Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Elmedin Osmanagić

Razvoj Windows Store aplikacij

diplomsko delo

visokošolski strokovni študijski program prve stopnje
računalništvo in informatika

Mentor: dr. Aljaž Zrnec

Ljubljana, 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00394 / 2013
Datum: 3.4.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ELMEDIN OSMANAGIĆ**

Naslov: **RAZVOJ WINDOWS STORE APLIKACIJ**
WINDOWS STORE APPLICATION DEVELOPMENT

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Operacijski sistem Windows 8 je prinesel številne novosti, med drugim tudi ti. aplikacije Windows Store apps. Nova aplikacijska arhitektura WinRT, ki tvori osnovo operacijskih sistemov Windows 8 in Windows RT, omogoča razvijalcem v jezikih C++, C# in VB razvoj podatkovno vodenih in grafično bogatih aplikacij Windows Store s pomočjo deklarativnega jezika XAML. Na analogen način je omogočen razvoj aplikacij Windows Store razvijalcem spletnih aplikacij, ki lahko uporabljajo kombinacijo HTML, CSS in JavaScript. Spletni razvijalci lahko dodatno uporabljajo novo JavaScript knjižnico WinJS, ki ponuja tudi množico bogatih, podatkovno vezanih HTML kontrol, integracijo z mapami Bing Maps ter servisi Windows Live. Z integriranim razvojnim okoljem Microsoft VS Express for Windows izdelajte prototip aplikacije Windows Store, ki jo najprej razvijete z uporabo C# in XAML, nato pa še z uporabo HTML, CSS in JavaScript. Primerjajte oba pristopa in komentirajte ugotovitve.

Mentor:
viš. pred. dr. Aljaž Zrnec

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Elmedin Osmanagić, z vpisno številko **24014582**,
sem avtor diplomskega dela z naslovom:

Razvoj Windows Store aplikacij

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom dr. Aljaža Zrneca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 4.2.2014 Podpis avtorja:

Zahvaljujem se mentorju dr. Aljažu Zrnecu za strokovne nasvete, podporo in potrpežljivost pri pisanju diplomskega dela.

Zahvaljujem se tudi ga. Metki Runovc za nasvete, pomoč, napotke, prijaznost, vljudnost in iskrene želje za uspešno končanje študija.

قل إن صلاتي ونسكي ومحياي ومماتي لله رب العالمين

Kazalo vsebine

Kazalo slik

Seznam uporabljenih tujk

Tujka		Prevod/Opis
Simbol	**Pomen**	
.NET	.NET Framework	Programsko ogrodje .NET
ADO.NET	ActiveX Data Objects .NET	.NET aktivni podatkovni objekti
API	Application Programming Interface	Aplikacijski programski vmesnik
APM	Asynchronous Programming Model	Asinhroni programski model
AQS	Advanced Query Syntax	Poizvedbe z napredno sintakso
ASP.NET	Active Server Pages .NET	Aktivne .NET strežniške strani
AJAX	Asynchronous JavaScript and XML	Asinhroni JavaScript in XML
BCL	Base Class Library	Osnovna knjižnica razredov
C#	C Sharp Programming Language	Programski jezik C Sharp
C++	C++ Programming Language	Programski jezik C++
CIL	Common Intermediate Language	Skupni vmesni jezik
CLI	Common Language Infrastructure	Skupna jezikovna infrastruktura
CLR	Common Language Runtime	Skupno jezikovno izvajanje
CLS	Common Language Specification	Skupna jezikovna specifikacija
CSS	Cascading Style Sheets	Kaskadne stilske podloge
DLL	Dynamic Link Library	Knjižnica za dinamično povezavo
DOM	Document Object Model	Objektni model za HTML dokumente
EAP	Event-based Asynchronous Pattern	Asinhroni vzorec baziran na dogodkih
ECMA	European Computer Manufacturers Association	Evropska zveza proizvajalcev računalnikov
FCL	.NET Framework Class Library	Knjižnica razredov ogrodja .NET
GIT	Software for distributed version control system	Programska oprema za verzioniranje različic datotek
GUI	Graphical User Interface	Grafični uporabniški vmesnik
GUID	Global Unique Identifier	Globalni enolični identifikator
HTML	HyperText Markup Language	Jezik za označevanje nadbesedila

Seznam uporabljenih tujk (nad.)

Tujka		Prevod/Opis
Simbol	**Pomen**	
HTML5	HTML revision 5	Peta revizija HTML
HSLS	High Level Shading Language for DirectX	Visokonivojski jezik za DirectX
IDE	Integrated Development Environment	Integrirano razvojno okolje
ISO	International Organization for Standardization	Mednarodna organizacija za standardizacijo
J#	J Sharp Programming Language	Programski jezik J Sharp
JS	JavaScript Programming Language	Programski jezik JavaScript
JSON	JavaScript Object Notation	JavaScript objektna notacija
KO	Knockout JavaScript Library	JavaScript knjižnica Knockout
LINQ	Language Integrated Query	Jezik za integrirane poizvedbe
LTR	Left-to-right Text Directionality	Usmeritev besedila z leve na desno
MVVM	Model View ViewModel Pattern	Vzorec model pogled pogledmodel
MSIL	Microsoft Intermediate Language	Microsoftov vmesni jezik
OOP	Object-Oriented Programming	Objektno orientirano programiranje
OS	Operating System	Operacijski sistem
PCL	Portable Class Library	Prenosljiva knjižnica razredov
PLINQ	Parallel Language Integrated Query	Paralelni LINQ
RAM	Random Access Memory	Spomin z naključnim dostopom
REST	Representational State Transfer	Predstavitveni prenos stanj
RTL	Right-to-left Text Directionality	Usmeritev besedila z desne na levo
SPA	Single Page Application	Enostranska aplikacija
SDK	Software Development Kit	Paket za razvoj programske opreme

Seznam uporabljenih tujk (nad.)

Tujka		Prevod/Opis
Simbol	**Pomen**	
SOAP	Simple Object Access Protocol	Preprosti protokol za dostop do objekta
TAP	Task-based Asynchronous Pattern	Asinhroni vzorec baziran na nalogah
TFVC	Team Version Control	Timska kontrola različic
URI	Uniform Resource Identifier	Enotni identifikator vira
URL	Uniform Resource Locator	Enotni lokator vira
VB	Visual Basic Programming Language	Programski jezik Visual Basic
VSO	Visual Studio Online	Microsoftov servis za upravljanje podatkov v oblaku
WCF	Windows Communication Foundation	Windows komunikacijska osnova
WF	Windows Workflow Foundation	Windows osnova za potek dela
Win 8	Windows 8	Operacijski sistem Windows 8
Win RT	Windows RT	Varianta operacijskega sistema Windows 8 prirejena za mobilne naprave na arhitekturi ARM
WinRT	Windows Runtime	Platformno homogena aplikacijska arhitektura na operacijskem sistemu Windows 8
WinJS	Windows Library for JavaScript	Windows knjižnica za JavaScript
WPF	Windows Presentation Foundation	Windows predstavitvena osnova
W3C	World Wide Web Consortium	Konzorcij World Wide Web
WNS	Windows Push Notification Services	Windows servisi za obvestila
XAML	EXtensible Application Markup Language	Razširljivi in označevalni aplikacijski jezik
XHR	XMLHttpRequest API	API, ki je na voljo skriptnim jezikom spletnih brskalnikov, npr. JavaScript
XML	EXtensible Markup Language	Razširljivi jezik za označevanje

Povzetek

Operacijski sistem Windows 8, kot naslednik priljubljenega sistema Windows 7, je prinesel kopico novosti, med drugim tudi aplikacije poimenovane Windows Store apps. Nova aplikacijska arhitektura WinRT, ki tvori osnovo operacijskih sistemov Windows 8 in Windows RT, omogoča razvijalcem v jezikih C++, C# in VB razvoj podatkovno vodenih in grafično bogatih aplikacij Windows Store s pomočjo deklarativnega jezika XAML. Na analogen način je omogočen razvoj aplikacij Windows Store razvijalcem spletnih aplikacij, ki lahko uporabljajo kombinacijo HTML, CSS in JavaScript. Spletni razvijalci lahko dodatno uporabljajo novo JavaScript knjižnico WinJS, ki ponuja tudi množico bogatih, podatkovno vezanih HTML kontrol, integracijo z mapami Bing Maps ter servisi Windows Live. V okviru diplomske naloge je z integriranim razvojnim okoljem Microsoft VS Express for Windows prikazana izdelava prototipov aplikacij Windows Store, kjer prva temelji na razvoju s C# in XAML, druga pa na razvoju s tehnologijami HTML, CSS in JavaScript. Primerjajte oba pristopa in komentirajte ugotovitve.

Ključne besede: Windows 8, Windows RT, aplikacije Windows Store, C#, XAML, HTML, CSS, JavaScript, servisi Windows Live.

Abstract

Windows 8 operating system, as a successor of the popular Windows 7, brought a set of novelties, amongst which are the Windows Store apps. The new application architecture WinRT which forms the basis of Windows 8 and Windows RT operating systems, enables developers using C++, C# and VB languages the development of Windows Store data-driven and rich UI applications with the help of declarative XAML language. The development is enabled in an analogue way to developers of web applications who can use the combination of HTML, CSS and JavaScript. Web developers can additionally use the new JavaScript library WinJS which among other features offers a set of rich, data-bound HTML controls and integration with Bing Maps and Windows Live services. As part of the thesis and with an integrated development environment Microsoft VS Express for Windows it is shown Windows Store applications prototype development, where the first one is based on development using C# and XAML and the other using HTML, CSS and JavaScript technologies. Compare the two approaches and comment on the findings.

Keywords: Windows 8, Windows RT, Windows Store apps, C#, XAML, HTML, CSS, JavaScript, Windows Live services.

1 Uvod

1.1 Uvod v razvoj aplikacij Windows Store

Aplikacije Windows Store predstavljajo nov tip aplikacij na operacijskih sistemih Windows 8 in Windows RT, ki so drugačne ne le vizualno, temveč se razlikujejo od klasičnih namiznih aplikacij tudi tako, da so optimizirane za naprave, ki imajo zaslon na dotik, kot ti. pametni telefoni (angl. *Smartphone*), hibridi med pametnimi telefoni in tabličnimi računalniki (angl. *Phablet)*, tablični računalniki (angl. *Tablet*) ter prenosni in namizni računalniki. Za aplikacije Windows Store je značilno, da so celozaslonske in se prikazujejo v enem samem okvirju (angl. *Frame*), ki nima robov in prav tako nobenih ikon za spreminjanje velikosti okvirja, pomanjšanje in povečanje ter izhod iz aplikacije. Nove lastnosti so pomenile odmik od klasičnega razvoja namiznih aplikacij in so na začetku begale ne samo uporabnike temveč tudi razvijalce, ki so se prilagajali novim principom razvoja aplikacij za sisteme Windows 8 in RT. Posledično je to pomenilo, da niso bile takoj priljubljene, kot se je to pričakovalo v krogih idejnih snovalcev in oblikovalcev, kar se je odrazilo tudi na trgu.

Vendar pojdimo na začetek. Windows je potreboval spremembo pri pametnih telefonih in ena večjih je bila opustitev razvoja zastarelega in preveč kompliciranega sistema za mobilne telefone Windows Mobile 6.5. Porodila se je drzna ideja ponuditi novo vrsto aplikacij z novim grafičnim vmesnikom (*Metro*), katerih ravoj bi temeljil na povsem novih principih. Tako se je razvil sistem Windows Phone 7, nato sta sledili verziji 7.5 in 7.8 ter končno nova verzija 8. S prihodom operacijskih sistemov Windows 8 in Windows RT so nastale aplikacije Windows Store. Trenutno zadnja različica sistema Windows 8 je 8.1, ki prinaša nekatere novosti tudi glede razvoja aplikacij.

Evolucija termina Windows Store je nekoliko neobičajna, saj naj bi na samem začetku podjetje Microsoft interno uporabljalo naziv *Immersive app*, nekoliko kasneje *Metro Style*, nato *Windows Store Design Language* ter končno preprosto *Windows Store apps* [24].

Koncept implementacije aplikacij za Windows Store je bil idejno zasnovan tako, da omogoča analogen razvoj različnim profilom oblikovalcev in razvijalcev programske opreme ne glede na to, katerega od podprtih pristopov izberejo. Logično sledi, da so si pristopi različni, že zaradi razlik v izbranih tehnologijah.

V diplomskem delu smo se posvetili primerjalni študiji razvoja aplikacij za Windows Store, kjer smo razvili na pogled identična prototipa grafično privlačne, za uporabnika enostavne in intuitivne aplikacije, ki je hkrati tudi hitra, fluidna in podatkovno vodena.

1.2 Predstavitev vsebine

Diplomska naloga obravnava dva od več pristopov implementacije aplikacij Windows Store in sicer enega z uporabo C# in XAML na eni ter HTML, CSS in JavaScript tehnologij na drugi strani. Pristopa sta si tehnološko različna, pa vendar obstajajo utemeljene skupne lastnosti razvoja med njima, pri čemer posebej izstopajo smernice za razvoj uporabniškega grafičnega vmesnika. Pristopa, na osnovi katerih sta bila razvita prototipa aplikacije Windows Store, sta najbolj primerna za klasične aplikacije Windows Store, medtem ko je pristop razvoja z uporabo programskega jezika C++ najbolj primeren za razvoj iger. Kot zanimivost naj povemo, da se lahko grafični vmesnik pri uporabi jezika C++ zgradi tudi z jezikom XAML, kar pa pri uporabi jezika JavaScript ni podprto, saj se za to opravilo uporabljata jezik HTML in kaskadne stilske podloge CSS.

V drugem poglavju predstavimo arhitekturo operacijskega sistema Windows 8, programsko ogrodje .NET in razvojno okolje. Potem opišemo lastnosti aplikacij Windows Store ter predstavimo principe stilskega oblikovanja le-teh. Nato obravnavamo aplikacijski model, življenski cikel, stanja in prehode med stanji ter navigacijo med stranmi aplikacij Windows Store. Na koncu poglavja predstavimo *Windows Runtime* in *Live Connect API*.

V tretjem poglavju predstavimo tematiko prototipa aplikacije, naredimo analizo zahtev, razložimo koncepte in paradigme razvoja aplikacij Windows Store ter se posvetimo skupnim modulom in podatkom. Zastavimo tudi primerjalno izhodišče pri razvoju prototipov, ki bo služilo za dejansko primerjavo le-teh.

