UDK 004.42: 004.51

[1]**Kravchenko Ihor,** Department of Computerized Control Systems, E-mail: 10@gmail.com, ORCID: 0000-0003-1751-6049, Odessa, Ukraine
[1]**Speranskyy Viktor,** Candidate of Technical Sciences, Associate Professor, Associate Professor at the Department of Computerized Control Systems, E-mail: speranskiyva@ukr.net, ORCID: 0000-0002-8042-1790, Odessa, Ukraine
[1]Odessa National Polytechnic University, Shevchenko ave., 1, Odessa, Ukraine

CROSS-PLATFORM PRACTICES FOR MOBILE APPLICATION DEVELOPMENT OF AUTOMATED TRADE ACCOUNTING

Abstract. *The problem of single application development that can work in widely used modern mobile platforms (Android and iOS) is dicussed. Current situation in building of crossplatform applicatioons is studied. The choise of appropriate development tools has been explained. The basic principles and rules of design and development of crossplatform mobile applications using chosen Xamarin.Forms technology has been described. The paper consists of two parts. The first part describes purposes and benefits of used Xamarin.Forms crossplatform technology and contains technical requirements. The Xamarin.Forms technology using with C# object oriented programming language. The second part describes the best practices of using this technology in current project: MVVM pattern definition for devlopement using best style OOP; C# asynchoronous programming for creating comfortable and fast for use application; custom controls creating used in current project for best UI experience; using platformspecific code with DependencyService; customization of standard controls with Renderers; final application optimization to reach maximum performance and minimum battery consumption at a time (results of battery time optimization are presented). Finally, studied and written about using of new features of Xamarin.Forms by big developers' community. Examples of software code and application screenshots used in application are given. The work shows the stages of the development of the mobile business application modules, which is already used in commercial product; all of the given examples are thoroughly tested during the development process and in real work, that allowed to make conclusions about best practices. The use of the developed sowftware allowed increasing the efficiency of trade accounting due to decreasing of monotonous operations quantity and as a result, the decreasing of errors in staff work, that already gave opportunity for money economy.*

Keywords: *mobile applications developing; crossplatform; Xamarin.Forms; Android; iOS; UWP; .NET; C#; MVVM*

Introduction

Problem statement. Before software development developers already know which devices and operating systems they will create a software product for. Nowadays, the most popular devices for today's business applications are personal computers, smartphones and tablets. Three most commonly used operating systems for personal computers are: Windows, Linux, macOS. For mobile devices are Android and iOS.

Current situation in building of crossplatform applicatioons is studied in [1; 14; 15]. The principle of abstracting a graphical interface in many cases solves the problem of cross-platform, but not always. Crossplatform application is not a single application for different platforms, but a single code base in different applications. And here it becomes very important to correctly build the architecture of the application, namely, the maximum separation of the interface and logical parts.

There is another option that is called hybrid applications that use web technologies rather than native development. The result is a web application that runs in a wrapper and is served not as a web page, but as a separate application requiring installation and having a separate icon. Hybrid solutions are quite popular due to the fact that the function of the web browser can handle virtually any mobile OS, which means if the application is already running under any mobile OS, then launch it to another will not work. Nevertheless, the use of such an approach does not allow for the high speed of the final development.The paper also includes UWP (Universal Windows Platform), which is not an operating system but is just a platform for the creation and run engaging and immersive applications that work across a wide variety of the Windows 10 device families. The API is implemented in C++ and is supported in VB.NET, C#, F#, and JavaScri

Typically, most software products are released immediately for multiple operating systems to reach larger audience of users. But at the same time, the price and development time are growing. So the problem is a to develop single application that can work everywhere with minimum of additional afterwork. In order to minimize and synthesize code writing for several operating systems, the crossplatform technologies for software development are used.

The aim of this work is to show the use of crossplatform technology Xamarin.Forms that was chosen due to convenient development tools and good performance of ready-made applications to automate the trade accounting with use of crossplatform application.

© Kravchenko I. A., Speranskyy V. O., 2018

Main part

There is a key difference between iOS and Android in terms of application execution — a way to precompile them. The Dalvik java virtual machine and Just-in-time compilation (on-the-fly compilation) are used to run applications on Android. The iOS uses Ahead-of-Time (compile before execution) for this. The difference is shown in Fig.1.

The Xamarin takes this distinction into account by providing separate compilers for each of these platforms which allows getting native applications that run outside the context of the browser and can use all the hardware and software resources of the platform.

The simplest way is to use Visual Studio 2017 development environment to develop using the Xamarin.Forms technology but it is need to activate Business or Enterprise licence that costs $999 or $1899 respectivly. The development of presented project is carried out using the C# programming language. However, it is possible to add projects written using the VB.NET programming language. For professional development of mobile applications it is needed to understand well the principles of object-oriented programming.

