame CornerRadius="15"
IsClippedToBounds="True" HorizontalOptions="Center"
BackgroundColor="Gray"
                                           WidthRequest="30"
HeightRequest="30" VerticalOptions="Start" Padding="0"
HasShadow="False">
                                        <Image
x:Name="SellerIcon" Source="{Binding SellerIconPath}"
VerticalOptions="FillAndExpand"
HorizontalOptions="FillAndExpand" />
                                    </Frame>
                                    <Label Text="{Binding
Username}" Padding="10,0"/>
                                </FlexLayout>
                                <StackLayout >
                                    <Image x:Name="PostImage"
Source="{Binding IconPath}"/>
                                    <FlexLayout
JustifyContent="SpaceBetween" Padding="20,0"
HeightRequest="50" AlignItems="Center">
                                        <StackLayout
Orientation="Horizontal">
                                            <Image
Source="{local:ImageResource
ShopsAggregator.images.heart.png}"
IsVisible="{Binding IsUserDontLikePost}">
<Image.GestureRecognizers>
<TapGestureRecognizer
```

```
Tapped="OnLikeImageTapped"

NumberOfTapsRequired="1" />

</Image.GestureRecognizers>
                                        </Image>
                                        <Image
Source="{local:ImageResource
ShopsAggregator.images.dislike.png}"

IsVisible="{Binding IsUserLikePost}">

<Image.GestureRecognizers>

<TapGestureRecognizer

Tapped="OnDislikeImageTapped"

NumberOfTapsRequired="1" />

</Image.GestureRecognizers>
                                        </Image>
                                    </StackLayout>
                                    <StackLayout>
                                        <Button
x:Name="DeliverButton" Text="+" Margin="0" Padding="0"
FontSize="30"

HeightRequest="30" Clicked="DeliverButton_OnClicked"/>
                                        <Button
Text="Заказать" FontSize="13" Padding="0" Margin="0"
HeightRequest="13"

Clicked="DeliverButton_OnClicked"/>
                                    </StackLayout>
                                </FlexLayout>
                                <Label Text="{Binding
Info}" Padding="20,3"/>
                            </StackLayout>
```

```
                                </StackLayout>
                            </ViewCell>
                        </DataTemplate>
                    </ListView.ItemTemplate>
                </ListView>
            </StackLayout>
        </ContentPage.Content>
    </ContentPage>
```

1.12 AllPostsPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Threading.Tasks;
using Newtonsoft.Json;
using RestSharp;
using ShopsAggregator.Models;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace ShopsAggregator.Views
{
    /// <summary>
    /// Код для страницы всех записей.
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class AllPostsPage : ContentPage
    {
        /// <summary>
        /// Экземпляр типа пользователя-покупателя.
        /// </summary>
        private Buyer _buyer;
        /// <summary>
        /// Новые записи, доступные пользователю-покупателю.
        /// </summary>
        private List<BuyerPostView> newPosts = new
List<BuyerPostView>();

        /// <summary>
        /// Конструктор страницы.
```

```
        /// </summary>
        /// <param name="buyer">Экземпляр типа пользователя-
покупателя.</param>
        public AllPostsPage(Buyer buyer)
        {
            InitializeComponent();
            _buyer = buyer;
            Posts.Refreshing += RefreshViewOnRefreshing;
            Posts.IsPullToRefreshEnabled = true;
        }

        /// <summary>
        /// Обновляет данные страницы по запросу пользователя.
        /// </summary>
        /// <param name="sender">Издатель события -
RefreshView.</param>
        /// <param name="e">Аргументы события.</param>
        private void RefreshViewOnRefreshing(object sender,
EventArgs e)
        {
            GetNewPostsAsync();
        }

        /// <summary>
        /// Получает данные о новых записях и вызывает базовое
поведение метода.
        /// </summary>
        protected override void OnAppearing()
        {
            base.OnAppearing();
            GetNewPostsAsync();
        }

        /// <summary>
        /// Получает новые записи, доступные пользователю.
        /// </summary>
        private async void GetNewPostsAsync()
        {
            if (!App.IsConnected())
            {
```

```
                await DisplayAlert("Ошибка", "Осутствует
подключение к интернету", "Поробовать снова");
                return;
            }
            Posts.IsRefreshing = true;
            RestClient client = new
RestClient($"{App.BaseUrl}api/posts/getBuyerNewPost?buyerId={_
buyer.Id}");
            var request = new RestRequest(Method.GET);
            request.AddHeader("Content-Type",
"application/text");
            var response = await client.ExecuteAsync(request);
            if (response.StatusCode ==
HttpStatusCode.BadRequest)
            {
                await DisplayAlert("Ошибка", "Не получилось
получить новые записи", "Попробовать снова");
                Posts.IsRefreshing = false;
                return;
            }
            List<BuyerPostView> newPosts = new
List<BuyerPostView>();
            try
            {
                newPosts =
JsonConvert.DeserializeObject<List<BuyerPostView>>(response.Co
ntent);
            }
            catch (Exception e)
            {
                await DisplayAlert("Ошибка", "Не получилось
получить новые записи", "Попробовать снова");
                Posts.IsRefreshing = false;
                return;
            }

            foreach (BuyerPostView buyerPostView in newPosts)
            {
                buyerPostView.BuyerId = _buyer.Id;
            }
```

```
            this.newPosts = newPosts;
            Posts.ItemsSource = this.newPosts;
            Posts.IsRefreshing = false;
        }

        /// <summary>
        /// Событие нажатия пользователем на картинку
"понравилась запись(лайк)".
        /// </summary>
        /// <param name="sender">Издатель события - картинка
"понравилась запись"</param>
        /// <param name="e">Аргументы события.</param>
        private void OnLikeImageTapped(object sender,
EventArgs e)
        {
            BuyerPostView post = (BuyerPostView) (((Image)
sender).ParentView.BindingContext);
            Task.Run(() =>
            {
                BuyerPostView currentPost =
                    (from newPost in newPosts where
newPost.PostId == post.PostId select
newPost).FirstOrDefault();
                if (currentPost == null)
                    return;
                currentPost.Likers.Add(_buyer.Id);
                Posts.ItemsSource = newPosts;
            });
            SendLikeTapped("addLike", post);
        }
        /// <summary>
        /// Событие нажатия пользователем на картинку "убрать
лайк".
        /// </summary>
        /// <param name="sender">Издатель события - картинка
"убрать лайк".</param>
        /// <param name="e">Аргументы события.</param>
        private void OnDislikeImageTapped(object sender,
EventArgs e)
```

```
        {
            BuyerPostView post = (BuyerPostView) (((Image)
sender).ParentView.BindingContext);
            Task.Run(() =>
            {
                BuyerPostView currentPost =
                    (from newPost in newPosts where
newPost.PostId == post.PostId select
newPost).FirstOrDefault();
                if (currentPost == null)
                    return;
                currentPost.Likers.Remove(_buyer.Id);
            });
            SendLikeTapped("removeLike", post);
        }

        /// <summary>
        /// Отправляет на сервер PUT запрос, чтобы добавить в
или убрать картинку из понравившееся у пользователя.
        /// </summary>
        /// <param name="param">Параметр с Url строки, в
зависимости от которого будет добавляться или убираться
лайк.</param>
        /// <param name="post">Запись, которая
понравилась/разонравилась пользователю.</param>
        private async void SendLikeTapped(String param,
BuyerPostView post)
        {
            if (!App.IsConnected())
            {
                await DisplayAlert("Ошибка", "Осутствует
подключение к интернету", "Поробовать снова");
                return;
            }
            Posts.ItemsSource = null;
            Posts.ItemsSource = newPosts;
            RestClient client = new
RestClient($"{App.BaseUrl}api/posts/{param}?likerId={_buyer.Id
}&postId={post.PostId}");
            var request = new RestRequest(Method.PUT);
```

```
            request.AddHeader("Content-Type",
"application/text");
            var response = await client.ExecuteAsync(request);
        }


        /// <summary>
        /// Отменяет событие нажатия на элемент из ListView.
        /// </summary>
        /// <param name="sender">Издатель события - ListView
Posts.</param>
        /// <param name="e">Аргументы события.</param>
        private void Posts_OnItemTapped(object sender,
ItemTappedEventArgs e)
        {
            if (sender is ListView listView)
            {
                listView.SelectedItem = null;
            }
        }

        /// <summary>
        /// Обрабатывает событие нажатия на иконку или имя
пользователя-продавца, который опубликовал запись.
        /// </summary>
        /// <param name="sender">Издатель события -
FlexLayout.</param>
        /// <param name="e">Аргументы события.</param>
        private async void OnPostHeaderTapped(object sender,
EventArgs e)
        {
            if (sender is FlexLayout flexLayout)
            {
                await Navigation.PushAsync(new
WatchSellerPage((flexLayout.Children[1] as
Label).Text,_buyer));
            }
        }

        /// <summary>
```

```
        /// Событие добавления пользователем записи к заказам.
        /// </summary>
        /// <param name="sender">Издатель события -
Button</param>
        /// <param name="e">Аргументы события.</param>
        private async void DeliverButton_OnClicked(object
sender, EventArgs e)
        {
            if (!App.IsConnected())
            {
                await DisplayAlert("Ошибка", "Нет подключения
к интернету", "Попробовать снова");
                return;
            }

            if (String.IsNullOrEmpty(_buyer.Info))
            {
                await DisplayAlert("Заполните поле информации
об аккаунте!",
                    "Так пользователь продавец будет знать как
с вами связаться", "Хорошо");
                return;
            }
            if (sender is Button button)
            {
                BuyerPostView view = (BuyerPostView)
button.ParentView.BindingContext;
                if (view == null)
                    return;
                RestClient client = new
RestClient($"{App.BaseUrl}api/orders/addOrder?buyerId={_buyer.
Id}&" +

$"postId={view.PostId}");
                var request = new RestRequest(Method.POST);
                request.AddHeader("Content-Type",
"application/text");
                var response = await
client.ExecuteAsync(request);
            }
```

```
        }
    }
}
```

1.13 Authentification.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:ShopsAggregator"
             xmlns:customControls="clr-
namespace:ShopsAggregator.CustomControls;assembly=ShopsAggrega
tor"
             x:Class="ShopsAggregator.Views.Authentification">
    <ContentPage.Content>
        <StackLayout VerticalOptions="CenterAndExpand"
HorizontalOptions="CenterAndExpand" Padding="20,0">
            <FlexLayout Direction="Column"
JustifyContent="SpaceAround">
                <StackLayout>
                    <Label Text="ShopYou"
TextColor="{StaticResource purple}"
HorizontalTextAlignment="Center" FontSize="44"
                           Margin="0,0,0,100"
FontFamily="SchlangeBold"/>

                    <customControls:CustomEditor
x:Name="username" Placeholder="Enter your buyername or email"
Margin="0,0,0,3"/>
                    <Entry x:Name="password" IsPassword="true"
Placeholder="Enter your password" Margin="0,0,0,7"/>
                    <Button Text="Войти как покупатель"
x:Name="buyerSignInButton"
Clicked="OnBuyerSignInButtonClicked"
                           Style="{StaticResource
buttonStyle}" CornerRadius="5"/>
                    <Button Text="Войти как магазин"
x:Name="sellerSignInButton"
Clicked="OnSellerSignInButtonClicked"
                           Style="{StaticResource
buttonStyle}" CornerRadius="5"/>
```

```
                    <Label Text="или" Margin="0,10"
HorizontalTextAlignment="Center" TextColor="{StaticResource
purple}"/>
                </StackLayout>
                <Button Text="Зарегистрироваться"
Clicked="OnRegistrationButtonClick"/>
                <Label x:Name="StatusOfSignIn"
FontSize="Small" TextColor="Red"
HorizontalTextAlignment="Center" Text="заглушка"
IsVisible="False"/>
            </FlexLayout>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