Četrto in peto poglavje sta povsem namenjena razvoju prototipov aplikacije Windows Store z uporabo C# in XAML ter HTML, CSS in JavaScript tehnologij, kjer podrobno razčlenimo razvojne koncepte in poudarimo pomembnosti, ki nam kasneje služijo kot oporne točke pri primerjavi obeh pristopov razvoja prototipa in zaključku. V četrtem poglavju tudi vizualno predstavimo nekaj zaslonskih posnetkov končnega prototipa in razložimo nekatere aspekte gradnje uporabniškega vmesnika z uporabo programskega jezika C# in razširljivega aplikacijskega označevalnega jezika XAML.

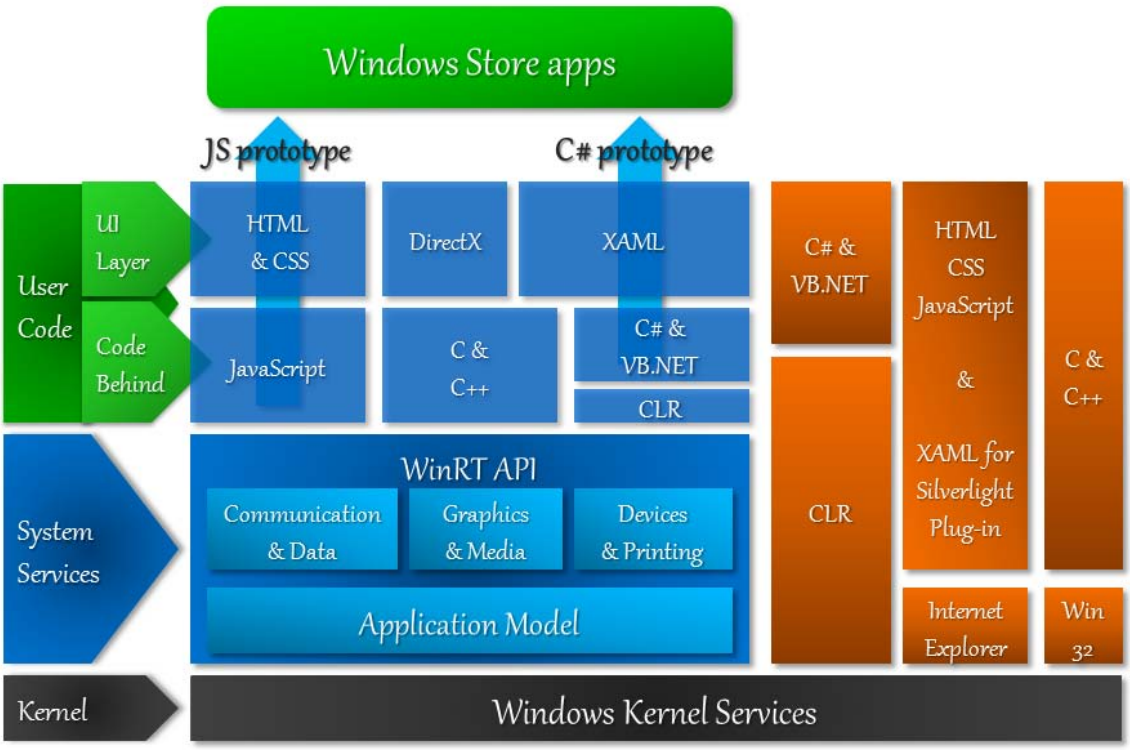
V šestem poglavju opravimo analizo in primerjavo obeh pristopov razvoja aplikacij Windows Store glede na koncepte, ki smo jih obravnavali v četrtem in petem poglavju ter na kratko opišemo prednosti in pomanjkljivosti, ki smo jih odkrili pri razvoju. Na koncu predstavimo rezultate primerjave razvoja obeh prototipov z radarskim grafom.

V sedmem poglavju strnemo zaključne ugotovitve v smiselno celoto in podamo osebno oceno razvoja aplikacij Windows Store.

2 Aplikacije Windows Store

2.1 Arhitektura operacijskega sistema Windows 8

Arhitekturo operacijskega sistema Windows 8 in njene osnovne module prikazuje slika 1. Na osnovi določenih arhitekturnih modulov smo vzporedno razvili prototipa aplikacije Windows Store z uporabo **C# in XAML** ter **HTML, CSS in JavaScript** tehnologij. Na arhitekturni sliki je označena razvojna plast *User Code*, kjer razvijalci implementirajo aplikacije Windows Store. Pod njo leži plast *System Services* oz. **WinRT API** z množico komponent (*Communication & Data*, *Graphics & Media*, *Devices & Printing*, *Application Model*), najnižje pa leži plast *Kernel* oz. **Windows Kernel Services**. Preostali moduli pa se uporabljajo za namizne, spletne in ostale Windows 8 aplikacije. Kot lahko vidimo na sliki, obstaja tudi tretji pristop razvoja aplikacij Windows Store in sicer z uporabo programskega jezika **C++** in knjižnic **DirectX**.



Slika 1: Arhitektura operacijskega sistema Windows 8

2.2 Programsko ogrodje .NET

.NET je zelo priljubljeno programsko ogrodje, ki ga je razvilo podjetje Microsoft konec leta 2002, trenutno zadnja različica pa je 4.5.1 [4]. .NET predstavlja *razvojno platformo za razvoj aplikacij na operacijskih sistemih Windows, Windows Phone in Windows Server ter platformo in infrastrukturo za storitve v oblaku Windows Azure*. .NET vključuje **Common Language Runtime (CLR)** in **.NET Framework Class Library (FCL)**, ki vsebuje razrede (angl. *classes*), vmesnike (angl. *interfaces*) in vrednostne tipe (angl. *value types*). Poleg tega je .NET upravljano izvajalno okolje, ki omogoča množico storitev .NET aplikacijam [12], kot npr.

- Upravljanje s spominom (angl. *Memory management*)
- Knjižnico razredov ogrodja .NET (angl. *.NET Framework Class Library (FCL)*)
- Razvojna ogrodja in tehnologije (angl. *Development frameworks and technologies*) **ASP.NET**, **ADO.NET**, **WCF**, **WPF** in **WF**
- Interoperabilnost jezikov (angl. *Language interoperability*)
- Združljivost različic (angl. *Version compatibility*)
- Izvajanje **side-by-side** (angl. *Side-by-side execution*)
- Prenosljivost knjižnic (angl. *Multitargeting using* **Portable Class Library (PCL)**)

Common Language Infrastructure (CLI)

CLI je specifikacija podjetja Microsoft, ki zajema ti. *Common Language Infrastructure (CLI)* in *Common Language Runtime (CLR)* ter definira okolje, ki dopušča uporabo več visokonivojskih jezikov na različnih platformah brez potrebe adaptacije izvorne kode za specifike posamezne arhitekture [9]. CLI je standardiziran s strani organizacij ECMA (pod oznako ECMA-335) [20] in ISO (ISO/IEC 23271:2012). CLI vsebuje tudi ti. osnovno knjižnico razredov (angl. **Base Class Library (BCL)**).

Common Intermediate Language (CIL)

CIL je definiran v specifikaciji *Common Language Infrastructure (CLI)* in se uporablja v ogrodjih .NET in Mono. CIL je vmesni jezik v katerega se prevedejo programi .NET,

napisani v programskih jezikih *C#*, *VB.NET* in *J#*. Nato *CLR* prevede CIL kodo v strojno kodo, ki je potem lahko izvaja na .NET platformi. Kot jezik je CIL na najnižjem nivoju, ki ga človek še lahko bere z razumevanjem kode. Prej je bil znan kot *Microsoft Intermediate Language (MSIL)* in je šele s standardizacijo dobil svoj sedanji naziv.

CIL je procesorsko in platformno neodvisen, kar v praksi pomeni, da se instrukcije, ki jih CIL podpira lahko izvajajo v kateremkoli okolju, ki podpira CLI.

Common Language Runtime (CLR)

CLR je programsko okolje v katerem se izvajajo programi .NET in predstavlja aplikacijski navidezni stroj (angl. *application virtual machine*), ki kot del ogrodja .NET ponuja storitve kot npr. varnost, upravljanje s spominom ter obravnavanje izjem in niti. CLR je zadolžen za zagon programov v ogrodju .NET, kar pomeni, da bo CLR zagnal program preveden v končno strojno kodo, ne glede na to, v katerem programskem jeziku ogrodja .NET je bil napisan. Pomembno je poudariti, da se lahko program sklicuje na eno ali več knjižnic, če tudi so bile le-te napisane v drugem programskem jeziku ogrodja .NET, npr. napišemo program v jeziku C#, ki kliče javne (angl. *public*) metode poljubnih tipov, ki so bili napisani v jeziku VB ter prevedeni v svojo .NET razredno knjižnico (angl. *.NET Class Library*). CLR podpira specifikacijo *Common Language Specification (CLS)*, ki definira osnovno množico lastnosti, ki jih podpira CLR.

Izvorna koda, napisana v kateremkoli programskem jeziku .NET, se najprej prevede z ustreznim prevajalnikom. Če je prevajanje uspešno, se generira jezikovno neodvisna koda **CIL**. CLR prevzame kodo CIL in jo prevede v strojno kodo, ki se potem lahko izvede na izvajalni platformi.

2.3 Razvojno okolje

Visual Studio Express 2013 for Windows

Pri razvoju prototipov aplikacije Windows Store smo uporabili integrirano razvojno okolje Visual Studio Express 2013 for Windows, ki ponuja praktično vsa potrebna orodja za moderni razvoj aplikacij .NET, kot npr. orodja ali module za kodiranje, testiranje, razhroščevanje in oblikovanje aplikacij ter gradnjo in namestitev paketov za Windows Store, kontrolo izvorne kode, namestitev paketov *NuGet* (odprtokodni upravljalnik paketov), itd. Visual Studio vključuje tudi urejevalnike besedila za jezike *JavaScript*,

HTML/XML/XAML, *CSS*, *C#*, *J#*, *Visual Basic*, *C++* in *HLSL* ter *IntelliSense*, orodje za samodejno dokončanje izrazov in parametrov v času pisanju kode. Poleg tega Visual Studio omogoča mnoga dodatna opravila kot npr. refaktoriranje izvorne kode, definiranje in vstavljanje odrezkov kode, gradnjo diagramov razredov ter povezovanje z mnogimi drugimi orodji, od katerih pa velja izpostaviti verjetno najbolj popularno razvojno orodje za .NET - *ReSharper* (refaktoriranje kode) podjetja *JetBrains*, ki je razvilo tudi druga zelo koristna orodja za .NET kot npr. *dotCover* (testiranje enot in pokritost kode), *dotTrace* (profiliranje kode) in *dotPeek* (pregled že prevedene kode in brskanje po knjižnicah DLL).

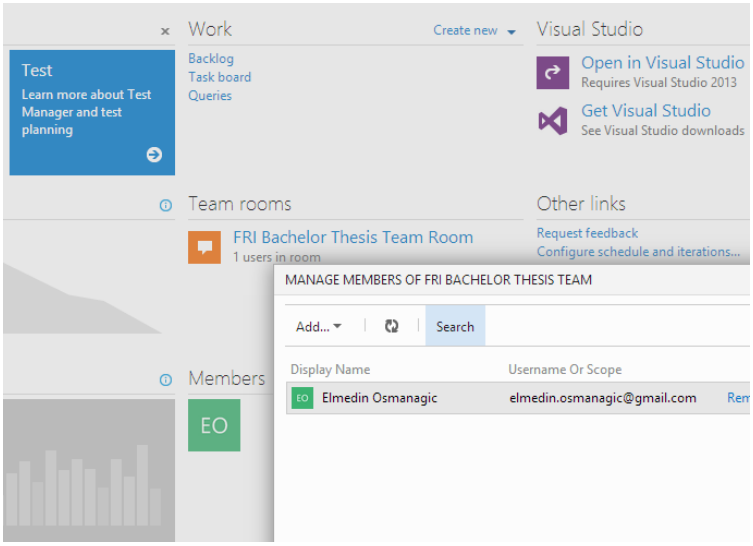
Blend for Visual Studio 2013

Tu bi posebej omenili orodje *Microsoft Blend for Visual Studio*, ki je namenjeno oblikovanju grafičnih uporabniških vmesnikov za spletne in namizne aplikacije, od pojave aplikacij Windows Store pa je omogočen tudi za le-te. Blend je baziran na XAML in HTML projektnih predlogah in je pakiran z Visual Studio 2013 plačljivo verzijo. Kot zanimivost omenimo, da če za Visual Studio pravijo, da je najmočnejši in najbolj napreden IDE na svetu, ne glede na jezik ali tehnologijo, pa je Blend močnejši glede grafičnega oblikovanja aplikacij Windows Store, *Windows Presentation Foundation* (WPF), *Windows Phone 8* in *Silverlight*, torej povsod tam, kjer se uporablja XAML. Poklicni razvijalci in strokovnjaki se načeloma brez izjem strinjajo, da je kombinacija orodij Visual Studio in Blend daleč najboljša za razvoj tovrstnih aplikacij.

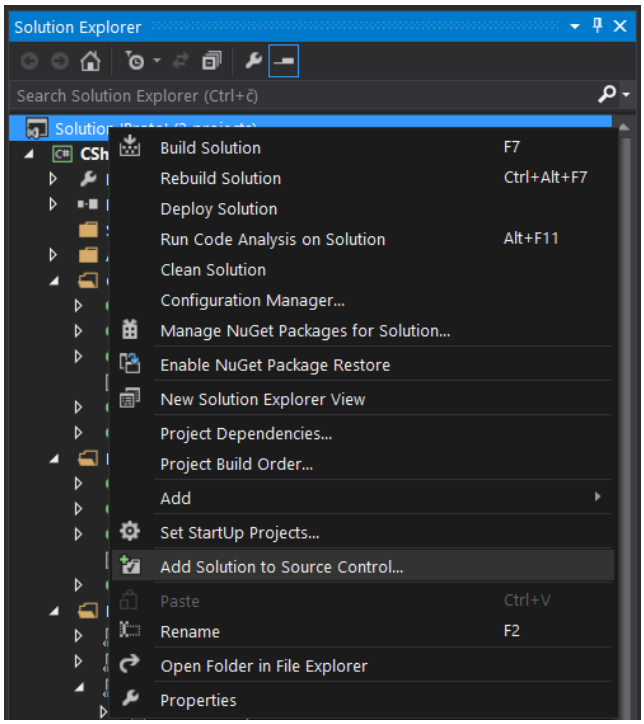
Visual Studio Online (Team Foundation Service)

Visual Studio Online (VSO) (prej znan kot *Team Foundation Service*) je servis podjetja Microsoft za shranjevanje in upravljanje podatkov v oblaku. VSO omogoča vzpostavitev delovnega okolja v oblaku in ponuja upravljanje različic izvorne kode z uporabo **Git** (orodje za porazdeljeno kontrolo različic in upravljanje izvorne kode) repozitorijev vsem uporabnikom orodja Visual Studio. Prav tako je mogoče izbrati *Team Version Control (TFVC)* namesto Git rešitve za kontrolo različic. V Visual Studio rešitvi se najprej povežemo z VSO servisom, nato pa izvorno kodo dodamo v VSO repozitorij, od koder potem lahko spremljamo spremembe. Prav tako lahko dodajamo testne plane, opravljamo testiranje enot izvorne kode, ustvarjamo opravila, prijavljamo hrošče, definiramo in izvajamo samodejna prevajanja izvorne kode ipd. VSO se sicer bolj uporablja pri agilnih pristopih razvoja programske opreme, ko imamo opravka z manjšo razvojno ekipo, pa vendar je uporaben tudi za posameznike.

