# Razvoj aplikacije za odabir i narudžbu fotografija

**Zupičić, Davor**

Sveučilište u Rijeci – Odjel za informatiku

Diplomski studij informatike – Poslovna informatika

Davor Zupičić

# Razvoj aplikacije za odabir i narudžbu fotografija

Diplomski rad

Mentor: doc. dr. sc. Marija Brkić Bakarić

Rijeka, rujan 2018.

# Sadržaj

# Sažetak

Cilj je ovog diplomskog rada prikazati izradu Windows desktop aplikacije kroz Windows Presentation Foundation pristup (skraćeno WPF) služeći se .NET tehnologijom. Osim WPF-a, navedeni su i objašnjeni pristupi izrade aplikacija kroz Windows Forms i Universal Windows Platform (skraćeno UWP). Izrađena je aplikacija za odabir i narudžbu fotografija te je prikazano korisničko sučelje aplikacije, nakon čega su navedene i objašnjene ključne klase i funkcije u kôdu.

Aplikacija funkcionira na način da korisnik kroz sučelje radi odabir fotografija te zadaje količinu, format i vrstu foto-papira na kojem želi izraditi fotografiju. Navedene se informacije pohranjuju u posebnu datoteku. Kasnije prilikom same izrade fotografija, čitaju se informacije iz posebne datoteke koja nam sve tražene fotografije spremi u jednu mapu iz koje završavamo proces narudžbe. Svaka fotografija ima u nazivu naznačenu količinu te se nalazi unutar mape sa željenim formatom i vrstom foto-papira.



**Ključne riječi:** aplikacija, WPF, UWP, Windows Forms, fotografija

# 1. Uvod

Tema ovog diplomskog rada je razvoj aplikacije za odabir i narudžbu fotografija za operacijski sustav Windows. Zamišljena je kao sustav kroz koji bi kupci, odnosno korisnici aplikacije, lakše mogli naručivati izradu fotografija. Koncept aplikacije zamišljen je na temelju osobnog iskustva stečenog kroz studentski posao u fotografskoj radnji.

U izradu ovog rada krenuo sam s vrlo malo predznanja o aplikacijama za operacijski sustav Windows, primarno jer sam se prethodno bavio isključivo izradom mobilnih aplikacija za Android operativni sustav, npr. moj završni rad na preddiplomskom studiju informatike bila je upravo izrada mobilne aplikacije za lokalnu trgovinu pri čemu su kupci imali mogućnost lakše doći do popisa artikala i informacija o njima. Za aplikaciju odabira i narudžbe fotografija krenuo sam proučavati objektno-orijentirani programski jezik C# i pristupe kojima se izrađuju programi odnosno aplikacije za operacijski sustav Windows. Od pristupa naveo bih *Windows Forms, Universal Windows Platform* i konačno pristup za koji sam se odlučio, *Windows Presentation Foundation.*

Način na koji se trenutno obavlja proces narudžbe fotografija je sljedeći: kupac u fotografskoj radnji zauzme jedno prijenosno računalo i pregledava mape prepune fotografija, zapisuje na kovertu njihove brojeve (npr. *051,072,101,…*) i količinu te naposljetku osobno zaposleniku fotografske radnje izražava želju formata fotografija i vrstu foto-papira – najčešće je to format *13x18cm* te sjajni foto-papir. Nakon što je kupac završio odabir fotografija, kuverta se dalje prosljeđuje djelatniku fotografske radnje koji mora na računalu otvoriti točnu mapu koja sadrži odabrane fotografije te jednu po jednu dodavati u proces izrade fotografija kroz zaseban program te odabrati vrstu foto-papira i format.

Radi se o jednostavnom procesu koji ne uzima toliko vremena pojedinačno, međutim zaposlenik utroši previše vremena na čitanje narudžbe s koverte, ponovno biranje fotografija, količine,… Ako se dogodi propust i izrađena je kriva fotografija, postavlja se pitanje je li kupac pisao nečitko ili je od silne količine fotografija zaposlenik zaboravio odabrati još neku fotografiju – problemi se nižu sami od sebe.

Upravo iz ovog opisa narudžbe jasno se nazire što je cilj ove aplikacije – korisniku će biti omogućeno da kroz aplikaciju odabire fotografije, označava njihovu količinu, format i vrstu foto-papira te u konačnici šalje narudžbu direktno zaposleniku koji će izraditi fotografije. Narudžba se zatim pokreće kao skripta koja pronalazi odabrane fotografije pomoću njihove putanje i kopira ih u zasebnu mapu, nazvanu po broju narudžbe. Zatim zaposlenik mora odabrati

sve fotografije u mapi narudžbe i pokrenuti njihovu izradu – više ne mora ručno „prepisivati“ fotografije i tražiti ih po mapama te se ne može dogoditi greška zaposlenika, jedino ako je kupac označio krivu fotografiju.

Smatram da ova aplikacija može pospješiti rad fotografskih radnji kroz više aspekata; ubrzava proces narudžbe, smanjuje se prostor za grešku pri izradi fotografije, zaposlenik troši manje vremena na (ponovni) odabir fotografija te ono najbitnije – kupac brže dobiva svoje fotografije.

# 2. Osnovni pojmovi

## 2.1. C#

Objektno orijentirani programski jezik C# (ili *C Sharp*) razvijen je u siječnju 2000. godine u timu kojeg je predvodio nizozemski programer Anders Hejlsberg.

C# razvijen je prvenstveno kao Microsoftov odgovor na širenje Jave kao programskog jezika – on služi kao pozadina iza Microsoftovog .NET okvira (engl. *Framework*) te za razliku od C++ programskog jezika sam brine o upravljanju memorijom što je velika prednost za programere.

## 2.2. .NET Framework

Microsoftov .NET okvir možemo najjednostavnije objasniti kao infrastrukturu pomoću koje se programi na Windows operacijskom sustavu lakše izrađuju i pokreću. .NET okvir sastoji se od *Common Language Runtime* komponente koja služi za izvršavanje .NET programa te *Framework Class Library*, repozitorija klasa, sučelja i skupa različitih vrsta podataka.

.NET čita se kao „dot-net“.

## 2.3. Windows Forms

Jedan dio Microsoft .NET okvira je Windows Forms, repozitorij s grafičkim sučeljem koji sadrži klase pomoću kojih se može izgraditi neka aplikacija vizualnim slaganjem gotovih elemenata te mijenjanjem karakteristika tih elemenata (npr. prozor u koji ćemo umetnuti grafički element slike, ispod njega tekstualni dio itd.)

Windows Forms je odličan pristup za izradu aplikacija koje nisu pretjerano ovisne o grafičkim sučeljima s previše sadržaja [1].

## 2.4. Windows Presentation Foundation

Skraćeno WPF, Windows Presentation Foundation jedan je pristup izradi aplikacija za Windows operacijski sustav pomoću .NET okvira – omogućava programerima da putem grafičkog sučelja lakše stavljaju grafičke elemente.

Windows Forms je za grafičke elemente koristio isti jezik u kojem je pisan ostatak programa, dok je Microsoft za WPF za grafički izgled ponudio XAML[1], njihovu verziju XML-a pomoću kojeg se definira grafički izgled programa. Prva verzija Windows Presentation Foundation-a je izašla 2006. godine u sklopu .NET okvira 3.0.

[1] XAML je skraćenica za eXtensible Application Markup Language

## 2.5. Extensible Application Markup Language

XAML jezik je za označavanje podataka, poseban po tome što skraćuje i pojednostavljuje kôd. Koristi se primarno u razvijanju WPF aplikacija, uostalom izašao je zajedno s WPF-om. Kako bi lakše predstavili važnost XAML-a u izradi grafičkog sučelja, uzmimo za primjer da želimo dodati grafički element tipke (engl. *Button*) unutar prozora programa – na slici 1 je prikaz WPF pristupa u XAML-u i istog kôda u C# programskom jeziku:

```
<StackPanel>
    <TextBlock Margin="20">Welcome to the World of XAML</TextBlock>
    <Button Margin="10" HorizontalAlignment="Right">OK</Button>
</StackPanel>
```

```
// Create the StackPanel
StackPanel stackPanel = new StackPanel();
this.Content = stackPanel;

// Create the TextBlock
TextBlock textBlock = new TextBlock();
textBlock.Margin = new Thickness(10);
textBlock.Text = "Welcome to the World of XAML";
stackPanel.Children.Add(textBlock);

// Create the Button
Button button = new Button();
button.Margin= new Thickness(20);
button.Content = "OK";
stackPanel.Children.Add(button);
```

*Slika 1 Usporedba kôda XAML i C#, preuzeto s [2]*

## 2.6. Universal Windows Platform

Skraćeno UWP, Universal Windows Platform najnoviji je od pristupa izrade aplikacija za Windows u usporedbi sa Windows Forms i WPF-om. UWP je sučelje za programiranje aplikacija[2] napravljeno radi lakšeg stvaranja aplikacija za Windows 10, Windows 10 Mobile, Xbox One i HoloLens; ukratko aplikacija se piše jednom, a može se pokrenuti na više računalnih platformi koje koriste Windows operativni sustav. Aplikacija se dakle ne mora pisati iznova za drugu platformu [3].

## 2.7. Microsoft Visual Studio

Microsoft je razvio Visual Studio kao integrirano razvojno okruženje u veljači 1997. godine, prije skoro dvadeset i dvije godine. Pokušao je ujediniti više programskih okruženja i više programskih jezika u jedno mjesto kako bi olakšao razvijanje programa. Prva verzija nazvana

[2] Sučelje za programiranje aplikacija (engl. *Application Programming Interface*, skraćeno API) predstavlja skup funkcija, metoda, klasa i ostalih komponenti koje služe da bi olakšale programeru pisanje neke aplikacije – npr. programer ne mora sam implementirati neku funkciju jer ona već postoji u sklopu sučelja, on ju samo koristi u pisanju kôda [8]

je Boston, a kasnije su slijedile nove verzije s imenima Aspen, Everett, itd., sve po američkim gradovima. Verzija u trenutku pisanja ovog diplomskog rada je Microsoft Visual Studio 2017, a izašla je u ožujku 2017. godine uz manje zakrpe koje su slijedile u nadolazećim mjesecima. Sljedeća verzija najavljena je za 2019. godinu.