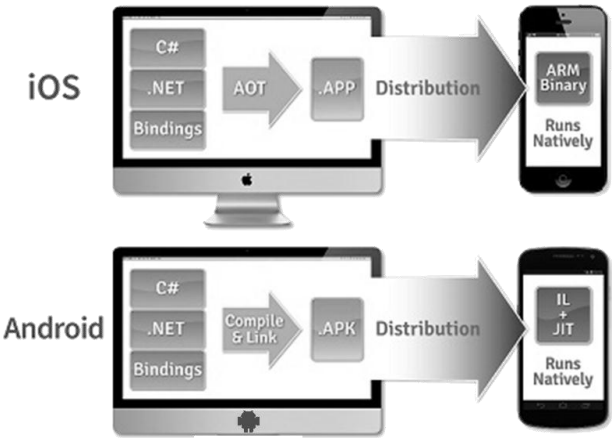

Fig. 1. Template and code sharing

Current work is created as Cross-Platform project in Visual Studio 2017 using the template of Mobile App (Xamarin.Forms) project of the Visual C#. Master Detail form pattern means that an application includes a page with a side menu. Typically, the Blank App is used, that is, a blank project that contains only the most essential components. In most cases, it is more convenient for developers to create blank projects to add only what they really need.

A list of platforms determines whether some platforms will be included in the project; you need to choose the platforms for which the application development is planned. It is better to choose everything: nothing will change if some platforms are not used at all.

The point that will affect further development is the code sharing strategy. To understand better what to choose, first consider the components of the project.

All projects created in Visual Studio have a Solution file (with a sln extension), and the project file itself (with the extension csproj, if the language is C #, or vbproj if the language is VB.NET). The project file contains a list of files used in the project: for example, folders, program code files, images, class diagrams, and more. However, sometimes one project is not enough. Therefore, file-based solutions are included that contain links to one or more projects. In addition, projects may have links to each other within a single solution. For a Xamarin.Forms project, a solution will be created for four several projects at the same time (Fig. 2).

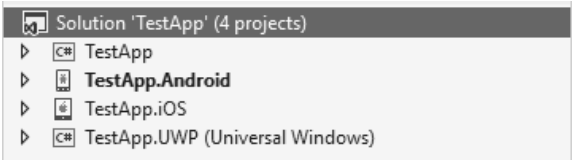

Fig. 2. Solution with projects

The TestApp project is a common project where most of the code is created. Other projects are created for each platform, where use of special code can be specified, inherent only on a separate platform. A project shown with bold text is used as a startup project and it can be changed to another type needed to run the application in another platform.

Considering the code distribution strategies, .NET Standard is a list of specifications to ensure the universality of libraries [2]. Because of this, the same libraries in different environments can be used, ensuring that these libraries meet the needs of.NET Standard. During compilation, a DLL-library is created for a general project that can be used anywhere, in this case, it uses a separate platform.

Shared Projects works differently: in each project that corresponds to the platform, there is a common project code. No libraries are created during compilation for the general project and the general code is packed together with the platform project.

That is, NET Standard uses the common code separately from the project platform, and Shared Projects uses the generic code inside the project platform.

For developers this means that Shared Projects allows you to use the compiler directives (for example, #if __ANDROID__) and use links to the libraries of individual platforms in a general code that is not possible for .NET Standard. However, .NET Standard makes universal code more versatile, and the Dependency Service is used to call the code implemented separately for each platform.

In any case, it is better to use .NET Standard – this will make the code universally available and allow the use of other libraries that support .NET Standard.

1. Components of the project

In Xamarin.Forms, objects that are situated on the device screen are called visual elements. They are split into three categories:

– page;
– layout;
– view.

Page is a visual element that occupies the entire screen of the device (or a large part of it). A mobile app can have one or more pages. A user can navigate from one page to another, for this there is a mechanism for navigating between pages in the project. Pages may vary, depending on the display and interaction with the user.

Layout (template) is a template that brings up the views of the page. There are various markups, for example, you can arrange items using the Grid markup, which allows them to be placed as a grid, or using the StackLayout layout that places elements one after the other.

View is the element used to display data and interact with the user. For example, a button, an input field, a label with a text, a switch and other items.

Therefore, we can say that the page contains a layout that contains a view. There are some features: the page cannot contain a view directly, it must necessarily have in itself only one layout. However, the layout may have other layouts that also contain one or more views.

Developer can create a visual interface in a program code or in XAML-files [3]. It is recommended to do this in XAML files, so the program code will be cleaner. In addition, in XAML-files developer can specify the properties of the objects that provide the data to display to the user.

2. Use of MVVM design pattern

In modern programming, it is important to be able to divide the code describing the UI from code with business logic. To do this, many design templates were created for developers. The recommended design template for Xamarin.Forms is MVVM [4-5].

MVVM (Model – View – ViewModel) is a design template that uses the properties of objects to provide data to the elements of the interface. It consists of three parts:

1) View is a visual element that displays user data.

2) Model – a data model that retains some kind of data.

3) ViewModel – a class that collects data from a model and provides them with a visual element using a special mechanism called Binding.

The main feature of MVVM is that Binding can work in both directions: to take data from a model into a visual element, and vice versa – to take data from a visual element into a model, or in both directions at the same time.

For example, there is a counter-agent edit page where the user can fill out his data. For this page, you need to create a ViewModel that contains a reference to an instance of the ContractorProxy class (contr-agent) that contains the necessary properties (name, email, comment, etc.).