Upravljanje z uporabniki servisa VSO nam prikazuje slika 2, medtem ko smo na sliki 3 prikazali dodajanje izvorne kode iz orodja Visual Studio v VSO.



Slika 2: Upravljanje uporabnikov v VSO

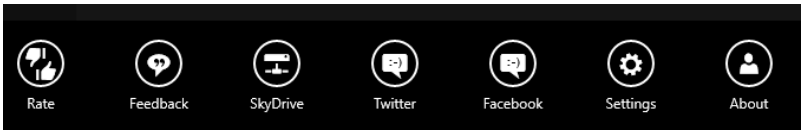


Slika 3: Dodajanje izvorne kode v VSO repozitorij

2.4 Lastnosti aplikacij Windows Store

Prenovljeni grafični vmesnik

Uporabnikom aplikacij Windows Store se ponuja popolnoma nova izkušnja s prenovljenim grafičnim uporabniškim vmesnikom (angl. *Graphical User Interface - GUI* ali skrajšano tudi *User Interface - UI*), ki se precej razlikuje od klasičnih vmesnikov namiznih aplikacij. Razlike se pokažejo predvsem v tem, da imajo tradicionalne namizne aplikacije vedno vidno menijsko vrstico (angl. *Menu Bar*), modalne dialoge (angl. *Modal Dialogs*), okvir (angl. *Frame*) in množico gumbov za pomanjšanje, povečanje in zaprtje (angl. *Minimize, Maximize, Close*) aplikacije, medtem ko se aplikacije Windows Store ponašajo z množico grafičnih površin, kjer se lahko uporabljajo kontrole kot recimo aplikacijsko platno (angl. *Application Canvas*), čari (angl. *Charms*), orodna ali aplikacijska vrstica (angl. *Application Bar*), dialogi (angl. *Dialogs*) ter novost med kontrolami kot je npr. kontrola *Flyout*. Aplikacijsko oz. orodno vrstico prototipa lahko vidimo na sliki 4, kjer vidimo gumbe za različne akcije.



Slika 4: Aplikacijska vrstica v prototipu

Grafični snovalci stilsko dodelanih spletnih aplikacij uporabljajo različna grafična ogrodja, ki definirajo okvirje, v katerih najdemo vse klasične kontrole grafičnih vmesnikov, pa tudi nekatere sestavljene tipe. Kot zanimivost naštejmo nekaj alternativnih grafičnih predlog [5] za grafične oblikovalce, ki se načeloma ne uporabljajo pri razvoju aplikacij Windows Store, lahko pa služijo za primerjavo:

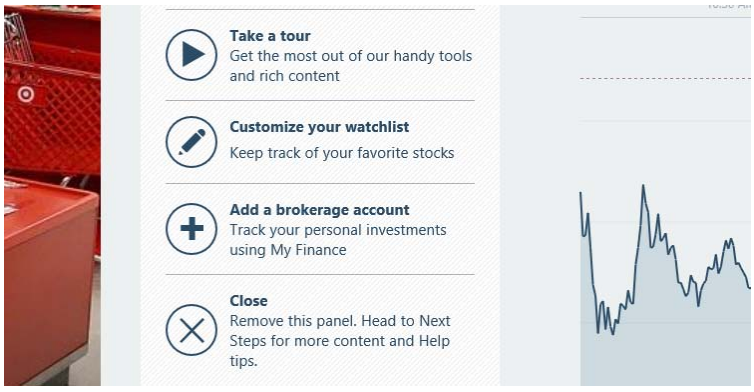
- **Dark Amber UI**
- **Derailed UI**
- **Futurico**
- **Impressionist UI Free**
- **jQuery UI**
- **Noire UI Kit**
- **Perfectjane UI Kit**

- **Square UI**
- **Vector User Interface Elements**

Aplikacije z menijem *File-Edit-View-Help* se ne pojavljajo več v aplikacijah Windows Store. Še več, aplikacije nimajo niti gumba za izhod iz aplikacije, temveč se aplikacijo zapre tako, da se jo zagrabi in povleče proti spodnjemu dela ekrana. Ob takem dogodku bo operacijski sistem (angl. *Operating System (OS)*) dal aplikacijo v prekinjeno (angl. *Suspended*) stanje in jo bo zadrževal toliko časa v delovnem spominu, dokler ni ta del spomina sproščen zaradi potreb po drugih virih ali po prejetju zahteve za prekinitev aplikacije. Hkrati to pomeni, da se bo aplikacija ob ponovni zahtevi po zagonu hitreje naložila (prehod iz stanja *Suspended* v delujoče stanje (angl. *Running*)), kar pa si bomo pogledali v nadaljevanju, kjer smo podrobneje opisali aplikacijski model aplikacij Windows Store in prehode med stanji le-teh.

Revizija grafičnih kontrol

Tovrstne aplikacije ponujajo tudi nekatere nove kontrole, ki jih klasične aplikacije ali ne definirajo ali pa so zasnovane čisto drugače. Ena takšnih novosti je tudi malo drugačna aplikacijska vrstica (angl. *Application Bar*), ki je privzeto skrita, pojavi pa se na uporabnikovo gesto *Swipe*, ko uporabnik povleče s prstom od spodaj navzgor po robu zaslona ali ko se zgodi desni klik miške. Klasični **ASP.NET** kontroli *Menu* in menijska vrstica *MenuItem* ter **Windows Forms** kontrole *Menu*, *MenuItem* in *MenuStrip* prav tako ne obstajajo več. Glavni razlog je stalna vidljivost takšnih in podobnih menijev, kar je za uporabnika lahko moteče. Ena od alternativ za to kontrolo je aplikacijska vrstica, ki jo razvijalec postavi na zgornji ali spodnji del zaslona. Druga alternativa je ti. vstavljeni meni (angl. *Inline Menu*), ki ga razvijalci postavijo na primerno mesto med vsebino. Primer takšnega menija kaže slika 5.



Slika 5: Vstavljeni meni

Podpora več zaslonskih postavitev

Aplikacije ponujajo tudi več različnih postavitev na zaslonu, kar je sicer v osnovi omogočeno z operacijskim sistemom Windows 8. Tako si npr. lahko dve aplikaciji delita zaslonski prostor v razmerju, ki ga določi uporabnik (samo od Windows 8.1 dalje). Naloga razvijalcev pa je, da definirajo in implementirajo odzivno metodo ali funkcijo (angl. *Event Handler*), ki se izvede ob tem dogodku (angl. *Event*) ter prerazporedi grafične elemente na način, da lahko uporabnik še vedno nemoteno uporablja aplikacijo.

Žive ploščice in značke

Žive ploščice (angl. **Live Tiles**) ali tudi samo ploščice (angl. **Tiles**) in značke (angl. **Badges**) predstavljajo posebnost aplikacij Windows Store v primerjavi z namiznimi aplikacijami, saj slednje ne poznajo tega koncepta, medtem ko ploščice predstavljajo osnovo novega **Start** zaslona v Windows 8.

Ploščice so nadomestile klasične sistemske ikone. Ob namestitvi aplikacije se na Start zaslonu pojavi tudi njena živa ploščica, ki služi za zagon aplikacije ter prikaz dodatne vsebine z animacijo (odtod tudi ime "žive"). Velikosti ploščic se da spreminjati, kar pa ni bilo podprto pri statičnih ikoncah. Trenutno so podprte naslednje velikosti: velika (angl. *Large*), široka (angl. *Wide*), srednja (angl. *Medium*) in majhna (angl. *Small*). Kot zanimivost naj povemo, da se tudi namizne aplikacije lahko pripnejo na Start zaslon, vendar se v tem primeru ploščice obnašajo kot statične ikonce ter kot take služijo samo kot bližnjice za zagon aplikacije. Ploščice ponujajo tudi funkcionalnost, ki jih ikone nimajo - aplikacije Windows Store lahko uporabniku ponudijo vsebino tudi ko niso v aktivnem stanju, npr. aplikacija je konfigurirana tako, da periodično poišče posodobitve o novicah, vremenu, športnih rezultatih, ipd. ter jih prikaže v svoji ploščici.

Šarmi

Šarmi ali čare (angl. **Charms**) so posebna skupina gumbov, ki se pri sistemski jezikovni nastavitvi LTR (angl. *Left-to-right*) pojavi na desni strani zaslona in pri RTL (angl. *Right-to-left*) na levi. Skupina trenutno obsega gumbe *Search*, *Share*, *Start*, *Devices* in *Settings*, služi pa predvsem iskanju vsebine znotraj aplikacije, na spletu in v trgovini Windows Store, priklopu zunanjih naprav ter posebnim nastavitvam.

Dodatne lastnosti aplikacij

Za zaključek omenimo še nekatere dodatne lastnosti aplikacij Windows Store, ki naj bi jih razvijalci upoštevali pri razvoju:

- Pregledna in odprta postavitev grafičnih kontrol
- Čista hierarhija informacij
- Windows silhueta
- Neposredne interakcije
- Izkoriščanje robov
- Hitrost in fluidnost
- Smiselne in namenske animacije
- Oblikovanost za dotik
- Vgrajene kontrole
- Prilagoditev ne glede na velikost zaslona

2.5 Principi stilskega oblikovanja

Principi modernega oblikovanja grafičnih uporabniških vmesnikov bazirajo na nekaterih starejših principih in šolah, kot npr. **The Bauhaus**, **Motion Design** in **International typographic style (Swiss Style)**. Le-ti so v podjetju Microsoft predstavljali izhodiščno točko in navdih za definicijo novih principov stilskega oblikovanja:

- **Show pride in craftsmanship** (slov. Pokaži ponos v mojstrski izdelavi)
- **Do more with less** (slov. Naredi več z manj)
- **Be fast and fluid** (slov. Bodi hiter in tekoč)
- **Be authentically digital** (slov. Bodi verodostojno digitalen)
- **Win as one** (slov. Zmagaj kot eden)

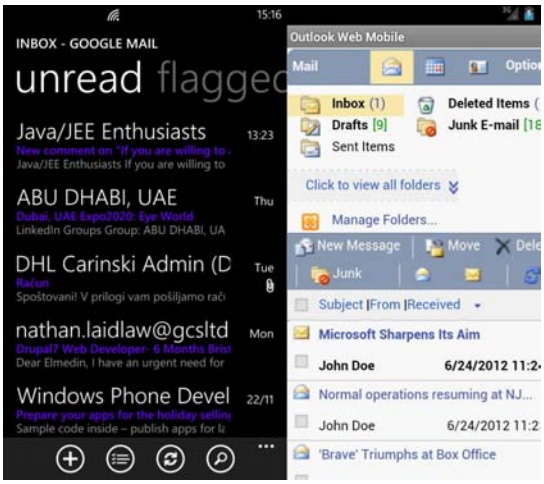
Pomembnost sledenja principom slogovnega oblikovanja za aplikacije Windows Store izraža tudi dejstvo, da Visual Studio ponuja obilico predlog za kreiranje projektov za Windows Windows, tako za C# kot tudi za JavaScript implementacije, kjer takoj opazimo, da se vse predloge podrejajo tem principom. V naslednjih sekcijah bomo zatorej predstavili principe, ki smo jih uporabili pri razvoju prototipov.

Show pride in craftsmanship

Že samo ime principa pove veliko. Poudarek je na posvečanju časa in energije razvijalcev in oblikovalcev oblikovanju aplikacije do najmanjše potankosti. Gre za osebnostni izziv oblikovati aplikacijo, na katero bomo ponosni. Hkrati naj bo aplikacija varna in zanesljiva. Ponavadi pri takšnih opravilih vložimo vso svojo osebnost, kot bi kreirali umetnino.

Do more with less

Sinonim principa *Do more with less* je izraz **Content over Chrome**. Ideja je v kreaciji izrazito koristnega grafičnega vmesnika aplikacije, pri čemer velja, da naj se uporabijo dejansko samo tisti grafični elementi, ki so pomembni in ki pripomorejo uporabniku osredotočenost na vsebino aplikacije in ki z odvečnimi elementi ne odvračajo uporabnikove pozornosti od vsebine.



Slika 6: Primer Content over Chrome in Chrome over Content aplikacij

Primer aplikacij, ki sta si - glede na ta princip - popolnoma diametralni, nam prikazuje slika 6. Na levi polovici slike imamo aplikacijo, za katero lahko rečemo Content over

Chrome, medtem ko imamo na desni polovici primer aplikacije, za katero bi lahko rekli, da sledi obratnem principu tj. *Chrome over Content*. Uporabniški vmesnik na desni je prenatrpan in nepregleden, ponuja opcije za upravljanje s sporočili vedno, tudi ko to ni potrebno, prav tako so vidne skoraj vse mape sporočil, koledar, itd. Skratka, prikazane so stvari, ki so sekundarnega pomena, na nepregleden način. Po drugi strani pa je grafični uporabniški vmesnik na levi strani veliko preglednejši in omogoča uporabniku fokusiranje na pomembne stvari, kot npr. vsebina, predmet sporočila, itd.

Be fast and fluid

Princip *Be fast and fluid* narekuje način uporabe animacij, prehodov in odzivov, ki so do uporabnika prijazni v smislu, da v ospredje postavlja ljudi, v ozadje pa tehnologijo. Aplikacija naj bo tekoča, kot bi predstavljala zgodbo skozi smiselno uporabo gibanja uporabnika skozi aplikacijo.

