University Of Zakho

Faculty of Science

Department Of Computer Science

Shopping Application

Prepared By:

Zana Jaseem Ibrahim

Zakya Sabah Muhammad

Adar Muhammad Hashim

Rihan Shaaban Salih

Supervisor:

Mr. Yousif Garabet Arshak

Study Year (2021-2022)

Table of Contents

Abstract

Shopping is an application intended for an online shop. The main objective of this application is to make it interactive and easy to use. It would make searching, viewing, and selection of a product easier. It contains a sophisticated search engine for the users to search for products specific to their needs. The search engine provides an easy and convenient way to search for products where a user can search for a product interactively and the search engine would refine the products available based on the user's input. The user can then view the complete specification of each product. They can also view the product reviews and also write their own reviews. The application also provides a drag and drop feature so that a user can add a product to the shopping cart by dragging the item into the shopping cart. The main emphasis lies in providing a user-friendly search engine for effectively showing the desired results and its drag and drop behavior.

Introduction

E-Commerce has improved business methods by giving businesses the opportunity of selling goods and services on a universal basis. The Internet provides traders the possibility of spreading out their shops into an infinite number of sites, and also gave consumers the advantage of shopping across all borders. To give the customers not only the advantage of buying anytime but also the advantage of buying anywhere, using a mobile device for E-commerce has become an alternative.

Progression of wireless technology made mobile devices more popular. From the customers' point of view, on one hand, it is advantageous to use mobile devices such as smartphones and take advantage of their mobility while surfing an online store, while on the other hand, the shop owners benefit from fulfilling the customer's desires. This new method of purchasing, which is more convenient for the consumers and more profitable for the shop owners, is called mobile commerce or in abbreviation m-commerce.

The problem with browsing online stores with smartphones is that these web shops are designed to be used with a device of big size screen and their functionality is optimized for those kinds of devices in comparison, smartphones have a much smaller screen size. As a result, the navigation process of listing the categories, selecting a product, viewing the details, adding it to the shopping cart, and issuing the order on the mobiles web browser is pretty tough, boring, and time-consuming.

The solution could be using the mobile device features in a suitable way for webshops to be optimized for that size of the screen. Taking the same functionality and adapting them to be used on a smartphone application could lead to a much better experience than using the native web browsing option.[1]

What are the benefits of an eCommerce mobile app?

1. **Increased brand recognition**
   One of the chief reasons to choose mobile eCommerce app development is increased brand visibility. Also, it’s the best advantage of mobile commerce.
2. **Improved marketing communication**
   With mobile gadgets, today, customers stay connected with the brands, 24/7. Such mobile devices have improved the way users interact with brands, get information, and shop. More and more customers prefer using mobile to shop online. So, it’s important for businesses to include such devices in their marketing approaches. This will assist in enhancing the way of interaction between the brand and customers.
3. **Enhanced customer experience**
   Trendy customers want a consistent and personalized experience all through their journey with brands. And using just websites as a medium to deal with your consumers would not make this happen. If you want your customers to be repetitive, opt for mobile apps to make this possible easily.
4. **Improved visitors’ engagement**
   Let you know that you can take advantage of your customers’ devices also. You can integrate the features of users’ phones with your application to simplify navigation and increase customer engagement.
5. **Increase average order value**
   Well, it’s not that easy to point out what motivates customers to spend more while shopping from your eStore, but here we will sum up a few factors.

   - **Push notifications**
     As discussed above, it notifies users about special discounts, sales, deals, or when the item comes back in stock.

   - **Easy Payment**
     You may give varied options of payment methods. Allow them to add a card they want for speedy transactions.

   - **One-Click Ordering**
     The simple and easy layout of the app makes checkout easier. So, you

should offer your customers one button as a one-click ordering option.

6. **Reduce cart abandonment rates**
   If we check the past records, we will find that mobile apps have lower car abandonment rates because of a simple checkout process. The system stores shipping and payment details that permit the users to checkout with just one click. This allows customers to complete checkout faster with no distractions. [1]

App Structure

Add HomeMasterDetailPage

First of all, let’s add a new page type off ***MasterDetailPage*** called HomeMasterDetailPage, in which we ‘ll define a menu and Hamburger button to show or hide the menu. Essentially, the menu will contain different product categories like (Dresses, Pants, Shoes …).