## 2.8. DataBinding

WPF je u svom pristupu uveo mehanizam po imenu DataBinding, a služi jednosmjernoj i dvosmjernoj vezi podataka. Jednosmjerna veza podataka podrazumijeva da će se promjenom izvora podataka automatski promijeniti odredište podataka (npr. unos podataka u neku varijablu i prikaz te varijable istovremeno pored polja za unos). [4] Iako u početku zvuči kompliciran, zapravo pojednostavljuje kôd – u XAML-u je potrebno neki element vezati pomoću neke ključne riječi – slijedi primjer iz ove aplikacije na slici 2.

```
<Viewbox Grid.Row="1" Grid.Column="3" Grid.RowSpan="3">
    <Image Source="{Binding}" Name="Velika"/>
</Viewbox>
```

*Slika 2 DataBinding primjer s jednosmjernom vezom*

Postavljen je element tipa *Image* s imenom *Velika*, a zatim postavili da je izvor te fotografije povezan – dok je u kôdu potrebno dodati jedino

```
Velika.Source = new BitmapImage(new Uri(Images[0].path));
```

pri čemu dalje program povezuje to dvoje, bez deklariranja; jasno mu je da je to isti element, a specifično u ovom primjeru postavljamo da veliki element slike dobiva putanju prve po redu fotografije u nekoj mapi s fotografijama (npr. 001.jpg, najčešće).

Dvosmjerna veza podataka podrazumijeva da se izmjena vrijednosti može odvijati bilo na izvoru ili odredištu. Primjer dvosmjerne veze prikazan je na slici 3.

```
<DataGrid Grid.Row="1" Name="Order_data" ItemsSource="{Binding Order, Mode=TwoWay,
                      UpdateSourceTrigger=PropertyChanged}" AutoGenerateColumns="False"
                      CanUserAddRows="False" ScrollViewer.VerticalScrollBarVisibility="Auto">
```

*Slika 3 DataBinding primjer s dvosmjernom vezom*

Primjer će biti jasan kada se pogleda slika 18 – promjena vrijednosti u kartici **Pregled narudžbe** utjecat će na vrijednosti u kartici **Odabir fotografija**, dakle neovisno o mjestu gdje mijenjamo podatke, na drugom mjestu su također promijenjeni.

## 2.9. Lambda izraz

Lambda izrazi su anonimne (bezimene) funkcije koje vraćaju konstantne vrijednosti, a nije od nikakve važnosti imati posebnu funkciju koja će obavljati taj zadatak. Za primjer navodim sljedeću liniju kôda:

```
currentImage = Images.Find(x => x.path == path);
```

Ovaj dio kôda radi sljedeće – varijabli *currentImage* potrebno je dodijeliti fotografiju koja se u tom trenutku pregledava kako bismo preko putanje imali podatke trenutne fotografije. Dakle kroz listu fotografija traži se ona fotografija čija lokacija odgovara onoj trenutne fotografije i na taj način se mogu s lakoćom dohvatiti ostali bitni podaci.

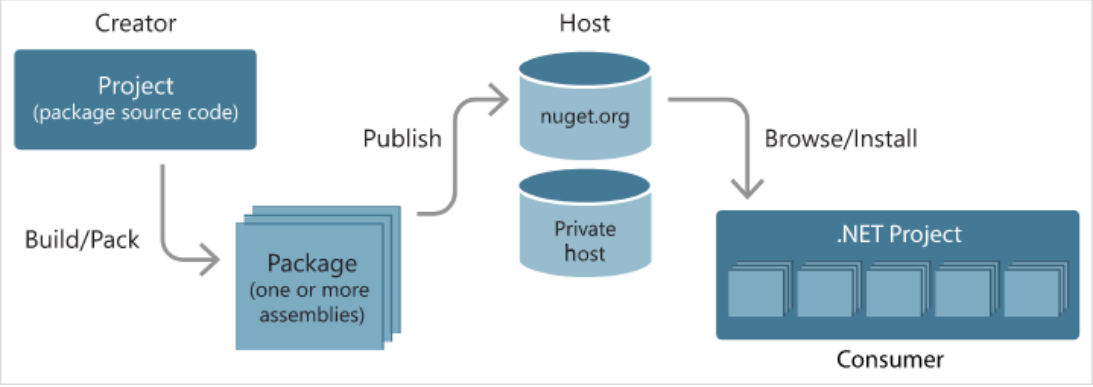
## 2.10. Ključna riječ *using*

.NET okvir sam brine o alociranju i čišćenju memorije. Svaki put kada se kreira novi objekt, alocira se određeni dio memorije te će sve funkcionirati dokle god ima dovoljno memorije. Međutim, povremeno se uključuje .NET-ov *Garbage Collector* koji provjerava koji se objekti ne koriste te obavlja čišćenje memorije i oslobađa je za sljedeće objekte. [5]

Ako želimo osigurati da se čistač memorije aktivira na vrijeme, možemo koristiti ključnu riječ *Using* koja automatski javlja da treba očistiti memoriju nakon što je završila sa svojim zadatkom, stoga je i kreiranje .bat datoteke obavljeno unutar te funkcije.

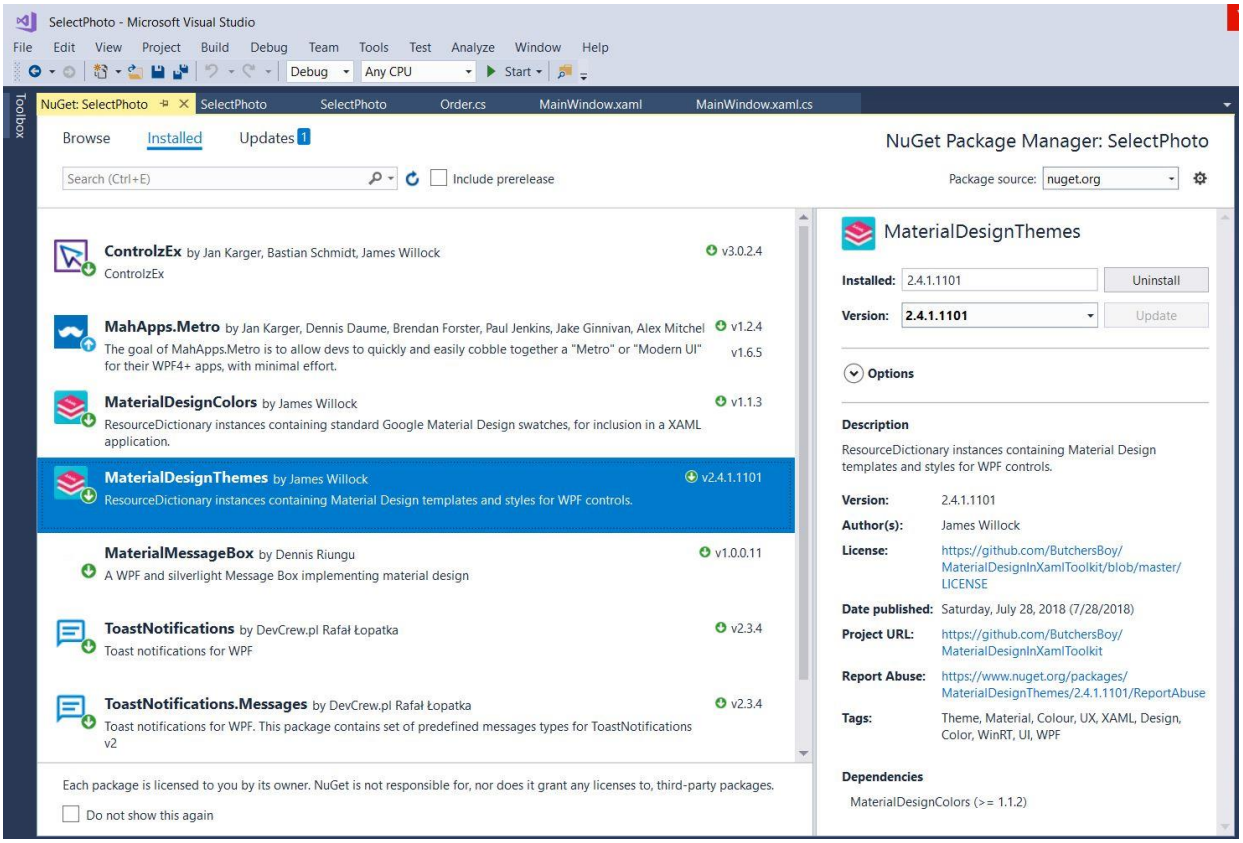
## 2.11. NuGet repozitorij

NuGet repozitorij je Microsoftov mehanizam za dijeljenje kôdova, sličan Githubu. Na slici 4 prikazano je povezivanje s NuGet platformom, preuzeto s [6].




*Slika 4 Povezivanje s NuGet platformom*

NuGet repozitorij sastoji se od paketa koje jednostavnom instalacijom dodajemo u aplikaciju. Opcija *Manage NuGet Packages* nalazi se u Microsoft Visual Studiu u alatnoj traci pod *Project*. Klikom na tu opciju, otvara se sljedeći zaslon kao na slici 5.




*Slika 5 Upravljanje NuGet paketima u Microsoft Visual Studiu*

Proces instalacije je jednostavan – pretragom se pronađe potrebna biblioteka i potom se instalira pritiskom na *Install* tipku u desnom dijelu prozora. U ovom projektu korišteno je sedam različitih NuGet paketa): ControlzEx[3], MahApps.Metro[4], MaterialDesignColors i MaterialDesignThemes,[5] MaterialMessageBox[6], ToastNotifications i ToastNotifications.Messages[7].

[3] https://www.nuget.org/packages/ControlzEx/, autori Jan Karger, Bastian Schmidt, James Willock
[4] https://www.nuget.org/packages/MahApps.Metro, autori Jan Karger, Dennis Daume, Brendan Forster, Paul Jenkins, Jake Ginnivan, Alex Mitchell
[5] https://www.nuget.org/packages/MaterialDesignColors i https://www.nuget.org/packages/MaterialDesignThemes, autor James Willock
[6] https://www.nuget.org/packages/MaterialMessageBox, autor Dennis Riungu
[7] https://www.nuget.org/packages/ToastNotifications i https://www.nuget.org/packages/ToastNotifications.Messages, autor DevCrew.pl Rafał Łopatka