Be authentically digital

Bodi verodostojno digitalen. Princip, ki z vso prednostjo izkorišča dejstvo, da so aplikacije v končni fazi točke na zaslonu in ne kopije fizičnega sveta. Uporaba storitev v oblaku za dosego medsebojne povezanosti uporabnikov. Uporaba ikonc, ki niso kopija realnega sveta, temveč metafore. Poudarek na uporabi tipografije in drznih barv.

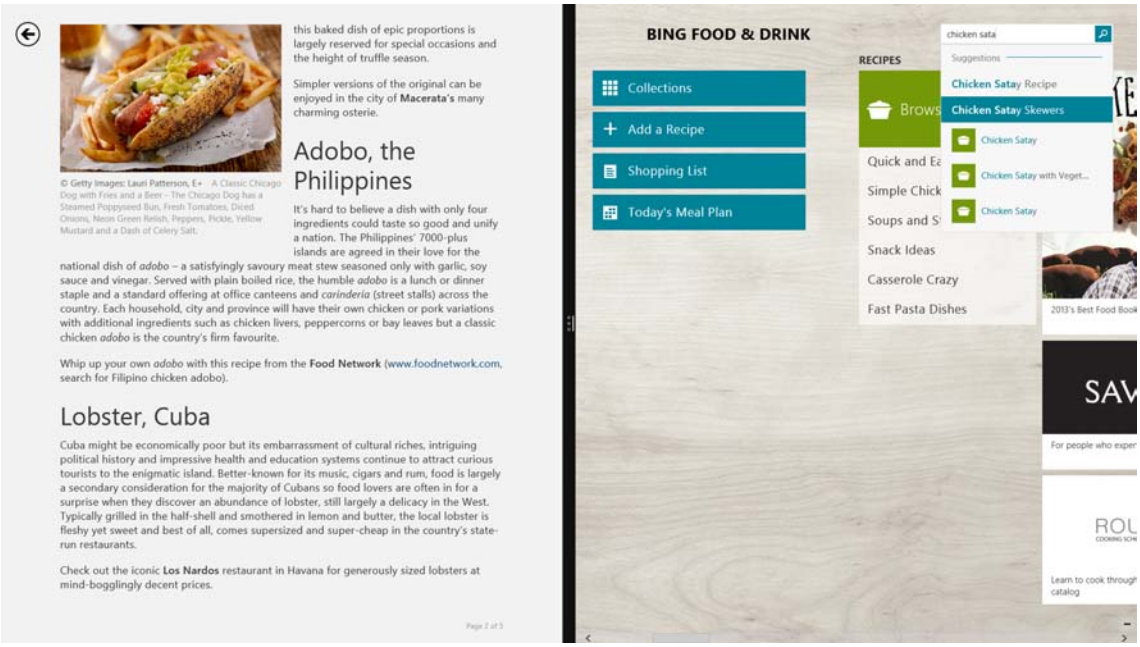
Win as one

Princip, ki zaokroži vse ostale principe. Ustvarjanje celostne podobe in scenarijev, kjer se uporabniki ne rabijo učiti tistega, kar že poznajo. Postavitev GUI modela v celoten mozaik okolja sistemov Windows 8 in Windows RT, znižana odvečnost, redundanca v GUI modelu. Uporaba predlogov zaradi učinka boljše konsistence slogovnega stila aplikacij.

Referenčni primer

Bing Travel in Bing Food sta lep primer referenčnih aplikacij, ki sledita principom slogovnega oblikovanja aplikacij Windows Store. Aplikaciji sta prikazani v deljenem zaslonskem načinu na sliki 7 in uporabljata projektni vzorec **Hub Page**, ki je podprt tako v C# kot tudi v JavaScript projektih. Glavna značilnost vzorca je, da se uporabljata kontroli

Hub in **Hub Section**, ki omogočata vodoravni premik vsebine, kjer so posamezne sekcije navpično omejene. Kot zanimivost naj povemo, da so klasične spletne strani pravo nasprotje aplikacij Windows Store (v načinu Hub), saj omogočajo navpični premik vsebine, dočim so vodoravno omejene na zaslonsko širino.



Slika 7: Aplikaciji v deljenem načinu

2.6 Stanja in prehodi med stanji aplikacij Windows Store

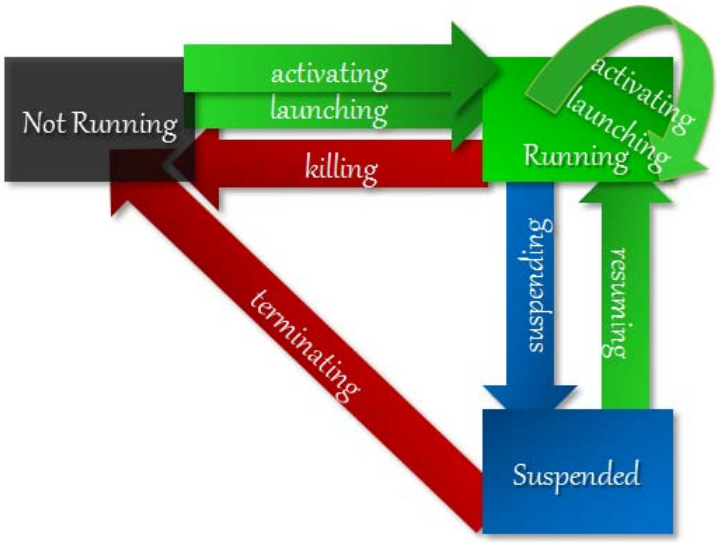
Stanja aplikacij Windows Store se razlikujejo od klasičnih namiznih Windows aplikacij tako, da je poleg stanj **delujoče** (angl. *Running State*) in **nedelujoče** (angl. *Not Running State*), za njih definirano še **prekinjeno** stanje (angl. *Suspended State*). To stanje je predvideno za hitro vračanje aplikacije v delujoče stanje, saj je v prekinjenem stanju aplikacija še vedno naložena v delovnem spominu in je zato ob prehodu v delujoče stanje operacijski sistem ne rabi ponovno nalagati. V primeru potrebe po sistemskih virih za druga opravila, operacijski sistem sprosti vire, ki jih aplikacija zaseda in njeno izvajanje zaustavi [6]. Ponavadi so ta opravila npr. zagon drugih aplikacij, upravljanje sistemskih storitev, itd.

Upravljanje z življenskim ciklom in stanji aplikacij Windows Store je pomembno kajti gre za proces, ki ga lahko delno kontroliramo tudi razvijalci, saj tako dodatno pripo-

moremo k boljši uporabniški izkušnji uporabnikov z našo aplikacijo. Razvijalci naj bi zagotovili predvsem naslednje:

- Ko uporabnik zapusti aplikacijo, npr. odpre drugo aplikacijo, naj bi se prvotna aplikacija znašla v enakem stanju kot je bila, preden jo je uporabnik zapustil
- Ko uporabnik zapre aplikacijo, naj bi ob njenem ponovnem zagonu dobil začetno sejo aplikacije

Slika 8 prikazuje možna stanja v katerih se aplikacija lahko nahaja in akcije, ki povzročijo prehode med stanji aplikacije iz enega v drugo [3].



Slika 8: Stanja in prehodi med stanji aplikacij Windows Store

Možni prehodi stanj aplikacij Windows Store so definirani kot:

- **Launching** - prehod iz stanja *Not Running*
- **Activating** - prehod iz stanja *Not Running* ali iz stanja *Running* v isto stanje
- **Suspending** - prehod iz stanja *Running* v stanje *Suspended*
- **Resuming** - prehod iz stanja *Suspended* v stanje *Running*
- **Killing** - aplikacija je bila zaustavljena s strani uporabnika
- **Terminating** - aplikacija je bila prekinjena s strani operacijskega sistema zaradi pomanjkanja virov ipd.

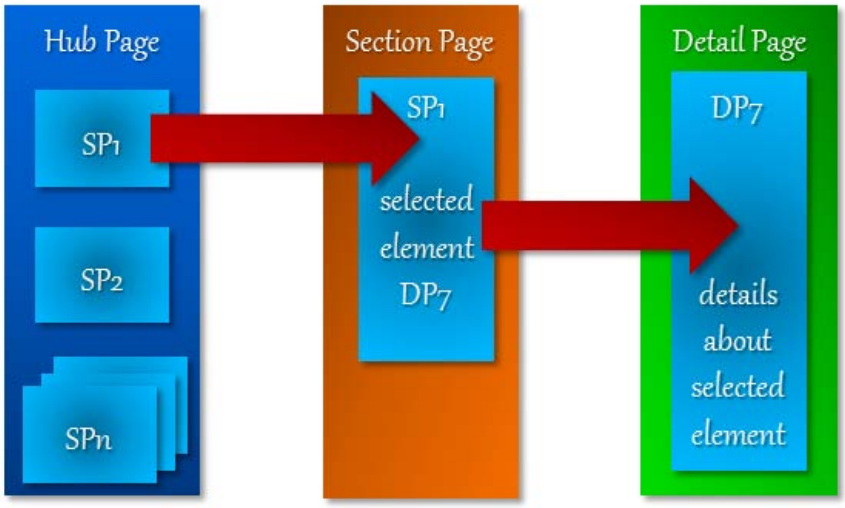
2.7 Navigacijski vzorci

Navigacija je v aplikacijah Windows Store zasnovana na naslednjih vzorcih [16]:

- **Hierarhični** (angl. *Hierarchical*)
- **Ploščati** (angl. *Flat*)
- **Kombinirani** (angl. *Combined*)

Navigacijski vzorci definirajo organizacijo vsebine aplikacij in navigacijo med njenimi stranmi. Izbira pravega vzorca je v mnogočem odvisna od tematike aplikacije, zato je dobro pred dejansko implementacijo skrbno načrtovati uporabniški vmesnik, postavitev in razdelitev vsebine ter definirati navigacijske poti, kot bomo videli kasneje, ko bomo opisali navigacijo med stranmi in razreda *NavigationHelper (C#)* in *PageControlNavigator (JavaScript)*. Pri načrtovanju uporabniškega vmesnika prototipov in vsebine aplikacije smo se odločili za hierarhični vzorec, saj le-tega uporabljata tudi ti. *Windows.UI.Xaml.Controls.Hub* in *WinJS.UI.Hub* kontroli, ki ju bomo posebej predstavili.

Slika 9 vizualno predstavlja navigacijo med stranmi po hierarhičnem vzorcu, kjer lahko vidimo tri strani aplikacije: začetno stran, ki poudarja posamezne sekcije aplikacije, ki so detajlno predstavljene na drugi strani z množico elementov ter končno tretjo stran, ki kaže detajle o izbranem elementu.



Slika 9: Navigacija med stranmi po hierarhičnem navigacijskem vzorcu

2.8 Windows Runtime API

S prihodom operacijskega sistema Windows 8 se je pojavila tudi nova aplikacijska arhitektura, ki ponuja naslednji množici Windows Runtime API ter kot taka definira novo platformo za razvoj aplikacij Windows Store:

- **Windows Runtime (WinRT)**
- **Windows Library for JavaScript (WinJS)**

Izvorna koda je bila izrecno napisana v jeziku C++ in je projicirana v jezike C# in JavaScript ter Visual Basic (VB). Programsko ogrodje .NET za namen razvoja aplikacij Windows Store ponuja podmnožico tipov, ki se uporabljajo v razvojnem pristopu z uporabo jezikov C# in XAML. Podmnožica teh tipov je poimenovana **.NET for Windows Store apps**, njeni tipi pa se kličejo neposredno iz Windows Runtime API. Razločujemo jih lahko po začetku imenskih prostorov (angl. *namespace*) in sicer tako, da če je API vsebovan v imenskem prostoru **System**, potem vemo, da gre za .NET tip, če pa je vsebovan v imenskem prostoru **Windows** je nedvomno Windows Runtime tip.

Nekateri .NET tipi so bili izpuščeni iz omenjene .NET podmnožice zaradi jasnosti in preglednosti, drugi pa zaradi potrebe, kot npr. ASP.NET tipi, potem nasplošno opuščeni in prekrivajoči se tipi, itd. Primer izpuščenega tipa, ki se uporablja za konzolne operacije je **System.Console**, kajti aplikacije Windows Store nimajo definirane klasične vhodno izhodne konzole ter prav tako ne standardnih tokov za vhod, izhod in napake.

Windows Runtime (WinRT)

Pri razvoju C# in XAML prototipa aplikacije smo uporabljali nekatere od naslednjih WinRT C# API modulov:

- **Core** - osnovni modul, ki vključuje tipe kot npr. *Windows.ApplicationModel*, *Windows.Foundation*, *Windows.System*, *Windows.UI*, ipd.
- **Controls** - modul, ki vsebuje tipe za GUI, s pomočjo katerih lahko uporabnik pregleduje in izbira datoteke *Windows.Storage.Pickers* ter modula za podporo sistemskih in uporabniških kontrol *Windows.UI.Xaml.Controls* ter komponentne dele kontrol *Windows.UI.Xaml.Controls.Primitives*.
- **Data & Content** - moduli, ki vsebujejo tipe za infrastrukturo povezovanja aplikacijskih podatkov, tip za delo s HTML podatki in JSON objekti, potem modul za podporo XML dokumentnega objektnega modela (angl. *Document Object Model - DOM*), itd.

- **Devices** - modul s tipi za podporo naprav ter tako funkcij za tipala, geolokacijo, pametne kartice, povezavo do omrežja Wi-Fi, upravljanje audio in video naprav, pošiljanje kratkih sporočil, itd.

- **Files & Folders** - modul, ki ponuja podporo za upravljanje datotek, imenikov in aplikacijskih nastavitev *Windows.Storage*, modul za napredno iskanje vsebine (angl. *Advanced Query Syntax (AQS)*, itd.

- **Globalization** - modul za podporo jezikovnih profilov, geografskih regij in mednarodnih koledarjev *Windows.Globalization*, itd.

- **Graphics** - moduli za delo z zaslonom, grafiko, slikovnimi datotekami in animacijami kot npr. *Windows.Graphics.Display*, *Windows.Graphics.Imaging*, *Windows.UI.Xaml*, itd.

- **Media** - moduli za delo s fotografijami ter avdio in video zapisi in napravami kot npr. *Windows.Media*, *Windows.Media.Devices* in *Windows.Ui.Xaml.Media*

- **Networking** - moduli, ki definirajo HTTP razrede in njihovo uporabo, potem ti. *push* notifikacije ter moduli za delo z omrežjem, kot npr. *Windows.Networking* in *Windows.Web*.

- **Presentation** - eden najpomembnejših modulov, kjer so definirane XAML kontrole, navigacije med stranmi ter osnovne komponente grafičnih elementov, kot npr. *Windows.UI*, *Windows.UI.Xaml*, in *Windows.UI.Xaml.Controls*.