When the user clicks on the menu item, so we drive the user to ***ProductsPage*** and switch the clicked category.

```
<?xml version="1.0" encoding="utf-8" ?>
<MasterDetailPage xmlns="http://xamarin.com/schemas/2014/forms"
                  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                  xmlns:prism="http://prismlibrary.com" xmlns:Views="clr-
namespace:Shopping.Views"
                  prism:ViewModelLocator.AutowireViewModel="True"
                  x:Class="Shopping.Views.HomeMasterDetailPage"
                  NavigationPage.HasNavigationBar="False"
                  x:Name="MasterPage">
    <MasterDetailPage.Master>
        <NavigationPage Title="Menu"
                        IconImageSource="ic_hamburger.png">
            <x:Arguments>
                <ContentPage NavigationPage.HasBackButton="False"
                             Title="Menu"
                             Padding="0">
                    <Grid RowDefinitions="*,auto">
                        <StackLayout Margin="10">
                            <!--Dresses-->
                            <StackLayout >
```

```
                                <Label Text="Dresses"
                                       Style="{StaticResource MenuItemLabelStyle}"/>
                                <BoxView Style="{StaticResource 
BoxViewSeparatorStyle}"/>
                                <StackLayout.GestureRecognizers>
                                    <TapGestureRecognizer Command="{Binding 
Source={x:Reference MasterPage}, Path=BindingContext.MenuItemClickCommand}"
                                                          NumberOfTapsRequired="1"
                                                          CommandParameter="Dresses">
                                </StackLayout.GestureRecognizers>
                            </StackLayout>
                            <!--Pants-->
                            <StackLayout>
                                <Label Text="Pants"
                                       Style="{StaticResource MenuItemLabelStyle}"/>
                                <BoxView Style="{StaticResource 
BoxViewSeparatorStyle}"/>
                                <StackLayout.GestureRecognizers>
                                    <TapGestureRecognizer Command="{Binding 
Source={x:Reference MasterPage}, Path=BindingContext.MenuItemClickCommand}"
                                                          NumberOfTapsRequired="1"
                                                          CommandParameter="Pants"/>
                                </StackLayout.GestureRecognizers>
                            </StackLayout>
                            <!--Shoes-->
                            <StackLayout>
                                <Label Text="Shoes"
                                       Style="{StaticResource MenuItemLabelStyle}"/>
                                <BoxView Style="{StaticResource 
BoxViewSeparatorStyle}"/>
                                <StackLayout.GestureRecognizers>
                                    <TapGestureRecognizer Command="{Binding 
Source={x:Reference MasterPage}, Path=BindingContext.MenuItemClickCommand}"
                                                          NumberOfTapsRequired="1"
                                                          CommandParameter="Shoes"/>
                                </StackLayout.GestureRecognizers>
                            </StackLayout>
                        </StackLayout>
                    </Grid>
                </ContentPage>
            </x:Arguments>
        </NavigationPage>
    </MasterDetailPage.Master>
    <MasterDetailPage.Detail>
        <NavigationPage>
            <x:Arguments>
```

```
                <Views:MainPage/>
            </x:Arguments>
        </NavigationPage>
    </MasterDetailPage.Detail>
</MasterDetailPage>
```

Style

Like in web development, in Xamarin Forms we have App. xaml file, in which we can centralize the common style used for Menu items, frames… Obviously, the objective of this is to avoid code duplication.

```
<!--Colors-->
  <Color x:Key="PrimaryColor">#00416E</Color>
  <Color x:Key="SeparatorGrayColor">#DCDCDC</Color>
  <Color x:Key="BlackColor">#000</Color>
  <Color x:Key="WhiteColor">#FFF</Color>
  <Color x:Key="StylishGrayColor">#4B515D</Color>

  <!--Navigation bar-->
  <Style TargetType="NavigationPage">
      <Setter Property="BarBackgroundColor" Value="{StaticResource PrimaryColor}"/>
      <Setter Property="BarTextColor" Value="White"/>
  </Style>

  <Style x:Key="MenuItemLabelStyle" TargetType="Label">
      <Setter Property="FontSize" Value="20"/>
      <Setter Property="Margin" Value="10,0,0,0"/>
      <Setter Property="TextColor" Value="{StaticResource BlackColor}"/>
      <Setter Property="VerticalTextAlignment" Value="Center"/>
  </Style>

  <Style x:Key="BoxViewSeparatorStyle" TargetType="BoxView">
      <Setter Property="Margin" Value="0,3"/>
      <Setter Property="HeightRequest" Value="1"/>
      <Setter Property="HorizontalOptions" Value="FillAndExpand"/>
      <Setter Property="Color" Value="{StaticResource SeparatorGrayColor}"/>
  </Style>
```