As a Model we use data the ContractorProxy class:

```
Public Class ContractorProxy
  Private _PointName As String
  Public Property PointName As String
    Get
      Return _PointName
    End Get
    Set(ByVal value As String)
      _PointName = value
    End Set
  End Property
End Class
```

As a ViewModel using class ContractorEditorViewModel:

```
public class ContractorEditorViewModel
{
  private ContractorProxy _card;
  public ContractorProxy Card
  {
    get{return _card;}
  }
  public string Name
{
get { return this.Card?.PointName; }
set { if (this.Card?.PointName!= value)
    {
    this.Card.PointName = value;
    this.OnPropertyChanged(nameof(Na
me));
    }
}
}
```

In order to notify the page that the data has changed and new data has been displayed to the user, developer need to implement the INotifyPropertyChanged interface in ViewModel and add the following code:

```
public event PropertyChangedEventHandler PropertyChanged;
    protected void OnPropertChanged ([CallerMemberName] string propertyName = ""){
        var changed = PropertyChanged;
        if (changed == null)
                return;
        changed.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
```

The above program code generates a changed event (with a property name parameter whose value needs to be read), which will be somewhere "heard" and the visual element that "looks" on the property with that name, should read the value of this property. But to do this, you need to call the OnPropertyChanged method.

Now we can create a ContractorEditorPage page that contains the necessary visual elements.

```
    public partial class ContractorEditorPage : ContentPage
    {
       private ContractorEditorViewModel _viewModel;

       public ContractorEditorPage()
       {
          InitializeComponent();
          _viewModel = new ContractorEditorViewModel();
          this.BindingContext = _viewModel;
    }
    }
```

In the program code written above, the _viewModel object is created and assigned the BindingContext properties of the page. Now you can set the properties of the visual properties of a ViewModel property in the XAML of this page:

```
<Entry Text="{Binding Name}"/>
```

Now when the user puts the text in the field, the Name property in ViewModel will be automatically updated, and with it will be updated the Model itself, which is expressed by the Card property in ViewModel.

From this, we can draw the following conclusions: ViewModel provides data for View. View has a link to ViewModel, and ViewModel has a link to Model. However, Model does not know about ViewModel, just as if ViewModel knows nothing about who uses its data. Code can link directly to Model. However, this issue occurs only for small code examples: in this case, it is impossible to see the whole "tragedy" of mixing the code page with the code of business logic. The bigger the project the bigger the price of the error. The answer is, for example, other tablets can be used for tablets, which describes a more responsive interface on the big screen. In addition, the data will be used the same. That is, it is more profitable to allocate the program code that provides data to another class so that it can then be used elsewhere. Therefore, experienced developers always share the programmatic code of the visual interface from the program code of the business logic of the project: it provides the universality of the code and its reliability.

However, if a model class contains a large number of properties it is not necessary for each such property to create a wrapper property in ViewModel. Use the wrapper property (in this example, the Name property) when it is necessary to somehow prepare the data for the interfaces (for example, before the name of the counterpart, add the word "Name"). In this case, XAML can write:

```
<Entry Text = "{Binding Card.PointName}" />
```

Thus, the property of the Card located in ViewModel will find the PointName property whose value will be displayed in the input field.

Taking into account all of the above, one can formulate the pros and cons of this design template.

Pros:

– ensuring the reliability and universality of the code;

– Automatic updating of properties in ViewModel after changing the properties of visual elements;

Cons:

– increasing of links quantity in the project;

– the imperfection of XAML due to the lack of a checklist with the properties list in ViewModel.

It is worth noting that the given disadvantages are non-significant and offset against the pros.

3. Using benefits of OOP

The most important thing in a modern program is not the writing of software code, but the construction of a proper hierarchy of data models. The use of OOP capabilities allows us to reduce the amount of software code and its repeatability. Therefore, we will apply the OOP for the typesetting.

In most cases, mobile applications contain a large number of pages. In addition, for each page a template must be created. Often, pages can be similar to each other in a functional way. Contractor editor, product editor, document editor; list of counteragents, list of goods items, list of documents. Same things can be done in the basic functionality. To begin, create a basic viewmodel, where we will add a functionality for the INotifyPropertyChanged interface (code above).

```
public abstract class BaseViewModel
: INotifyPropertyChanged
{
  bool _isBusy = false;
  public bool IsBusy
  {
    get {return _isBusy; }
    set {SetProperty(ref _isBusy, val-
ue);}
  }
}
```

The IsBusy property is required to show the loading animation when performing some kind of action (downloading or sending data, etc.). Below you will find out how to work with it.

Consider an example with a checklist for a page with a list. For example, there is a page with a list of goods. We need to download a list of items as soon as the page is displayed on the screen. The page has an OnAppearing method, which can describe any actions that should occur as soon as the page is displayed on the screen. To begin, we will create a look-up model.

```
public class GoodsRowListViewModel :
BaseViewModel{
public new ObservableCollection
<GoodsRowListItemViewModel> Items
{
get { return (ObservableCollection
<GoodsRowListItemViewMod-
el>)base.Items;
}
public virtual async Task<bool>
LoadItems()
    {
      this.IsBusy=true;
      this.IsBusy=false;
    }
}
```

Items – is a collection that will be added to the list of loaded goods. The LoadItems method will perform actions to load data from services and add them to the Items collection. Now the page needs to add a ListView item that is required to display a list with data (this code will be omitted for visualization) and call the method to load the data.

```
protected async override void OnAppearing()
{
  base.OnAppearing();
  await _viewModel.LoadItems();
}
```

Now when we go to this page, data will be downloaded (Fig. 4).