- **Security** - moduli za delo z varnostnimi elementi in avtentikacijo na Windows Live, kot npr. *Windows.Security.Cryptography*, *Windows.Security.Credentials* in *Windows.Security.Authentication.Web*.

- **UI Automation** - moduli, ki definirajo podporne tipe za **Microsoft UI Automation** infrastrukturo.

- **User Interaction** - moduli, ki nudijo podporo za vhodne operacije oz. geste uporabnikov ali naprav za interakcijo, kot npr. dotik zaslona, miška in tipkovnica. Modul dodatno nudi tudi detekcijo gest in njihovo manipulacijo.

Windows Library for JavaScript (WinJS)

Analogno kot smo pri razvoju prototip aplikacije Windows Store z uporabo tehnologij C# in XAML uporabljali WinRT API, smo v JavaScript projektu izkoristili prednosti

JavaScript knjižnice, ti. WinJS API [24], ki ponuja posebno funkcionalnost za JavaScript razvoj aplikacij Windows Store. WinJS ponuja enumeracije (angl. *enumerations*), objekte (angl. *objects*), vmesnike (angl. *interfaces*) in funkcije (angl. *functions*). WinJS je del ti. *Windows API reference for Windows Store apps* in je na abstraktnem nivoju združen z *Windows Runtime (WinRT)* upravljanimi tipi, WinJS tipi pa se prepoznajo po njihovem imenskem prostoru (**WinJS**).

WinJS zajema naslednje imenske prostore:

- **Core**
  - **WinJS** - poseben imenski prostor za podporo posebne JavaScript funkcionalosti kot sta ti. *Promise* (slov. obljuba) in **xhr** (objekt, ki zavije *XMLHttpRequest* v obljubo), podobno kot **AJAX** tehnike, ki uporabljajo *XMLHttpRequest* objekt za asinhrono pošiljanje in sprejemanje podatkov.
  - **WinJS.Application** - posebna funkcionalnost na aplikacijskem nivoju, kot npr. aktivacija, shranjevanje in definicija aplikacijskih dogodkov.
  - **WinJS.Resources** - funkcije za dostop do aplikacijskih virov in lokalizirane vsebine.
- **Controls**
  - **WinJS.UI** - zelo pomemben modul, ki ponuja UI kontrole in objekte za podatkovno manipulacijo.
- **Data and content**
  - **WinJS.Binding** - funkcionalnost za povezovanje podatkov in predlog.
- **Helpers**
  - **WinJS.Class** - priročne funkcije za definicijo razredov.
  - **WinJS.Namespace** - podobno kot WinJS.Class - ponuja funkcije za definicijo imenskih prostorov.
  - **WinJS.Utilities** - ponuja priročne funkcije za delo s CCS razredi.
- **Presentation**
  - **WinJS.Navigation** - funkcionalnost za osnovno navigacijo med stranmi.
  - **WinJS.UI** - posebne kontrole in objekti, ki manipulirajo s podatki.
  - **WinJS.UI.Animation** - dostop do animacij z uporabo izvedenih kontrol.

– **WinJS.UI.Fragments** - funkcije, ki omogočajo nalaganje HTML vsebine programabilno.

– **WinJS.UI.Pages** - funkcije za definicijo in prikaz *PageControl* objektov.

2.9 Live Connect API

Microsoftov **Live Connect API** omogoča razvijalcem delo z *Live Connect* združljivimi servisi, kot npr. **Outlook**, **SkyDrive** in **Skype**.

Live Connect JavaScript API

Poleg Windows Runtime API modulov lahko pri razvoju uporabljamo še nekatere Microsoft API module, kot sta The Live Connect JavaScript API [13] in pa The Live Connect Managed API [15]. JavaScript API v kombinaciji z REST API modulom ponuja metode za uporabo servisov Windows Live v jeziku JavaScript, kot npr. prenos datotek z internetne shrambe za shranjevanje datotek **Microsoft SkyDrive**, prijava na dogodek, branje sejnega objekta, prikaz elementa grafičnega vmesnika za prijavo na storitve **Live**, itd.

Live Connect Managed API

Za razliko od JavaScript in REST API modulov, Live Connect Managed API modul ponuja C# API v imenskih prostorih *Microsoft.Live in Microsoft.Live.Controls*, uporablja se pa za poenostavljen dostop do uporabnikovih podatkov iz aplikacij Windows Store in Windows Phone, ponuja pa tudi grafične Live Connect elemente. Live API ni samodejno vključen s postavitvijo osnovnih komponent razvojnega okolja, temveč je potrebno namestiti poseben paket **Live SDK** (trenutno zadnja različica je 5.5) in nato v Visual Studio rešitvi referencah projektnih knjižnic dodati tudi Live API knjižnico tako, da izberemo *Add Reference*, nato označimo *Windows modul*, nato *Extensions* ter končno izberemo *Live SDK*. To je le osnova, da lahko prevedemo modul za povezavo z Live servisi, ne zadošča pa za dejansko povezavo. Potrebne korake za uspešno prijavo na Microsoft Live storitve smo opisali v naslednjem poglavju.

Live Connect REST API

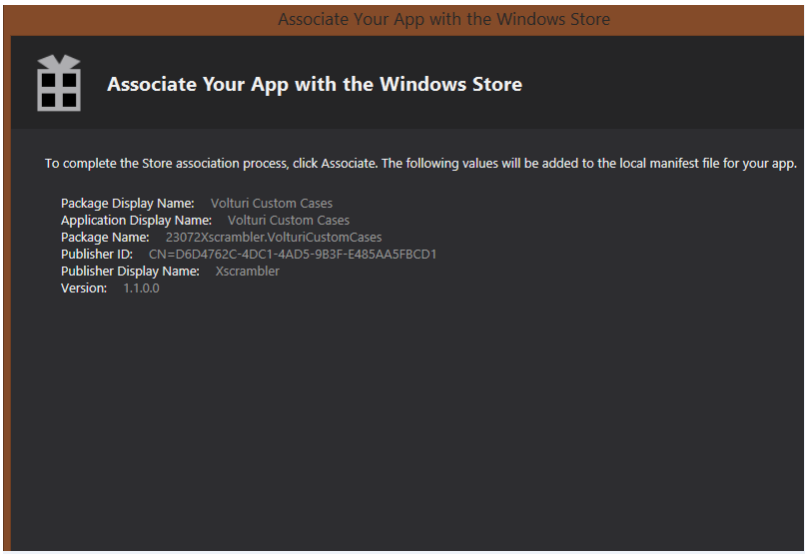
Live Connect REST API [19] omogoča aplikacijam, ki uporabljajo storitve Live Connect programabilen dostop do informacij o uporabnikih in njihovih atributih, kot so kontakti, prijatelji, aktivnosti, SkyDrive datoteke in drugi. REST API podpira naslednje HTTP metode: *GET* (vrne vir), *POST* (doda nov vir kolekciji objektov), *PUT* (posodobi vir na lokaciji kamor kaže URL ali naredi novega, če vir ne obstaja), *DELETE* (pobriše vir), *MOVE* (premakne lokacijo vira) in *COPY* (duplicira vir).

Registracija aplikacije na Windows Store in storitev SkyDrive

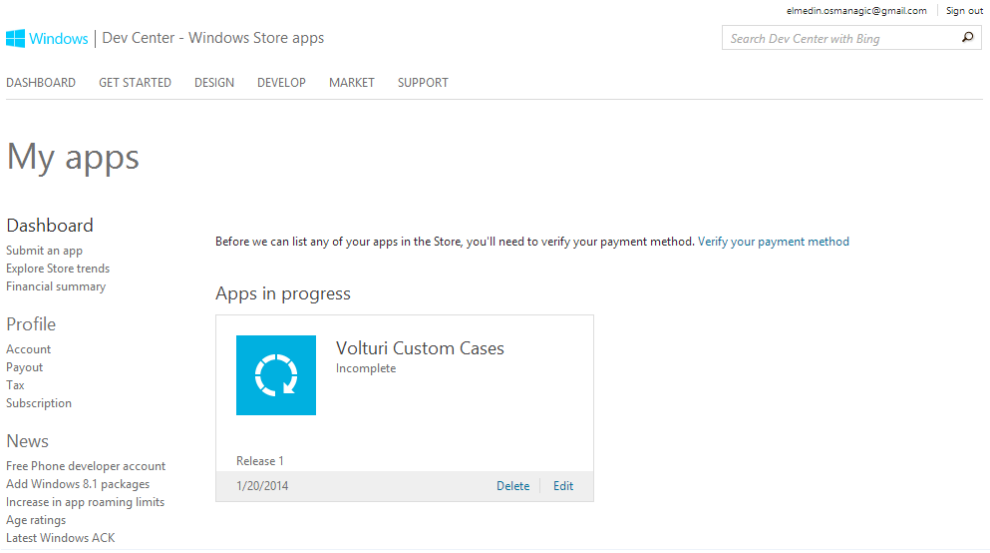
Storitev SkyDrive omogoča shranjevanje podatkov v oblaku, vendar moramo za uporabo le-tega imeti veljaven *Microsoft Live* račun, pred zagonom aplikacije pa moramo opraviti še naslednje korake:

- Ustvarimo ti. *Windows Live* uporabniški račun pri podjetju Microsoft.
- Nato se registriramo še kot razvijalec aplikacij Windows Store.
- Prijavimo se na spletno stran *Dev Center - Windows Store apps*.
- V zavihku *Dashboard* kliknemo na spletno povezavo *Submit an app* in v polje *App name* vpišemo ime aplikacije ter nato kliknemo na gumb *Reserve app name*, kar bo naredilo rezervacijo naziva naše aplikacije (če je naziv aplikacije še prost).
- Nato odpremo Visual Studio ter se postavimo na projekt ter z desnim klikom odpremo *Store - Associate App with the Store* in zaključimo proces asociacije.
- Po uspešnem zaključku asociacije se ustvari datoteka *Package.StoreAssociation.xml*, ki hrani podatke kot npr. *Publisher*, *ReservedNames*, *MainPackageIdentityName*, itd.
- Če želimo uporabljati servise **Windows Push Notification Services (WNS)**, moramo pridobiti še podatka *Client secret* ter *Package Security Identifier* za zaščito naše aplikacije, kajti **WNS** in **Live Connect** storitvi uporabljata te podatke za avtentikacijo komunikacije s strežnikom. Storitev WNS omogoča ti. *third-pary* razvijalcem pošiljanje posodobitev *toast*, *tile*, *badge* in surovih (angl. *raw*) podatkov z uporabo njihovih lastnih storitev v oblaku.

Na slikah 10 in 11 smo kot primer prikazali *Developer Center Dashboard* in končni korak asociacije aplikacije z Windows Store.



Slika 10: Developer Center Dashboard



Slika 11: Asociacija aplikacije z Windows Store

3 Prototip aplikacije Windows Store

3.1 Tematika

Ob snovanju ideje za prototip aplikacije smo se odločili razviti aplikacijo Windows Store **Volturi Custom Cases**, katere tematika so ročno narejene torbe za biljard palice. Aplikacija bo uporabnikom nudila pregled najpopularnejših torbic, izdelavo naročila nove torbice, odpošiljanje izdelanega naročila izdelovalcu, dodajanje torbice med priljubljene in odstranitev le-te iz skupine priljubljenih, prijavo na servise **Windows Live**, shranjevanje slik in naročila v **SkyDrive** mapo, ipd.

3.2 Analiza zahtev

Pri analizi zahtev se glede števila uporabniških zahtev nismo postavili preveč statično, saj namen diplomskega dela ni dokončna implementacija točno zastavljenega števila funkcionalnih in nefunkcionalnih zahtev, temveč je cilj izdelati primerjalno študijo o razvoju aplikacij za Windows Store. Zatorej smo zahteve specificirali bolj ohlapno in prototipa implementirali z bolj agilnim pristopom in z namenom, da pokrijemo nekatere najbolj važne segmente razvoja aplikacij za Windows Store.

Pri implementaciji prototipov nismo definirali nobenih nefunkcionalnih zahtev, razen da naj *razvoj prototipov sledi principom stilskega oblikovanja aplikacij za Windows Store.*

Definirali smo naslednje funkcionalne zahteve:

- **Uporaba osnovnih Windows Live storitev**
  - uporabnik se lahko prijavi na *Windows Live* račun (*login*)
  - uporabnik se lahko odjavi iz *Windows Live* računa (*logout*)
  - prikaz trenutnega stanja prijave *Windows Live* računa (*show login status*)
- **Naročilo torbice za biljard palice**
  - naročilo vsebuje sekcije podatkov kot npr. podatki o stranki (*customer info*), izbira obstoječega modela torbice na osnovi katerega bo narejena posebna verzija za stranko (*select base case model*), zunanjost torbice (*exterior*), notranjost torbice (*interior*), dodatki (*extras*) in dodatne beležke in pojasnila izdelovalcu (*notes*)

  - naročilo je mogoče shraniti lokalno (*save locally*) in tudi v *SkyDrive* mapo (*save to SkyDrive*) (le-ta predvideva uspešno predhodno prijavo v Windows Live račun)
  - naročilo je možno poslati izdelovalcu (*send order*) ali ga postaviti na privzete vrednosti (*clear order*)
  - naročilo je mogoče ponovno naložiti in ga urejati naprej (*load order*)

- **Priljubljene torbice**
  - prikaz priljubljenih torbic (*show favorites*)
  - dodajanje torbic v skupino priljubljenih (*add to favorites*)
  - odstranitev torbic iz skupine priljubljenih (*remove from favorites*)

- **Prezentacijski video posnetek**
  - uporabnik si lahko pogleda prezentacijski video posnetek (*show video clip*)
  - dodajanje komentarjev na video posnetek (*add comments to video clip*)
  - ocenjevanje video posnetka (*rate video clip*)

- **Dodatne zahteve**
  - zamenjava teme aplikacije (*change application theme*)
  - ocenjevanje aplikacije na *Windows Store* (*rate application*)
  - pošiljanje povratne informacije razvijalcu (*send feedback to developer*)
  - pregled *SkyDrive* imenikov (*show my SkyDrive folders*)
  - pošiljanje kratkih sporočil razvijalcu na *Twitter* (*send tweets to developer*)
  - obisk spletne strani *Volturi Custom Cases Facebook* (*visit Volturi Custom Cases Facebook web page*)
  - izdelava posebnega nastavitvenega modula (*settings module*)
  - vizitka razvijalca in podatki o aplikaciji (*contact info & about application*)