On the other side, we don’t have much to put in the viewmodel. Only the binding of the **MenuItemClickCommand**. So your ViewModel will look like:

```
public class HomeMasterDetailPageViewModel : ViewModelBase
    {
        #region Properties
        #endregion
        #region Commands
        public ICommand MenuItemClickCommand { get; private set; }
        #endregion
        public HomeMasterDetailPageViewModel(INavigationService navigationService)
            :base(navigationService)
        {
            MenuItemClickCommand = new DelegateCommand<string>(MenuItemClick);
        }
        private void MenuItemClick(string categoryName)
        {
            var nagivationParams = new NavigationParameters { { "CategoryName",
categoryName } };
            NavigationService.NavigateAsync($"{nameof(NavigationPage)}/{nameof(Produc
tsPage)}", nagivationParams);
        }
    }
```

ProductsService and ProductsPage

Now, we have gone to create a new page called ***ProductsPage***. On this page, we will display the list of products in the selected category. In general, we show a collection view that contains Thumbnails, the product name, and the price.

When the user taps on an item from this collection view, we navigate to a new page in which we can show more details of the selected product like description, technical details or the different sizes and colors are available, the price…. and eventually don’t forget the Buy button.

So, in the ***ProductsPage***, We have added the collection view. Bind the ***ItemsSrouce*** property to ProductsList, which will be defined in the ViewModel class. The ***ItemsLayout*** property is equal to “VerticalGrid, 2”, which means that the collection view will display items in a grid with 2 columns and has a vertical orientation.

```
<?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:prism="http://prismlibrary.com"
```

```
         prism:ViewModelLocator.AutowireViewModel="True"
         x:Class="Shopping.Views.ProductsPage"
         Title="{Binding Title}"
         x:Name="ProductsView">
<Grid>
    <CollectionView ItemsSource="{Binding ProductsList}"
                    ItemsLayout="VerticalGrid, 2"
                    Margin="5">
        <CollectionView.ItemTemplate>
            <DataTemplate>
                <StackLayout>
                <Grid RowDefinitions="auto,30,50,1"
                      ColumnDefinitions="*,70"
                      Margin="0,4,0,0">
                    <Frame BorderColor="LightGray"
                           Padding="3"
                           Grid.ColumnSpan="2">
                        <Image Source="{Binding Thumbnail}"
                           Aspect="AspectFill"
                           VerticalOptions="Center"
                           HorizontalOptions="Center"/>
                    </Frame>
                    <Label Grid.Row="1"
                           Grid.ColumnSpan="2"
                           Style="{StaticResource ProductTitleLabelStyle}"
                           HorizontalOptions="Start"
                           Text="{Binding Name}"/>
                    <!--<Label Grid.Row="2"
                           Grid.Column="1"
                           Style="{StaticResource ProductPriceLabelStyle}"
                           TextColor="{StaticResource BlackColor}"
                           HorizontalOptions="Start"
                           Text="{Binding Price, StringFormat='{0:C2}'}"/>-->
                    <Frame Grid.Row="2"
                           Grid.Column="1"
                           CornerRadius="5"
                           HasShadow="True"
                           Margin="0"
                           Padding="0"
                           BackgroundColor="{StaticResource StylishGrayColor}"
                           WidthRequest="60"
                           HeightRequest="35"
                           VerticalOptions="Center"
                           HorizontalOptions="Center"
                           BorderColor="Gray">
```

```
                                    <Label   Text="{Binding Price, 
StringFormat='{0:C2}'}"
                                            Style="{StaticResource 
ProductPriceLabelStyle}"/>
                        </Frame>

                        <Grid.GestureRecognizers>
                            <TapGestureRecognizer Command="{Binding 
Source={x:Reference ProductsView}, Path=BindingContext.ShowProductDetailsCommand}"
                                                  NumberOfTapsRequired="1"
                                                  CommandParameter="{Binding .}"/>
                        </Grid.GestureRecognizers>
                        <!--<Frame Grid.Row="2"
                               Grid.Column="1"
                               CornerRadius="50"
                               VerticalOptions="Center"
                               HorizontalOptions="Center"
                               Padding="5"
                               Margin="0,0,5,5">
                            <Image Source="add_to_basket.png"/>
                        </Frame>-->
                    </Grid>
                    </StackLayout>
                </DataTemplate>
            </CollectionView.ItemTemplate>
        </CollectionView>
    </Grid>
</ContentPage>
```