# 3. Usporedba Windows Forms, UWP i WPF pristupa

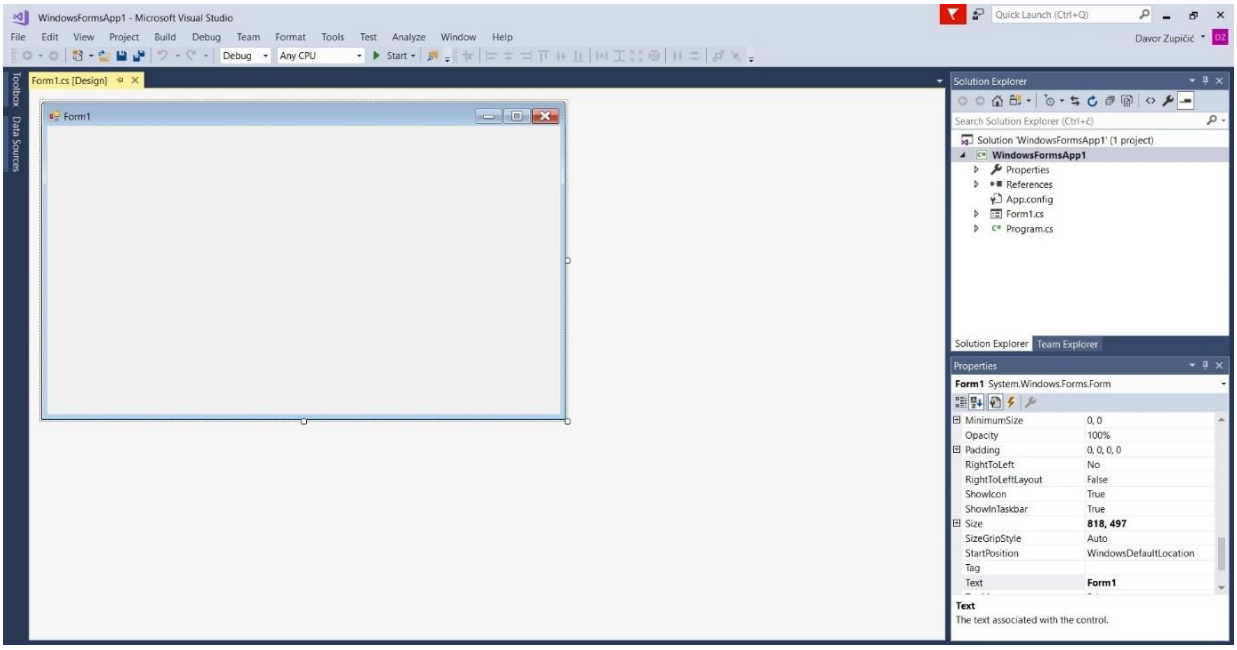
U ovom dijelu diplomskog rada usporedit ću tri pristupa izrade aplikacija za operacijski sustav Windows.

Prije svega, kada sam započinjao s izradom ove aplikacije, proveo sam neko vrijeme smišljajući kako bih zapravo htio da aplikacija izgleda, koje funkcionalnosti bi trebala nuditi. Obzirom da sam u prošlosti radio uglavnom mobilne aplikacije za Android operativni sustav, čak sam razmišljao prebaciti aplikaciju u mobilni oblik, međutim mobitel bi bio potpuno krivi medij za pregled fotografija – uvijek treba gledati fotografije na velikom zaslonu, a osim toga opseg posla koji je u fotografskoj radnji ne bi dopuštao pregled fotografija na mobilnom uređaju (npr. tabletu), jednostavno bi bilo nepraktično. Istraživanjem sam došao do tri glavna pristupa – Windows Forms, Windows Presentation Foundation te Universal Windows Platform.

## 3.1. Pristup izrade u Windows Forms

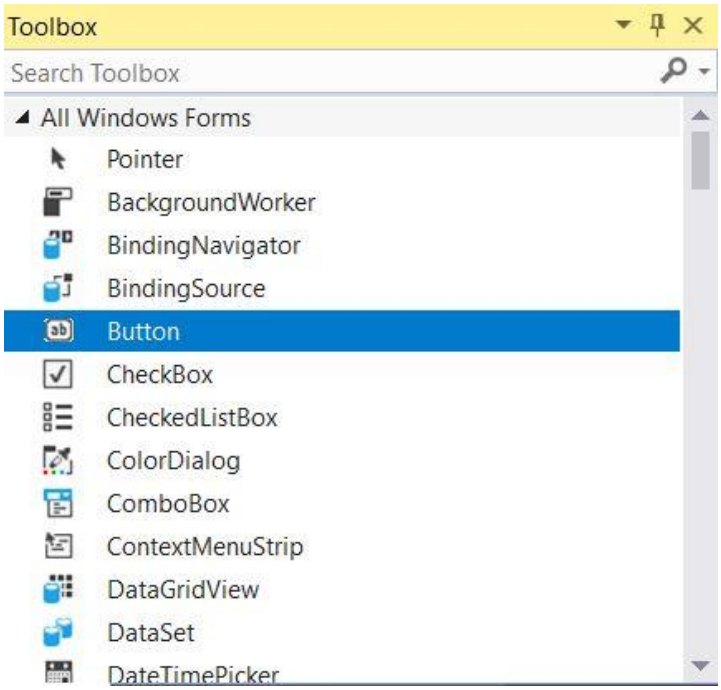
Microsoft ima snažnu dokumentaciju za izradu aplikacije te potiče korisnike da odaberu .NET okvir za izradu (među koje spadaju gornja tri pristupa). Osim .NET-a, također ima riječi o izradi aplikacija kroz programski jezik C++ i Win32 [7].

Na slici 6 prikaz je novog Windows Forms projekta u sklopu Microsoft Visual Studia.




*Slika 6 Novi Windows Forms projekt u Microsoft Visual Studiu*

Kako bi počeli dodavati elemente, potrebno je kroz padajući izbornik *View* u alatnoj traci otvoriti *Toolbox* kroz koji ćemo moći birati elemente za slaganje prozora, prikazan na slici 7.



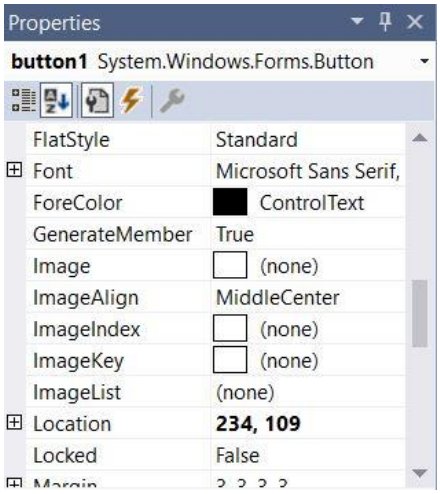

*Slika 7 Toolbox u Windows Forms pristupu*

Jedna od zamjerki Windows Forms pristupu vezana je uz način izrade grafičkog sučelja aplikacije. Radi se o uskoj povezanosti korisničkog sučelja i kôda na sljedeći način: nakon što smo dodali neki element putem *Toolboxa*, istovremeno se stvorio taj element u kôdu. Primjer je prikazan na slici 8.

```
this.button1 = new System.Windows.Forms.Button();
this.SuspendLayout();
//
// button1
//
this.button1.Location = new System.Drawing.Point(234, 109);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(75, 23);
this.button1.TabIndex = 0;
this.button1.Text = "button1";
this.button1.UseVisualStyleBackColor = true;
this.button1.Click += new System.EventHandler(this.button1_Click);
```

*Slika 8 Windows Forms - kôd za jedan Button element*

Kako bi razumjeli izradu grafičkog sučelja aplikacije kroz WindowsForms, morali smo znati programski jezik u kojem je pisan kôd. (C#, VB.NET ili C++) [8]. Naravno, na svojstva elementa možemo utjecati kroz izbornik *Properties*, prikazan na slici 9.

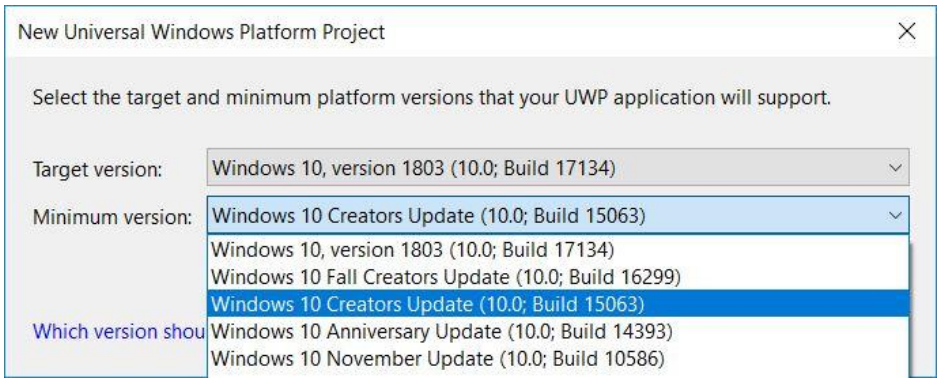



*Slika 9 Svojstva jednog Buttton elementa*

Promjenom vrijednosti u kôdu, utječemo na izgled elementa, kao što bi promjenom vrijednosti u izborniku *Properties* došlo do izmjene kôda.

## 3.2. Pristup izrade u UWP-u

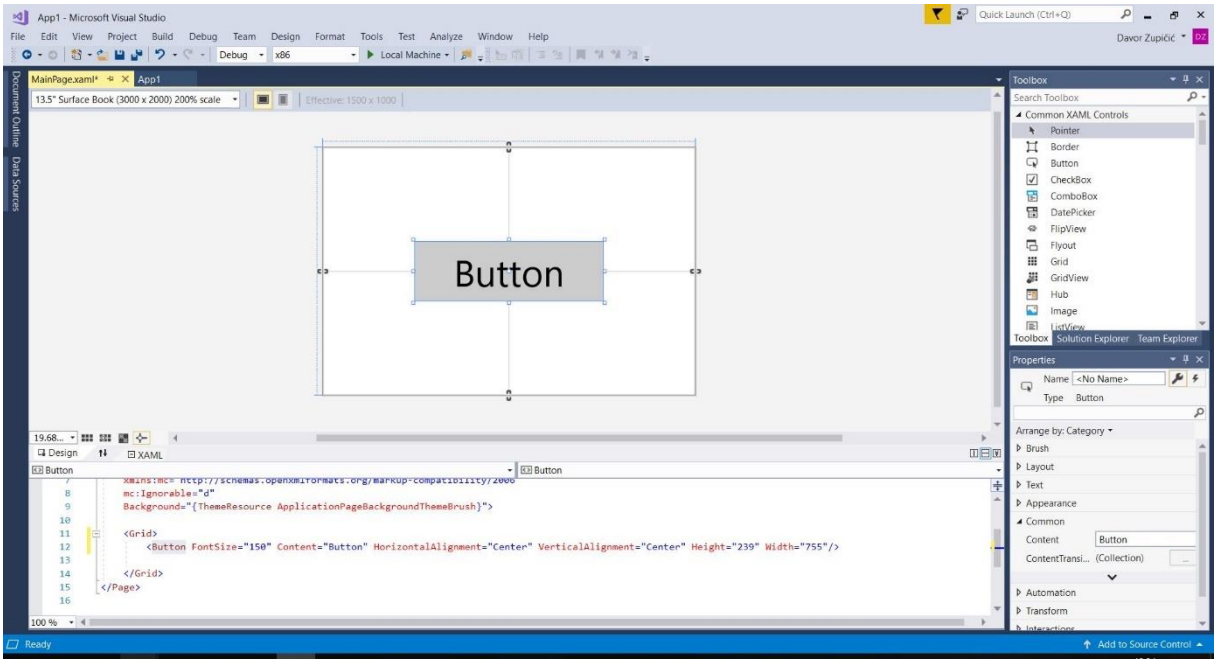
Prilikom pokretanja novog UWP projekta u Microsoft Visual Studiu, potrebno je odabrati minimalnu verziju na kojoj će aplikacija raditi te optimalnu verziju operativnog sustava kao što je prikazano na slici 10.