1.14 Authentification.xaml.cs

```
using System;
using System.Net;
using Newtonsoft.Json;
using RestSharp;
using ShopsAggregator.Models;
using Xamarin.Forms;

namespace ShopsAggregator.Views
{
    /// <summary>
    /// Код страницы авторизации пользователя.
    /// </summary>
    public partial class Authentification : ContentPage
    {
        /// <summary>
        /// Текст сообщения о том, что поле имени пользователя
пустое.
        /// </summary>
        private const String
IncorrectUsernameInputErrorMessage = "Имя ползователя не может
быть пустым";
        /// <summary>
        /// Текст сообщения о том, что поля пароля пустое.
        /// </summary>
```

```
        private const String
IncorrectPasswordInputErrorMessage = "Необходимо ввести
пароль";
        /// <summary>
        /// Заголовок сообщения о неудачном входе.
        /// </summary>
        private const String BadSignInTryAlertTitle =
"Неудачная попытка входа";
        /// <summary>
        /// Текст сообщения о неудачном входе.
        /// </summary>
        private const String BadSignInTryAlertCancelText =
"Попробовать снова";

        /// <summary>
        /// Конструктор страницы.
        /// </summary>
        public Authentification()
        {
            InitializeComponent();
            NavigationPage.SetHasNavigationBar(this, false);
        }

        /// <summary>
        /// Устанавливает цвет границ кнопок авторизации и
вызывает базовое поведение метода.
        /// </summary>
        protected override void OnAppearing()
        {
            base.OnAppearing();
            buyerSignInButton.BorderColor =
sellerSignInButton.BorderColor =  Color.FromRgba(12, 12, 12,
1);
        }

        /// <summary>
        /// Событие попытки входа в аккаунт пользователя-
покупателя.
        /// </summary>
```

```
        /// <param name="sender">Издатель события - Button
buyerSignInButton.</param>
        /// <param name="e">Аргументы события.</param>
        private void OnBuyerSignInButtonClicked(object sender,
EventArgs e)
        {
            if (!CheckIsBoxesAndConnectionCorrect())
                return;
            Buyer user = null;
            try
            {
                user = TrySignIn<Buyer>(username.Text,
password.Text, "authBuyer");
            }
            catch (Exception exception)
            {
                DisplayAlert(BadSignInTryAlertTitle, "Ошибка
сервера", BadSignInTryAlertCancelText);
                return;
            }
            if (user == null)
            {
                DisplayAlert(BadSignInTryAlertTitle,
"Проверьте введеные данные", BadSignInTryAlertCancelText);
                return;
            }

            var mainTabbedPage = new MainTabbedPage(user);
            NavigationPage.SetHasNavigationBar(mainTabbedPage,
true);
            NavigationPage.SetHasBackButton(mainTabbedPage,
false);
            Navigation.PushAsync(mainTabbedPage);
        }

        /// <summary>
        /// Событие попытки входа в аккаунт пользователя-
продавца.
        /// </summary>
```

```
        /// <param name="sender">Издатель события - Button
sellerSignInButton.</param>
        /// <param name="e">Аргументы события.</param>
        private void OnSellerSignInButtonClicked(object
sender, EventArgs e)
        {
            if (!CheckIsBoxesAndConnectionCorrect())
                return;
            Seller user = null;
            try
            {
                user = TrySignIn<Seller>(username.Text,
password.Text, "authSeller");
            }
            catch (Exception)
            {
                DisplayAlert(BadSignInTryAlertTitle, "Ошибка
сервера", BadSignInTryAlertCancelText);
                return;
            }
            if(user == null)
            {
                DisplayAlert(BadSignInTryAlertTitle,
"Проверьте введеные данные", BadSignInTryAlertCancelText);
                return;
            }
            var mainTabbedPage = new MainTabbedPage(user);
            NavigationPage.SetHasNavigationBar(mainTabbedPage,
true);
            NavigationPage.SetHasBackButton(mainTabbedPage,
false);
            Navigation.PushAsync(mainTabbedPage);
        }

        /// <summary>
        /// Проверяет есть ли соединение с интернетом и все ли
поля заполнены верно.
        /// </summary>
        /// <returns>Результат проверки.</returns>
        private Boolean CheckIsBoxesAndConnectionCorrect()
```

```
        {
            if (!App.IsConnected())
            {
                DisplayAlert("Нет подключения к интернету",
"Проверьте подключение к интернету", "Попробовать снова");
                return false;
            }
            if (String.IsNullOrWhiteSpace(username.Text))
            {
                MessageAboutIncorrectBox(username,
IncorrectUsernameInputErrorMessage);
                return false;
            }

            if (String.IsNullOrEmpty(password.Text))
            {
                MessageAboutIncorrectBox(password,
IncorrectPasswordInputErrorMessage);
                return false;
            }

            return true;
        }

        /// <summary>
        /// Отправляет запрос на сервер для входа в аккаунт.
        /// </summary>
        /// <param name="login">Логин, под которым
пользователь хочет войти в аккаунт.</param>
        /// <param name="password">Пароль, под которым
пользователь хочет войти в аккаунт.</param>
        /// <param name="authType">Строка, под каким типом
хочет войти пользователь.</param>
        /// <typeparam name="T">Ожидаемый тип возврата с
сервера.</typeparam>
        /// <returns>Результат запроса в виде экземпляра типа
T.</returns>
        private T TrySignIn<T>(String login, String password,
String authType) where T : class
        {
```

```
            var client = new
RestClient($"{App.ServerUrl}{authType}?login={login}&password=
{password}");
            client.Timeout = 10000;
            var request = new RestRequest(Method.GET);
            IRestResponse response = client.Execute(request);
            User user;
            if (response.StatusCode ==
HttpStatusCode.BadRequest)
            {
                return null;
            }
            return
JsonConvert.DeserializeObject<T>(response.Content);
        }

        /// <summary>
        /// Вызывает анимацию встряски View элемента и выводит
сообщение о том что поле заполнено неверно.
        /// </summary>
        /// <param name="view">View элемент, который заполнен
неверно.</param>
        /// <param name="errorMsg">Сообщение об ошибки,
котороые необходимо вывести.</param>
        private void MessageAboutIncorrectBox(Entry view,
String errorMsg)
        {
            App.Shake(view);
            StatusOfSignIn.Text = errorMsg;
            StatusOfSignIn.IsVisible = true;
            view.TextChanged += OnEntryTextChanged;
        }

        /// <summary>
        /// Убирает сообщение о статусе входа при измененнии
полей входа.
        /// </summary>
        /// <param name="sender">Издатель события -
Entry.</param>
```

```
        /// <param name="e">Аргументы события.</param>
        private void OnEntryTextChanged(object sender,
TextChangedEventArgs e)
        {
            if (sender is Entry entry)
            {
                StatusOfSignIn.IsVisible = false;
                entry.TextChanged -= OnEntryTextChanged;
            }
        }


        /// <summary>
        /// Перенаправляет пользователя на страницу
регистрации.
        /// </summary>
        /// <param name="sender">Издатель события -
Button.</param>
        /// <param name="e">Аргументы события.</param>
        private async void OnRegistrationButtonClick(object
sender, EventArgs e)
        {
            await Navigation.PushAsync(new
RegistrationPage());
        }
    }
}
```

1.15 BuyerMainPage.xaml

```
<?xml version="1.0" encoding="utf-8"?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:shopsAggregator="clr-
namespace:ShopsAggregator;assembly=ShopsAggregator"
             x:Class="ShopsAggregator.Views.BuyerMainPage"
             Title="Аккаунт">
    <ContentPage.ToolbarItems>
        <ToolbarItem Text="Выйти"
Clicked="OnExitToolbarItemTapped" />
```

```
    </ContentPage.ToolbarItems>
    <ContentPage.Content>
        <RefreshView Refreshing="RefreshView_OnRefreshing"
x:Name="RefreshView">
        <ScrollView>
        <StackLayout Margin="0,10,0,0" Padding="50,0">
            <Frame CornerRadius="100" IsClippedToBounds="True"
HorizontalOptions="Center" BackgroundColor="Gray"
                   WidthRequest="200" HeightRequest="200"
VerticalOptions="Start" Padding="0" HasShadow="True"
Margin="0,20">
                <Frame.GestureRecognizers>
                    <TapGestureRecognizer Tapped="GetPhoto"/>
                </Frame.GestureRecognizers>
                <StackLayout HorizontalOptions="Center"
VerticalOptions="Center">
                    <Image x:Name="Icon"
HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand"
                           Source="{Binding IconPath}"/>
                    <Label x:Name="ImagePickerText"
Text="Нажмите чтобы выбрать фотографию"
TextColor="{StaticResource cloudWhite}"
                           FontSize="Small"
HorizontalTextAlignment="Center"
VerticalTextAlignment="Center"/>
                </StackLayout>
            </Frame>
            <StackLayout x:Name="UserInfoStack"
HorizontalOptions="FillAndExpand">
                <Label Text="{Binding Username}"
FontSize="Large" HorizontalTextAlignment="Center" />
                <Label Text="{Binding Email}"
TextColor="{StaticResource grey}" FontSize="Small"
HorizontalTextAlignment="Center"/>
                <Editor x:Name="InfoEditor" Text="{Binding
Info}" Placeholder="Введите адрес и другую ифнорацию, чтобы
продавец смог связаться с вами"

Unfocused="VisualElement_OnUnfocused"/>
```

```
            </StackLayout>
            <StackLayout HorizontalOptions="Center">
                <Label x:Name="SubscribedCounter" Text="0"
FontSize="Large" HorizontalTextAlignment="Center"/>
                <Label x:Name="SubscribedLabel"
Text="Подписан" FontSize="Medium" TextColor="{StaticResource
grey}"
                       HorizontalTextAlignment="Center"/>
            </StackLayout>
            <Label FontSize="Title"
HorizontalTextAlignment="Start">Понравившиеся товары</Label>
            <ListView x:Name="Posts"
CachingStrategy="RecycleElementAndDataTemplate"
HasUnevenRows="True"
                      SeparatorVisibility="None"
ItemTapped="OnPostsItemTapped">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <StackLayout Padding="0,5"
Margin="0,0,0,10">
                                <Label Text="{Binding PostId}"
IsVisible="False" />
                                <FlexLayout
JustifyContent="Start" Padding="5" AlignItems="Center">

<FlexLayout.GestureRecognizers>
                                        <TapGestureRecognizer

Tapped="OnPostHeaderTapped"/>

</FlexLayout.GestureRecognizers>
                                    <Frame CornerRadius="15"
IsClippedToBounds="True" HorizontalOptions="Center"
BackgroundColor="Gray"
                                           WidthRequest="30"
HeightRequest="30" VerticalOptions="Start" Padding="0"
HasShadow="False">
```

```
                                        <Image
x:Name="SellerIcon" Source="{Binding SellerIconPath}"
VerticalOptions="FillAndExpand"

HorizontalOptions="FillAndExpand" />
                                    </Frame>
                                    <Label Text="{Binding
Username}" Padding="10,0"/>
                                </FlexLayout>
                                <StackLayout >
                                    <Image x:Name="PostImage"
Source="{Binding IconPath}"/>
                                    <FlexLayout
JustifyContent="Start" HeightRequest="50" AlignItems="Center">
                                        <StackLayout
Orientation="Horizontal">
                                            <Image
Source="{shopsAggregator:ImageResource
ShopsAggregator.images.dislike.png}" >

<Image.GestureRecognizers>

<TapGestureRecognizer Tapped="SendDislikeImageTapped"/>

</Image.GestureRecognizers>
                                            </Image>
                                            <Label
HorizontalTextAlignment="Center" Text="{Binding LikesCount}"
FontSize="Small"/>
                                        </StackLayout>
                                    </FlexLayout>
                                    <Label Text="{Binding
Info}" Padding="0,3"/>
                                </StackLayout>
                            </StackLayout>
                        </ViewCell>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
        </StackLayout>
```

```
            </ScrollView>
            </RefreshView>
        </ContentPage.Content>
    </ContentPage>
```

1.16 BuyerMainPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using Newtonsoft.Json;
using Plugin.Media;
using Plugin.Media.Abstractions;
using RestSharp;
using ShopsAggregator.Models;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace ShopsAggregator.Views
{
    /// <summary>
    /// Код страницы пользователя-покупателя.
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class BuyerMainPage : ContentPage
    {
        /// <summary>
        /// Ссылка на иконку пользователя.
        /// </summary>
        private String iconPath;
        /// <summary>
        /// Экземпляр типа пользователя-покупателя.
        /// </summary>
        private Buyer _buyer;
        /// <summary>
        /// Заголовок сообщения об ошибке подключения к
    серверу.
        /// </summary>
```

```
        private const String
FailedConnectionToServerAlertTitle = "Ошибка подключения к
серверу";
        /// <summary>
        /// Список записей, которые понравились пользователю.
        /// </summary>
        private List<BuyerPostView> likedPosts = new
List<BuyerPostView>();
        /// <summary>
        /// Конструктор страницы.
        /// </summary>
        /// <param name="buyer">Экземпляр типа пользователя-
покупателя.</param>
        public BuyerMainPage(Buyer buyer)
        {
            InitializeComponent();
            _buyer = buyer;
            this.BindingContext = _buyer;
        }

        /// <summary>
        /// Устанавливает ширину ListView Posts, получает
понравившиеся записи и вызывает базовое поведение метода.
        /// </summary>
        protected override void OnAppearing()
        {
            Posts.WidthRequest = App.Current.MainPage.Width;
            SubscribedCounter.Text =
_buyer.Subscribed.Count.ToString();
            GetBuyerLikedPosts();
            base.OnAppearing();
        }

        /// <summary>
        /// Получает с сервера записи, которые понравились
пользователю.
        /// </summary>
        private async void GetBuyerLikedPosts()
        {
            if (!App.IsConnected())
```

```
            {
                await DisplayAlert("Нет подключения к
интернету", "Проверьте подключение к интернету", "Попробовать
снова");
                return;
            }
            RefreshView.IsRefreshing = true;
            var client = new
RestClient($"{App.BaseUrl}api/posts/getBuyerLikedPosts?buyerId
={_buyer.Id}");
            client.Timeout = -1;
            var request = new RestRequest(Method.GET);
            request.AddHeader("Content-Type",
"application/text");
            IRestResponse response = await
client.ExecuteAsync(request);
            if (response.StatusCode ==
HttpStatusCode.BadRequest)
            {
                await DisplayAlert("Ошибка", "Не получилось
удалось понравившиеся записи", "Попробовать снова");
                return;
            }

            try
            {
                likedPosts =
JsonConvert.DeserializeObject<List<BuyerPostView>>(response.Co
ntent);
            }
            catch (Exception e)
            {
                await DisplayAlert("Ошибка", "Не получилось
удалось понравившиеся записи", "Попробовать снова");
                return;
            }