3.3 Koncepti in paradigme

Pri razvoju prototipov smo se osredotočili na bolj pomembne koncepte in paradigme razvoja aplikacij Windows Store. Le-ti so večinoma različni že zaradi uporabljenih tehnologij, pa vendar si oba pristopa razvoja aplikacij za Windows Store na abstraktnem nivoju delita iste principe, paradigme in abstrakcije, zato smo pri razvoju obravnavali naslednje izbrane module aplikacij Windows Store:

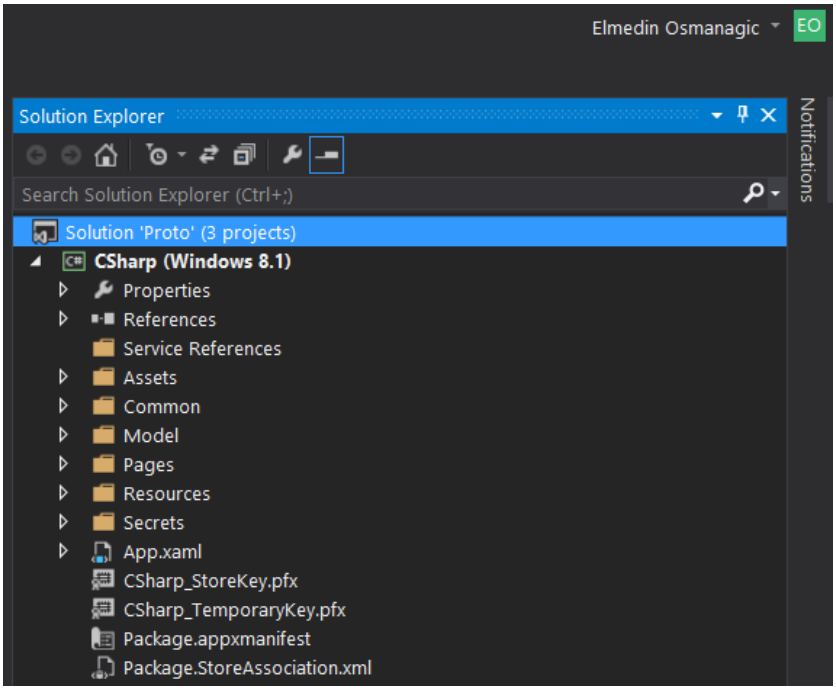
- **Asinhrono programiranje** (angl. *Asynchronous Programming*)
- **Aplikacijski model** (angl. *Application Model*)
- **Navigacija med stranmi** (angl. *Page Navigation*)
- **Kontrole** (angl. *Controls*)
- **Stili** (angl. *Styles*)
- **Obravnava dogodkov** (angl. *Event Handling*)
- **Povezovanje podatkov** (angl. *Data Binding*)
- **Oblikovanje uporabniškega vmesnika** (angl. *User Interface Design*)

Slednji so nam služili za dejansko primerjavo razvoja obeh prototipov in kot taki tvorijo osnovo za primerjalno izhodišče. Čeprav so nekateri koncepti na abstraktnem nivoju enaki pri obeh pristopih razvoja, kot npr. aplikacijski model, obravnava dogodkov, ipd. pa se implementacija obeh drastično razlikuje, kar bomo posebej obravnavali. Vsaka od naštetih sekcij vsebuje množico nastavljivih vrednosti, ki jih nastavimo deklarativno. Nastavljanje vrednosti je sicer neodvisno od uporabljenega pristopa razvoja aplikacij Windows Store.

3.4 Visual Studio rešitev

Visual Studio IDE omogoča dodajanje različnih tipov projektov v eno samo datoteko (angl. *Visual Studio Solution File*). Našo prototip rešitev smo poimenovali enostavno **Proto**. Dodali smo projekta **CSharp** in **JS**, kjer smo v prvem implementirali prototip z uporabo C# in XAML, drugega pa z uporabo HTML, CSS in JavaScript tehnologij. Visual Studio omogoča nastavitev posebne opcije projektu ti. **StartUp Project**, kar nam omogoča da poljubno nastavimo, kateri projekt v rešitvi se bo zagnal. Takšna organizacija rešitve omogoča boljši pregled nad izvorno kodo, lažjo prenosljivost ter tudi lažje vzporedno delo na več projektih. Poleg tega smo dodali še testni projekt **CSharpTestLibrary** za testiranje C# projekta CSharp.

Detajle o organizaciji in vsebini C# projekta CSharp nam ponazarja slika 12, dočim je JavaScript projekt JS organiziran podobno.



Slika 12: Visual Studio rešitev Proto

V CSharp projektu smo poleg privzetih imenikov *Properties* (tu se nahaja datoteka AssemblyInfo.cs, ki nastavi parametre pripadajoče datoteke DLL, kot npr. *AssemblyTitle*, *AssemblyVersion* ipd.) in *References* (tu se nahajajo reference na *Windows* in *.NET for Windows Store apps* knjižnice), definirali še nekatere. Poleg omenjenih dodatek Visual Studio samodejno vključi še datoteko s cerfitikatom (*CSharp_TemporaryKey.pfx*), ki je nujna za podpis namestitvenega paketa aplikacije. Analogno je takšna datoteka vključena tudi v JS projekt (*JS_TemporaryKey.pfx*). Poleg tega se vključi tudi datoteka v *Personal Information Exchange (PFX)* formatu *CSharp_StoreKey.pfx*, za podpis aplikacije s privatno/javnim ključem (uporablja se *Authenticode technology*).

Celotna organizacija rešitve za oba projekta pa je po sekcijah razdeljena tako:

- **Projekt *CSharp***
  - **Properties** - *AssemblyInfo.cs* datoteka.
  - **References** - Reference na SDK API knjižnice *.NET for Windows Store app*, *Windows* in *Live SDK*.
  - **Service References** - Reference spletnih servisov.
  - **Assets** - Logotipi, slike in video material.

  - **Common** - Skupni razredi, ki niso tipa *Page*, npr. *SuspensionManager* ali niso za uporabo v imenskem prostoru *Model*.
  - **Model** - Aplikacijski podatki (JSON datoteka) in razredi, ki implementirajo model kot npr. *CustomerInfo*, *CustomCueOrder*, ipd.
  - **Pages** - XAML datoteke in pripadajočimi C# datotekami.
  - **Resources** - Izvedeni stili za XAML kontrole.
  - **Secrets** - Datoteka s podatki *key*, *secret* in *application* za uporabo **Flickr** spletnega servisa.
- **Projekt *JS***
  - **References** - Referenca na *Windows Library for JavaScript 2.0* API, ki vsebuje *CSS* in *JavaScript* datoteke, npr. css/ui-dark.css, css/ui-light.css, base.js, en-us/ui.js, itd.
  - **css** - CSS datoteke, npr. default.css
  - **assets** - Logotipi, slike in video material.
  - **js** - Preostale JavaScript datoteke, ki funkcionalno ne spadajo v imenik *pages*.
  - **model** - Aplikacijski podatki (JSON datoteka).
  - **pages** - HTML strani aplikacije organizirane v imenike s pripadajočimi datotekami (**.html*), (**.css*) in (**.js*).
  - **secrets** - Datoteka s podatki *key*, *secret* in *application* za uporabo **Flickr** spletnega servisa.
  - **strings** - Viri za JSON podatke (*resources.resjson*).

3.5 Konfiguracija aplikacije

Parametri aplikacije so nastavljeni v konfiguracijski datoteki ***package.appxmanifest*** [3], ki je identična za oba projekta, vendar vsebujeta vsak svojo kopijo, sicer se ne bi uspešno prevedla. Zaradi obilice parametrov smo jih nastavili samo nekaj (**Display name** = Volturi Custom Cases, **Entry point** = CSharp.App, **Default language** = en-US, **Supported rotations** = Landscape, **Capabilities** = Internet (Client), Location, Pictues Library, **PackageName** = 23072Xscrambler.VolturiCustomCases), tiste manj pomembne za prototip pa smo izpustili (kot npr. parametre, ki zadevajo dejansko objavo aplikacije na Windows Store kot npr. Publisher display name, kopica logotipov, ipd.).

Datoteka trenutno definira naslednje sekcije in parametre:

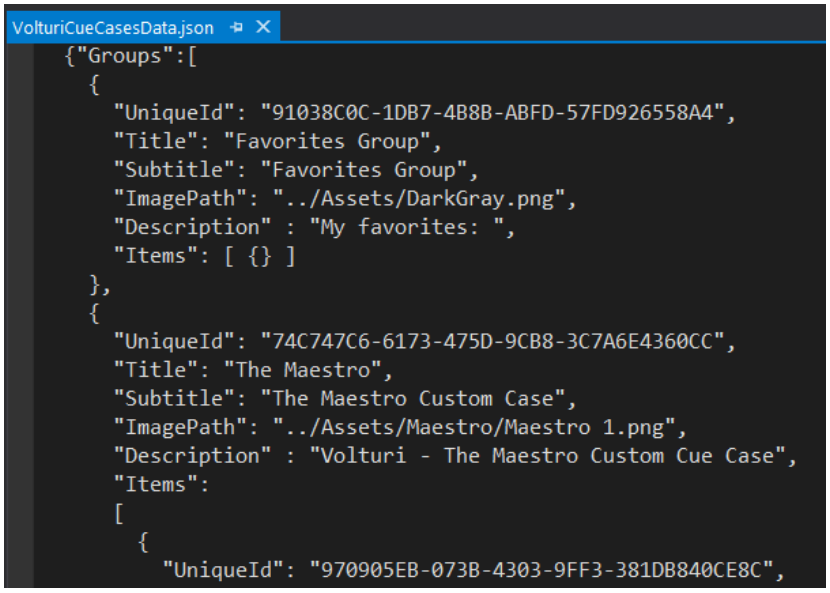
- **Application** (slov. *Aplikacija*) - *Display name*, *Entry point*, *Default language*, *Description*, *Supported orientations*, *Minimum width*, *Notification*, *Tile update*, itd.
- **Visual Assets** (slov. *Vizualna sredstva*) - *Tile*, *Splash screen*, *Square 70x70 logo*, *Square 150x150 logo*, *Wide 310x150 logo*, *Store logo*, *Badge logo*, itd.
- **Capabilities** (slov. *Zmogljivosti*) - *Enterprise Authentication*, *Internet*, *Location*, *Microphone*, *Music, Pictures & Video Library*, *Webcam*, itd.
- **Declarations** (slov. *Deklaracije*) - *Description*, *Properties*, *Available & Supported Declarations*
- **Content URIs** (slov. *Vsebinski URI naslovi*) - URI & Rule
- **Packaging** (slov. *Pakiranje*) - *Package name*, *Package display name*, *Version*, *Publisher*, *Package family name*, *Generate app bundle*

3.6 Aplikacijski podatki

Aplikacije Windows Store so večinoma aplikacije z obilico podatkov in malo poslovne logike, razen nekaterih izjem. Lahko bi tudi rekli, da so vglavnem upravljane s podatki (angl. *Data-Driven Applications*). V tem kontekstu lahko zaključimo, da je prikaz rezultatov bolj odvisen od podatkov samih kot od poslovne logike. Podatki so v aplikacijah Windows Store shranjeni na naslednjih podatkovnih lokacijah:

- **Lokalna** (angl. *Local*) - Stalni podatki, ki obstajajo samo na trenutni napravi.
- **Gostujoča** (angl. *Roaming*) - Podatki, ki obstajajo na vseh napravah, na katerih je uporabnik namestil aplikacijo.
- **Začasna** (angl. *Temporary*) - Začasni podatki, ki jih sistem lahko pobriše kadarkoli.

Prototip lahko dostopa do vseh podatkov razen do začasnih sistemskih podatkov. Nekateri podatki so statični na nivoju aplikacije, kar omogoča vsem modulom aplikacije njihov dostop in obdelavo. Tu gre za podatke o stanju aplikacije, shranjenem naročilu biljard torbice, grupah vseh biljard torbic, ki so na voljo in njihovih detajlih, ipd. Te podatke lahko aplikacija zapiše lokalno ali v načinu gostovanja, nekateri od le-teh pa so shranjeni v paketu aplikacije v tekstualni datoteki v formatu **JavaScript Object Notation (JSON)**. Datoteko smo poimenovali preprosto *VolturiCueCasesData.json*, del podatkov pa je prikazan na sliki 13.



Slika 13: Podatkovna datoteka *VolturiCueCasesData* v formatu JSON

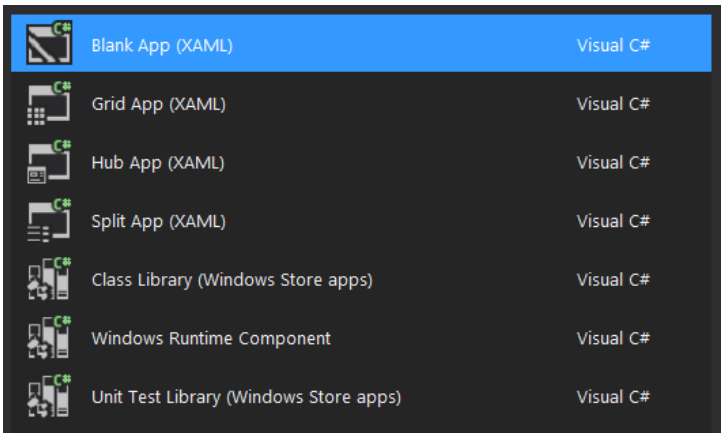
Podatki v datoteki so organizirani tako, da je na najvišjem nivoju definiran objekt *Groups*, ki vsebuje matriko objektov, ki vsebujejo elemente. Izjema je prvi objekt poimenovan *Favorites Group*, ki izjemoma ne vsebuje nobenega elementa. Naslednja na sliki prikazana matrika je poimenovana *The Maestro* in vsebuje podatke o grupi, kot npr. **enolični identifikator grupe** (tip GUID), **naslov** (tip String) oz. naziv ter podnaslov grupe (*Title* & *Subtitle*) (tip String), **relativno pot do slikovne datoteke** (tip String), ki predstavlja grupo (*ImagePath*), **opis** (*Description*) (tip String) in **listo elementov**, ki predstavljajo posamezne slike biljard torbice (*Items*) (tip VolturiCueCasesDataItem[]).