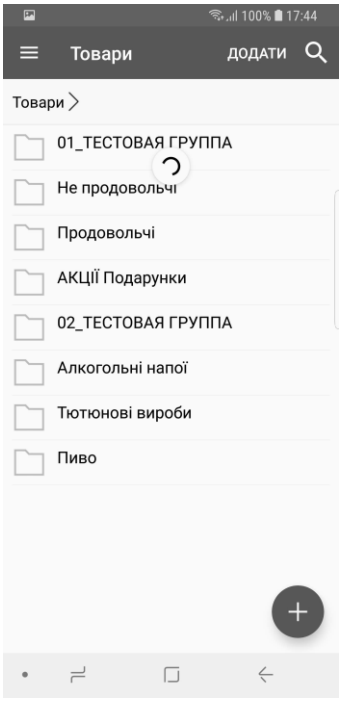

Fig. 4. Data loading

The boot animation connects by using the ListView property of the IsRefreshing property via Binding to the IsBusy property in the lookup model.

However, the pages with lists in the application may be many. Therefore, we can allocate a common functionality for lists to the upper level of the hierarchy, creating a ListViewModel class for that and load the LoadItems method there.

```
protected abstract void AddItem(Object item);
protected abstract
Task<IEnumerable<object>> GetList-
FromDataSource();
protected abstract object CreateItem(object
listItem);

public virtual async Task LoadItems()
{
  this.IsBusy = true;
  IEnumerable<object> serviceResult = await
GetListFromDataSource();
  foreach(object proxyObject in serviceRe-
sult)
  {
      this.AddItem(this.CreateItem(proxyObj
ect));
  }
  this.IsBusy = false;
}
```

Now for each page on the list, you need to create a lookup model, follow it from ListViewModel, and implement abstract methods. Thus, most of the functionality and code remain the same, ensuring the versatility of the code and its

suitability for change. If you want to add some functionality to all pages with lists, then you can add it to the ListViewModel class. But if you want to expand the functionality only in some classes, you can select a general code for them and put it on the upper level of the abstraction.

4. Asynchronous programming in C#

The current development of applications requires knowledge of asynchronous programming. It is needed to create a responsive interface: that is, long-term operations (downloading data from the network or performing complex tasks) should not block UI-thread. The programming language C # for such purposes provides handy tools in the form of async and awaits operators [6]. It should be noted that asynchrony does not mean multithreading. Indeed, a long-term task can be performed in a separate thread, but this is not required. For an example, consider the code above to download data after the page is displayed.

```
protected async override void OnAppearing()
{
   base.OnAppearing();
   await _viewModel.LoadItems();
   DoSomething();
}
```

The await statement indicates that the execution of the program will be suspended until the code after the await operator is executed. The LoadItems method will be launched in a separate thread (run it in a separate thread decides the compiler, but usually uses a separate thread), and when it is executed, the program will continue its execution. In some cases, if the next program code does not depend on the outcome of the asynchronous operation, the compiler can continue the program execution. However, the wait operator will be ignored if the word header does not add the word async.

Thus, the program remains "sensitive" to interact with the user.

The await method can be used only for those methods returned by Task. Using void in the asynchronous method is possible only for events, in other cases it is not recommended (otherwise it will not be possible to catch the exception).

```
public async Task LoadCount()
{
this.Count = await GetCountAsync();
}
```

But often the method should return some data. To retrieve data, you must use the typed Task class.

```
public async Task<int> LoadCount()
{
int count = await GetCountAsync();
return count;
}
```

For any method, we can create an asynchronous version of it. For example, there is an ordinary GetCount method that returns int. How to return from it the Task <int> to execute it in a separate thread?

```
public async Task<int> GetCountAsync()
{
return await Task.Run(() => GetCount());
}
```

Another way:

```
public async Task<int> GetCountAsync()
{
return await Task.Factory.StartNew(() =>
GetCount());
}
```

The difference lies in the fact that in the second case, we can use the additional parameters of launching the method in a separate thread (in this paper these details will be omitted).

Note: In the VB.NET programming language, the same asynchronous mechanism is used but it is based on the syntax of the language.

5. Creating custom visual elements

Because Xamarin.Forms combines all three platforms, a small number of visual elements are available from the start, that is, only those that are present on all three platforms at once. However, developers have the ability to create their own visual elements, and there are also many free user libraries with visual components.

We can use the ContentView class [7] to create our own visual element. To do this, we will create a new element that follows from ContentView with the XAML code. Consider an example of creating a checkbox (let us call it Checker), this component is not present in Xamarin.Forms. Instead, a Switch performs the same functions as the classic checkbox (check box), but has a slightly different interpretation. For example, the check box is "yes" or "no", while the switch is set to "on" or "off", although for the code it is true or false in any case.