            Posts.ItemsSource = likedPosts;
            RefreshView.IsRefreshing = false;
```

```
        }

        /// <summary>
        /// Отправляет запрос с обновлением фотографии
пользователя.
        /// </summary>
        /// <param name="form">Форма с информацией о
пользователе и его новой фотграфией.</param>
        private async void SendUpdateUserPut(IconPostForm
form)
        {
            String json = JsonConvert.SerializeObject(form);
            var client = new
RestClient($"{App.ServerUrl}addBuyerIcon");
            client.Timeout = -1;
            var request = new RestRequest(Method.PUT);
            request.AddHeader("Content-Type",
"application/json");
            request.AddParameter("application/json", json,
ParameterType.RequestBody);
            IRestResponse response = await
client.ExecuteAsync(request);
            if (response.StatusCode ==
HttpStatusCode.RequestEntityTooLarge)
            {
                await
DisplayAlert(FailedConnectionToServerAlertTitle,
response.Content, "Попробовать снова");
                return;
            }

            if (response.StatusCode ==
HttpStatusCode.BadRequest)
            {
                await
DisplayAlert(FailedConnectionToServerAlertTitle,
response.Content, "Попробовать снова");
                return;
            }
```

```
            _buyer.IconPath = _buyer.Username + "icon.jpeg";
        }

        /// <summary>
        /// При нажатии пользователя на иконку предоставляет
пользователю выбрать фотографию.
        /// </summary>
        /// <param name="sender">Издатель события -
Frame.</param>
        /// <param name="e">Аргументы события.</param>
        private async void GetPhoto(object sender, EventArgs
e)
        {
            if (CrossMedia.Current.IsPickPhotoSupported)
            {
                MediaFile photo = await
CrossMedia.Current.PickPhotoAsync();
                iconPath = photo.Path;
                Icon.Source = ImageSource.FromFile(iconPath);
                IconPostForm form = new
IconPostForm(_buyer.Id);
                GetPhotoBytes(iconPath, form);
                if (!App.IsConnected())
                {
                    return;
                }
                SendUpdateUserPut(form);
            }
        }

        /// <summary>
        /// Получает байты фотографии, которую выбрал
пользователь.
        /// </summary>
        /// <param name="path">Путь к файлу с указаной
фотографией.</param>
        /// <param name="form">Модель, в которую записываются
байты фотографии.</param>
        private void GetPhotoBytes(String path, IconPostForm
form)
```

```
        {
            List<Int32> bytes = new List<Int32>();
            using (FileStream fs = new FileStream(path,
FileMode.Open))
            {
                while(fs.Position != fs.Length)
                    bytes.Add(fs.ReadByte());
            }

            form.IconBytesArr = bytes.ToList();
        }

        /// <summary>
        /// Обработчик события нажатия на кнопку выхода.
        /// </summary>
        /// <param name="sender">Издатель события -
Button.</param>
        /// <param name="e">Аргументы события.</param>
        private async void OnExitToolbarItemTapped(object
sender, EventArgs e)
        {
            await Navigation.PopToRootAsync();
        }

        /// <summary>
        /// Обновляет информацию о пользователе.
        /// </summary>
        /// <param name="sender">Издатель события -
RefreshView.</param>
        /// <param name="e">Аргументы события.</param>
        private void RefreshView_OnRefreshing(object sender,
EventArgs e)
        {
            this.BindingContext = _buyer;
            GetBuyerLikedPosts();
        }
        /// <summary>
        /// Открывает страницу пользователя, который
опубликовал запись.
        /// </summary>
```

```
        /// <param name="sender">Издатель события -
FlexLayout.</param>
        /// <param name="e">Аргументы события.</param>
        private async void OnPostHeaderTapped(object sender,
EventArgs e)
        {
            if (sender is FlexLayout flexLayout)
            {
                await Navigation.PushAsync(new
WatchSellerPage((flexLayout.Children[1] as
Label).Text,_buyer));
            }
        }

        /// <summary>
        /// Отправляет запрос на сервер, чтобы убрать запись
из понравившегося.
        /// </summary>
        /// <param name="sender">Издатель события -
Image.</param>
        /// <param name="e">Аргументы события.</param>
        private async void SendDislikeImageTapped(object
sender, EventArgs e)
        {
            if (!App.IsConnected())
            {
                await DisplayAlert("Нет подключения к
интернету", "Проверьте подключение к интернету", "Попробовать
снова");
                return;
            }
            if (sender is Image image)
            {
                BuyerPostView bindingPost = (BuyerPostView)
(image.ParentView.BindingContext);
                BuyerPostView post =
                    (from selectionPost in likedPosts
                        where selectionPost.PostId ==
bindingPost.PostId
```

```
                        select
selectionPost).FirstOrDefault();
                if (post == null)
                    return;
                likedPosts.Remove(post);
                RestClient client = new
RestClient($"{App.BaseUrl}api/posts/removeLike?likerId={_buyer
.Id}&postId={post.PostId}");
                var request = new RestRequest(Method.PUT);
                request.AddHeader("Content-Type",
"application/text");
                var response = await
client.ExecuteAsync(request);
                Posts.ItemsSource = null;
                Posts.ItemsSource = likedPosts;
            }
        }

        /// <summary>
        /// Отправляет запрос, чтобы изменить информацию о
пользователе.
        /// </summary>
        /// <param name="sender">Издатель события -
Editor.</param>
        /// <param name="e">Аргументы события.</param>
        private async void VisualElement_OnUnfocused(object
sender, FocusEventArgs e)
        {
            if (!App.IsConnected() ||
String.IsNullOrWhiteSpace(_buyer.Info))
            {
                return;
            }
            RestClient client = new
RestClient($"{App.BaseUrl}api/users/setBuyerInfo?buyerId={_buy
er.Id}&info={_buyer.Info}");
            var request = new RestRequest(Method.PUT);
            request.AddHeader("Content-Type",
"application/text");
            var response = await client.ExecuteAsync(request);
```

```
        }

        /// <summary>
        /// Отменяет нажатие на элемент ListView.
        /// </summary>
        /// <param name="sender">Издатель события -
ListView.</param>
        /// <param name="e">Аргументы события.</param>
        private void OnPostsItemTapped(object sender,
ItemTappedEventArgs e)
        {
            if (sender is ListView listView)
            {
                listView.SelectedItem = null;
            }
        }
    }
}
```

1.17 BuyerOrdersPage.xaml

```
<?xml version="1.0" encoding="utf-8"?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="ShopsAggregator.Views.BuyerOrdersPage"
             Title="Мои заказы">
    <ContentPage.Content>
        <StackLayout>
            <ListView x:Name="Orders"
SeparatorVisibility="None" IsPullToRefreshEnabled="True"
Refreshing="OnOrdersUpdate"
                      HasUnevenRows="True" Footer="Это все
заказы" ItemTapped="OnOrdersItemTapped">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <StackLayout Padding="0,5"
Margin="0,0,0,10">
                                <Label Text="{Binding
OrderId}" IsVisible="False" />
```

```
                                    <Label Text="{Binding
SellerPost.Postd}" IsVisible="False" />
                                    <FlexLayout
JustifyContent="Start" Padding="5" AlignItems="Center">

<FlexLayout.GestureRecognizers>
                                        <TapGestureRecognizer

Tapped="OnPostHeaderTapped"/>

</FlexLayout.GestureRecognizers>
                                    <Frame CornerRadius="15"
IsClippedToBounds="True" HorizontalOptions="Center"
BackgroundColor="Gray"
                                       WidthRequest="30"
HeightRequest="30" VerticalOptions="Start" Padding="0"
HasShadow="False">
                                    <Image
x:Name="SellerIcon" Source="{Binding OrderSeller.IconPath}"
VerticalOptions="FillAndExpand"

HorizontalOptions="FillAndExpand" />
                                    </Frame>
                                    <Label Text="{Binding
OrderSeller.Username}" Padding="10,0"/>
                                </FlexLayout>
                                <StackLayout >
                                    <Image x:Name="PostImage"
Source="{Binding SellerPost.ImagePath}"/>
                                    <Label Text="{Binding
SellerPost.Info}" Padding="20,3"/>
                                </StackLayout>
                                <StackLayout
Orientation="Horizontal" Padding="20,5">
                                    <Label Text="Статус
заказа: " FontSize="Header" />
                                    <Label Text="{Binding
OrderStatus}" FontSize="Header" TextColor="{StaticResource
purple}" />
                                </StackLayout>
```

```
                                        <Button Text="Удалить заказ"
Clicked="DeleteButtonClickedAsync" />
                                    </StackLayout>
                                </ViewCell>
                            </DataTemplate>
                        </ListView.ItemTemplate>
                    </ListView>
                </StackLayout>
            </ContentPage.Content>
        </ContentPage>
```

1.18 BuyerOrdersPage.xaml.cs

```
        using System;
        using System.Collections.Generic;
        using System.Linq;
        using System.Net;
        using Newtonsoft.Json;
        using RestSharp;
        using ShopsAggregator.Models;
        using Xamarin.Forms;
        using Xamarin.Forms.Xaml;

        namespace ShopsAggregator.Views
        {
            /// <summary>
            /// Код страницы заказов пользователя.
            /// </summary>
            [XamlCompilation(XamlCompilationOptions.Compile)]
            public partial class BuyerOrdersPage : ContentPage
            {
                /// <summary>
                /// Экземпляр типа пользователя.
                /// </summary>
                private Buyer _buyer;
                /// <summary>
                /// Список заказов пользователя.
                /// </summary>
                private List<BuyerOrderView> buyerOrders = new
        List<BuyerOrderView>();
                /// <summary>
                /// Конструктор страницы.
```

```
        /// </summary>
        /// <param name="buyer">Экземпляр типа пользователя-
покупателя.</param>
        public BuyerOrdersPage(Buyer buyer)
        {
            InitializeComponent();
            _buyer = buyer;
        }

        /// <summary>
        /// Получает заказы пользователя и вызывает базовое
поведение метода.
        /// </summary>
        protected override void OnAppearing()
        {
            UpdateOrdersAsync();
            base.OnAppearing();
        }

        /// <summary>
        /// Открывает страницу пользователя, который
опубликовал запись.
        /// </summary>
        /// <param name="sender">Издатель события -
FlexLayout.</param>
        /// <param name="e">Аргументы события.</param>
        private async void OnPostHeaderTapped(object sender,
EventArgs e)
        {
            if (sender is FlexLayout flexLayout)
            {
                await Navigation.PushAsync(new
WatchSellerPage((flexLayout.Children[1] as
Label).Text,_buyer));
            }
        }

        /// <summary>
        /// Событие обновления заказов.
        /// </summary>
```

```
        /// <param name="sender">Издатель события -
ListView.</param>
        /// <param name="e">Аргументы события.</param>
        private void OnOrdersUpdate(object sender, EventArgs
e)
        {
            UpdateOrdersAsync();
        }


        /// <summary>
        /// Отправляет на сервер запрос, чтобы получить заказы
пользователя.
        /// </summary>
        private async void UpdateOrdersAsync()
        {
            Orders.IsRefreshing = true;
            if (!App.IsConnected())
            {
                await DisplayAlert("Ошибка", "Осутствует
подключение к интернету", "Поробовать снова");
                return;
            }
            RestClient client = new
RestClient($"{App.BaseUrl}api/orders/getBuyerOrders?buyerId={_
buyer.Id}");
            var request = new RestRequest(Method.GET);
            request.AddHeader("Content-Type",
"application/text");
            var response = await client.ExecuteAsync(request);
            if (response.StatusCode ==
HttpStatusCode.BadRequest)
            {
                await DisplayAlert("Ошибка", "Не удалось
получить данные", "Поробовать снова");
                return;
            }

            try
            {
```

```
                buyerOrders =
JsonConvert.DeserializeObject<List<BuyerOrderView>>(response.C
ontent);
            }
            catch (Exception)
            {
                await DisplayAlert("Ошибка", "Ошибка в
обработке данных", "Поробовать снова");
                return;
            }

            Orders.ItemsSource = buyerOrders;
            Orders.IsRefreshing = false;
        }

        /// <summary>
        /// Отменяет нажатие на элемент ListView.
        /// </summary>
        /// <param name="sender">Издатель события -
ListView.</param>
        /// <param name="e">Аргументы события.</param>
        private void OnOrdersItemTapped(object sender,
ItemTappedEventArgs e)
        {
            if (sender is ListView listView)
            {
                listView.SelectedItem = null;
            }
        }

        /// <summary>
        /// Обработчик события удаления заказа.
        /// </summary>
        /// <param name="sender">Издатель события -
Button.</param>
        /// <param name="e">Аргументы события.</param>
        private async void DeleteButtonClickedAsync(object
sender, EventArgs e)
        {
            if (!App.IsConnected())
```

```
            {
                await DisplayAlert("Ошибка", "Осутствует
подключение к интернету", "Поробовать снова");
                return;
            }

            if (sender is Button button)
            {
                BuyerOrderView order = (BuyerOrderView)
(button.ParentView.BindingContext);
                if (order == null)
                    return;
                RestClient client = new
RestClient($"{App.BaseUrl}api/orders/deleteOrder?orderId={orde
r.OrderId}");
                var request = new RestRequest(Method.DELETE);
                request.AddHeader("Content-Type",
"application/text");
                var response = await
client.ExecuteAsync(request);
                if (response.StatusCode ==
HttpStatusCode.BadRequest)
                {
                    await DisplayAlert("Ошибка", "Не удалось
удалить заказ", "Поробовать снова");
                    return;
                }