3.7 Projektna predloga

Visual Studio Express 2013 for Windows vsebuje kopico projektnih predlog pri začetni postavitvi projektov za razvoj aplikacij. Razlogov za nastanek projektnih predlog je več, kot najpomembnejše pa bi navedli večkratno ustvarjanje različnih projektov z uporabo identičnih predlog, odstranitev potrebe razvijalcem za ustvarjanje okostja projekta od začetka in na novo, ustvarjanje datotečne strukture po v praksi uveljavljenih tehnikah, omogočanje lastnih predlog ter hitrejši, lažji in bolj sistematičen pristop k razvoju in testiranju aplikacij. Visual Studio trenutno ponuja projektne predloge *Blank App*, *Grid App*, *Hub App* in *Split App* za C# in JavaScript, za slednjega pa definira še predlogo *Navigation App*. Omenimo lahko še projektne predloge **Class Library**, **Windows Runtime Component** ter **Unit Test Library**, katerih namen pa ni postavitev ogrodja za apli-

kacije Windows Store, temveč implementaciji DLL in Windows Runtime komponent ter knjižnic za testiranje enot aplikacij.

Slika 14 prikazuje XAML predloge, ki so trenutno definirane za C#. Predloge vsebujejo osnovne C# in XAML datoteke za gradnjo prototipa aplikacije.



Slika 14: Projektne predloge za XAML

Pri razvoju prototipov smo uporabili projektno predlogo **Hub App** za C# in JavaScript, saj smo pri analiziranju referenčnih Windows Store aplikacij ugotovili, da so ravno tiste, ki uporabljajo to predlogo, tudi najbolje oblikovane in ponujajo najlažjo in najbolj smiselno navigacijo med stranmi.

4 Razvoj prototipa aplikacije Windows Store z uporabo tehnologij C# in XAML

V tem delu diplomske naloge bomo pokazali značilnosti razvoja aplikacij Windows Store z uporabo tehnologij C# in XAML. Opisali bomo uporabljene tehnologije razvoja programske opreme, nekatere koncepte programskih jezikov C# in XAML ter opisali, kako smo razvili prototip aplikacije za Windows Store *Volturi Custom Cases*.

4.1 Tehnologije

Programski jezik C#

C# (tudi **C Sharp** ali **Visual C#**) je multi paradigmatični, moderni, objektno orientirani in tipsko varni programski jezik, ki ga je razvilo podjetje Microsoft [8]. Tako kot prej omenjeni **CLI**, je tudi programski jezik C# standardiziran s strani organizacij ECMA (pod oznako ECMA-334) in ISO (ISO/IEC 23270), trenutno zadnja različica pa je 5.0. Danes je C# eden najpopularnejših objektno orientiranih jezikov. Razlogi za njegovo popularnost so, poleg ostalih, tudi dejstva, da je C# programski jezik, ki ...

- ... je primeren za splošno uporabo
- ... je komponentno orientiran
- ... je strukturiran jezik, ki se ga je enostavno naučiti
- ... je del popularne platforme .NET
- ... zadovoljuje principe OOP (enkapsulacija, dedovanje, abstrakcija in polimorfizem)

C# ima obilico uporabnih funkcij in dobrih značilnosti, kot npr.

- Asinhrono programiranje z modifikatorjem *async* in operatorjem *await* (angl. *Asynchronous programming using* ***async*** *modificator &* ***await*** *operator*)
- LINQ/PLINQ izrazi (angl. ***LINQ/PLINQ*** expressions)
- Izrazi lambda (angl. ***lambda*** expressions)
- Bloki using (angl. ***using*** *blocks*)
- Delegatni in anonimni tipi (***delegate*** & ***anonymous*** types)
- Dinamične variable (angl. ***dynamic*** *variables*)

Razširljivi aplikacijski označevalni jezik XAML

XAML je deklarativni razširljivi aplikacijski označevalni jezik, ki ga je razvilo podjetje Microsoft. XAML je baziran na jeziku XML in je bil razvit z namenom olajšati razvijalcem zasnovo uporabniških grafičnih vmesnikov, saj je deklarativna gradnja le-teh veliko lažja kot programabilna. Poudariti je treba, da je izvorna koda XAML (datoteke XAML imajo končnico *.xaml*) sama po sebi v teoriji dovoljena in podprta, vendar v praksi obstaja zelo redko, saj je ponavadi vedno v paru z izvorno kodo C# (datoteke C# imajo končnico *.cs*). XAML se torej uporablja za oblikovanje in gradnjo grafičnih vmesnikov in s tem uporabo barv, pisav, postavitve grafičnih elementov oziroma kontrol, deklaracijo odzivnih metod (angl. Event Handlers), povezovanje podatkov (angl. Data Binding), definicijo stilov, animacije, ipd.

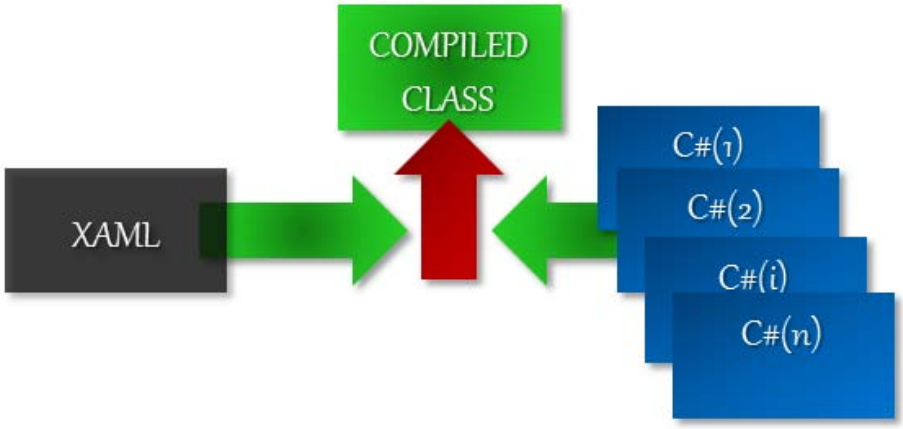
XAML torej poskrbi za kreiranje instanc razredov grafičnih elementov in manipulacijo njihovih lastnosti in atributov. Prednost kreiranja grafičnih vmesnikov z jezikom XAML napram C# se kaže tudi v tem, da lahko tako dosežemo identičen rezultat z veliko manj vrstic izvorne kode, boljšim pregledom, lažjo predstavitvijo, vstavitev ter gnezdenje grafičnih elementov, hitrejšo obravnavo napak (Visual Studio Designer nam takoj javi morebitne napake, kar se pri C# kodi lahko pokaže šele v času prevajanja kode ali izvajanja aplikacije) ter še marsikaj drugega.

Po drugi strani izvorna datoteka XAML predstavlja stran aplikacije, analogija temu je stran HTML. Identično velja tudi pri navigaciji z ene strani na drugo. Kakor lahko navigiramo med HTML stranmi, lahko tudi med XAML stranmi. Koncept navigacije med stranmi je pomemben pri aplikacijah Windows Store, zato ga bomo podrobneje opisali v posebnem poglavju. Naj povemo še to, da XAML v kombinaciji z C# kodo uporablja koncept parcialnih razredov, ki ga bomo predstavili v sekciji, ki sledi.

4.2 Parcialni razredi in princip *code-behind*

Konstrukta parcialni razredi (angl. **partial classes**) in princip **code-behind** sta neločljiva v smislu, da princip definira ločitev izvorne C# in deklarativne XAML kode v času implementacije skozi dve ali več izvornih datotek. Po drugi strani konstrukt parcialnih razredov definira, kako označiti razrede v C# in XAML, da bo za njih veljalo, da jih bo prevajalnik obravnaval kot parcialne. To je klasičen način, kako razdeliti izvorno kodo nekega razreda na deklarativni in programski del, tj. deklarativno kodo za gradnjo grafičnega vmesnika deklarativno (XAML) ter preostalo kodo, kot npr. odzivne metode, poslovno logiko, itd. (C#). Za parcialne razrede velja, da je poljubni C# razred deklariran kot parcialni, če

mu pri deklaraciji dodamo ključno besedo **partial**. Hkrati mora veljati, da morajo vse deklaracije istega razreda vsebovati isto ključno besedo, če je razred porazdeljen na več izvornih datotek. Datoteka XAML je lahko le ena, medtem ko je C# datotek lahko več, kot je razvidno iz slike 15. Parcialni razred lahko obstaja v tudi eni sami izvorni datoteki. V času prevajanja, C# prevajalnik poišče vse deklaracije parcialnih razredov in jih združi po razredih v eno celoto.



Slika 15: Primer parcialnega razreda

4.3 Asinhrono programiranje

Eden od ključnih konceptov programiranja aplikacij Windows Store je asinhrono programiranje, ki omogoča večjo fluidnost in boljšo odzivnost aplikacije. Asinhrono programiranje je bilo seveda mogoče že v prejšnjih verzijah platforme .NET po vzorcih *Event-based Asynchronous Pattern (EAP)* in *Asynchronous Programming Model (APM)*, vendar je zdaj vpeljan nov vzorec asinhronega programiranja - **Task-based Asynchronous Pattern (TAP)** [21], ki omogoča poenostavljeno programiranje in hitrejši razvoj aplikacij ter se posebej izkaže kot zelo zanesljiv pri razvoju kompleksnejših grafičnih vmesnikih. Prav za ta namen .NET vsebuje obsežno množico asinhronih metod, ki dopolnjujejo njihove sinhrone različice. Na splošno velja, da lahko slabo zasnovan uporabniški vmesnik obstane v čakanju na končanje sinhrone operacije (npr. pridobivanje velike količine podatkov prek spleta ali katerakoli dolgo trajajoča metoda) ter s tem blokira glavno nit uporabniškega vmesnika, kar pomeni blokado same aplikacije. Glavna nit naj bi bila čim manj obremenjena s procesiranjem vhodnih podatkov, opravljanja vzhodno izhodnih operacij, pridobivanja podatkov prek spleta, ipd. Zatorej smo tudi v prototipih uporabljali **TAP** asinhrone metode namesto njihovih sinhronih različic vedno, ko je to bilo smiselno.

Asinhrone metode z modifikatorjem *async* in operatorjem *await*

Metode, ki imajo v svoji signaturi dodan modifikator **async** so deklarirane kot asinhrone metode. Poleg le-teh so asinhrone lahko tudi **anonimne metode** (angl. *Anonymous Methods*) in **lambda izrazi** (angl. *Lambda Expressions*). Povratni tip pri asinhronih metodah je lahko **Task**, **Task<TResult>** ali **void**. Generični tip Task<TResult> implementira vmesnik IAsyncResult ter tako predstavlja asinhrono operacijo, ki lahko vrača neko vrednost. Neformalno je Task obljuba (angl. *Promise*), da bo klicatelj dobil rezultat (v primeru, da je bil rezultat zahtevan) enkrat v prihodnosti.

```
private async Task GetDataAsync()
{
    if (this._volturiCueCasesDataGroup.Count != 0)
        return;

    var uri = new Uri(Constant.JSON_DATA_LOCAL_URI);
    var file = await StorageFile.GetFileFromApplicationUriAsync(uri);
    var jsonText = await FileIO.ReadTextAsync(file);
    var jsonObject = JsonObject.Parse(jsonText);
    var jsonArray = jsonObject["Groups"].GetArray();

    foreach (JsonValue groupValue in jsonArray)
    {
        var groupObject = groupValue.GetObject();
        var groupUniqueId = groupObject["UniqueId"].GetString();
        if (groupUniqueId.Equals(Constant.FAVORITE_GROUP_UNIQUE_ID))
```

Slika 16: Primer asinhrone pridobitve podatkov z *async/await*

Asinhrone metode, ki nimajo vsaj enega *await* operatorja, se bodo izvedle sinhrono na glavni niti. V telesu asinhrone metode moramo vsaj enkrat uporabiti *await* operator, saj le tako omogočimo asinhrono izvedbo operacije. V primeru, da ima metoda dodan modifikator *async* in uporablja operator *await*, se bo še vedno izvajala sinhrono na glavni niti, dokler izvajanje ne pride do operatorja *await*, ko se bo začelo asinhrono izvajanje zahtevanega stavka. Na sliki 16 vidimo asinhrono metodo, ki pridobi podatke o iskanem objektu *VolturiCueCasesDataItem*, kot argument pa sprejme njegov identifikator. Metoda najprej asinhrono odpre in prebere JSON datoteko ter vsebino priredi spremenljivki, ki je tipa *JsonObject*, nakar vrne kazalec JSON objekta na matriko grup. Končno v zanki vse prebrane elemente shranimo v *VolturiCueCasesDataGroup* objekt.

4.4 Aplikacijski model

O stanjih in prehodih med stanji aplikacije Windows Store smo govorili že v poglavju Stanja in prehodi med stanji aplikacij Windows Store. V tem poglavju obravnavamo specifike aplikacijskega modela, vstopni razred in vstopno točko aplikacije. Aplikacija definira svoj vstopni razred v že omenjeni **package.appxmanifest** datoteki in sicer se v sekciji *Application* nastavi vrednost ključa *Entry point* na razred, ki se bo najprej naložil ob zagonu aplikacije. V C# prototipu smo to vrednost postavili privzeto **CSharp.App**, kjer je CSharp imenski prostor, App pa razred tipa *Windows.Ui.Xaml.Application* in predstavlja tip, ki zajame aplikacijo in servise aplikacije. Privzeti App konstruktor inicializira razred App kot **singleton** tj. hkrati lahko obstaja samo ena instanca tega razreda. Konstruktor je v tem primeru logični ekvivalent metod *main()* ali *WinMain()* [25]. Prva metoda, ki se mora izvesti je *InitializeComponent()*, ki poskrbi za inicializacijo instance, v kateri se izvaja, nato se registrira odzivna metoda *OnSuspending* kot je prikazano na sliki 17.

```
sealed partial class App : Application
{
    public static LiveConnectSessionStatus LiveStatus = LiveConnectSessionStatus.Unknown;
    public static Type HomePageType = typeof(Pages.MainPage);
    public static VolturiCueCasesDataGroup FavoriteCasesGroup;

    /// <summary>
    /// Initializes the singleton Application object.  This is the first line of authored
    /// code executed, and as such is the logical equivalent of main() or WinMain().
    /// </summary>
    public App()
    {
        this.InitializeComponent();
        this.Suspending += OnSuspending;
    }
```

Slika 17: Metoda *InitializeComponent* in registracija odzivne metode *OnSuspending*

Takoj za konstruktorjem se izvede asinhrona metoda **OnLaunched** - ki je ena od vstopnih točk aplikacije - v primeru, da je bila aplikacija zagnana s strani končnega uporabnika. Lahko se izvede tudi katera od drugih vstopnih točk kot npr. *OnFileActivated*, *OnSearchActivated*, *OnShareTargetActivated*, itd. Vse te metode so torej vstopne točke aplikacije in so definirane v razredu *Application*, kdaj se bo katera izvedla pa je odvisno od prožilne akcije, npr. če uporabnik poišče aplikacijo z uporabo kombinacije tipk *Windows+Q* in zažene najdeno aplikacijo, se bo takoj za konstruktorjem izvedla metoda *OnSearchActivated* in ne *OnLaunched*. Vstopna točka mora nato poskrbeti za inicializacijo ti. vizualnega korena aplikacijskega okna (*Frame*), potem poskrbi za nastavitev privzetega jezika aplikacije ter končno za registracijo odzivnih metod za navigacijo. Nato navigiramo ali do glavne ali do zadnje obiskane strani.