So, adding a regular label (Label) to XAML.

```
<ContentView.Content>
  <Grid RowSpacing="0" >
    <Grid.RowDefinitions>
      <RowDefinition />
    </Grid.RowDefinitions>
    <Label Grid.Row="0" x:Name="IconLabel"
FontSize="32" HorizontalOptions="Center"
VerticalTextAlignment="Center" VerticalOptions="Center">
      <Label.GestureRecognizers>
```

```
        <TapGestureRecognizer
    Tapped="OnCheckTapped"/>
      </Label.GestureRecognizers>
      </Label>
    </Grid>
    </ContentView.Content>
```

In the code of the visual element, the following is added:

```
public static readonly BindableProperty Is-
SelectedProperty = BindableProper-
ty.Create(nameof(IsSelected), typeof(bool),
        typeof(Checker), false, Binding-
Mode.TwoWay, propertyChanged: OnIs-
SelectedPropertyChanged);

public bool IsSelected
{
get {return
(bool)GetValue(IsSelectedProperty);}
set {
  SetValue(IsSelectedProperty, value);}}
  private static void OnIsSelectedProperty
Changed(BindableObject bindable, object
oldvalue, object newvalue){
if (!(bindable is Checker selector))
  return;
selector.SetIcon((bool)newvalue);
}
```

Property IsSelected will be used to indicate the status of the checkbox. The IsSelectedProperty property has a BindableProperty type that allows us to use the IsSelected property for the Binding mechanism. When we create it, we must specify the name and type of the property that changes-Xia, its default value, the Binding mode, and the method to be called when changing the property value. The OnIsSelectedPropertyChanged method changes the icon depending on the state of the property. To do this, use the so-called icon fonts (for example, FontAwesome), which in fact are fonts, but display the text as an image. Add another important OnCheckTapped method, which will be called after clicking on the icon.

This method changes the value of the IsSelected property to the opposite; in the property, the set will be called, where a new value for IsSelectedProperty will be called, which, in turn, will call the OnIsSelectedPropertyChanged method, which will change the image for the icon. This way, the user will see that the check mark has been flagged.

Using the Checker element looks like this:

```
<views:Checker IsSelected="{Binding Is-
Selected}"/>
```

We can also add other properties, such as color, size, and more.

That is, we must use BindableProperty to write our own visual elements. This allows us to "look" at the property of the element through the Binding mechanism, which is not contrary to the requirements of MVVM.

6. Using platform-dependent code

All three platforms are different between cobundles, so Xamarin. Forms's "out of the box" cannot cover all the required functionality. However, developers have the opportunity to "get" specific data from a specific platform or perform some actions on their own. To do this, there is a DependencyService mechanism [9]. Let us consider it on an example of receiving the serial number of the device. Device serial number is a unique ID that can be used for various purposes, such as adding a device to a list of trusted user devices.

First, the IDevice interface was created (in the C# programming language there is a rule called interfaces with letter I).

```
public interface IDevice
{
  string DeviceSerialNumber { get; }
}
```

Now each platform project needs to implement this interface. Let us start with android.

```
[assembly: Dependency
(typeof(TcuClientStandard.Droid.Helpers.Devi
ce))]
namespace TcuClientStandard.Droid.Helpers
{
public class Device : IDevice
{
    public string DeviceSerialNumber
    {
    get { return Android.OS.Build.Serial;    }}
}}
```

Above the namespace, it is needed to specify an attribute that in this class will be used for DependencyService. So, it will be seen from the general project during execution.

The same code will be for iOS:

```
  [assembly: Dependency
  (typeof(TcuClientStandard.iOS.Helpers.Devic
  e))]
  namespace TcuClientStandard.iOS.Helpers
  {
  public class Device : IDevice{
    public string DeviceSerialNumber
        {
        get { return UIK-
  it.UIDevice.CurrentDevice.IdentifierForVend
  or.AsString(); }
        }
```

```
}}
```

Now we can use the platform-dependent code in the general project:

```
public static string DeviceSerialNumber{
  get {
     return DependencyService.
Get<IDevice>().DeviceSerialNumber;
}}
```

Thus, the same functionality in each platform is implemented in its own way, and then used in the general project.

7. Customizing visual elements with renders

Sometimes situations arise when we need to change the appearance of standard items or add a new functionality that is only on a separate platform. For this, there is a mechanism for ExportRenderer [10]. Consider using it as an example of extending the function of the standard Label. This was performed to add the ability to display the underlined text.

For this, an ExtendedLabel class was created that is inherited from Label, and the IsUnderline property using BindableProperty.

```
public static readonly BindableProperty IsUnder-
linеProperty = BindableProper-
ty.Create("IsUnderline", typeof(bool),
typeof(ExtendedLabel), false, Binding-
Mode.OneWay);
public bool IsUnderline
{  get{ return (bool)GetValue(IsUnderlineProperty);
}
  set{
    SetValue(IsUnderlineProperty, value);
}}
```

Now add the ExtendedLabelRenderer to the Android project, which is inherited from LabelRenderer. In order to change the functionality of an element, we need to redefine the OnElementChanged method. This method will be called once when creating a visual element.

```
protected override void OnElement-
Changed(ElementChangedEventArgs<Label> e){
base.OnElementChanged(e);
var view = (ExtendedLabel)Element;
var control = Control;
UpdateUi(view, control);
}
```

In the renderer classes, there are two main properties: Element and Control. Element means the visual element used in Xamarin.Forms. Control is a visual element of the platform. That is, in this case Element is ExtendedLabel, and Control is an android textView element.