                BuyerOrderView orderFromList =
                    (from orderList in buyerOrders where
orderList.OrderId == order.OrderId select orderList)
                    .FirstOrDefault();
                buyerOrders.Remove(orderFromList);
            }
        }
    }
}
```

1.19 CreatePostPage.xaml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="ShopsAggregator.Views.CreatePostPage"
             Title="Добавить товар">
    <ContentPage.Content>
        <ScrollView>
            <StackLayout HorizontalOptions="Center">
                <Label FontSize="Large" Text="Добавить новый
товар" HorizontalTextAlignment="Center" />
                <Image x:Name="PostImage" />
                <Button x:Name="GetPostPhoto" Text="Выбрать
фотографию" Clicked="OnGetPostPhotoClicked" />
                <Entry Placeholder="Введите информацию о
товаре" Text="{Binding Info}"/>
                <Button Text="Добавить" Style="{StaticResource
buttonStyle}" Clicked="CreatePostButtonClicked" />
            </StackLayout>
        </ScrollView>
    </ContentPage.Content>
</ContentPage>
```

1.20 CreatePostPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
using Newtonsoft.Json;
using RestSharp;
using ShopsAggregator.Models;
using ShopsAggregator.Services;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace ShopsAggregator.Views
{
    /// <summary>
    /// Код страницы создания записи.
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class CreatePostPage : ContentPage
```

```
    {
        /// <summary>
        /// Экземпляр типа пользователя-покупателя.
        /// </summary>
        private Seller seller;
        /// <summary>
        /// Формы создания записи.
        /// </summary>
        private CreatePostForm form;
        /// <summary>
        /// Конструктор страницы.
        /// </summary>
        /// <param name="seller">Экземпляр типа пользователя-
покупателя.</param>
        public CreatePostPage(Seller seller)
        {
            InitializeComponent();
            this.seller = seller;
            form = new CreatePostForm(seller.Id);
            this.BindingContext = form;
        }

        /// <summary>
        /// Позволяет пользователю выбрать фотографию после
нажатия на кнопку.
        /// </summary>
        /// <param name="sender">Издатель события -
Button.</param>
        /// <param name="e">Аргументы события.</param>
        private async void OnGetPostPhotoClicked(object
sender, EventArgs e)
        {
            try
            {
                String imagePath = String.Empty;
                await CrossMedia.Current.Initialize();
                if (CrossMedia.Current.IsPickPhotoSupported)
                {
                    MediaFile photo = await
CrossMedia.Current.PickPhotoAsync();
```

```
                    imagePath = photo.Path;
                    PostImage.Source =
ImageSource.FromFile(imagePath);
                    GetImageBytesFromPath(imagePath, form);
                    if (!App.IsConnected())
                    {
                        return;
                    }

                    GetPostPhoto.Text = "Изменить фотографию";
                }
            }
            catch (Exception)
            {
                await DisplayAlert("Ошибка", "Что-то пошло не
так", "Попробовать снова");
            }
        }

/// <summary>
        /// Получает байты изображения, выбранного
пользователем.
        /// </summary>
        /// <param name="path">Путь к фотографии.</param>
        /// <param name="form">Форма создания записи.</param>
        private void GetImageBytesFromPath(String path,
CreatePostForm form)
        {
            List<Int32> bytes = new List<Int32>();
            using (FileStream fs = new FileStream(path,
FileMode.Open, FileAccess.Read))
            {
                while(fs.Position != fs.Length)
                    bytes.Add(fs.ReadByte());
            }

            form.ImageBytes= bytes.ToList();
        }
        /// <summary>
```

```
        /// Обработчик события нажатия на кнопку создания
записи.
        /// </summary>
        /// <param name="sender">Издатель события -
Button.</param>
        /// <param name="e">Аргументы события.</param>
        private async void CreatePostButtonClicked(object
sender, EventArgs e)
        {
            if (form.ImageBytes.Count == 0)
            {
                await DisplayAlert("Ошибка", "Выберите
фотографию", "Выбрать");
                return;
            }

            if (!App.IsConnected())
            {
                await DisplayAlert("Нет подключения к
интернету", "Проверьте подключение к интернету", "Попробовать
снова");
                return;
            }

            if (form.CreatorId == 0)
            {
                await DisplayAlert("Ошибка в данных
пользователя", "Зайдите снова, чтобы ее устранить", "Ок");
                return;
            }
            var client = new
RestClient(App.BaseUrl+"api/posts/create");
            client.Timeout = -1;
            var request = new RestRequest(Method.POST);
            request.AddHeader("Content-Type",
"application/json");
            String json = JsonConvert.SerializeObject(form);
            request.AddParameter("application/json", json,
ParameterType.RequestBody);
```

```
            IRestResponse response = await
client.ExecuteAsync(request);
            if (response.StatusCode ==
HttpStatusCode.BadRequest)
            {
                await DisplayAlert("Ошибка", "Не удалось
добавить запись", "Попробовать снова");
                return;
            }

            await DisplayAlert("Запись успешно добавлена", "",
"Хорошо");
        }
    }
}
```

1.21 RegistrationPage.xaml

```
<?xml version="1.0" encoding="utf-8"?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="ShopsAggregator.Views.RegistrationPage">
    <ContentPage.Content>
        <StackLayout Margin="0,40,0,0" Padding="30,0"
                     VerticalOptions="FillAndExpand">
            <StackLayout  x:Name="Layout"
VerticalOptions="Center" HorizontalOptions="CenterAndExpand">
                <Label FontAttributes="Bold" FontSize="34"
Text="Регистрация" Margin="0,0,0,15"/>
                <Label FontSize="Medium"
Margin="0,10,0,0">Введите свой активный email</Label>
                <Entry x:Name="EmailEntry" Keyboard="Email"
IsPassword="False"
                       Placeholder="example@example.com"
Text="{Binding Email}"/>
                <Label FontSize="Medium"
Margin="0,10,0,0">Пользователи будут видеть вас под этим
именем</Label>
                <Entry x:Name="UsernameEntry" Keyboard="Text"
IsPassword="False"
```

```
                        Placeholder="nickname" Text="{Binding
Username}"/>
                <Label FontSize="Medium"
Margin="0,10,0,0">Придумайте надежный пароль</Label>
                <Entry x:Name="PasswordEntry"
IsPassword="True" Placeholder="########"
                        Text="{Binding Password}"/>
                <Label FontSize="Medium"
Margin="0,10,0,0">Введите пароль повторно</Label>
                <Entry x:Name="PasswordAgainEntry"
IsPassword="True" Placeholder="########" Text=""/>
                <Label FontSize="Medium"
Margin="0,10,0,0">Хотите быть продавцом?</Label>
                <Switch x:Name="IsSeller" IsToggled="{Binding
IsSeller}"/>
                <StackLayout
HorizontalOptions="StartAndExpand" Orientation="Horizontal"
Margin="0,10">
                    <CheckBox x:Name="CheckBox"
HorizontalOptions="Start" CheckedChanged="CheckBoxChanged"
Color="{StaticResource purple}"/>
                    <Label x:Name="CheckBoxText"
HorizontalOptions="FillAndExpand" Text="Согласен с политикой
конфиденциальности и условиями пользования"
FontSize="Medium"/>
                </StackLayout>
                <Button FontSize="Medium"
Style="{StaticResource buttonStyle}" Text="Зарегистрироваться"
Clicked="OnRegistrationButtonCLicked"/>
                <Label x:Name="ErrorRegistration"
FontSize="Medium" TextColor="{StaticResource errorColor}"
HorizontalTextAlignment="Center" IsVisible="False"/>
            </StackLayout>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

1.22 RegistrationPage.xaml.cs

```
using System;
using System.Globalization;
using System.Net;
```

```
using System.Text.RegularExpressions;
using Newtonsoft.Json;
using RestSharp;
using ShopsAggregator.Models;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace ShopsAggregator.Views
{
    /// <summary>
    /// Код страницы регистрации пользователя.
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class RegistrationPage : ContentPage
    {
        /// <summary>
        /// Текст сообщения о неверном Email.
        /// </summary>
        private const String IncorrectEmailMessage = "Email
должен иметь вид example@example.com";
        /// <summary>
        /// Текст сообщения о неверном Username.
        /// </summary>
        private const String IncorrectUsernameMessage = "Имя
пользователя может содержать только латинские символы или
цифры";
        /// <summary>
        /// Текст сообщения о неверном пароле.
        /// </summary>
        private const String IncorrectPasswordMessage=
            "Пароль должен содержать не менее 8 символов и
состоять из заглавных и строчных латинских букв и цифр";
        /// <summary>
        /// Текст сообщения о неверном повторном пароле.
        /// </summary>
        private const String IncorrectPasswordAgainMessage =
"Введеные пароли не совпадают";
        /// <summary>
        /// Заголовок сообщения о неудачном создании аккаунта.
        /// </summary>
```

```
        private const String BadAccountCreateAlertTitle = "Не
удалось создать акканут";
        /// <summary>
        /// Текст кнопки сообщения о неудачном создании
аккаунта.
        /// </summary>
        private const String BadAccountCreateAlertCancelText =
"Попробовать снова";
        /// <summary>
        /// Текст сообщения о неудачном создании аккаунта.
        /// </summary>
        private const String BadRequestAlertMessage = "Ошибка
подключения к серверу";
        /// <summary>
        /// Текст кнопки сообщения об удачном создании
аккаунта.
        /// </summary>
        private const String SuccessAccountCreateTitle =
"Поздравляю!!";
        /// <summary>
        /// Текст сообщения об удачном создании аккаунта.
        /// </summary>
        private const String SuccessAccountCreateMessage =
"Аккаунт успешно создан";
        /// <summary>
        /// Заголовок сообщения об удачном создании аккаунта.
        /// </summary>
        private const String SuccessAccountCreateCancel =
"Ура";
        /// <summary>
        /// Модель регистрации пользователя.
        /// </summary>
        private RegistrationForm form = new
RegistrationForm();
        /// <summary>
        /// Конструктор страницы.
        /// </summary>
        public RegistrationPage()
        {
            InitializeComponent();
```

```
            CheckBox.CheckedChanged += (sender, args) =>
CheckBox.Color = (Color)App.Current.Resources["purple"];
            this.BindingContext = form;
        }

        /// <summary>
        /// Проверяет все ли поля заполнены верно и вызывает
метод отправки формы на сервер.
        /// </summary>
        /// <param name="sender">Издатель события -
Button.</param>
        /// <param name="e">Аргументы события.</param>
        private async void OnRegistrationButtonCLicked(object
sender, EventArgs e)
        {
            if (!App.IsConnected())
            {
                await DisplayAlert("Ошибка", "Осутствует
подключение к интернету", "Поробовать снова");
                return;
            }
            if (!IsBoxesCorrect())
            {
                return;
            }

            Boolean result = false;
            try
            {
                result = SendRegistrationPost(form);
            }
            catch (Exception)
            {
                await
DisplayAlert(BadAccountCreateAlertTitle,BadRequestAlertMessage
, BadAccountCreateAlertCancelText);
            }
            if (!result)
                return;
```

```
            await DisplayAlert(SuccessAccountCreateTitle,
SuccessAccountCreateMessage, SuccessAccountCreateCancel);
            await Navigation.PopAsync();
        }

        /// <summary>
        /// Отправляет на сервер запрос создания аккаунта.
        /// </summary>
        /// <param name="user">Модель регистрацции
пользователя, которая отправляется на сервер.</param>
        /// <returns>Результат отправки запроса на
сервер.</returns>
        private Boolean SendRegistrationPost(RegistrationForm
user)
        {
            String json = JsonConvert.SerializeObject(user);
            var client = new
RestClient($"{App.ServerUrl}reg");
            client.Timeout = -1;
            var request = new RestRequest(Method.POST);
            request.AddHeader("Content-Type",
"application/json");
            request.AddParameter("application/json", json,
ParameterType.RequestBody);
            IRestResponse response = client.Execute(request);
            if (response.StatusCode ==
HttpStatusCode.BadRequest)
            {
                DisplayAlert(BadAccountCreateAlertTitle,
response.Content, BadAccountCreateAlertCancelText);
                return false;
            }

            return true;
        }

        /// <summary>
        /// Изменяет цвет CheckBox и его текста при изменении
полей ввода данных.
        /// </summary>
```

```
        /// <param name="sender">Издатель события -
Entry.</param>
        /// <param name="e">Аргументы события.</param>
        private void CheckBoxChanged(object sender, EventArgs
e)
        {
            CheckBox.Color = CheckBoxText.TextColor =
Color.Default;
        }

        /// <summary>
        /// Проверяет все ли поля для заполнения заполнены
верно.
        /// </summary>
        /// <returns>Результат проверки.</returns>
        private Boolean IsBoxesCorrect()
        {
            if (!IsTextboxCorrect(EmailEntry, ValidateEmail))
            {
                MessageAboutIncorrectBox(EmailEntry,
IncorrectEmailMessage);
                return false;
            }

            if (!IsTextboxCorrect(UsernameEntry,
                (Char symb) => { return symb >= 'A' && symb <=
'z' || Char.IsDigit(symb); }))
            {
                MessageAboutIncorrectBox(UsernameEntry,
IncorrectUsernameMessage);
                return false;
            }

            if (!IsTextboxCorrect(PasswordEntry,
ValidatePassword))
            {
                MessageAboutIncorrectBox(PasswordEntry,
IncorrectPasswordMessage);
                return false;
            }
```

```
            if (!IsTextboxCorrect(PasswordAgainEntry, (String
pwd) => pwd == PasswordEntry.Text))
            {
                MessageAboutIncorrectBox(PasswordAgainEntry,
IncorrectPasswordAgainMessage);
                App.Shake(PasswordEntry);
                return false;
            }
            if (!CheckBox.IsChecked)
            {
                CheckBoxText.TextColor = CheckBox.Color =
Color.Red;
                App.Shake(CheckBoxText);
                App.Shake(CheckBox);

                CheckBox.CheckedChanged += (sender, e) =>
CheckBoxText.TextColor = Color.Default;
                return false;
            }
            return true;
        }

        /// <summary>
        /// Встряхивает неверное поле с данными и выводит
соответствующее сообщение.
        /// </summary>
        /// <param name="view">Невернре поле данных.</param>
        /// <param name="errorMsg">Сообщени об ошибке.</param>
        private void MessageAboutIncorrectBox(Entry view,
String errorMsg)
        {
            App.Shake(view);
            ErrorRegistration.Text = errorMsg;
            ErrorRegistration.IsVisible = true;
            view.TextChanged += OnEntryTextChanged;
        }

        /// <summary>
        /// Событие об изменении Entry.
```

```
        /// </summary>
        /// <param name="sender">Издатель события -
Entry.</param>
        /// <param name="e">Аргументы события.</param>
        private void OnEntryTextChanged(object sender,
TextChangedEventArgs e)
        {
            if (sender is Entry entry)
            {
                ErrorRegistration.IsVisible = false;
                entry.TextChanged -= OnEntryTextChanged;
            }
        }

        /// <summary>
        /// Проверяет введенные данные.
        /// </summary>
        /// <param name="box">Entry для проверки.</param>
        /// <param name="func">Метод проверки данных.</param>
        /// <returns>Результат проверки.</returns>
        private Boolean IsTextboxCorrect(Entry box, Func<Char,
Boolean> func)
        {
            if (String.IsNullOrWhiteSpace(box.Text))
                return false;
            Char[] symbols = box.Text.ToCharArray();
            return Array.TrueForAll(symbols, symb =>
func(symb));
        }

        /// <summary>
        /// Проверяет введенные данные.
        /// </summary>
        /// <param name="box">Entry для проверки.</param>
        /// <param name="func">Метод проверки данных.</param>
        /// <returns>Результат проверки.</returns>
        private Boolean IsTextboxCorrect(Entry box,
Func<String, Boolean> func)
        {
            if (String.IsNullOrWhiteSpace(box.Text))
```

```
                return false;

            return func(box.Text);
        }

        /// <summary>
        /// Проверяет, является ли введеные email верным.
        /// </summary>
        /// <param name="email">Строка с email.</param>
        /// <returns>Результат проверки.</returns>
        private Boolean ValidateEmail(String email)
        {
            if (string.IsNullOrWhiteSpace(email))
                return false;

            try
            {
                email = Regex.Replace(email, @"(@)(.+)$",
DomainMapper,
                    RegexOptions.None,
TimeSpan.FromMilliseconds(200));

                string DomainMapper(Match match)
                {

                    var idn = new IdnMapping();

                    var domainName =
idn.GetAscii(match.Groups[2].Value);

                    return match.Groups[1].Value + domainName;
                }
            }
            catch (RegexMatchTimeoutException e)
            {
                return false;
            }
            catch (ArgumentException e)
            {
                return false;
```

```
            }

            try
            {
                return Regex.IsMatch(email,
                    @"^(?(""")(""".+?(?<!\\)""@)|(([0-9a-
z]((\.(?!\.))|[-!#\$%&'\*\+/=\?\^`\{\}\|~\w])*)(?<=[0-9a-
z])@))" +
                    @"(?(\[)(\[(\d{1,3}\.){3}\d{1,3}\])|(([0-
9a-z][-0-9a-z]*[0-9a-z]*\.)+[a-z0-9][\-a-z0-9]{0,22}[a-z0-
9]))$",
                    RegexOptions.IgnoreCase,
TimeSpan.FromMilliseconds(250));
            }
            catch (RegexMatchTimeoutException)
            {
                return false;
            }
        }

        /// <summary>
        /// Проверяет является ли введеный пароль допустимым.
        /// </summary>
        /// <param name="input">Введеный пароль.</param>
        /// <returns>Результат проверки.</returns>
        private Boolean ValidatePassword(String input)
        {
            var hasNumber = new Regex(@"[0-9]+");
            var hasUpperChar = new Regex(@"[A-Z]+");
            var hasMinimum8Chars = new Regex(@".{8,}");

            return hasNumber.IsMatch(input) &&
hasUpperChar.IsMatch(input) &&
hasMinimum8Chars.IsMatch(input);
        }

    }
}
```