Eventually, use the ***TapGestureRecognizer*** like in the ***HomeMasterDetailPage*** to bind the user click on product to ***ShowProductDetailsCommand***. When user selects a product, he will be redirected to ***ProductDetailsPage***.

Before going to the ViewModel, we gonna create a new folder called Services, in which we create a new class called ***ProductsService*** and its interface ***IProductsService***.

In fact, the services play the role of provider. For example, we call the Web API to get the list of products. At the moment, we will make a stub to return a list of fake products.

So the service looks like this:

```
using Shopping.Models.Enums;
using Shopping.Models.Models;
using Shopping.Services.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;

namespace Shopping.Services
{
    public class ProductsService : IProductsService
    {
        /// <summary>
        /// Gets the available products list by category name.
        /// </summary>
        /// <param name="categoryName">The target category Name</param>
        /// <returns>The found prodycts.</returns>
        public IEnumerable<ProductModel> GetProductsByGategoryName(string
categoryName)
        {
            ProductsEnum productCategory;
            List<ProductModel> mockList = new List<ProductModel>
            {
                new ProductModel{ID = 1, Category = ProductsEnum.Pants, Name = "High
rise skinny jeans", Price = 50, Thumbnail = "https://m1.jeans-
industry.fr/201863-thickbox/jeans-bleu-delave-taille-haute.jpg" },
                new ProductModel{ID = 2, Category = ProductsEnum.Pants, Name = "High
waist destroyed jeans", Price = 50, Thumbnail = "https://m1.jeans-
industry.fr/210026-thickbox/jeans-taille-haute-destroy.jpg" },
                new ProductModel{ID = 3, Category = ProductsEnum.Dresses, Name =
"Long dress", Price = 50, Thumbnail = "https://m1.jeans-industry.fr/195837-
thickbox/robe-longue.jpg" },
                new ProductModel{ID = 4, Category = ProductsEnum.Dresses, Name =
"Ribbed belt long dress", Price = 50, Thumbnail = "https://m3.jeans-
industry.fr/190595-thickbox/robe-longue-cotele-a-ceinture.jpg" },
                new ProductModel{ID = 5, Category = ProductsEnum.Dresses, Name =
"Embroidery and lace dress", Price = 50, Thumbnail = "https://m3.jeans-
industry.fr/181489-thickbox/robe-a-broderies-et-dentelles.jpg" },
                new ProductModel{ID = 6, Category = ProductsEnum.Dresses, Name =
"Ribbed gathered dress", Price = 50, Thumbnail = "https://m2.jeans-
industry.fr/227238-thickbox/robe-cotelee-froncee.jpg" },
            };
            productCategory = (ProductsEnum)Enum.Parse(typeof(ProductsEnum),
categoryName);
            return mockList.Where(x => x.Category == productCategory);
        }
    }
}
```