*Slika 10 Odabir verzije za UWP projekt*

Ciljana verzija odgovara najnovijoj –nema smisla raditi projekt za stariju verziju. Ovakvu vrstu odabira verzija često nalazimo i kod izrade mobilnih aplikacija za Android operativni sustav – kada krećemo u pisanje Android aplikacije, moramo odabrati koja će najstarija verzija Androida biti podržana. Odabir najstarije verzije vrlo je bitan u pisanju aplikacije jer neka opcija koja je moguća u najnovijoj verziji neće biti dostupna u nekoj starijoj, što automatski znači da nam je ta opcija isključena. Ukoliko ju pokušamo uključiti, upozoreni smo da nam je minimalna verzija prestara za tu opciju i da se preporuča promjena minimalne verzije naše aplikacije.

Kako bi počeli pisati kôd, potrebno je uključiti Razvojni način (engl. *Developer mode*) u postavkama operacijskog sustava Windows – bez toga programiranje u UWP-u nije moguće jer je potrebno moći instalirati aplikacije koje pišemo, a ne nalaze se na Microsoft Store-u. Na slici 11 prikazan je novi UWP projekt s jednim *Button* elementom.




*Slika 11 Novi UWP projekt s jednim Button elementom*

Gotovo da nema razlike između WPF-a i UWP-a po pitanju slaganja elemenata u XAML datoteci, međutim UWP je usmjeren upravljanju putem dodira (engl. *Touch*) tako da će potpora za tu funkcionalnost biti jača nego u WPF-u.

Microsoft je današnji UWP uveo 2015. godine i namijenio ga razvoju za Windows 10 operacijski sustav, što znači da je razvoj za starije verzije Windowsa onemogućen. Windows je svojevremeno ukinuo Windows Phone[8] i zamijenio ga s Windows 10 Mobile tako da je UWP korišten primarno za razvoj aplikacija za Windows 10 operacijski sustav, Windows 10 Mobile te Xbox One, Microsoftovu platformu za videoigre. [9]

Glavni nedostatak UWP pristupa smatra se njegova „ograničenost“ za Windows 10 – razvoj poslovnih aplikacija kroz UWP pristup jednostavno se ne preporuča jer brojne tvrtke i dalje koriste starije verzije Windowsa te se traži svojevrsna jednostavnost u upisivanju i dohvaćanju podataka. Kao što je Windows Forms jedan sloj iznad samog operacijskog sustava

[8] Windows Phone bio je pametni telefon kojeg je Windows plasirao na tržište kako bi konkurirao Appleovom iPhoneu te Android pametnim uređajima. Zadnja verzija operacijskog sustava za Windows Phone izdana je 2015. godine te je Windows Phone zamijenjen s Windows 10 Mobile

Windows, tako je UWP sloj iznad Windowsa 10 i nudi ipak manje slobode nego WPF (iako će se to najvjerojatnije promijeniti kroz godine). Naravno, osim što ovisi koja je svrha aplikacije i gdje će se koristiti i u kakvom okruženju, programeri odlučuju o pristupu kroz koji će razvijati aplikaciju.

Uglavnom je navika krenuti u sustav koji je neko vrijeme stabilan – a to je u ovom slučaju WPF. Kroz godine, kako Windows bude nadograđivao UWP, zasigurno će postati praktičnije razvijati aplikacije u njemu. Uostalom, to je bio i slučaj s WPF-om – kada je izašao, većina programera je čekala da izađe malo stabilnije izdanje, odnosno da se počne više aplikacija raditi u njemu.

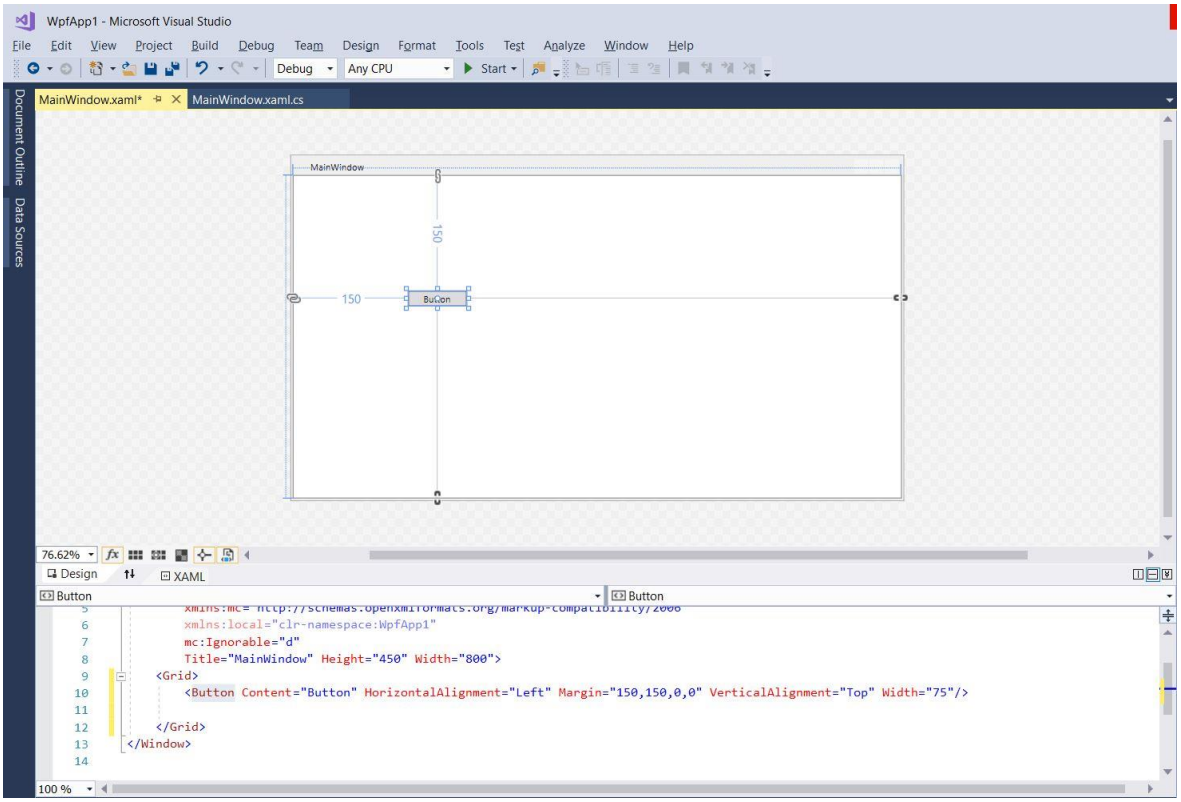
## 3.3. Pristup izrade u WPF-u

Odabrao sam WPF pristup izrade aplikacije zbog njegove prednosti (u odnosu na UWP) da podržava starije verzije Windowsa i dopušta veću slobodu u programiranju. UWP je vrlo kratko bio opcija o kojoj sam razmišljao za razvoj aplikacije tako da sam uglavnom kasnije razmišljao hoću li u izradu aplikacije krenuti Windows Forms pristupom ili WPF pristupom.

Kada uspoređujem Windows Forms i WPF, prednosti i nedostatke teško je navesti jer ovisi što programer preferira. Windows Forms pruža jednostavnost gdje se upravlja elementima kroz tablicu svojstava (engl. *properties*) poput pozadinske slike, margina, maksimalne veličine, vidljivosti, prozirnosti i ostalo. Naravno, sve ovo možemo postići u kôdu, ali svakako je korisno što se program slaže primarno vizualno pa se kasnije programiranjem postižu veze između elemenata. U neku ruku, to je nešto što WPF također nudi – slaganje elemenata iz *Toolboxa*.

Razlika je u tome što je Windows Forms zapravo samo sloj iznad Windowsa – prilikom programiranja nudi se sve ono što Windows ima u svojim programima pa je poprilično ograničeno što se može s postojećim opcijama dok Windows Presentation Foundation pristup nudi fleksibilnost jer nije ovisan o funkcionalnosti Windowsa koliko Windows Forms. WPF je također skalabilan i pruža više mogućnosti za izradu interaktivnih grafičkih sučelja za korisnike (što će biti vidljivo iz primjera ove aplikacije).

Na izvoru [10] navodi se prednost Windows Forms pristupa utoliko što ga je moguće pokrenuti na Windowsima starijim od Windowsa 2000 premda osobno smatram da to nije nedostatak WPF pristupa. Na slici 12 prikazan je novi WPF projekt s jednim *Button* elementom.




*Slika 12 Novi WPF projekt s jednim Button elementom*

Kada je .NET okvir 2006. godine izašao s verzijom 3.0, donio je brojne mogućnosti za upravljanje grafičkim elementima u aplikacijama. Prije verzije 3.0 razvoj grafičkog sučelja neke Windows Forms aplikacije podrazumijevao je da se programer zna služiti s raznim API-jevima kako bi npr. dodao video u aplikaciju. Na slici 13 je primjer, preuzet s [11]

| **Desired Functionality** | **Pre-.NET 3.0 Solution** |
|---|---|
| Forms, dialog boxes, controls | Windows Forms, VB6, MFC, and so on |
| 2D graphics | System.Drawing.dll (e.g., GDI+) or raw C-based GDI |
| 3D graphics | DirectX |
| Streaming video | Windows Media Player API or third party APIs |
| Fixed / Flow documents | Third party APIs |

*Slika 13 Razvoj grafičkog sučelja prije .NET 3.0 verzije*

Počevši s verzijom 3.0 pa nadalje, .NET okvir omogućio je da se svaka željena funkcionalnost rješava putem Windows Presentation Foundation pristupa.

Nakon što sam razmotrio sva tri pristupa, odlučio sam se za WPF. Iako je Universal Windows Platform novija tehnologija te će Microsoft sigurno više ulagati u razvoj tog pristupa,

za izradu ove aplikacije WPF jednostavno je bolji izbor. Novija tehnologija od Windows Forms pristupa, fleksibilnost izrade, skalabilnost aplikacije i mogućnost upravljanja grafičkim elementima kroz XAML pokazali su se kao prikladan izbor za izradu aplikacije za odabir i narudžbu fotografija. Naravno, tu je i činjenica da se WPF aplikacija može izraditi za Windowse starije od trenutne verzije Windows 10, što mi svakako odgovara.

Smatram da će u budućnosti ipak UWP prevladati, ali za sada WPF i Windows Forms ostaju najbolji pristupi za izradu poslovnih aplikacija.