1.23 SearchPage.xaml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="ShopsAggregator.Views.SearchPage"
             Title="Поиск">
    <ContentPage.Content>
        <StackLayout>
            <SearchBar Placeholder="Введите имя пользователя"
SearchButtonPressed="OnSearchButtonPressed"/>
            <ListView x:Name="UsersListView"
ItemSelected="OnSearchItemSelected">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <TextCell Text="{Binding .}" />
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

1.24 SearchPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Newtonsoft.Json;
using RestSharp;
using ShopsAggregator.Models;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace ShopsAggregator.Views
{
    /// <summary>
    /// Код страницы поиска пользователей-продавцов.
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class SearchPage : ContentPage
    {
        /// <summary>
```

```
        /// Заголовок сообщения о неудачном запросе к серверу.
        /// </summary>
        private const String connectionAlertTitle =
"Невозможно сделать запрос";
        /// <summary>
        /// Текст сообщения об отсутствии интернета.
        /// </summary>
        private String connectionAlertContent = "Отсутствует
подключение к интернету";
        /// <summary>
        /// Текст кнопки сообщения.
        /// </summary>
        private const String alertCancel = "Попробовать
снова";
        /// <summary>
        /// Заголовок сообщения о неудачном запросе к серверу.
        /// </summary>
        private const String getResponseAlertTitle = "Ошибка
запроса";
        /// <summary>
        /// Текст сообщения о неудачном получении данных.
        /// </summary>
        private const String getResponseAlertContent = "Не
удалось получить данные";
        /// <summary>
        /// Список найденных пользователей.
        /// </summary>
        private List<String> searchedSellers;
        /// <summary>
        /// Экземпляр типа пользователя-покупателя.
        /// </summary>
        private Buyer _buyer;
        /// <summary>
        /// Конструктор страницы.
        /// </summary>
        /// <param name="buyer">Экземпляр типа пользователя-
покупателя.</param>
        public SearchPage(Buyer buyer)
        {
            InitializeComponent();
```

```
            _buyer = buyer;
        }

        /// <summary>
        /// Обработчик события поиска пользователя по имени.
        /// </summary>
        /// <param name="sender">Издатель события -
SearchBar.</param>
        /// <param name="e">Аргументы события.</param>
        private async void OnSearchButtonPressed(object
sender, EventArgs e)
        {
            if (!App.IsConnected())
            {
                await DisplayAlert(connectionAlertTitle,
connectionAlertContent, alertCancel);
                return;
            }
            if (sender is SearchBar searchBar)
            {
                if (String.IsNullOrWhiteSpace(searchBar.Text))
                    return;
                Search(searchBar.Text);
            }
        }

        /// <summary>
        /// Вызывает метод с запросом на сервер, получает
результат работы сервера и обрабатывает их.
        /// </summary>
        /// <param name="searchText">Текст запроса
пользователя.</param>
        private async void Search(String searchText)
        {
            String responseContent = String.Empty;
            try
            {
                responseContent = await
TryGetSearchResult(searchText);
            }
```

```
            catch (Exception)
            {
                await DisplayAlert(getResponseAlertTitle,
getResponseAlertContent, alertCancel);
                return;
            }

            List<String> sellers;
            try
            {
                sellers =
JsonConvert.DeserializeObject<List<String>>(responseContent);
            }
            catch (Exception)
            {
                await DisplayAlert(getResponseAlertTitle,
getResponseAlertContent, alertCancel);
                return;
            }

            searchedSellers = sellers;
            UsersListView.ItemsSource = sellers;
        }

        /// <summary>
        /// Отправляет запрос на сервер для поиска
пользователя.
        /// </summary>
        /// <param name="searchLine">Строка поиска
пользователя.</param>
        /// <returns>Результат запроса.</returns>
        private async Task<String> TryGetSearchResult(String
searchLine)
        {
            var client = new
RestClient($"{App.BaseUrl}api/search?q={searchLine}");
            client.Timeout = -1;
            var request = new RestRequest(Method.GET);
            IRestResponse response = await
client.ExecuteAsync(request);
```

```
            return response.Content;
        }

        /// <summary>
        /// Обработчик события нажатия на элемент списка
найденных пользователей.
        /// </summary>
        /// <param name="sender">Издатель события -
ListView.</param>
        /// <param name="e">Аргументы события.</param>
        private async void OnSearchItemSelected(object sender,
SelectedItemChangedEventArgs e)
        {
            if (sender is ListView list)
            {
                if (searchedSellers == null)
                    return;
                Int32 index = e.SelectedItemIndex;
                if (index < 0 || index >=
searchedSellers.Count)
                    return;
                list.SelectedItem = null;
                await Navigation.PushAsync(new
WatchSellerPage(searchedSellers[index], _buyer));
            }
        }
    }
}
```

1.25 SellerMainPage.xaml

```
<?xml version="1.0" encoding="utf-8"?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="ShopsAggregator.Views.SellerMainPage"
             Title="Аккаунт">
    <ContentPage.ToolbarItems>
        <ToolbarItem Text="Выйти"
Clicked="OnExitToolbarItemTapped" />
    </ContentPage.ToolbarItems>
```

```
    <ContentPage.Content>
        <RefreshView x:Name="RefreshView">
        <ScrollView x:Name="ScrollView">
        <StackLayout Margin="0,10,0,0">
            <StackLayout x:Name="InfoStack" Padding="50,0">
                <Frame CornerRadius="100"
IsClippedToBounds="True" HorizontalOptions="Center"
BackgroundColor="Gray"
                       WidthRequest="200" HeightRequest="200"
VerticalOptions="Start" Padding="0" HasShadow="True"
Margin="0,20">
                    <Frame.GestureRecognizers>
                        <TapGestureRecognizer
Tapped="GetPhoto"/>
                    </Frame.GestureRecognizers>
                    <StackLayout HorizontalOptions="Center"
VerticalOptions="Center">
                        <Image x:Name="Icon"
HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand" Aspect="AspectFill"
                               Source="{Binding IconPath}"/>
                        <Label x:Name="ImagePickerText"
Text="Нажмите чтобы выбрать фотографию"
TextColor="{StaticResource cloudWhite}"
                               FontSize="Small"
HorizontalTextAlignment="Center"
VerticalTextAlignment="Center"/>
                    </StackLayout>
                </Frame>
                <StackLayout x:Name="UserInfoStack">
                    <Label Text="{Binding Username}"
FontSize="Large" HorizontalTextAlignment="Center" />
                    <Label Text="{Binding Email}"
TextColor="{StaticResource grey}" FontSize="Small"
HorizontalTextAlignment="Center"/>
                    <Editor x:Name="InfoEditor" Text="{Binding
Info}" Placeholder="Расскажите покупателям о себе"

Unfocused="VisualElement_OnUnfocused"/>
                </StackLayout>
```

```
                <StackLayout
HorizontalOptions="FillAndExpand">
                    <Label x:Name="SubscribersCounter"
FontSize="Large" HorizontalTextAlignment="Center"/>
                    <Label Text="Подписчики" FontSize="Medium"
TextColor="{StaticResource grey}"
HorizontalTextAlignment="Center"/>
                </StackLayout>
            </StackLayout>
            <Label FontSize="Title"
HorizontalTextAlignment="Center" Padding="0,15">Мои
товары</Label>
            <ListView x:Name="Posts"
CachingStrategy="RecycleElementAndDataTemplate"
HasUnevenRows="True"
                      ItemSelected="Posts_OnItemSelected"
SeparatorVisibility="None">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <StackLayout x:Name="StackLayout"
Margin="0,0,0,10">
                                <Image x:Name="PostImage"
Source="{Binding ImagePath}"/>
                                <Label Text="{Binding Info}"
Padding="30, 0" />
                                <StackLayout
Orientation="Horizontal">
                                    <Label Text="Понравилось:
" FontSize="Body"/>
                                    <Label Text="{Binding
LikesCount}" FontSize="Body" TextColor="{StaticResource
purple}"/>
                                </StackLayout>
                            </StackLayout>
                        </ViewCell>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
        </StackLayout>
```