Next will be called the UpdateUi method, which will make the text highlighted if the desired property is enabled.

```
private static void UpdateUi(ExtendedLabel view,
TextView control)
{
if (view.IsUnderline)
{
control.PaintFlags = control.PaintFlags |
PaintFlags.UnderlineText;
}}
```

But in order to allow the underscore to be "on the fly", we need to redefine the OnElementPropertyChanged method, which will be used to change any visual properties of the Xamarin.Forms property.

```
protected override void OnElementProperty-
Changed(object sender, PropertyChangedEventArgs
e){
base.OnElementPropertyChanged(sender, e);
var view = (ExtendedLabel)Element;
if (e.PropertyName == Extended-
Label.IsUnderlineProperty.PropertyName)
{
   Control.PaintFlags = view.IsUnderline ? Con-
   trol.PaintFlags | PaintFlags.UnderlineText : Con-
   trol.PaintFlags &= ~PaintFlags.UnderlineText;
}}
```

Now, in order for a custom renderer to work, you need to put the ExportRenderer attribute above the namespace.

```
[assembly:
ExportRenderer(typeof(ExtendedLabel),
typeof(ExtendedLabelRenderer))]
```

The first parameter means for which visual element the renderer will be used, and the second parameter indicates which renderer will be used.

The use of Extended-Label with underline in the fields "Customer" and "Article" is shown in Fig. 5.

8. Performance and battery time optimization

Weak place for mobile apps (especially on Android) is ListView. Wrong work with lists can cause fading during scrolling. This is because the application performs a lot of work in the main thread, which results in framerate (the number of frames per second). To optimize the work of lists in Xamarin.Forms, caching elements is used [11].

That is, when scrolling, the same visual elements are used, only the data that is displayed to the user is changed. To enable this option, you need to set CashingStrategy = "RecycleElement" for ListView [12].

Here are some more tips:

– use the same height for the cells;

– use AbsoluteLayout, RelativeLayout, Grid with a fixed-size line instead of StackLayout;

– avoid a complex investment of some elements to others;

– avoid using a large number of items: for example, Label can use the FormattedString property instead of a large number of labels;

– if pictures are used in the list, they should be downloaded asynchronously, using the OnItemAppearing event in the list;

– if standard images are used in the Image class, it is advisable to use the same ImageSource property for identical images, thus, one source will be used for all images, which will greatly save memory.

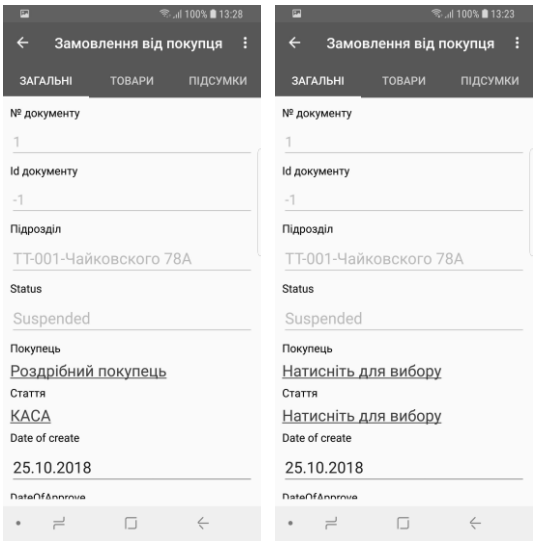

Fig. 5. Result of custom renderers using

For diagnostics, we can use some features in the developer's parameters [13].

An important diagnostic feature is the "Graphics Processor Profile". You can use Android Device Monitor to investigate which process blocks the UI stream (Fig. 6).

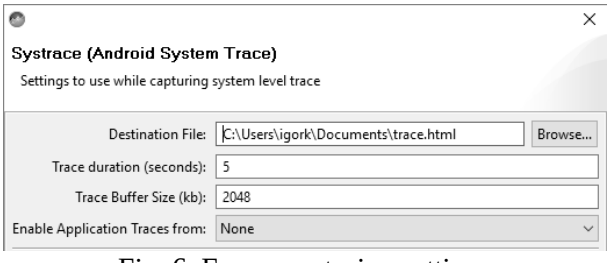

Fig. 6. Frame capturing settings

This program has the capture capability of the device. When you click the "Capture system wide trace using Android systrace" button, you must set the capture time (usually 5 seconds). After clicking "OK" you should immediately start to scroll the list on the device within the specified time. When you open the trace.html file, you can see which frames took more time than allowed for comfortable perception. Such frames occupy more space on the schedule. By clicking on the frame, you can see which task was performed at that time.

Another diagnostic feature used is the option "Adjust GPU overlay" that shows the degree of nesting of visual elements (Fig. 7).