# 4. Opis aplikacije

Aplikacija za odabir i narudžbu fotografija služi kao jednostavno grafičko sučelje za korisnike kojima će narudžba fotografija biti olakšana. Ideja je zamijeniti aktualni proces narudžbe – ručnog zapisivanja brojeva digitalnih fotografija i naknadno ručno pronalaženje istih fotografija kroz razne mape – novim softverskim rješenjem koji skraćuje postupak i direktno olakšava korisniku narudžbu i djelatniku fotografske radnje izradu fotografija.

Nakon što je korisnik odabrao fotografije, djelatnik fotografske radnje završava proces narudžbe ručnim unosom broja narudžbe u za to predviđeno polje. Ovaj korak je obvezan obzirom da fotografska radnja već ima spremne koverte s jedinstvenim brojem narudžbe te se isti zapravo prepisuje u aplikaciju, a izrađene fotografije se spremaju u već gotove koverte. Podatci narudžbe ne spremaju se u bazu podataka već se izvoze kao .bat datoteka s brojem narudžbe koji smo definirali u prethodnom koraku.

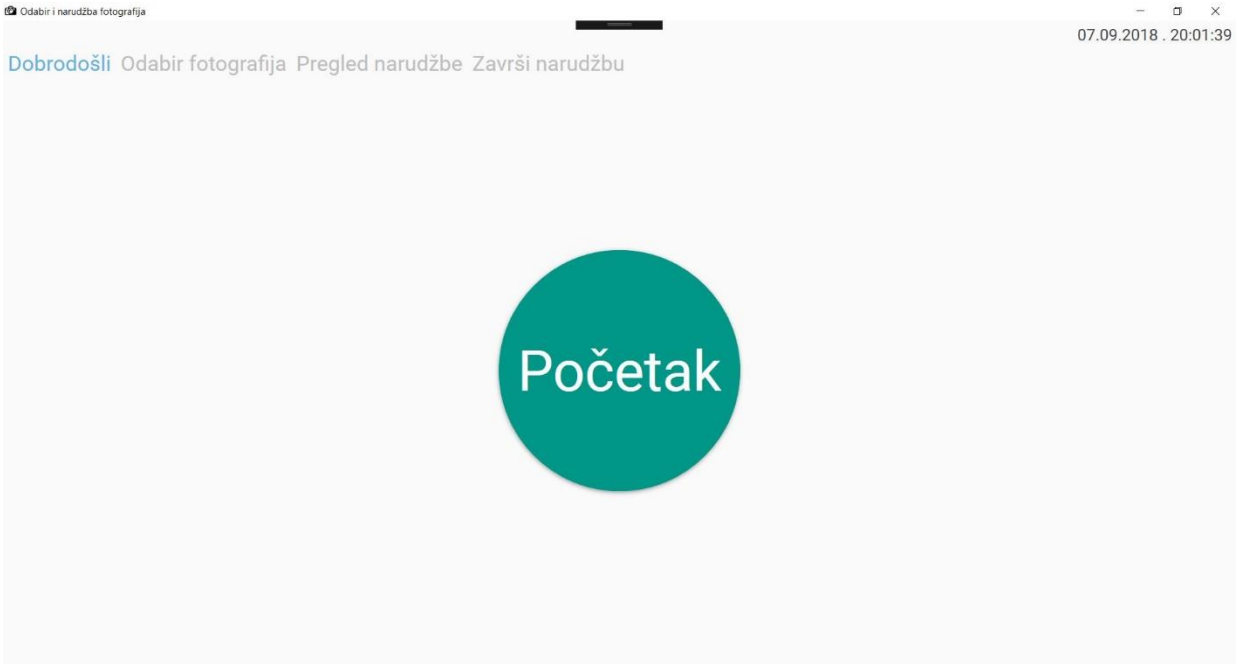
Datoteka narudžbe u .bat formatu je *batch*[9] datoteka koja sadrži naredbe kopiranja fotografija u novi direktorij nazivom narudžbe na računalu djelatnika radnje koji zatim te fotografije izrađuje u zasebnom softveru od stroja u fotografskoj radnji. Razlog za naknadno kopiranje fotografija je taj što korisnik odnosno kupac pregledava fotografije male rezolucije zbog bržeg pregleda dok su fotografije u visokoj rezoluciji pohranjene na računalu djelatnika radnje. Prednost ovog postupka je što baza podataka nije potrebna – djelatniku radnje .bat datoteka ostaje sačuvana, a korisnik bilo kada može ponoviti izradu svih ili pojedinih fotografija po jedinstvenom broju narudžbe.

Aplikacija je izrađena u razvojnom okruženju Microsoft Visual Studio uz Windows Presentation Foundation odnosno WPF pristup.

[9] *Batch* datoteka je skripta koja izvršava naredbe liniju po liniju

## 4.1. Korisničko sučelje

Prilikom pokretanja aplikacije, korisniku je prikazan početni ekran s tipkom za početak narudžbe. U gornjem desnom prikazani su vrijeme i datum. Na slici 14 je prikaz zaslona.




*Slika 14 Početni zaslon aplikacije, kartica **Dobrodošli***

Cijela aplikacija se izvršava unutar jednog prozora, a dijeli se na kartice **Dobrodošli**, **Odabir fotografija**, **Pregled narudžbe** te **Završi narudžbu**. Navigacija nije omogućena putem klika, već putem za to predviđenih tipki. Radi primjera, klikom na **Početak** otvorit će se direktorij:

*C:\fotke -small\00 ZARKOVICA*

Na slici 15 prikazan je sadržaj kartice **Odabir fotografija** na kojoj je već odabrano nekoliko fotografija s raznim formatima i vrstom foto-papira.

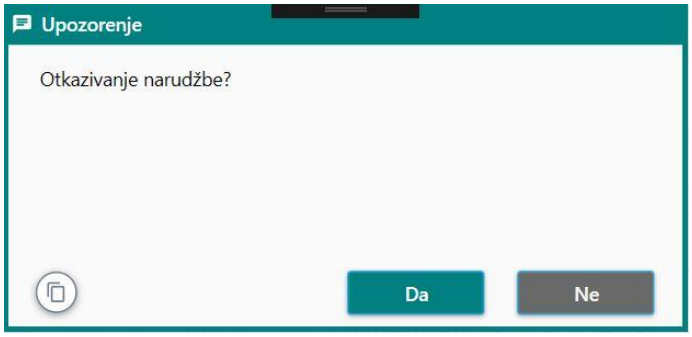

*Slika 15 Kartica **Odabir fotografija** s nekoliko izabranih fotografija*

U ovoj kartici automatski se učitavaju fotografije mape u kojoj se nalazimo – u ovom slučaju to je *00 ZARKOVICA*. Slika 16 prikazuje tipke za promjenu količine, foto-papira i formata fotografije te je uvijek naznačena količina koju smo odabrali za trenutnu fotografiju.


*Slika 16 Tipke za pojedinosti narudžbe*

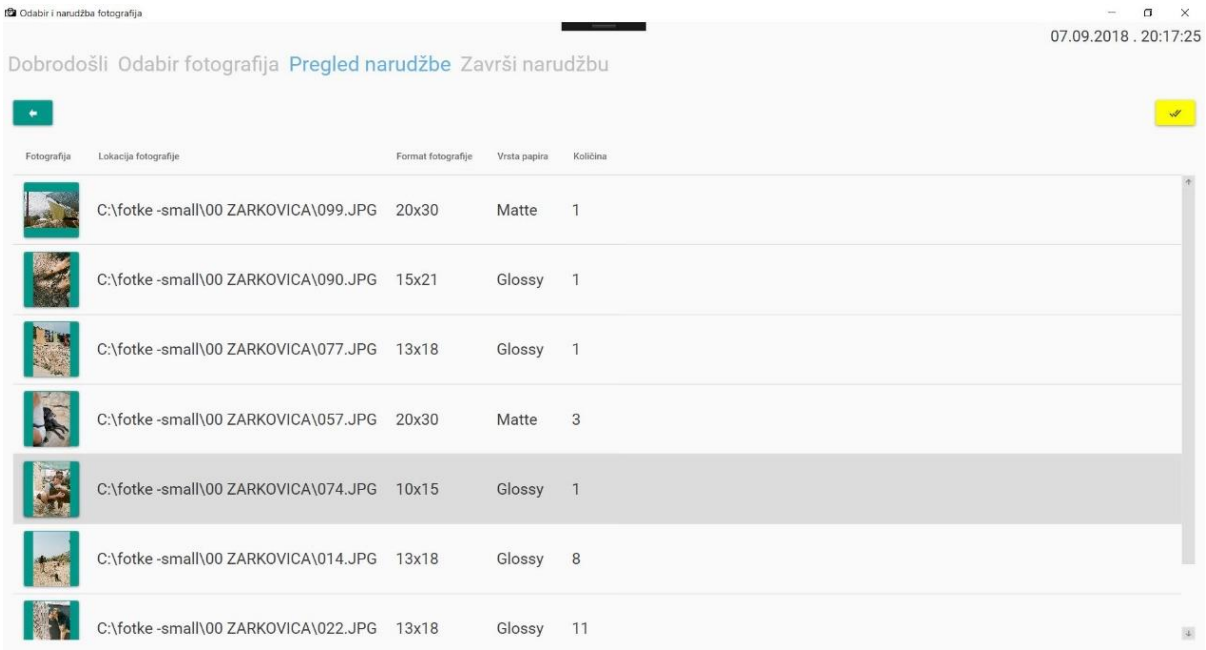
Korisnici aplikacije mogu odabrati vrste foto-papira i formate fotografija koje ta fotografska radnja nudi, u primjeru su to *glossy* i *matte* foto-papir (u prijevodu, nude se sjajne i mat fotografije) te veličine *10x15*, *13x18*, *15x21*, *20x30* u centimetrima. Navigacija između

fotografija omogućena je pritiskom na strelice na tipkovnici ili klikom miša na manje fotografije u popisu lijevo. Na slici 17 prikazano je upozorenje za otkazivanje narudžbe ukoliko je korisnik pritisnuo tipku za povratak na karticu **Dobrodošli** (samo ako je korisnik već odabrao neku fotografiju).




*Slika 17 Prozor s upozorenjem za otkazivanje narudžbe*

Slika 18 prikazuje karticu **Pregleda narudžbe** s popisom izabranih fotografija i njihovom količinom, formatom te vrstom foto-papira.




*Slika 18 Kartica **Pregled narudžbe** s pregledom izabranih fotografija*

Korisniku se pruža opcija da putem padajućih izbornika mijenja narudžbu (tako da npr. klikom na trenutni format može ga promijeniti u neki drugi). Isto vrijedi za vrstu foto-papira i količinu koju korisnik može sam upisati putem tipkovnice. Sljedeći korak je kliknuti mišem na žutu ikonu koja nas vodi na karticu **Završi narudžbu**, prikazanu na slici 19.