```
            </ScrollView>
            </RefreshView>
        </ContentPage.Content>
    </ContentPage>
```

1.26 SellerMainPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using Newtonsoft.Json;
using Plugin.Media;
using Plugin.Media.Abstractions;
using RestSharp;
using ShopsAggregator.Models;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace ShopsAggregator.Views
{
    /// <summary>
    /// Код страницы пользователя-продавца.
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class SellerMainPage : ContentPage
    {
        /// <summary>
        /// Заголовок сообщения об отсутствии подключения.
        /// </summary>
        private const String
FailedConnectionToServerAlertTitle = "Ошибка подключения к
серверу";
        /// <summary>
        /// Экземпляр типа пользователя-продавца.
        /// </summary>
        private Seller _seller;
        /// <summary>
        /// Список записей пользователя.
        /// </summary>
        private List<Post> sellerPosts = new List<Post>();
```

```
        /// <summary>
        /// Конструктор страницы.
        /// </summary>
        /// <param name="seller">Экземпляр типа пользователя-
продавца.</param>
        public SellerMainPage(Seller seller)
        {
            InitializeComponent();
            this._seller = seller;
            this.BindingContext = seller;
            RefreshView.Refreshing += PostsOnRefreshing;
            Posts.WidthRequest = App.Current.MainPage.Width;
        }


        /// <summary>
        /// Обработчик события обновления данных.
        /// </summary>
        /// <param name="sender">Издатель события -
RefreshView.</param>
        /// <param name="e">Аргументы события.</param>
        private void PostsOnRefreshing(object sender,
EventArgs e)
        {
            GetSellerPosts(_seller.Username);
        }

        /// <summary>
        /// Получает записи пользователя и вызывает базовое
поведение метода.
        /// </summary>
        protected override void OnAppearing()
        {
            SubscribersCounter.Text =
_seller.Subscribers.Count.ToString();
            GetSellerPosts(_seller.Username);
            base.OnAppearing();
        }
```

```
        /// <summary>
        /// Получает записи пользователя с сервера.
        /// </summary>
        /// <param name="sellerName">Имя пользователя.</param>
        private async void GetSellerPosts(String sellerName)
        {
            if (!App.IsConnected())
            {
                await DisplayAlert("Нет подключения к
интернету", "Проверьте подключение к интернету", "Попробовать
снова");
                return;
            }
            RefreshView.IsRefreshing = true;
            var client = new
RestClient($"{App.BaseUrl}api/posts/getSellerPosts?sellerName=
{sellerName}");
            client.Timeout = -1;
            var request = new RestRequest(Method.GET);
            request.AddHeader("Content-Type",
"application/text");
            IRestResponse response = await
client.ExecuteAsync(request);
            if (response.StatusCode ==
HttpStatusCode.BadRequest)
            {
                return;
            }

            try
            {
                sellerPosts =
JsonConvert.DeserializeObject<List<Post>>(response.Content);
            }
            catch(Exception e){}
            Posts.ItemsSource = sellerPosts;
            RefreshView.IsRefreshing = false;
        }

        /// <summary>
```

```
        /// Обработчик нажатия на кнопку выхода.
        /// </summary>
        /// <param name="sender">Издатель события -
Button.</param>
        /// <param name="e">Аргументы события.</param>
        private void OnExitToolbarItemTapped(object sender,
EventArgs e)
        {
            Navigation.PopAsync();
        }

        /// <summary>
        /// Отправляет запрос на сервер для обновления
фотографии пользователя.
        /// </summary>
        /// <param name="form">Модель формы с информацией о
фотографии.</param>
        private async void SendUpdateUserPut(IconPostForm
form)
        {
            if (!App.IsConnected())
            {
                return;
            }
            String json = JsonConvert.SerializeObject(form);
            var client = new
RestClient($"{App.ServerUrl}addSellerIcon");
            client.Timeout = -1;
            var request = new RestRequest(Method.PUT);
            request.AddHeader("Content-Type",
"application/json");
            request.AddParameter("application/json", json,
ParameterType.RequestBody);
            IRestResponse response = await
client.ExecuteAsync(request);
            if (response.StatusCode ==
HttpStatusCode.RequestEntityTooLarge)
            {
```

```
                await
DisplayAlert(FailedConnectionToServerAlertTitle,
response.Content, "Попробовать снова");
                return;
            }

            if (response.StatusCode ==
HttpStatusCode.BadRequest)
            {
                await
DisplayAlert(FailedConnectionToServerAlertTitle,
response.Content, "Попробовать снова");
                return;
            }

            _seller.IconPath = _seller.Username + "icon.jpeg";
        }

        /// <summary>
        /// Обработчик события получения пользователем
фотографии.
        /// </summary>
        /// <param name="sender">Издатель события -
Frame.</param>
        /// <param name="e">Аргументы события.</param>
        private async void GetPhoto(object sender, EventArgs
e)
        {
            String iconPath = String.Empty;
            await CrossMedia.Current.Initialize();
            if (CrossMedia.Current.IsPickPhotoSupported &&
CrossMedia.IsSupported)
            {
                MediaFile photo = await
CrossMedia.Current.PickPhotoAsync();
                if (photo == null)
                    return;
                iconPath = photo.Path;
                Icon.Source =
ImageSource.FromFile(photo.Path);
```

```
                IconPostForm form = new
IconPostForm(_seller.Id);
                GetPhotoBytes(iconPath, form);
                if (!App.IsConnected())
                {
                    return;
                }
                SendUpdateUserPut(form);
            }        }

        /// <summary>
        /// Получает байты фотографии.
        /// </summary>
        /// <param name="path">Путь к файлу с
фотографией.</param>
        /// <param name="form">Форма для запроса, в которую
записываются байты фотографии.</param>
        private void GetPhotoBytes(String path, IconPostForm
form)
        {
            List<Int32> bytes = new List<Int32>();
            using (FileStream fs = new FileStream(path,
FileMode.Open))
            {
                while(fs.Position != fs.Length)
                    bytes.Add(fs.ReadByte());
            }

            form.IconBytesArr = bytes.ToList();
        }

        /// <summary>
        /// Отменяет событие нажатия на элемент из ListView.
        /// </summary>
        /// <param name="sender">Издатель события -
ListView.</param>
        /// <param name="e">Аргументы события.</param>
        private void Posts_OnItemSelected(object sender,
SelectedItemChangedEventArgs e)
        {
```

```
            if (sender is ListView listView)
            {
                listView.SelectedItem = null;
            }
        }

        /// <summary>
        /// Отправляет запрос, чтобы изменить информацию о
пользователе.
        /// </summary>
        /// <param name="sender">Издатель события -
Editor.</param>
        /// <param name="e">Аргументы события.</param>
        private async void VisualElement_OnUnfocused(object
sender, FocusEventArgs e)
        {
            if (!App.IsConnected() ||
String.IsNullOrWhiteSpace(_seller.Info))
            {
                return;
            }
            RestClient client = new
RestClient($"{App.BaseUrl}api/users/setSellerInfo?sellerId={_s
eller.Id}&info={_seller.Info}");
            var request = new RestRequest(Method.PUT);
            request.AddHeader("Content-Type",
"application/json");
            var response = await client.ExecuteAsync(request);
        }
    }
}
```

1.27 SellerOrdersPage.xaml

```
<?xml version="1.0" encoding="utf-8"?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="ShopsAggregator.Views.SellerOrdersPage"
             Title="Заказы пользователей">
    <ContentPage.Content>
```

```
        <StackLayout>
            <ListView x:Name="Orders"
SeparatorVisibility="None" IsPullToRefreshEnabled="True"
Refreshing="OnOrdersUpdate"
                      HasUnevenRows="True"
ItemTapped="OnOrdersItemTapped">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <StackLayout Padding="0,5"
Margin="0,0,0,10">
                                <Label Text="{Binding
OrderId}" IsVisible="False" />
                                <StackLayout
Margin="0,0,0,10">
                                    <Image x:Name="PostImage"
Source="{Binding CurrentPost.ImagePath}"/>
                                    <Label Text="{Binding
CurrentPost.Info}" Padding="20,3"/>
                                </StackLayout>
                                <StackLayout
Orientation="Horizontal" Margin="0,0,0,10" Padding="20,5">
                                    <Label Text="Заказчик:"
FontSize="Header"/>
                                    <StackLayout>
                                        <StackLayout
Orientation="Horizontal">
                                            <Label Text="Имя
пользователя: " TextColor="{StaticResource purple}"
FontSize="Body"/>
                                            <Label
Text="{Binding OrderBuyer.Username}" FontSize="Body" />
                                        </StackLayout>
                                        <StackLayout
Orientation="Horizontal">
                                            <Label
Text="Email: " TextColor="{StaticResource purple}"
FontSize="Body"/>
                                            <Label
Text="{Binding OrderBuyer.Email}" FontSize="Body" />
```

```
                                        </StackLayout>
                                        <StackLayout
Orientation="Horizontal">
                                            <Label
Text="Информация о пользователе: " TextColor="{StaticResource
purple}" FontSize="Body"/>
                                            <Label
Text="{Binding OrderBuyer.Info}" FontSize="Body" />
                                        </StackLayout>
                                    </StackLayout>
                                </StackLayout>
                                <StackLayout
Orientation="Horizontal" Padding="20,5">
                                    <Label Text="Статус
заказа: " FontSize="Header" />
                                    <Label Text="{Binding
OrderStatus}" FontSize="Header" TextColor="{StaticResource
purple}" />
                                </StackLayout>
                                <FlexLayout
JustifyContent="SpaceAround">
                                    <Button
BackgroundColor="{StaticResource successColor}"
TextColor="{StaticResource cloudWhite}"
                                            Text="Подтвердить
заказ" Padding="10,0" Clicked="SetOrderSuccess"/>
                                    <Button
BackgroundColor="{StaticResource errorColor}" Text="Отклонить
заказ"

TextColor="{StaticResource cloudWhite}" Padding="10,0"
Clicked="SetOrderCanceledStatus"/>
                                </FlexLayout>
                            </StackLayout>
                        </ViewCell>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
        </StackLayout>
    </ContentPage.Content>
```

```
</ContentPage>
```

1.28 SellerOrdersPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Net;
using Newtonsoft.Json;
using RestSharp;
using ShopsAggregator.Models;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace ShopsAggregator.Views
{
    /// <summary>
    /// Код страницы заказы пользователя-продавца.
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class SellerOrdersPage : ContentPage
    {
        /// <summary>
        /// Экземпляр типа пользователя-продавца.
        /// </summary>
        private readonly Seller _seller;
        /// <summary>
        /// Список заказов пользователя-продавца.
        /// </summary>
        private List<SellerOrderView> sellerOrders = new
List<SellerOrderView>();

        /// <summary>
        /// Конструктор страницы.
        /// </summary>
        /// <param name="seller">Экземпляр типа пользователя-
продавца.</param>
        public SellerOrdersPage(Seller seller)
        {
            _seller = seller;
            InitializeComponent();
        }
```

Изм.	Лист	№ док.	Подп.	Дата
RU.17701729.04.01-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам инв. №	Инв. № дубл.	Подп. и дата

```
        /// <summary>
        /// Вызывает метод получения заказов пользователя и
вызывает базовое поведение метода.
        /// </summary>
        protected override void OnAppearing()
        {
            UpdateOrdersAsync();
            base.OnAppearing();
        }

        /// <summary>
        /// Обработчик события обновления заказов.
        /// </summary>
        /// <param name="sender">Издатель события -
ListView.</param>
        /// <param name="e">Аргументы события.</param>
        private void OnOrdersUpdate(object sender, EventArgs
e)
        {
            UpdateOrdersAsync();
        }

        /// <summary>
        /// Обновляет заказы пользователя.
        /// </summary>
        private async void UpdateOrdersAsync()
        {
            Orders.IsRefreshing = true;
            if (!App.IsConnected())
            {
                await DisplayAlert("Ошибка", "Осутствует
подключение к интернету", "Поробовать снова");
                return;
            }
            RestClient client = new
RestClient($"{App.BaseUrl}api/orders/getSellerOrders?sellerId=
{_seller.Id}");
            var request = new RestRequest(Method.GET);
            request.AddHeader("Content-Type",
"application/text");
```

```
            var response = await client.ExecuteAsync(request);
            if (response.StatusCode ==
HttpStatusCode.BadRequest)
            {
                await DisplayAlert("Ошибка", "Не удалось
получить данные", "Поробовать снова");
                return;
            }

            try
            {
                sellerOrders =
JsonConvert.DeserializeObject<List<SellerOrderView>>(response.
Content);
            }
            catch (Exception)
            {
                await DisplayAlert("Ошибка", "Ошибка в
обработке данных", "Поробовать снова");
                return;
            }

            Orders.ItemsSource = sellerOrders;
            Orders.IsRefreshing = false;
        }

        /// <summary>
        /// Отменяет событие нажатия на элемент из ListView.
        /// </summary>
        /// <param name="sender">Издатель события -
ListView.</param>
        /// <param name="e">Аргументы события.</param>
        private void OnOrdersItemTapped(object sender,
ItemTappedEventArgs e)
        {
            if (sender is ListView listView)
            {
                listView.SelectedItem = null;
            }
        }
```