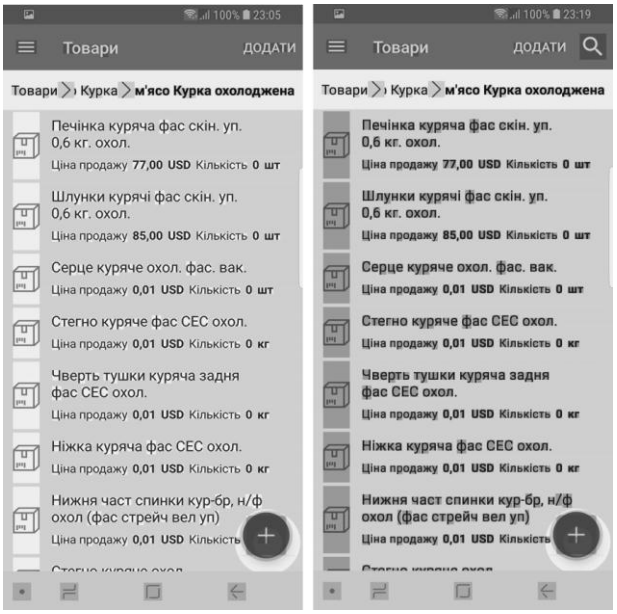

Fig. 7. Elements overlay (little left, a lot of right)

Having the results of diagnostics written earlier, the optimization of developed application were performed. The optimization of application performance and number of ambigious operations perfomed by staff using mobile device were studied. The results of battery time for the same device before (blue color) and after (orange color) performing all optimizations according to the level of screen brightness are presented in Fig. 8.

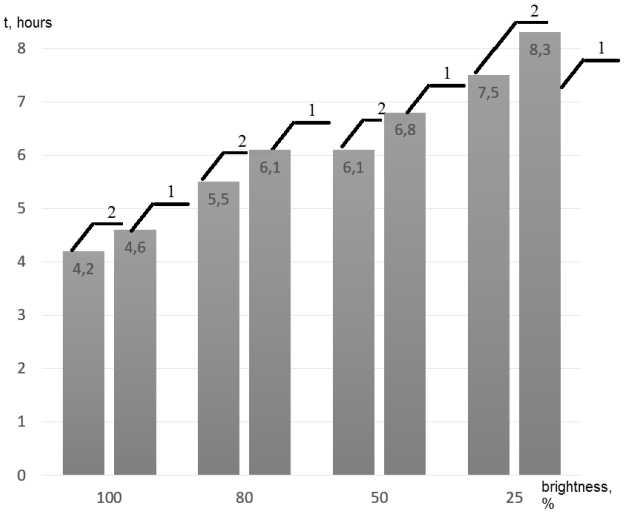

Fig. 8. Battery time dependence at predefined screen brightness level before (2) and after (1) software optimization

Conclusion

Professional development of mobile applications with Xamarin.Forms technology has become more real than ever. The technology is constantly updated, new features are added, more and more users are joining the Xamarin developer community. A lot of articles and recommendations were created by collaborative efforts of developers, answers to a lot of questions were provided in the forums, and application setup became more convenient. For those developers who just start using Xamarin.Forms, you just need to find the best practices and use them in your own projects. This paper contains the answers for most urgent questions: which design template to use, how to work with the main technology links and how to optimize the work with the graphical user interface. The software optimization resulted 10-12% increase of time using battery and speed of applicatin work at the same device. The described work is the part of the commercial project [16]. Its implementation allowed to increase the efficiency of trade accounting due to decreasing of the number of monotonous operations and as a result the decreasing of human factor in everyday work.

References

1. Kravchenko, I. A., & Speransky, V. O. (2018), "Analysis of technologies for creating a client application on mobile platforms Android and iOS for trading accounting system for small and medium businesss", Information Sciences, Information Systems and Technologies: Abstracts of the 15-th All-Ukrainian Conference of Students and Young Scientists. Odessa, April 27, 2018, pp. 48-51.

2. Sharing code overview [Electronic Resource]. – Access Mode https: //docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/code-sharing.

3. Markup Extensions for XAML Overview [Electronic Resource]. – Access Mode https : //docs.microsoft.com/en-us/dotnet/framework/ xaml-services/markup-extensions-for-xaml-overview.

4. From Data Bindings to MVVM. [Electronic Resource]. – Access Mode https : //docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/xaml-basics/data-bindings-to-mvvm.

5. Simple Activity Indicator: Xamarin Forums. [Electronic Resource]. – Access Mode: https ://forums.xamarin.com/discussion/comment/346268/ #Comment_346268.

6. Asynchronous programming with async and await (C#) [Electronic Resource]. – Access Mode: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/,

7. Creating Custom Controls with Bindable Properties in Xamarin.Forms [Electronic Resource]. – Access Mode: https://mindofai.github.io/Creating-Custom-Controls-with-Bindable-Properties-in-Xamarin.Forms/.

8. Xamarin Forms Pages Forms [Electronic Resource]. – Access Mode : https://docs.microsoft.com/ru-ru/xamarin/xamarin-forms/user-interface/controls/pages

9. Xamarin Forms Layouts Forms [Electronic Resource]. – Access Mode: https://docs.microsoft.com/ru-ru/xamarin/xamarin-forms/user-interface/controls/layouts,

10. Introduction to DependencyService [Electronic Resource]. – Access Mode https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/dependency-service/introduction.

11. Xamarin. Forms Custom Renderers [Electronic Resource]. – Access Mode https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/custom-renderer/.

12. Optimizing Xamarin.Forms Apps for Maximum Performance [Electronic Resource]. – Access Mode https://blog.xamarin.com/optimizing-xamarin-forms-apps-for-maximum-performance/.