Finally, let jump into the ViewModel file and add the ***ProductsList*** property and the ***ShowProductDetailsCommand***.

```
using Newtonsoft.Json;
using Prism.Commands;
using Prism.Navigation;
using Shopping.Models.Models;
using Shopping.Services.Interfaces;
using Shopping.Views;
using System.Collections.Generic;
using System.Windows.Input;

namespace Shopping.ViewModels
{
    public class ProductsPageViewModel : ViewModelBase
    {
        #region Properties
        private readonly IProductsService productsService;

        private IEnumerable<ProductModel> productsList;

        public IEnumerable<ProductModel> ProductsList
        {
            get => productsList;
            set => SetProperty(ref productsList, value);
        }

        private ProductModel selecetdProduct;
        public ProductModel SelectedProduct
        {
            get => selecetdProduct;
            set => SetProperty(ref selecetdProduct, value);
        }
        #endregion

        #region Commands
        public ICommand ShowProductDetailsCommand { get; private set; }
        #endregion

        public ProductsPageViewModel(INavigationService navigationService,
IProductsService productsService)
            : base(navigationService)
        {
            this.productsService = productsService;
            ShowProductDetailsCommand = new
DelegateCommand<ProductModel>(ShowProductDetails);
        }

        private void ShowProductDetails(ProductModel product)
        {
            var nagivationParams = new NavigationParameters { { "Product",
JsonConvert.SerializeObject(product) } };
            NavigationService.NavigateAsync(nameof(ProductDetailsPage),
nagivationParams);
        }
        public override void OnNavigatedTo(INavigationParameters parameters)
        {
        }
        public override void Initialize(INavigationParameters parameters)
        {
            string categoryName = parameters.GetValue<string>("CategoryName");
            Title = categoryName;
            InitData(categoryName);
        }
        private void InitData(string categoryName)
        {
            ProductsList = productsService.GetProductsByGategoryName(categoryName);
        }
    }
}
```

Add ProductDetailsPage

by adding a new tag called ***CarouselView*** through which we will display the different product images. Bind the ItemsSource to ProductImagesList that will be defined in the view model.
So the code will look like this:

```
<Grid RowDefinitions="3.8*,30,50,auto,*"
          Margin="5"
          ColumnDefinitions="*,*">
        <CarouselView ItemsSource="{Binding ProductImagesList}"
                      Margin="0,5"
                      Grid.ColumnSpan="2"
                      IndicatorView="IndicatorView">
            <CarouselView.ItemTemplate>
                <DataTemplate>
                    <StackLayout>
                        <Frame HasShadow="True"
                               BorderColor="DarkGray"
                               CornerRadius="5"
                               HorizontalOptions="Center"
                               Padding="5"
                               VerticalOptions="Center">
                            <Image Source="{Binding }"
                                   Aspect="AspectFill"/>
                        </Frame>
                    </StackLayout>
                </DataTemplate>
            </CarouselView.ItemTemplate>
        </CarouselView>
</Grid>
```

Finally, in this view, add the product label, price, and buy button to add the product to the basket. Note that styles were already defined in ***App.xaml*** file.

```
<Label Grid.Row="2"
               Text="{Binding ProductName}"
               Style="{StaticResource ProductTitleLabelStyle}"/>
        <Frame Grid.Row="2"
               Grid.Column="1"
               CornerRadius="5"
               HasShadow="True"
               Margin="0"
               Padding="0"
               BackgroundColor="{StaticResource StylishGrayColor}"
               WidthRequest="100"
               HorizontalOptions="Center"
               BorderColor="Gray">
            <Label  Text="{Binding Price, StringFormat='{0:C2}'}"

                    Style="{StaticResource ProductPriceLabelStyle}"/>
        </Frame>
        <Label Grid.Row="3"
               Grid.ColumnSpan="2"
               Style="{StaticResource ProductDescriptionLabelStyle}"
               Text="Premium dress with embroidery and lace Composition: 100%
POLYESTER"/>
        <Frame Grid.Row="4"
               Grid.ColumnSpan="2"
               Style="{StaticResource ButtonFrameStyle}">
            <Button Style="{StaticResource ButtonStyle}"
                    Text="Buy"/>
        </Frame>
```

App.xaml file:

```
<Style x:Key="ProductTitleLabelStyle" TargetType="Label">
            <Setter Property="TextColor" Value="{StaticResource BlackColor}"/>
            <Setter Property="FontSize" Value="18" />
            <Setter Property="VerticalOptions" Value="Center"/>
            <Setter Property="HorizontalOptions" Value="Center"/>
        </Style>
        <Style x:Key="ProductPriceLabelStyle" TargetType="Label">
            <Setter Property="VerticalOptions" Value="Center"/>
            <Setter Property="HorizontalOptions" Value="Center"/>
            <Setter Property="FontSize" Value="15" />
            <Setter Property="TextColor" Value="{StaticResource WhiteColor}" />
        </Style>
        <Style x:Key="ProductDescriptionLabelStyle" TargetType="Label"
BasedOn="{StaticResource ProductTitleLabelStyle}">
            <Setter Property="FontSize" Value="15" />
        </Style>
```