```
        /// <summary>
        /// Ставит статус заказа одобреным и отправляет запрос
на сервер.
        /// </summary>
        /// <param name="sender">Издатель события -
Button.</param>
        /// <param name="e">Аргументы события.</param>
        private void SetOrderSuccess(object sender, EventArgs
e)
        {
            if (sender is Button button)
            {
                Int32 orderId = ((SellerOrderView)
(button.ParentView.BindingContext)).OrderId;
                SendOrderStatusAsync("setOrderSuccess",
orderId);
                foreach (SellerOrderView sellerOrderView in
sellerOrders)
                {
                    if (sellerOrderView.OrderId == orderId)
                        sellerOrderView.IsSucceded = true;
                }
            }
        }

        /// <summary>
        /// Ставит статус заказа отклоненым и отправляет
запрос на сервер.
        /// </summary>
        /// <param name="sender">Издатель события -
Button.</param>
        /// <param name="e">Аргументы события.</param>
        private void SetOrderCanceledStatus(object sender,
EventArgs e)
        {
            if (sender is Button button)
            {
                Int32 orderId = ((SellerOrderView)
(button.ParentView.BindingContext)).OrderId;
```

```
                SendOrderStatusAsync("setOrderCanceled",
orderId);
                foreach (SellerOrderView sellerOrderView in
sellerOrders)
                {
                    if (sellerOrderView.OrderId == orderId)
                        sellerOrderView.IsCanceled = true;
                }
            }
        }

        /// <summary>
        /// Отправляет запрос на сервер чтобы обновить статус
заказа.
        /// </summary>
        /// <param name="reqStatus">Строка запроса на
сервер.</param>
        /// <param name="orderId">Id заказа.</param>
        private async void SendOrderStatusAsync(String
reqStatus, Int32 orderId)
        {
            Orders.IsRefreshing = true;
            if (!App.IsConnected())
            {
                await DisplayAlert("Ошибка", "Осутствует
подключение к интернету", "Поробовать снова");
                return;
            }
            RestClient client = new
RestClient($"{App.BaseUrl}api/orders/{reqStatus}?orderId={orde
rId}");
            var request = new RestRequest(Method.PUT);
            request.AddHeader("Content-Type",
"application/text");
            var response = await client.ExecuteAsync(request);
            if (response.StatusCode ==
HttpStatusCode.BadRequest)
            {
                await DisplayAlert("Ошибка", "Не удалось
изменить статус заказа", "Поробовать снова");
```

```
                return;
            }

            Orders.ItemsSource = sellerOrders;
            Orders.IsRefreshing = false;
        }
    }
}
```

1.29 WatchSellerPage.xaml

```
<?xml version="1.0" encoding="utf-8"?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-
namespace:ShopsAggregator;assembly=ShopsAggregator"
             x:Class="ShopsAggregator.Views.WatchSellerPage">
    <ContentPage.Content>
        <StackLayout>
            <RefreshView x:Name="RefreshView">
            <ScrollView>
                <StackLayout Margin="0,10,0,0">
                    <StackLayout x:Name="InfoStack"
Padding="50,0">
                        <Frame CornerRadius="100"
IsClippedToBounds="True" HorizontalOptions="Center"
BackgroundColor="Gray"
                               WidthRequest="200"
HeightRequest="200" VerticalOptions="Start" Padding="0"
HasShadow="True" Margin="0,20">
                            <Image x:Name="Icon"
HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand" Aspect="AspectFill"
                                   Source="{Binding
IconPath}"/>
                        </Frame>
                        <StackLayout x:Name="UserInfoStack"
HorizontalOptions="FillAndExpand">
                            <Label Text="{Binding Username}"
FontSize="Large" HorizontalTextAlignment="Center" />
```

```
                            <Label Text="{Binding Email}"
TextColor="{StaticResource grey}" FontSize="Small"
HorizontalTextAlignment="Center"/>
                            <Label Text="{Binding Info}" />
                            <Button
x:Name="SubscribeSatusButton" HorizontalOptions="Center"
Style="{StaticResource buttonStyle}"
                                    Padding="30,5"/>
                        </StackLayout>
                        <Label FontSize="Title"
HorizontalTextAlignment="Center" Padding="0,15">Товары
пользователя</Label>
                        <ListView x:Name="Posts"
CachingStrategy="RecycleElementAndDataTemplate"
HasUnevenRows="True"

ItemSelected="Posts_OnItemSelected"
SeparatorVisibility="None">
                            <ListView.ItemTemplate>
                                <DataTemplate>
                                    <ViewCell>
                                        <StackLayout
Margin="0,0,0,10">
                                            <Label
Text="{Binding PostId}" IsVisible="False" />
                                            <Image
x:Name="PostImage" Source="{Binding IconPath}"/>
                                            <FlexLayout
JustifyContent="Start" Padding="20,0" HeightRequest="50"
AlignItems="Center">
                                                <StackLayout
Orientation="Horizontal">
                                                    <Image
Source="{local:ImageResource
ShopsAggregator.images.heart.png}"

IsVisible="{Binding IsUserDontLikePost}">

<Image.GestureRecognizers>
```

```
<TapGestureRecognizer

Tapped="OnLikeImageTapped"

NumberOfTapsRequired="1" />

</Image.GestureRecognizers>
                                                    </Image>
                                                    <Label
HorizontalTextAlignment="Center" Text="{Binding LikesCount}"
FontSize="Small"/>
                                                </StackLayout>
                                                <Image
Source="{local:ImageResource
ShopsAggregator.images.dislike.png}"

IsVisible="{Binding IsUserLikePost}">

<Image.GestureRecognizers>

<TapGestureRecognizer

Tapped="OnDislikeImageTapped"

NumberOfTapsRequired="1" />

</Image.GestureRecognizers>
                                                    </Image>
                                            </FlexLayout>
                                            <Label
Text="{Binding Info}" Padding="20,3"/>
                                        </StackLayout>
                                    </ViewCell>
                                </DataTemplate>
                            </ListView.ItemTemplate>
                        </ListView>
                    </StackLayout>
                </StackLayout>
            </ScrollView>
```

```
                </RefreshView>
            </StackLayout>
        </ContentPage.Content>
    </ContentPage>
```

1.30 WatchSellerPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Threading.Tasks;
using Newtonsoft.Json;
using RestSharp;
using ShopsAggregator.Models;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace ShopsAggregator.Views
{
    /// <summary>
    /// Код страницы просмотра страницы пользователя-продавца
пользователем-покупателем.
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class WatchSellerPage : ContentPage
    {
        /// <summary>
        /// Экземпляр типа пользователя-продавца.
        /// </summary>
        private Seller _seller;
        /// <summary>
        /// Имя пользователя-продавца.
        /// </summary>
        private String _sellerName;
        /// <summary>
        /// Список записей пользователя-продавца.
        /// </summary>
        private List<BuyerPostView> _sellerPosts = new
List<BuyerPostView>();
        /// <summary>
        /// Экземпляр типа пользователя-покупателя.
```

```
        /// </summary>
        private Buyer _buyer;
        /// <summary>
        /// Конструктор страницы. Получает экземпляр
пользователя-продавца с сервера.
        /// </summary>
        /// <param name="sellerName">Имя пользователя-
продавца.</param>
        /// <param name="buyer">Экземпляр типа пользователя-
покупателя.</param>
        public WatchSellerPage(String sellerName, Buyer buyer)
        {
            InitializeComponent();
            _buyer = buyer;
            GetSellerByNameAsync(sellerName);
            _sellerName = sellerName;
            RefreshView.Refreshing += RefreshViewOnRefreshing;
        }

        /// <summary>
        /// Устанавливает кнопки статус подписки.
        /// </summary>
        private void SetSubscribeButton()
        {
            SubscribeSatusButton.BackgroundColor = (Color)
App.Current.Resources["purple"];
            SubscribeSatusButton.TextColor = (Color)
App.Current.Resources["buttonTextColor"];
            SubscribeSatusButton.Text = "+Подписаться";
            SubscribeSatusButton.Clicked -=
UnsubscribeFromSeller;
            SubscribeSatusButton.Clicked += SubscribeToSeller;
        }
        /// <summary>
        /// Устанавливает кнопки статус отписки.
        /// </summary>
        private void SetUnsubscribeButton()
        {
            SubscribeSatusButton.BackgroundColor = (Color)
App.Current.Resources["cloudWhite"];
```

```
            SubscribeSatusButton.TextColor = (Color)
App.Current.Resources["grey"];
            SubscribeSatusButton.Text = "Отписаться";
            SubscribeSatusButton.Clicked +=
UnsubscribeFromSeller;
            SubscribeSatusButton.Clicked -= SubscribeToSeller;
        }

        /// <summary>
        /// Обработчик собтыия подписки на пользователя-
продавца.
        /// </summary>
        /// <param name="sender">Издатель события -
Button.</param>
        /// <param name="e">Аргументы события.</param>
        private async void SubscribeToSeller(object sender,
EventArgs e)
        {
            if (!App.IsConnected())
            {
                await DisplayAlert("Нет подключения к
интернету", "Проверьте подключение к интернету", "Попробовать
снова");
                return;
            }
            SetUnsubscribeButton();
            RestClient client = new
RestClient($"{App.BaseUrl}api/sub/addSub?buyerId={_buyer.Id}&s
ellerName={_seller.Username}");
            var request = new RestRequest(Method.PUT);
            request.AddHeader("Content-Type",
"application/text");
            var response = await client.ExecuteAsync(request);
            if (response.StatusCode ==
HttpStatusCode.BadRequest)
            {
                SetSubscribeButton();
                await DisplayAlert("Ошибка подписки", $"Не
удалось подписаться на пользователя ${_seller.Username}",
"Жаль");
```

```
            }
        }
        /// <summary>
        /// Обработчик собтыия отписки от пользователя-
продавца.
        /// </summary>
        /// <param name="sender">Издатель события -
Button.</param>
        /// <param name="e">Аргументы события.</param>
        private async void UnsubscribeFromSeller(object
sender, EventArgs e)
        {
            if (!App.IsConnected())
            {
                await DisplayAlert("Нет подключения к
интернету", "Проверьте подключение к интернету", "Попробовать
снова");
                return;
            }
            SetSubscribeButton();
            RestClient client = new
RestClient($"{App.BaseUrl}api/sub/rmSub?buyerId={_buyer.Id}&se
llerName={_seller.Username}");
            var request = new RestRequest(Method.PUT);
            request.AddHeader("Content-Type",
"application/text");
            var response = await client.ExecuteAsync(request);
            if (response.StatusCode ==
HttpStatusCode.BadRequest)
            {
                SetUnsubscribeButton();
                await DisplayAlert("Ошибка отписки", $"Не
удалось отписаться на пользователя ${_seller.Username}",
"Жаль");
            }
        }

        /// <summary>
        /// Обработчик события обновления информации о
пользователе-продавце.
```

```
        /// </summary>
        /// <param name="sender">Издатель события -
RefreshView.</param>
        /// <param name="e">Аргументы события.</param>
        private void RefreshViewOnRefreshing(object sender,
EventArgs e)
        {
            GetSellerByNameAsync(_sellerName);
        }


        /// <summary>
        /// Получает с сервера пользователя-продавца по имени.
        /// </summary>
        /// <param name="sellerName">Имя пользователя-
продавца.</param>
        private async void GetSellerByNameAsync(String
sellerName)
        {
            if (!App.IsConnected())
            {
                await DisplayAlert("Нет подключения к
интернету", "Проверьте подключение к интернету", "Попробовать
снова");
                return;
            }
            RefreshView.IsRefreshing = true;
            RestClient client =new
RestClient($"{App.BaseUrl}api/search/getSeller?sellerName={sel
lerName}");
            var request = new RestRequest(Method.GET);
            request.AddHeader("Content-Type",
"application/text");
            var response = await client.ExecuteAsync(request);
            if (response.StatusCode ==
HttpStatusCode.BadRequest)
            {
                await DisplayAlert("Ошибка", "Не получилось
получить данные о пользователе", "Поробовать снова");
                await Navigation.PopAsync();
```

```
                return;
            }

            try
            {
                _seller =
JsonConvert.DeserializeObject<Seller>(response.Content);
            }
            catch (Exception e)
            {
                await DisplayAlert("Ошибка", "Не получилось
получить данные о пользователе", "Поробовать снова");
                await Navigation.PopAsync();
                return;
            }

            this.BindingContext = _seller;
            GetSellerPostsAsync(_seller.Username);
            if (_seller.Subscribers == null ||
_seller.Subscribers.Contains(_buyer.Id))
            {
                SetUnsubscribeButton();
            }
            else
            {
                SetSubscribeButton();
            }
            RefreshView.IsRefreshing = false;
        }

        /// <summary>
        /// Получает записи пользователя-продавца с сервера.
        /// </summary>
        /// <param name="sellerName">Имя пользователя-
продавца.</param>
        private async void GetSellerPostsAsync(String
sellerName)
        {
            if (!App.IsConnected())
            {
```

```
                await DisplayAlert("Нет подключения к
интернету", "Проверьте подключение к интернету", "Попробовать
снова");
                return;
            }
            RefreshView.IsRefreshing = true;
            var client = new
RestClient($"{App.BaseUrl}api/posts/getSellerPosts?sellerName=
{sellerName}");
            client.Timeout = -1;
            var request = new RestRequest(Method.GET);
            request.AddHeader("Content-Type",
"application/text");
            IRestResponse response = await
client.ExecuteAsync(request);
            if (response.StatusCode ==
HttpStatusCode.BadRequest)
            {
                await DisplayAlert("Ошибка", "Не получилось
получить данные о пользователе", "Поробовать снова");
                return;
            }

            List<Post> posts = new List<Post>();
            try
            {
                posts =
JsonConvert.DeserializeObject<List<Post>>(response.Content);
            }
            catch (Exception e)
            {
                await DisplayAlert("Ошибка", "Не получилось
получить данные о пользователе", "Поробовать снова");
            }

            foreach (Post post in posts)
            {
                BuyerPostView postView = new
BuyerPostView(post);
                postView.BuyerId = _buyer.Id;
```

```
                _sellerPosts.Add(postView);
            }
            Posts.ItemsSource = _sellerPosts;
            RefreshView.IsRefreshing = false;
        }

        /// <summary>
        /// Отменяет событие нажатия на элемент из ListView.
        /// </summary>
        /// <param name="sender">Издатель события -
ListView.</param>
        /// <param name="e">Аргументы события.</param>
        private void Posts_OnItemSelected(object sender,
SelectedItemChangedEventArgs e)
        {
            if (sender is ListView listView)
            {
                listView.SelectedItem = null;
            }
        }

        /// <summary>
        /// Событие нажатия пользователем на картинку
"понравилась запись(лайк)".
        /// </summary>
        /// <param name="sender">Издатель события - картинка
"понравилась запись"</param>
        /// <param name="e">Аргументы события.</param>
        private void OnLikeImageTapped(object sender,
EventArgs e)
        {
            if (sender is Image image)
            {
                BuyerPostView post = (BuyerPostView)
image.ParentView.BindingContext;
                Task.Run(() =>
                {
                    BuyerPostView currentPost =
```

```
                        (from newPost in _sellerPosts where
newPost.PostId == post.PostId select
newPost).FirstOrDefault();
                    if (currentPost == null)
                        return;
                    currentPost.Likers.Add(_buyer.Id);
                });
                SendLikeTapped("addLike", post);
            }
        }
        /// <summary>
        /// Событие нажатия пользователем на картинку "убрать
лайк".
        /// </summary>
        /// <param name="sender">Издатель события - картинка
"убрать лайк".</param>
        /// <param name="e">Аргументы события.</param>
        private void OnDislikeImageTapped(object sender,
EventArgs e)
        {
            BuyerPostView post = (BuyerPostView) (((Image)
sender).ParentView.BindingContext);
            Task.Run(() =>
            {
                BuyerPostView currentPost =
                    (from newPost in _sellerPosts where
newPost.PostId == post.PostId select
newPost).FirstOrDefault();
                if (currentPost == null)
                    return;
                currentPost.Likers.Remove(_buyer.Id);
            });
            SendLikeTapped("removeLike", post);
        }

        /// <summary>
        /// Отправляет на сервер PUT запрос, чтобы добавить в
или убрать картинку из понравившееся у пользователя.
        /// </summary>
```

```
        /// <param name="param">Параметр с Url строки, в
зависимости от которого будет добавляться или убираться
лайк.</param>
        /// <param name="post">Запись, которая
понравилась/разонравилась пользователю.</param>
        private async void SendLikeTapped(string param,
BuyerPostView post)
        {
            if (!App.IsConnected())
            {
                await DisplayAlert("Нет подключения к
интернету", "Проверьте подключение к интернету", "Попробовать
снова");
                return;
            }
            RestClient client = new
RestClient($"{App.BaseUrl}api/posts/{param}?likerId={_buyer.Id
}&postId={post.PostId}");
            var request = new RestRequest(Method.PUT);
            request.AddHeader("Content-Type",
"application/text");
            var response = await client.ExecuteAsync(request);
            Posts.ItemsSource = null;
            Posts.ItemsSource = _sellerPosts;
        }
    }
}
```