13. ListView Performance [Electronic Resource]. – Access Mode https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/listview/performance.

14. Tips for Creating a Smooth and Fluid Android UI [Electronic Resource]. – Access Mode https : //blog.xamarin.com/tips-for-creating-a-smooth-and-fluid-android-ui/.

15. Xanthopoulos S., and Xinogalos S. (2013), "A comparative analysis of cross-platform development approaches for mobile applications", Proceedings of the 6-th Balkan Conference in Informatics, ACM, pp. 213-220 .

16. Trade Accounting [Electronic Resource]. – Access Mode https://andriy.co/TCUMobile-Sistema_ucheta_mobilnoi_torgovli_i_distributsii_dlya_KPK .aspx

Received 12.11.2018

[1]**Кравченко Ігор Андрійович ,** кафедри комп'ютеризованих систем управління, E-mail: 10@gmail.com, ORCID: 0000-0003-1751-6049, м. Одесса, Украина
[1]**Сперанський Віктор Олександрович,** кандидат технічних наук кафедри комп'ютеризованих систем управління, E-mail: speranskiyva@ukr.net, ORCID: 0000-0002-8042-1790, м. Одесса, Украина
[1]Одесский национальный политехнический университет, пр-т Шевченко, 1, м. Одесса, 65044, Украина

КРОСПЛАТФОРМОВА ПРАКТИКА РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ ДЛЯ АВТОМАТИЗОВАНОГО ТОРГІВЕЛЬНОГО ОБЛІКУ

Анотація. *В роботі описано основні принципи та правила проектування та розробки мобільних додатків, що використовують кросплатформову технологію Xamarin.Forms. Описано основні принципи та правила проектування та розробки мобільних додатків, що використовують перехресну технологію Xamarin.Forms. Робота базується на розробці мобільного бізнес-додатку, який вже використовується в комерційній компанії, всі наведені приклади перевірено в реальній роботі. Стаття складається з двох частин. Перша частина описує цілі та переваги використовуваної технології Xamarin.Forms і містить технічні вимоги. Технологія Xamarin.Forms вимагає використання об'єктно-орієнтованого програмування в C#. У другій частині описуються найкращі практики використання цієї технології в поточному проекті: визначення шаблонів MVVM, асинхронне програмування C#, створення користувальницьких елементів керування, використання платформозалежного коду з DependencyService, налаштування стандартних елементів управління з використанням Renderers і оптимізація програми для максимальної продуктивності. Описано додавання нових можливостей до Xamarin.Forms великої спільноти розробників. Наведено приклади програмного коду та скріншоти програм. Використання розробленого програмного забезпечення дозволило підвищити ефективність торговельного обліку за рахунок зменшення кількості монотонних операцій і, як наслідок, зменшити кількість технічних помилок у роботі персоналу.*

Ключові слова: *розробка мобільних додатків; кросплатформовість; Xamarin.Forms; Android; Ios; UWP; .NET; C#; MVVM*

[1]**Кравченко Игорь Андреевич ,** кафедри компьютеризированных систем управления,E-mail: 10@gmail.com, ORCID: 0000-0003-1751-6049, г. Одесса, Украина
[1]**Сперанский Виктор Александрович,** кандидат технических наук кафедры компьютеризированных систем управления,, E-mail: speranskiyva@ukr.net, ORCID: 0000-0002-8042-1790, г. Одесса, Украина
[1]Одесский национальний политехнический университет, пр-т Шевченко, 1, г. Одесса, 65044, Украина

КРОССПЛАТФОРМЕННАЯ ПРАКТИКА РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ ДЛЯ АВТОМАТИЗИРОВАННОГО ТОРГОВОГО УЧЁТА

Аннотация. *Обсуждается проблема разработки единого приложения, способного работать на большинстве современных мобильных платформ. В работе описаны основные принципы и правила проектирования и разработки мобильных приложений, использующих кроссплатформенную технологию Xamarin.Forms. Описаны основные принципы и правила проектирования и разработки мобильных приложений, использующих кроссплатформенную технологию Xamarin.Forms. Работа базируется на разработке мобильного бизнес-приложения, которое уже используется в коммерческой компании, все приведенные примеры проверено в реальной работе. Статья состоит из двух частей. Первая часть описывает цели и преимущества используемой технологии Xamarin.Forms и содержит технические требования. Технология Xamarin.Forms требует использования объектно-ориентированного программирования в C#. Во второй части описываются лучшие практики использования этой технологии в текущем проекте: определение шаблонов MVVM, асинхронное программирование в C#, создание пользовательских элементов управления, использования платформозависимого кода используя DependencyService, настройки стандартных элементов управления с использованием Renderers и оптимизация программы для максимальной производительности. Описаны дополнения новых возможностей большого сообщества разработчиков к Xamarin.Forms. Приведены примеры программного кода и скриншоты программ. Использование разработанного программного обеспечения позволило повысить эффективность торгового учета за счет уменьшения количества монотонных операций и, как следствие, уменьшить количество технических ошибок в работе персонала.*

Ключевые слова: *разработка мобильных приложений; кроссплатформенность; Xamarin.Forms; Android; iOS; UWP; NET; C#; MVVM*