add the product size list, quantity, and the buy button. So the Product details page code will look like this:

```
<?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:prism="http://prismlibrary.com"
             prism:ViewModelLocator.AutowireViewModel="True"
             x:Class="Shopping.Views.ProductDetailsPage"
             Title="{Binding Title}">
    <ScrollView>
    <Grid RowDefinitions="400,30,50,auto,auto,80,80"
```

```
  Margin="5"
  ColumnDefinitions="*,*">
<CarouselView ItemsSource="{Binding ProductImagesList}"
              Margin="0,5"
              Grid.ColumnSpan="2"
              IndicatorView="IndicatorView">
    <CarouselView.ItemTemplate>
        <DataTemplate>
            <StackLayout>
                <Frame HasShadow="True"
                       BorderColor="DarkGray"
                       CornerRadius="5"
                       HorizontalOptions="Center"
                       Padding="5"
                       VerticalOptions="Center">
                    <Image Source="{Binding }"
                           Aspect="AspectFill"/>
                </Frame>
            </StackLayout>
        </DataTemplate>
    </CarouselView.ItemTemplate>
</CarouselView>
<IndicatorView x:Name="IndicatorView"
               Grid.ColumnSpan="2"
               Grid.Row="1"
               IndicatorColor="LightGray"
               SelectedIndicatorColor="DarkGreen"
               HorizontalOptions="Center"/>
<Label Grid.Row="2"
       Margin="20,0,0,0"
       Text="{Binding ProductName}"
       Style="{StaticResource ProductTitleLabelStyle}"/>
<Frame Grid.Row="2"
       Grid.Column="1"
       CornerRadius="5"
       HasShadow="True"
       Margin="0"
       Padding="0"
       BackgroundColor="{StaticResource StylishGrayColor}"
       WidthRequest="100"
       HorizontalOptions="Center"
       BorderColor="Gray">
    <Label  Text="{Binding Price, StringFormat='{0:C2}'}"
           Style="{StaticResource ProductPriceLabelStyle}"/>
</Frame>
```

```
        <Label Grid.Row="3"
               Grid.ColumnSpan="2"
               Margin="20,10,10,10"
               Style="{StaticResource ProductDescriptionLabelStyle}"
               Text="Premium dress with embroidery and lace Composition: 100%
POLYESTER"/>
        <StackLayout Grid.Row="4"
                     Grid.ColumnSpan="2"
                     Margin="20,0,0,0"
                     Orientation="Horizontal">
            <Label Text="Size:"
                    TextColor="{StaticResource BlackColor}"
                   VerticalTextAlignment="Center"/>
            <Picker ItemsSource="{Binding SizeList}" WidthRequest="40"
HorizontalOptions="Center"
                    SelectedItem="{Binding SelectedSize}" />
        </StackLayout>
        <StackLayout Grid.Row="5"
                     Grid.ColumnSpan="2"
                     Orientation="Horizontal"
                     Margin="20">
            <Label Text="Quantity:"
                   TextColor="{StaticResource BlackColor}"
                   VerticalTextAlignment="Center" />
            <Label Text="{Binding Quantity}"
                   TextColor="{StaticResource BlackColor}"
                   VerticalTextAlignment="Center"/>
            <Button Text="-"
                    Command="{Binding DecreaseQuantityCommand}"
                    VerticalOptions="Center"
                    WidthRequest="40"/>
            <Button Text="+"
                    Command="{Binding IncreaseQuantityCommand }"
                    VerticalOptions="Center"
                    WidthRequest="40"/>
        </StackLayout>
        <Frame Grid.Row="6"
               Grid.ColumnSpan="2"
               Style="{StaticResource ButtonFrameStyle}">
            <Button Style="{StaticResource ButtonStyle}"
                    Text="Buy"/>
        </Frame>
    </Grid>
    </ScrollView>
</ContentPage>
```