1.31 App.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="ShopsAggregator.App">
    <Application.Resources>
        <ResourceDictionary>
            <Color x:Key="purple">#9b59b6</Color>
            <Color x:Key="grey">#95a5a6</Color>
            <Color x:Key="cloudWhite">#ecf0f1</Color>
            <Color x:Key="buttonTextColor">#ecf0f1</Color>
            <Color x:Key="errorColor">#e74c3c</Color>
```

```
            <Color x:Key="successColor">#2ecc71</Color>
            <x:String x:Key="appName">ShopYou</x:String>
            <Style x:Key="buttonStyle" TargetType="Button">
                <Setter Property="BackgroundColor"
Value="{StaticResource purple}"/>
                <Setter Property="TextColor"
Value="{StaticResource buttonTextColor}"></Setter>
            </Style>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

1.32 App.xaml.cs

```
using System;
using System.Globalization;
using System.IO;
using System.Threading;
using Newtonsoft.Json;
using ShopsAggregator.Models;
using ShopsAggregator.Services;
using ShopsAggregator.Views;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
using Xamarin.Essentials;


[assembly: XamlCompilation(XamlCompilationOptions.Compile)]
[assembly: ExportFont("SchlangeBold.ttf", Alias =
"SchlangeBold")]
namespace ShopsAggregator
{
    /// <summary>
    /// Основная информация о приложении хранится в этом
классе.
    /// </summary>
    public partial class App : Application
    {
        /// <summary>
        /// Ссылка сервера на контроллер users.
        /// </summary>
```

```
        public static String ServerUrl =
"http://62.113.116.228/api/users/";
        /// <summary>
        /// Базовый адрес сервера.
        /// </summary>
        public static String BaseUrl =
"http://62.113.116.228/";
        /// <summary>
        /// Конструктор. Устанавливает язык приложения и
интерфейса.
        /// </summary>
        public App()
        {
            Thread.CurrentThread.CurrentCulture = new
CultureInfo("en-US");
            Thread.CurrentThread.CurrentUICulture =
CultureInfo.CurrentCulture;
            InitializeComponent();

            var navPage = new NavigationPage(new
Authentification());
            MainPage = navPage;
        }

        /// <summary>
        /// Анимирует встряхивание элемента.
        /// </summary>
        /// <param name="element">View элемент.</param>
        public static async void Shake(View element)
        {
            UInt32 timer = 50;
            await element.TranslateTo(-15, 0, timer);
            await element.TranslateTo(15, 0, timer);
            await element.TranslateTo(-10, 0, timer);
            await element.TranslateTo(10, 0, timer);
            await element.TranslateTo(-5, 0, timer);
            await element.TranslateTo(5, 0, timer);
            element.TranslationX = 0;
        }
```

```
        /// <summary>
        /// Проверяет, есть ли у приложения подключение к
интернету.
        /// </summary>
        /// <returns>Результат проверки.</returns>
        public static Boolean IsConnected()
        {
            var current = Connectivity.NetworkAccess;
            return current != NetworkAccess.None;
        }
    }
}
```

1.33 ImageResourceExtension.cs

```
using System;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace ShopsAggregator
{
    /// <summary>
    /// Расширение функционала Image view.
    /// </summary>
    [ContentProperty("Source")]
    public class ImageResourceExtension : IMarkupExtension
    {
        /// <summary>
        /// Путь к файлу с изображением.
        /// </summary>
        public string Source { get; set; }

        /// <summary>
        /// Получает значени из view.
        /// </summary>
        /// <param name="serviceProvider">Механизм извлечения
данных.</param>
        /// <returns>Расположение изображения.</returns>
        public object ProvideValue(IServiceProvider
serviceProvider)
        {
            if (Source == null)
```

```
                {
                    return null;
                }
                var imageSource =
ImageSource.FromResource(Source);

                return imageSource;
            }
        }
    }
```

1.34 MainTabbedPage.xaml

```
<?xml version="1.0" encoding="utf-8"?>

<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="ShopsAggregator.MainTabbedPAge">

</TabbedPage>
```

1.35 MainTabbedPage.xaml.cs

```
using ShopsAggregator.Models;
using ShopsAggregator.Views;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace ShopsAggregator
{
    /// <summary>
    /// Код для страницы с множеством вкладок.
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class MainTabbedPage : TabbedPage
    {
        /// <summary>
        /// Общий конструктор. Устанавливает заголовок
страницы.
        /// </summary>
        public MainTabbedPage()
        {
```

```
            Title = "ShopYou";
        }
        /// <summary>
        /// Добавляет страницы если пользователем является
пользователь-покупатель.
        /// </summary>
        /// <param name="buyer">Экземпляр типа пользователя-
покупателя.</param>
        public MainTabbedPage(Buyer buyer) : this()
        {
            var userSettingsPage = new BuyerMainPage(buyer);
            userSettingsPage.IconImageSource =
ImageSource.FromResource("ShopsAggregator.images.accounticon.p
ng");
            var searchPage = new SearchPage(buyer);
            searchPage.IconImageSource =
ImageSource.FromResource("ShopsAggregator.images.searchicon.pn
g");
            var postsPage = new AllPostsPage(buyer);
            postsPage.IconImageSource =
ImageSource.FromResource("ShopsAggregator.images.gallery.png")
;
            var ordersPage = new BuyerOrdersPage(buyer);
            ordersPage.IconImageSource =
ImageSource.FromResource("ShopsAggregator.images.orders.png");
            Children.Add(postsPage);
            Children.Add(searchPage);
            Children.Add(ordersPage);
            Children.Add(userSettingsPage);
        }
        /// <summary>
        /// Добавляет страницы если пользователем является
пользователь-продавец.
        /// </summary>
        /// <param name="seller">Экземпляр типа пользователя-
продавца.</param>
        public MainTabbedPage(Seller seller) : this()
        {
            var sellerMainPage = new SellerMainPage(seller);
```

```
            sellerMainPage.IconImageSource =
ImageSource.FromResource("ShopsAggregator.images.accounticon.p
ng");
            var createPostPage = new CreatePostPage(seller);
            createPostPage.IconImageSource =
ImageSource.FromResource("ShopsAggregator.images.plus.png");
            var ordersPage = new SellerOrdersPage(seller);
            ordersPage.IconImageSource =
ImageSource.FromResource("ShopsAggregator.images.orders.png");
            Children.Add(ordersPage);
            Children.Add(createPostPage);
            Children.Add(sellerMainPage);
        }
    }
}
```

1.36 MainActivity.cs

```
using System.IO;
using System.Threading.Tasks;
using Android.App;
using Android.Content;
using Android.Content.PM;
using Android.OS;

namespace ShopsAggregator.Android
{
    [Activity(Label = "ShopYou", Theme = "@style/MainTheme",
MainLauncher = true,
        ConfigurationChanges = ConfigChanges.ScreenSize |
ConfigChanges.Orientation)]
    public class MainActivity :
global::Xamarin.Forms.Platform.Android.FormsAppCompatActivity
    {
        protected override void OnCreate(Bundle
savedInstanceState)
        {
            TabLayoutResource = Resource.Layout.Tabbar;
            ToolbarResource = Resource.Layout.Toolbar;

            base.OnCreate(savedInstanceState);
```

```
            global::Xamarin.Forms.Forms.Init(this,
savedInstanceState);
            LoadApplication(new App());
            Instance = this;
        }

        internal static MainActivity Instance { get; private
set; }
        // ...
        // Field, property, and method for Picture Picker
        public static readonly int PickImageId = 1000;

        public TaskCompletionSource<Stream>
PickImageTaskCompletionSource { set; get; }

        protected override void OnActivityResult(int
requestCode, Result resultCode, Intent intent)
        {
            base.OnActivityResult(requestCode, resultCode,
intent);

            if (requestCode == PickImageId)
            {
                if ((resultCode == Result.Ok) && (intent !=
null))
                {
                    global::Android.Net.Uri uri = intent.Data;
                    Stream stream =
ContentResolver.OpenInputStream(uri);

                    // Set the Stream as the completion of the
Task

PickImageTaskCompletionSource.SetResult(stream);
                }
                else
                {

PickImageTaskCompletionSource.SetResult(null);
                }
```

```
            }
        }
    }
}
```

1.37 ApDelegate.cs

```
using Foundation;
using UIKit;

namespace ShopsAggregator.iOS
{
    // The UIApplicationDelegate for the application. This
class is responsible for launching the
    // User Interface of the application, as well as listening
(and optionally responding) to
    // application events from iOS.
    [Register("ApDelegate")]
    public partial class AppDelegate :
global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
    {
        //
        // This method is invoked when the application has
loaded and is ready to run. In this
        // method you should instantiate the window, load the
UI into it and then make the window
        // visible.
        //
        // You have 17 seconds to return from this method, or
iOS will terminate your application.
        //
        public override bool FinishedLaunching(UIApplication
app, NSDictionary options)
        {
            global::Xamarin.Forms.Forms.Init();
            LoadApplication(new App());

            return base.FinishedLaunching(app, options);
        }
    }
}
```

1.38 Main.cs

```
using UIKit;

namespace ShopsAggregator.iOS
{
    public class Application
    {
        // This is the main entry point of the application.
        static void Main(string[] args)
        {
            // if you want to use a different Application
Delegate class from "AppDelegate"
            // you can specify it here.
            UIApplication.Main(args, null, "AppDelegate");
        }
    }
}
```

2 Список литературы

2.1 ГОСТ 19.401-78. ЕСПД. Текст программы. Требования к содержанию и оформлению. – М.: ИПК Издательство стандартов, 2001.

2.2 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
	Измененных	Замененных	новых	Аннулированных				

Изм.	Лист	№ док.	Подп.	Дата
RU.17701729.04.01-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам инв. №	Инв. № дубл.	Подп. и дата