*Slika 19 Kartica **Završi narudžbu** sa sažetkom narudžbe*

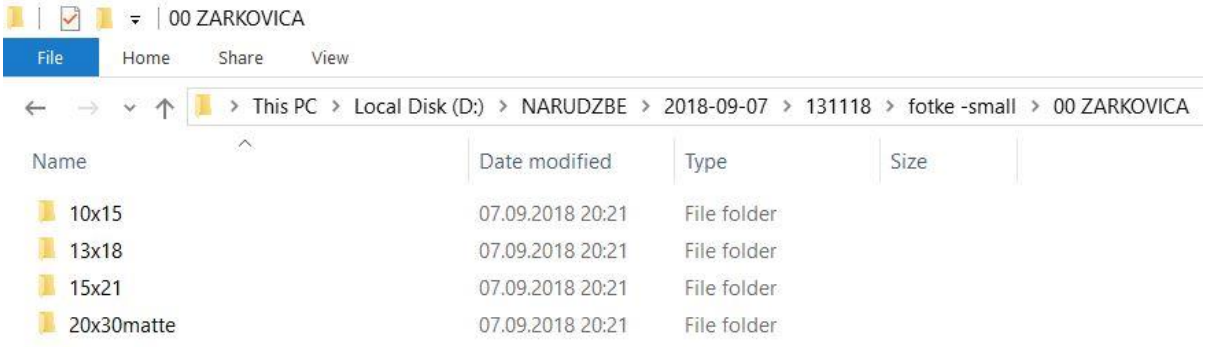
Zadnja kartica sadrži ukupan iznos fotografija po formatima te ukupnu količinu fotografija koje je korisnik odabrao. Djelatniku fotografske radnje preostaje da upiše broj narudžbe ovisno o tome koja je koverta sljedeća na redu. Ovdje završava dio u kojem je korisnik uključen u proces narudžbe te djelatnik radnje obavlja samu izradu fotografija.

Iako će ostatak procesa bit objašnjen detaljno u pregledu kôda, slijedi kratki prikaz. Završetkom narudžbe, stvorena je *.bat* datoteka u koju je zapisana putanja do svake fotografije, njezina količina, format te vrsta foto-papira. Djelatnik zatim sjeda za računalo u radnji te pokreće *batch* datoteku s odgovarajućim brojem narudžbe. Zapis unutar *batch* datoteke je zapravo skup naredbi kopiranja fotografija u novi direktorij, prikazan na slici 20.

```
ECHO F | XCOPY /F /Y "D:\small\00 ZARKOVICA\099.JPG" "D:\NARUDZBE\2018-09-07\131118\fotke -small\00 ZARKOVICA\20x30matte\099_1x.JPG"
ECHO F | XCOPY /F /Y "D:\small\00 ZARKOVICA\090.JPG" "D:\NARUDZBE\2018-09-07\131118\fotke -small\00 ZARKOVICA\15x21\090_1x.JPG"
ECHO F | XCOPY /F /Y "D:\small\00 ZARKOVICA\077.JPG" "D:\NARUDZBE\2018-09-07\131118\fotke -small\00 ZARKOVICA\13x18\077_1x.JPG"
ECHO F | XCOPY /F /Y "D:\small\00 ZARKOVICA\057.JPG" "D:\NARUDZBE\2018-09-07\131118\fotke -small\00 ZARKOVICA\20x30matte\057_3x.JPG"
ECHO F | XCOPY /F /Y "D:\small\00 ZARKOVICA\074.JPG" "D:\NARUDZBE\2018-09-07\131118\fotke -small\00 ZARKOVICA\10x15\074_1x.JPG"
ECHO F | XCOPY /F /Y "D:\small\00 ZARKOVICA\014.JPG" "D:\NARUDZBE\2018-09-07\131118\fotke -small\00 ZARKOVICA\13x18\014_8x.JPG"
```

*Slika 20 Sadržaj .bat datoteke narudžbe*

Izvođenjem ovih naredbi, dobiva se sljedeće na računalu djelatnika kao na slici 21.




*Slika 21 Kopirane fotografije u pripadajuće mape (po formatima)*

Mapa u kojoj se narudžba nalazi jasno govori da je preuzeta čitava struktura direktorija u kojem su se nalazile fotografije, a unutar direktorija (u kojem bi inače odmah imali fotografije) sada imamo dodatne direktorije s formatima unutar kojih se nalaze odabrane fotografije. Ovakva podjela direktorija će djelatniku radnje uvelike olakšati izradu fotografija – ako ima više fotografija istog formata, jednostavno ih sve označi unutar te mape i promijeni im format (u posebnom programu za izradu fotografija), dakle ne mora ih pojedinačno nalaziti i mijenjati im format. Slika 22 prikaz je sadržaja mape nekog formata nakon izvođenja naredbi u .bat datoteci.


*Slika 22 Prikaz mape iz narudžbe - format 20x30*

Ulaskom u mapu nekog formata (u ovom slučaju i vrste foto-papira *matte*) može se uočiti da se pri kopiranju fotografija visoke rezolucije promijenio naziv – nadodana je količina tako da djelatnik radnje u programu za izradu fotografija treba jedino postaviti količinu, bez da „ručno“ ponavlja izradu istih fotografija (pojednostavljeno, datoteka je preimenovana umjesto kopirana onoliko puta koliko je naručena).

## 4.2. Klase i funkcije aplikacije

U ovom dijelu diplomskog rada biti će opisane klase i funkcije unutar aplikacije. Svaka WPF aplikacija sastoji se prije svega od Window klase koja služi vizualnom izgledu aplikacije – u tu klasu postavljamo grafičke elemente koje potom povezujemo dalje u kodu odnosno pripadajućoj MainWindow.xaml.cs datoteci. Takva datoteka može se nazvati i *Code-Behind* obzirom da se C# kôd piše u njoj.

Ova aplikacija napravljena je s idejom da je sve jedan prozor uz kartice, tako da imamo samo jednu Window klasu.

### 4.2.1. SelectPhoto.MainWindow

Window klasa u ovoj aplikaciji zove se *MainWindow*, a *SelectPhoto* označava kojoj aplikaciji 'pripada', odnosno uvijek naziv aplikacije prethodi imenu klase *Window*. Unutar klase raspodijeljeni su elementi po karticama. Na slici 23 prikaz je kartice početnog zaslona u XAML-u.

```
<!-- KARTICA POČETNOG ZASLONA -->
<TabItem Header="{x:Static properties:Resources.tab_welcome}" Name="Welcome" IsEnabled="False">
    <Grid IsEnabled="True">
        <Button Style="{StaticResource MaterialDesignFloatingActionButton}"
                Click="Start" Height="300" Width="300" FontSize="70"
                Content="{x:Static properties:Resources.button_start}"/>
    </Grid>
</TabItem>
```

*Slika 23 Prikaz kartice početnog zaslona (XAML)*

Kako bi se uputilo korisnika da se služi isključivo predviđenim načinom navigacije kroz kartice, isključena je mogućnost klikanja po nazivima kartica – zato je *IsEnabled="False"*. Međutim, da bi sam sadržaj unutar kartice bio dostupan, ista vrijednost unutar *Grida* (u prijevodu s engleskog: rešetka) postavljena je na *True*. Radi lakšeg dizajna aplikacije, preuzet je „stil" odnosno izgled tipke **Početak** – primijenjen je *MaterialDesignFloatingActionButton*[10], jedan od elemenata u skupini grafičkih dodataka u sklopu *MaterialDesignInXaml* biblioteke dostupne na Githubu.

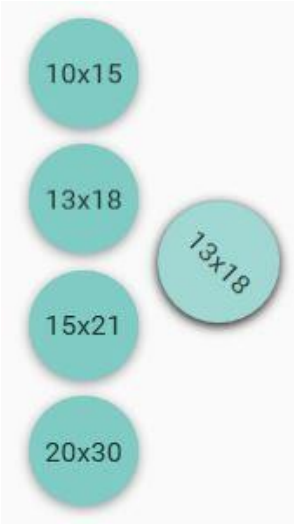
U primjeru se također vidi lakoća povezivanja XAML i njegovog „Code-Behinda" – *Click="Start"* omogućuje stvaranje funkcije *Start* u kojoj se definira što se događa kada se klikne na **Početak**. Isto tako je na slici 24 prikazano povezivanje XAML-a i „*Code-Behinda*" s *Click="Change_format_click"* preko koje stvaramo funkciju odabira formata putem klika.

[10] Preuzeto s https://github.com/MaterialDesignInXAML

```
<materialDesign:PopupBox Name="Format"
                Style="{StaticResource MaterialDesignMultiFloatingActionLightPopupBox}"
                PlacementMode="LeftAndAlignMiddles"
                ToolTipService.Placement="Left"
                Grid.Row="4" HorizontalAlignment="Center">
    <materialDesign:PopupBox.ToggleContent>
        13x18
    </materialDesign:PopupBox.ToggleContent>
    <StackPanel>
        <Button Click="Change_format_click" Height="50" Width="50">10x15</Button>
        <Button Click="Change_format_click" Height="50" Width="50">13x18</Button>
        <Button Click="Change_format_click" Height="50" Width="50">15x21</Button>
        <Button Click="Change_format_click" Height="50" Width="50">20x30</Button>
    </StackPanel>
</materialDesign:PopupBox>
```

*Slika 24 MaterialDesign PopupBox u kôdu*

Još jedan primjer korištenja elemenata iz MaterialDesign biblioteke je *PopupBox*, vizualno atraktivna metoda koja mijenja padajuće izbornike, prikazana na slici 25.




*Slika 25 Vizualni prikaz MaterialDesign PopupBoxa*

### 4.2.2. MainWindow.xaml.cs

Kako je spomenuto, kôd u kojem popunjavamo sadržaj iza XAML datoteke prozora upravo je ova Code-Behind klasa sa slike 26, odnosno smatra se da su XAML i XAML.cs dvije definicije iste klase.

```
public MainWindow()
{
    InitializeComponent();
    Timer.Tick += new EventHandler(Timer_Click);

    Timer.Interval = new TimeSpan(0, 0, 1);

    Timer.Start();
    loadDirectory(mainPath);

}
```

*Slika 26 Konstruktor*

Kôd započinje s inicijalizacijom cijelog programa s *InitializeComponent();* a potom se pokreće učitavanje direktorija dohvaćanjem putanje spremljene u *string* varijablu *mainPath*. Putanja se za sada mijenja isključivo u kôdu, a radi se o tome da treba posebno prilagoditi putanju fotografskoj radnji, kao i druge elemente (vrste foto-papira, formate i drugo).

*Timer* je pokrenut u konstruktoru jer se očekuje da radi cijelo vrijeme dok je program pokrenut – nije neophodan za aplikaciju, međutim smatrao sam da je koristan jer stalno prikazuje datum i vrijeme.