In ViewModel, define the size list, selected size, quantity and the different commands like increase or decrease the quantity.

```
using Newtonsoft.Json;
using Prism.Commands;
using Prism.Navigation;
using Shopping.Models.Models;
using Shopping.Services.Interfaces;
using System.Collections.Generic;
using System.Windows.Input;
namespace Shopping.ViewModels
{
    public class ProductDetailsPageViewModel : ViewModelBase
    {
        #region Properties
        private readonly IProductsService productsService;
        private IEnumerable<string> productImagesList;
        public IEnumerable<string> ProductImagesList
        {
            get => productImagesList;
            set => SetProperty(ref productImagesList, value);
        }
        private string productName;
        public string ProductName
        {
            get => productName;
            set => SetProperty(ref productName, value);
        }
        private float price;
        public float Price
        {
            get => price;
            set => SetProperty(ref price, value);
        }
        private int quantity;
        public int Quantity
        {
            get => quantity;
            set=> SetProperty(ref quantity, value);
        }
        private IEnumerable<string> sizeList;
        public IEnumerable<string> SizeList
        {
            get => sizeList;
            set => SetProperty(ref sizeList, value);
        }
        private string selectedSize;
        public string SelectedSize
        {
            get => selectedSize;
            set => SetProperty(ref selectedSize, value);
        }
        #endregion
        #region Commands

        public ICommand IncreaseQuantityCommand { get; set; }
        public ICommand DecreaseQuantityCommand { get; set; }
        #endregion
        public ProductDetailsPageViewModel(INavigationService navigationService,
IProductsService productsService)
            : base(navigationService)
        {
            this.productsService = productsService;
            IncreaseQuantityCommand = new DelegateCommand(IncreaseQuantity);
            DecreaseQuantityCommand = new DelegateCommand(DecreaseQuantity);
        }
        private void DecreaseQuantity()
        {
            if(Quantity >= 2)
            {
                Quantity--;
            }
        }
        private void IncreaseQuantity()
        {
            Quantity++;
        }
        public override void Initialize(INavigationParameters parameters)
        {
            ProductModel productModel =
JsonConvert.DeserializeObject<ProductModel>(parameters.GetValue<string>("Product"));
            Title = productModel.Name;
            ProductImagesList = productsService.GetProductsImages(productModel.ID);
            ProductName = productModel.Name;
            Price = productModel.Price;

            // Fake data
            SizeList = new List<string>() { "S", "M", "L" };
            Quantity = 1;
        }
```

```
    }
}
```

Card Button (Basket)

First of all, We added a custom navigation which contains the title, the card button, and the number of products available in the basket. So this code will be used on several pages like ***ProductDetails*** and ***ProductsPage***. To avoid code duplication, we created reusable ***UserControl*** called ***BasketHeader*** under ***Views/UserControls***.

Code will be like:

```
<?xml version="1.0" encoding="utf-8" ?>
<Grid   xmlns="http://xamarin.com/schemas/2014/forms"
         xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  x:Class="Shopping.Views.UserControls.BasketHeaderUserControl"
        ColumnDefinitions="*,60"
       x:Name="BasketHeader">
    <Label Text="{Binding Title}"
                   TextColor="White"
                   FontSize="20"
                   VerticalTextAlignment="Center"/>
    <Frame Grid.Column="1"
                   Margin="0"
                   Padding="5"
                   BorderColor="Green"
                   HasShadow="True"
                   CornerRadius="50"
                   WidthRequest="40"
                   BackgroundColor="White"
                   HorizontalOptions="Center">
        <ImageButton WidthRequest="40"
                             HeightRequest="40"
                             Padding="5"
                             Source="add_to_basket.png"
                             BackgroundColor="Transparent"
                             VerticalOptions="Center"
                             HorizontalOptions="Center"/>
    </Frame>
    <Frame Grid.Column="1"
                   WidthRequest="20"
                   VerticalOptions="Center"
                   HorizontalOptions="Center"
```

```
                    CornerRadius="50"
                    BackgroundColor="Red"
                    Margin="30,-30,0,0"
                     HasShadow="true"
                    BorderColor="Black"
                    Padding="1">
            <Label Text="{Binding ProductCount}"
                        TextColor="White"
                        HorizontalTextAlignment="Center"
                        VerticalTextAlignment="Center"/>
        </Frame>
        <Grid.GestureRecognizers>
            <TapGestureRecognizer Command="{Binding GoToBasketCommand}"
                                  NumberOfTapsRequired="1"/>
        </Grid.GestureRecognizers>
</Grid>
```