### 4.2.3. Funkcija loadDirectory

Ova funkcija, prikazana na slici 27, bila je logičan prvi korak u izradi aplikacije –te popunjavanjem liste fotografija s elementima iz klase *Image* (na slici 28) u direktoriju s putanjom *mainPath*

```
//Popunjavanje liste fotografija s onima iz trenutnog foldera
int id = 0;
foreach (string file in files)
{
    BitmapImage image = new BitmapImage();
    image.BeginInit();
    image.UriSource = new Uri(file);
    image.EndInit();
    Images.Add(new Image {
        id = id++,
        image = image,
        path = file
    });
}
```

*Slika 27 Popunjavanje liste fotografija s onima iz mape*

Svakoj fotografiji pridodan je identifikacijski broj koji služi isključivo kao pomoć u navigaciji između fotografija u programu. Na primjer, recimo da kupac pregledava jednu mapu

fotografija i sviđaju mu se fotografije s *id = 1, id = 2, id = 3*. Ukoliko krene pregledavati fotografije u drugoj mapi, opet može odabrati fotografije s istim identifikacijskim brojem jer je on stvoren za svaku mapu jednako. Jedina stavka koja se može smatrati primarnim ključem je path, odnosno lokacija fotografije.

```
namespace SelectPhoto
{
    public class Image
    {
        public int id { get; set; }
        public BitmapImage image { get; set; }
        public string path { get; set; }
    }
}
```

*Slika 28 Klasa Image*

### 4.2.4. Funkcije Quantity_Add i Quantity_Remove

Uz spremanje narudžbe, najvažnije funkcije su dodavanje i oduzimanje elemenata narudžbe putem količine željenih fotografija, prikazane na slici 29.

```
private void Quantity_Add(object sender, RoutedEventArgs e)
{
    if (Order == null) return;
    Order currentOrder = Order.Find(x => x.path == currentImage.path);


    if (currentOrder == null) //ako ne postoji narudžba
    {
        Order.Add(new Order //napravi novu narudžbu s tim elementom
        {
            image = currentImage.image,
            path = currentImage.path,
            format = Format.ToggleContent.ToString(),
            quantity = 1,
            photopaper = Photo_Paper.ToggleContent.ToString()
        });
        ((Badged)currentButton.Parent).Badge = 1;
    } else //ako postoji taj element narudžbe, dodaj količinu
    {
        currentOrder.quantity++;
        ((Badged)currentButton.Parent).Badge = Int32.Parse(((Badged)currentButton.Parent).Badge.ToString()) + 1;
    }
    Update_quantity_label();
}
```

*Slika 29 Funkcija Quantity_Add*

S obzirom da su funkcije dodavanja i oduzimanja količine željenih fotografija gotovo iste, bit će objašnjeno kako funkcionira dodavanje količine. Prije svega, da bi se izbjegli problemi, potrebno je provjeriti ima li trenutna narudžba neke elemente – ako nema, napravit će se nova narudžba. Ako već postoji narudžba (i još ako je ta fotografija prethodno odabrana, tada će zbrajanje povećati količinu (te će logično oduzimanje maknuti količinu dok nije potrebno maknuti narudžbu).

Ovdje je u primjeni i lambda izraz; na slici 29 vidljiva je sljedeća linija kôda:

```
Order currentOrder = Order.Find(x => x.path == currentImage.path);
```

Lambda izraz će ovdje provjeriti postoji li u narudžbi fotografija koju se trenutno pregledava. On dakle pretražuje sve elemente narudžbe u listi *Order* za lokaciju fotografije koja se podudara s onom lokacijom koju se trenutno gleda. Ako se ne podudara, to znači da ta fotografija nije još uključena u narudžbu.

### 4.2.5. Kreiranje .bat datoteke

Najbitnija funkcija u aplikaciji nalazi se u kartici **Završi narudžbu**. Nakon što je zaposlenik unio broj narudžbe i klikom na tipku završio narudžbu, izvršava se zapisivanje podataka narudžbe u .bat datoteku kroz ključnu riječ *using* (koja je spomenuta u osnovnim pojmovima). Cijeli postupak kreiranja .bat datoteke razlomljen je u više segmenata; na slici 30 prikazan je početak u kojem smo postavili da se narudžba sprema na radnu površinu računala.

```
using (System.IO.StreamWriter file =
new System.IO.StreamWriter(Environment.GetFolderPath(Environment.SpecialFolder.Desktop)
                           + @"\" + Order_number.Text + ".bat"))
{
```

*Slika 30 Kreiranje .bat datoteke – lokacija spremanja .bat datoteke*

Na slici 31 je prikaz dohvaćanja putanje svake fotografije u narudžbi. Za primjer ove aplikacije, putanja „korisničkog računala“ postavljena je na *C:\fotke-small*, a putanja „djelatnikovog računala“ postavljena je na *D:\small*, dakle radi se o maloj promjeni. Navest ću još jedan primjer – korisnik gleda fotografiju s putanjom *C:\2018\09\PROMOCIJA INFORMATIKA\001.jpg*, ali na računalu djelatnika putanja iste fotografije u visokoj rezoluciji biti će *D:\DJELATNIK\2018\09\PROMOCIJA INFORMATIKA\001.jpg* – dakle jedina promjena je početak, ostatak putanje je identičan.

```
foreach (Order item in Order)
{
    string newpath = @"D:\NARUDZBE\" + DateTime.Now.ToString("yyyy-MM-dd") + @"\" + Order_number.Text + item.path.Split(':')[1];
    string custompath = @"D:\" + item.path.Split('-')[1];

    //sada imamo npr. D:\NARUDZBE\2018-09-05\narudzba001\small\00 ZARKOVICA\###.jpg
    int idx = newpath.LastIndexOf('\\');
    string prvidio = (newpath.Substring(0, idx)); // D:\NARUDZBE\2018-09-05\narudzba001\small\00 ZARKOVICA\
    string ime = (newpath.Substring(idx + 1)); // ###.jpg
```

*Slika 31 Kreiranje .bat datoteke - rastavljanje putanje*

Pomoću *item.path.Split(':')* funkcije putanja se rastavlja na dva dijela – prije i nakon ':' čime dobivamo npr. *C* i *\fotke -small\00 ZARKOVICA\001.jpg*. Idući korak je zamijeniti putanju fotografije u malenoj rezoluciji (koju gleda korisnik) s putanjom fotografije u visokoj rezoluciji

(pohranjenu na računalu djelatnika) – to je spremljeno u varijablu *custompath* gdje smo zamijenili *C* sa *D* diskom i uzeli putanju bez dijela „fotke -“.

Bitno je naglasiti varijablu *newpath* koja označava konačnu strukturu mapa unutar kojih će fotografija biti kopirana i označena s količinom. Kako bi imali točnu strukturu mapa, moramo rastaviti varijablu *newpath* na dva dijela – ovaj dio kôda je dokumentiran: prvi dio sadržavat će putanju sve do mape u kojoj se nalazi fotografija, a drugi dio odnosi se na sam naziv fotografije, npr. *051.jpg*. Na slici 32 prikaz je izmjene konačne putanje u koju se sprema fotografija.

```
//dodajemo folder s formatom te vrstom papira ukoliko nije sjajni papir
string folder = prvidio + @"\" + item.format + (item.photopaper.Equals("Matte") ? "matte" : "") + @"\";
string name;
if (item.quantity == 1) name = ime; //ako smo uzeli samo jednu fotografiju, nema potrebe pisati xxx_1x.jpg
else name = ime.Split('.')[0] + "_" + item.quantity + "x." + ime.Split('.')[1]; // xxx.jpg prelazi npr. u xxx_3x.jpg

//spajamo u jedno --> D:\NARUDZBE\2018-09-05\narudzba001\small\00 ZARKOVICA\13x18\xxx_3x.jpg
string destination = folder + name;
```

*Slika 32 Kreiranje .bat datoteke - nova putanja fotografije*

Ovisno o odabranom formatu fotografije, stvara se nova mapa koja će u nazivu sadržavati format fotografije i vrstu foto-papira (s time da se vrsta foto-papira navodi isključivo ako je odabrana *matte* opcija). Željena količina fotografije mora se također zapisati, ali samo ako je naručeno više od jednog primjerka te fotografije. U slučaju da su npr. naručena tri primjerka, 001.jpg će postati 001_3x.jpg. Konačna putanja fotografije spremljena je u varijablu *destination*. Na slici 33 prikaz je spremanja fotografije u .bat datoteku.

```
    //stvaramo redak za .bat file --> COPY source destination
    //koristimo XCOPY jer za razliku od COPY, on stvara direktorije kojih nema (npr. NARUDZBE i DATUM)
    //te s echo potvrđujemo da se radi o datoteci
    string bat_row = "ECHO F | XCOPY /F /Y \"" + custompath + "\" \"" + destination + "\"";
    file.WriteLine(bat_row);
}
```

*Slika 33 Kreiranje .bat datoteke - zapisivanje naredbe*

### 4.2.6. Funkcija resetDefaults

Ako korisnik želi otkazati narudžbu, a pregledavao je fotografije kroz razne mape, tada je potrebno postaviti sve vrijednosti natrag na one početne, a to se napravilo s funkcijom *resetDefaults* (slika 34) koja je pozvana na dva mjesta u aplikaciji:

- pri vraćanju na prvu karticu **Dobrodošli**
- kada je narudžba završena

```
private void ResetDefaults()
{
    currentButton = new Button();
    mainPath = @"C:\fotke -small\00 ZARKOVICA";
    nomanzone = @"C:\fotke -small";
    previousPath = "";
    currentPath = "";

    Format.ToggleContent = "13x18";
    Photo_Paper.ToggleContent = "Glossy";
    quantity_label.Content = "0";

    loadDirectory(mainPath);
}
```

*Slika 34 Funkcija resetDefaults*

U funkciji se vraća *mainPath* (početni direktorij pregledavanja fotografija) na izvornu vrijednost. *String nomanzone* predstavlja svojevrsnu zaštitu za sustav navigacije kroz mape. U funkciji *Back_Click*, implementiranoj za kretanje u prethodnu mapu, nalazi se sljedeća linija kôda:

*if (previousPath.Equals(nomanzone) || previousPath == "") return;*

Ako se pritisne tipka za kretanje u prethodnu mapu po strukturi direktorija, a ta mapa odgovara putanji spremljenoj u *nomanzone*, tada je kretanje kroz mape onemogućeno, odnosno želimo da korisnici vide isključivo one mape čiji im je pregled dozvoljen.

### 4.2.7. Resources.resx

Prilikom izrade aplikacije, znao sam da ću imati puno natpisa, naziva, upozorenja i drugih *stringova* – shvatio sam da bi bilo najbolje spremiti sve vrijednosti na jedno mjesto i zatim ih pozivati kroz kôd. Upravo tome služe .resx datoteke – pohranjuju resurse poput *stringova*, a prednost je to što je proces lokalizacije olakšan – uz manje preinake kopiramo naše resurse i prilagodimo za neki drugi jezik. U ovoj aplikaciji, pohranjeni su *stringovi* na hrvatskom jeziku. Na slici 35 prikazan je sadržaj *Resources.resx* datoteke.

Spremanje resursa napravio sam kroz opciju *Add Resource* pri čemu se prvo unosi naziv resursa (npr. *appName*), vrijednost resursa (*npr. Odabir i narudžba fotografija*) te komentar za pojedini resurs (*npr. Naziv aplikacije*, koji objašnjava čemu služi taj resurs ili gdje se koristi).

| Name | Value | Comment |
|---|---|---|
| appName | Odabir i narudžba fotografija | Naziv aplikacije |
| button_back | Natrag | Button za navigaciju kroz mape (folder up) |
| button_finishorder | Završi narudžbu | Button za kraj |
| button_start | Početak | Button za početak |
| hint_discardorder | Otkažite narudžbu. | Natpis za otkazivanje narudžbe |
| hint_order | Unesite broj narudžbe | Natpis za unos broja narudžbe |
| hint_proceedorder | Krenite na pregled narudžbe | Natpis za pregled narudžbe |
| tab_finish | Završi narudžbu | Kartica Finish |
| tab_makingorder | Odabir fotografija | Kartica Making Order |
| tab_orderdetails | Pregled narudžbe | Kartica Order Details |
| tab_welcome | Dobrodošli | Kartica Welcome |
| tablica_format | Format fotografije | Natpis u tablici |
| tablica_image | Fotografija | Natpis u tablici |
| tablica_path | Lokacija fotografije | Natpis u tablici |
| tablica_photopaper | Vrsta foto-papira | Natpis u tablici |
| tablica_quantity | Količina | Natpis u tablici |
| text_failure | Greška pri slanju narudžbe! | Tekst za neuspjeh narudžbe |
| text_no | Ne | Opcija za odbijanje |
| text_ordernumber | Broj narudžbe: | Tekst |
| text_success | Uspješno poslano! | Tekst za uspjeh narudžbe |
| text_total | Ukupno fotografija: | Tekst za total narudžbe |
| text_warning | Upozorenje | Tekst za upozorenje |
| text_yes | Da | Opcija za potvrdu |
| warning_cancelorder | Otkazivanje narudžbe? | Upit za otkazivanje |
| warning_noimages | Niste odabrali nijednu fotografiju. | Upozorenje da nisu odabrane fotografije |
| warning_ordernumber | Unesite broj narudžbe! | Upozorenje da broj narudžbe nedostaje |

*Slika 35 Resursi u aplikaciji*

Pristupanje određenom resursu je različito za XAML i njegov Code-Behind. Da bi u XAML-u došli do resursa, prvo je potrebno u Window dodati sljedeću liniju kôda:

*xmlns:properties="clr-namespace:SelectPhoto.Properties"*

Zatim se elementima koji trebaju prikazivati tekst dodaje sljedeće:

*Content="{x:Static properties:Resources.button_finishorder}"*

Time smo preko statičnog resursa dohvatili željeni *string* koji će prikazati „Završi narudžbu“. Ako npr. želimo dohvatiti vrijednost istog resursa u xaml.cs datoteci, tada npr. pišemo

*SelectPhoto.Properties.Resources.warning_noimages*

Ovaj dio kôda mora se napisati tamo gdje je *string* potreban, npr. notifier.showWarning();

Postoji i drugi način za dodavanje resursa - kroz kôd, putem *ResXResourceWriter* klase [12]. Prije svega potrebno je napomenuti da se u .resx datoteke ne smiju spremati lozinke ili ostali osjetljivi podatci jer je njima iznimno lako pristupiti. Na slici 36 prikazan je primjer u kojem se stvara nova .resx datoteka kroz kôd.

```
// Instantiate an Automobile object.
Automobile car1 = new Automobile("Ford", "Model N", 1906, 0, 4);
Automobile car2 = new Automobile("Ford", "Model T", 1909, 2, 4);
// Define a resource file named CarResources.resx.
using (ResXResourceWriter resx = new ResXResourceWriter(@".\CarResources.resx"))
{
   resx.AddResource("Title", "Classic American Cars");
   resx.AddResource("HeaderString1", "Make");
   resx.AddResource("HeaderString2", "Model");
   resx.AddResource("HeaderString3", "Year");
   resx.AddResource("HeaderString4", "Doors");
   resx.AddResource("HeaderString5", "Cylinders");
   resx.AddResource("Information", SystemIcons.Information);
   resx.AddResource("EarlyAuto1", car1);
   resx.AddResource("EarlyAuto2", car2);
}
```

*Slika 36 Primjer stvaranja .resx datoteke u kôdu, preuzeto s [12]*

U aplikaciji ovog diplomskog rada nije bilo potrebno na taj način zapisivati resurse obzirom da korisnici aplikacije ništa ne unose, već samo djelatnik u fotografskoj radnji unosi broj narudžbe na kraju procesa. Jedino se može razmisliti o naknadnom uvođenju funkcionalnosti da fotografska radnja sama postavi početni direktorij pregledavanja fotografija (za sada je to određeno u kôdu, bez mogućnosti izmjene).

# 5. Zaključak

U ovom diplomskom radu prikazan je razvoj aplikacije za odabir i narudžbu fotografija za operacijski sustav Windows. Aplikacija je izrađena u integriranom razvojnom okruženju Microsoft Visual Studio u objektno orijentiranom programskom jeziku C# kroz Windows Presentation Foundation pristup.

Korisniku aplikacije omogućen je pregled mapa s fotografijama i naručivanje tih fotografija te odabir količine, formata i foto-papira. Nakon što je korisnik završio odabir fotografija, narudžba se prosljeđuje djelatniku fotografske radnje koji pokreće skriptu s kojom se sve odabrane fotografije kopiraju u zajedničku mapu te su spremne za izradu.

Osim WPF-a, sažeto su objašnjeni pristupi izrade aplikacija u Windows Forms te Universal Windows Platform pristupu, koje sam usporedio kroz njihove prednosti i nedostatke. Obzirom da mi UWP pristup nije odgovarao zbog njegove ograničenosti za starije verzije Windowsa, krenuo sam u izradu aplikacije kroz WPF pristup.

S obzirom da sam prethodno pisao mobilne aplikacije za Android operativni sustav, primjećujem da sam se brzo snašao u razvijanju jer je C# vrlo sličan Javi, što mi je odgovaralo kao i mogućnost slaganja elemenata u XAML-u, kojeg smatram velikim unaprjeđenjem u odnosu na slaganje elemenata u Android operativnom sustavu (s time da je zadnja verzija Androida za koju sam pisao aplikacije bila Marshmallow s minimalnom verzijom postavljenom na KitKat).

U radu su opisane klase i funkcije unutar glavnog prozora aplikacije. Priložene su slike zaslona aplikacije koje opisuju korake koje korisnik prolazi u kreiranju narudžbe. Smatram da ova aplikacija odnosno proces narudžbe koji je predstavljen u ovoj aplikaciji može unaprijediti poslovanje fotografske radnje te olakšati korisnicima i djelatnicima neke poteškoće koje se javljaju u trenutnom, fizičkom načinu naručivanja fotografija.

# 6. Popis slika

# 7. Popis literature


[1] Technopedia, »What is Windows Forms?,« [Mrežno]. Available: https://www.techopedia.com/definition/24300/windows-forms-net. [Pokušaj pristupa 9 Rujan 2018].

[2] C. Moser, »WPF Tutorial | XAML,« 13 Prosinac 2013. [Mrežno]. Available: https://www.wpftutorial.net/XAML.html. [Pokušaj pristupa 8 Rujan 2018].

[3] Wikipedia, »Universal Windows Platform,« [Mrežno]. Available: https://en.wikipedia.org/wiki/Universal_Windows_Platform. [Pokušaj pristupa 14 Rujan 2018].

[4] Microsoft, »Data Binding Overview,« 30 Ožujak 2017. [Mrežno]. Available: https://docs.microsoft.com/en-us/dotnet/framework/wpf/data/data-binding-overview. [Pokušaj pristupa 14 Rujan 2018].

[5] Microsoft, »Garbage Collection,« 3 Ožujak 2017. [Mrežno]. Available: https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/index. [Pokušaj pristupa 9 Rujan 2018].

[6] K. Nandwani, M. Wenzel, K. Brockshmidt i A. Myers, »What is NuGet and what does it do?,« 10 Siječanj 2018. [Mrežno]. Available: https://docs.microsoft.com/en-us/nuget/what-is-nuget. [Pokušaj pristupa 08 Rujan 2018].

[7] Microsoft, »Choose Your Technology,« 31 Svibanj 2018. [Mrežno]. Available: https://docs.microsoft.com/en-us/windows/desktop/choose-your-technology. [Pokušaj pristupa 9 Rujan 2018].

[8] N. L, »MigrateTO.NET,« Lipanj 2016. [Mrežno]. Available: https://www.migrateto.net/wp-content/uploads/2016/06/Whitepaper-WHY-USE-WPF-INSTEAD-OF-WINFORMS.pdf. [Pokušaj pristupa 16 Rujan 2018].

[9] »Which .NET Framework for Windows: UWP, WPF OR WINDOWS FORMS?,« 29 Siječanj 2018. [Mrežno]. Available: https://www.itwriting.com/blog/10182-which-net-

framework-for-windows-uwp-wpf-or-windows-forms.html. [Pokušaj pristupa 14 Rujan 2018].

[10] N. Matkovska, »Let the battle begin: WinForms vs. WPF,« 8 Kolovoz 2016. [Mrežno]. Available: https://www.linkedin.com/pulse/let-battle-begin-winforms-vs-wpf-natalie-matkovska/. [Pokušaj pristupa 9 Rujan 2018].

[11] Intertech, »Introducint WPF,« 2014. [Mrežno]. Available: https://www.intertech.com/Downloads/WhitePapers/Intertech-Complete-WPF-Chapter-1.pdf. [Pokušaj pristupa 16 Rujan 2018].

[12] Microsoft, »Working with .resx Files Programmatically,« 3 Ožujak 2017. [Mrežno]. Available: https://docs.microsoft.com/en-us/dotnet/framework/resources/working-with-resx-files-programmatically. [Pokušaj pristupa 14 Rujan 2018].

[13] J. Freeman, »What is an API? | Application Programming Interfaces explained,« 9 Svibanj 2018. [Mrežno]. Available: https://www.infoworld.com/article/3269878/apis/what-is-an-api-application-programming-interfaces-explained.html. [Pokušaj pristupa 14 Rujan 2018].