After that, we added 3 bindable properties Title, ProductCount and **GoToBasketCommand** in **BasketHeaderUserControl.cs .**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Shopping.Views.UserControls
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class BasketHeaderUserControl : Grid
    {
        public static readonly BindableProperty TitleProperty =
            BindableProperty.Create("Title", typeof(string), 
typeof(BasketHeaderUserControl));
        public static readonly BindableProperty ProductCountProperty =
          BindableProperty.Create("ProductCount", typeof(int), 
typeof(BasketHeaderUserControl));
        public static readonly BindableProperty GoToBasketCommandProperty =
            BindableProperty.Create("GoToBasketCommand", typeof(ICommand), 
typeof(BasketHeaderUserControl));
        public int ProductCount
        {
            get => (int)GetValue(ProductCountProperty);
            set => SetValue(ProductCountProperty, value);
        }
        public string Title
        {
            get => (string)GetValue(TitleProperty) ;
            set => SetValue(TitleProperty, value);
        }
        public ICommand GoToBasketCommand
        {
            get => (ICommand)GetValue(GoToBasketCommandProperty);
            set => SetValue(GoToBasketCommandProperty, value);
        }
        public BasketHeaderUserControl()
        {
            InitializeComponent();
        }
    }
}
```

MessagingCenter

MessagingCenter implements the publish-subscribe pattern. This mechanism allows also, publishers and subscribers to communicate without having a reference for each other. begin by ***send*** method which takes 3 parameters. We find the sender, the message, and the data to send.

the code looks like this:

```
public ICommand AddProductToBasketCommand { get; set; }
public ProductDetailsPageViewModel(INavigationService navigationService,
IProductsService productsService)
            : base(navigationService)
{
            this.productsService = productsService;
            IncreaseQuantityCommand = new DelegateCommand(IncreaseQuantity);
            DecreaseQuantityCommand = new DelegateCommand(DecreaseQuantity);
            AddProductToBasketCommand = new DelegateCommand(AddProductToBasket);
}
private void AddProductToBasket()
{
    MessagingCenter.Send("UpdateBasket", "Add product", productModel);
}
```

The most appropriate place to register for the product add event is ***ViewModelBase***. begin by adding a new method called ***SubscribeToEvents***.

```
private void SubscribeToEvents()
{
     MessagingCenter.Unsubscribe<string, ProductModel>("UpdateBasket", "Add 
product");
     MessagingCenter.Subscribe<string, ProductModel>("UpdateBasket", "Add product", 
(sender, args) => {

     List<ProductModel> productsList = new List<ProductModel>();
     var jsonProductsList = Preferences.Get("BasketList", null);
     if (!string.IsNullOrEmpty(jsonProductsList))
     {
        productsList = 
JsonConvert.DeserializeObject<List<ProductModel>>(Preferences.Get("BasketList", 
null));
     }
     productsList.Add(args);
     ProductCount = productsList.Count;
     Preferences.Set("BasketList", JsonConvert.SerializeObject(productsList));
     });
}
```

Conclusion

Obviously, there is still a lot to work on in the mobile application if I want to launch it to the market. There is also the possibility of developing the iOS version of the application. In fact, even if the application doesn’t bring the company many orders (through the application), it could serve as a marketing strategy. This will be done after adding the push notifications to it and sending notifications to users whenever there is a new product or a new discount. That is among the reasons why we chose to have the user register in the application, in order to have his data. This will allow us to send him promotions to his phone as well as his email address.

References

[1] "businessofapps," [Online]. Available: https://www.businessofapps.com/insights/benefits-of-building-a-mobile-app-for-ecommerce-business/. [Accessed 5 april 2022].

[2] "learntechnologies," [Online]. Available: https://learntechnologies.fr/2020/12/09/xamarin-forms-e-commerce-application/. [Accessed 6 april 2022].

[3] M. Hossein, M. Dirin, N. Mohammad, and H. Z. Khiabani, “Degree project E-Commerce on Android OS,” 2011.