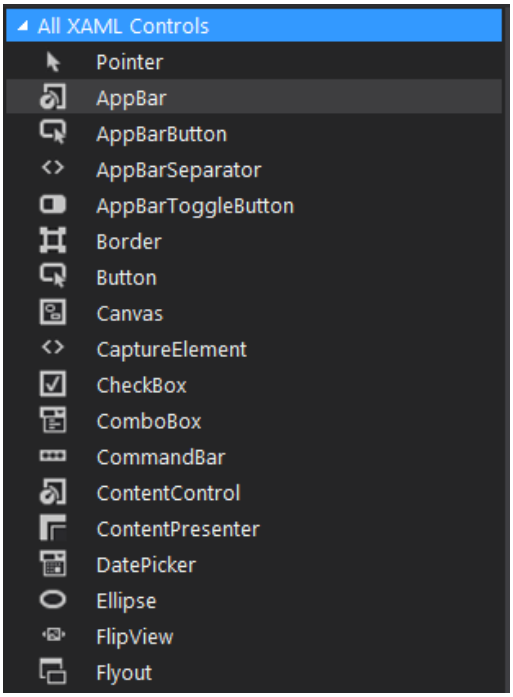
4.5 Navigacija med stranmi

Navigacija med stranmi je pomemben del pri programiranju Windows Store aplikacij. To lahko vidimo tudi po tem, da skoraj vse Visual Studio projektne predloge za Windows Store že vključujejo implementiran razred *NavigationHelper*. Naloga le-tega je, da zagotovi navigacijske metode, kot npr. *GoBack()* (pojdi nazaj) ali *GoForward()* (pojdi naprej), ki omogočata navigacijo na stran naprej ali nazaj, če je to mogoče. Strani, ki smo jih že obiskali se pomnijo v posebnih spremenljivkah *BackStack* in *ForwardStack*, ki sta tipa *IList<PageStackEntry>*. Navigacija je zelo preprosto implementirana - poljubna stran aplikacije, ki je instanca razreda *Page*, npr. *HomePage*, instancira razred NavigationHelper tako, da v konstruktorju referencira samo sebe tj. objekt *this*, kjer se mapira ti. *application view state* in se registrirajo metode za obravnavo dogodkov, ko se stran naloži in odstrani (*Page.Loaded*, *Page.Unloaded*). CSharp prototip privzeto uporablja razred **NavigationHelper**, kateremu smo dodali nekaj svojih metod, recimo *GoBackOrGoHome()*, ki v primeru, da navigacija nazaj ni mogoča, navigira na prvotno stran aplikacije. Strani, ki so definirane v prototipnem projektu *CSharp* pa so naslednje:

- **MainPage.xaml** - vstopna ali glavna stran aplikacije, ki združuje vse sekcije
- **DesignCasePage.xaml** - stran omogoča uporabniku urejevanje naročila za biljard torbe, shranjevanje naročila v lokalno ali SkyDrive datoteko in oddajo naročila
- **FavoritesPage.xaml** - stran je inicialno prazna, sicer pa služi za prikaz priljubljenih torbic
- **FeaturedPage.xaml** - stran prikazuje različne biljard torbe, njihovo izbiro ter prikaz detajlov ter dodajanje torbic med priljubljene
- **ItemDetailPage.xaml** - stran prikazuje vse detajle o izbrani torbici, kot npr. pregled vseh slik torbice v večji velikosti, podatke o imenu in modelu torbice, možnosti dodajanja torbice med priljubljene, itd.
- **PresentationalClipPage.xaml** - stran vsebuje video prezentacijo o *Volturi Custom Cases*, oddajo komentarjev, itd.

4.6 Kontrole

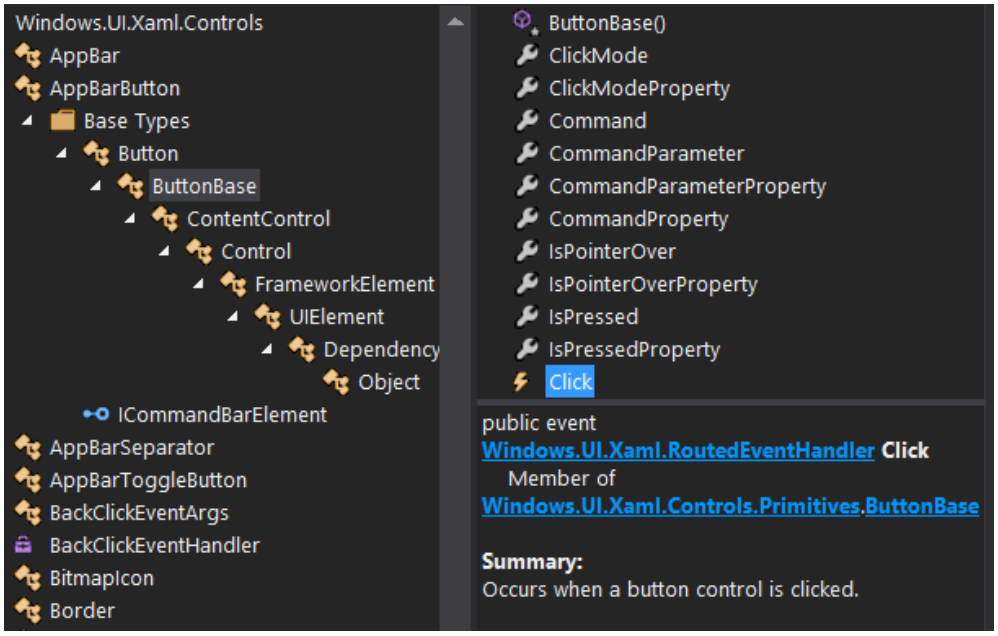
XAML ponuja kopico vgrajenih grafičnih kontrol (angl. *embedded controls*) za gradnjo uporabniškega vmesnika, razvijalci pa lahko razvijajo tudi kontrole po meri (angl. *custom controls*). Nekatere od vgrajenih XAML kontrol [2] prikazuje slika 18.



Slika 18: Vgrajene XAML kontrole

Za XAML kontrolo štejemo vsak razred, ki je definiran v imenskem prostoru **Windows.UI.Xaml.Controls** in hkrati implicitno implementira razred **DependencyObject**. Imenski prostor poleg kontrol definira še razrede, ki definirajo tip vhodnih argumentov (variante razreda **EventArgs**) ter delegate, ki predstavljajo odzivne metode (variante razreda **EventHandler**) za odziv na dogodke, katere bomo spoznali v naslednjem podpoglavju.

Primer strukture dedovanja kontrole, ki jo v prototipu uporabljamo v aplikacijski vrstici (*ApplicationBar*) na glavni strani, prikazuje slika 19. To je *AppBarButton* kontrola, katere osnovni razred je *Button*, ki pa deduje od razreda *ButtonBase*. Lahko vidimo, da razred AppBarButton eksplicitno ne definira dogodka *Click*, temveč je ta definiran v razredu ButtonBase, ki predstavlja osnovni (angl. *base*) razred za vse kontrole tipa *gumb*, npr. AppBarButton, Button, RepeatButton, HyperlinkButton, ipd.



Slika 19: Struktura dedovanja kontrole *AppBarButton*

Pomembno je omeniti, da lahko kontrole vsebujejo tudi druge kontrole ter da lahko vsaki kontroli priredimo tudi svoj stil (angl. *style*), na splošno pa jih lahko razdelimo na **Content**, **Items**, **Text**, **Images**, **Audio, Video & Speech** in **Other** [2]. V prototipni rešitvi smo uporabili kopico kontrol, med katerimi so nepogrešljive *Grid*, *Hub*, *StackPanel*, *Image*, *TextBlock*, *TextBox*, *CommandBar*, *AppBar*, *AppBarButton*, *ComboBox*, ipd.

Podrobnosti o stiliranju kontrol in povezovanju podatkov med njimi ter načinu gradnje uporabniškega vmesnika z gnezdenjem kontrol so opisani v poglavjih, ki sledijo.

4.7 Stili

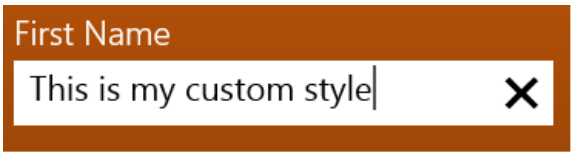
Stili (angl. *Styles*) so preproste entitete [3], ki jih lahko apliciramo na XAML kontrole in s tem deklarativno opišemo izgled posamezne kontrole. Konkretno so stili prezentirani z razredom *Windows.UI.Xaml.Style* in grupirajo lastnosti ciljne kontrole (angl. *target type*). XAML stili so konkretna analogija stilov **CSS**, saj z njimi stiliramo skupino grafičnih elementov brez ponavljanja deklarativne kode na vsaki posamezni kontroli. Seveda se stil določene kontrole lahko spreminja tudi programabilno, vendar se to v praksi uporablja redkeje. V prototipu smo definirali nekaj stilov, ki smo jih aplicirali na različne kontrole.

Za zgled si poglejmo stil *DesignCaseTextBox*, katerega cilja kontrola je *TextBox*. Stil smo definirali v datoteki **App.xaml** v sekciji *Application.Resources*, kot je to prikazano na sliki 20.

```
<Application.Resources>
    <Style x:Name="DesignCaseTextBox" TargetType="TextBox">
        <Setter Property="Height" Value="80"/>
        <Setter Property="Width" Value="350"/>
        <Setter Property="Margin" Value="0"/>
        <Setter Property="BorderThickness" Value="0"/>
        <Setter Property="FontSize" Value="22"/>
        <Setter Property="FontWeight" Value="Normal"/>
        <Setter Property="HorizontalAlignment" Value="Left"/>
        <Setter Property="VerticalAlignment" Value="Stretch"/>
    </Style>
```

Slika 20: Primer definicije stila za kontrolo *TextBox*

Da bi dejansko aplicirali stil na določeni kontroli, moramo v XAML datoteki ciljni kontroli postaviti vrednost lastnosti *Style* na stil, s katerim želimo kontrolo stilirati, kot to prikazuje slika 21.

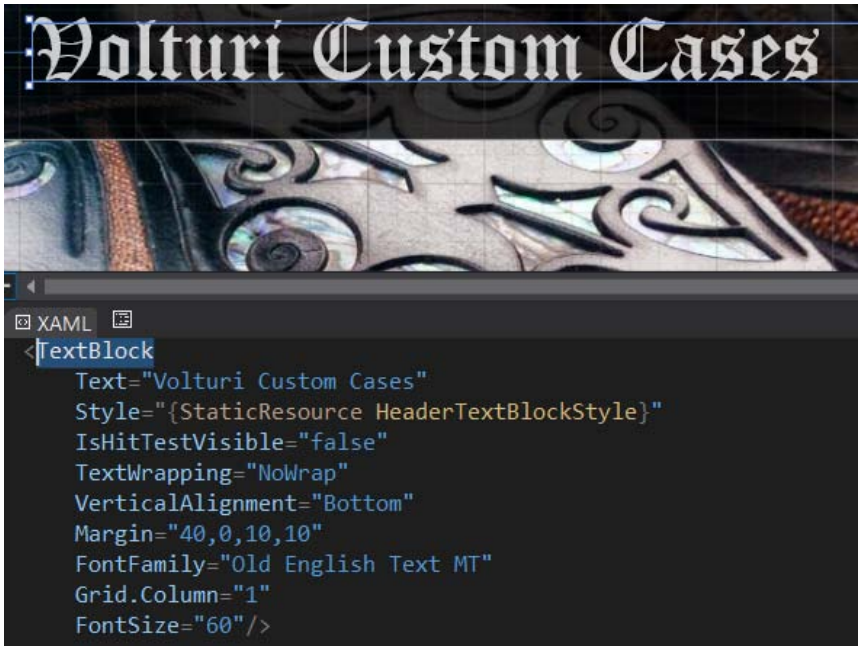



```
<TextBox x:Name="tbCustomerInfoFirstName"
         Style="{ThemeResource DesignCaseTextBox}">
    <TextBox.HeaderTemplate>
        <DataTemplate>
            <TextBlock Text="First Name"/>
        </DataTemplate>
    </TextBox.HeaderTemplate>
</TextBox>
```

Slika 21: Stiliranje kontrole *TextBox*

Na ta način se izognemo večkratnemu popravljanju vrednosti lastnosti za kontrole, katere želimo identično stilirati. V našem primeru so to preostale kontrole za vpis podatkov o stranki in sicer *Last Name*, *E-mail*, *Repeat E-mail*, *Street*, *City*, *Post Code* in *Country*. Za vse te kontrole torej zadostuje sprememba definicije stila in v *Visual Studio Designer* orodju bomo spremembe stila vidne takoj na vseh omenjenih kontrolah.

Stil, ki smo ga aplicirali uporabniški kontroli *TextBox* je sicer tipa **ThemeResource**, poznamo pa tudi stil tipa **StaticResource** npr. *BaseTextBlockStyle*, ki predstavljajo statične vire za stile in so del predefiniranih XAML stilov in jih razvijalec ne more spreminjati, lahko pa jih uporablja in kombinira lastnosti XAML kontrole z dodajanjem dodatnih lastnosti, kot se to vidi na sliki 22, kjer smo s stilom *BaseTextBlockStyle* stilirali naziv aplikacije na glavni strani. Ta stil definirati lastnosti *FontSize*, *FontWeight* in *FontLine*. Problem se lahko pojavi, ker smo v deklaraciji elementa *TextBlock* najprej navedli stil in nato ostale lastnosti, od katerih je *FontSize* identična lastnost, ki jo definira že sam stil. V tem primeru sicer ne bo prišlo do napake, saj bo stavek *FontSize="60"* prepisal vrednost iz stila tj. "56", zato je potrebno paziti pri uporabi stilov, kajti takšni prepisi vrednosti se ponavadi težje odkrijejo.



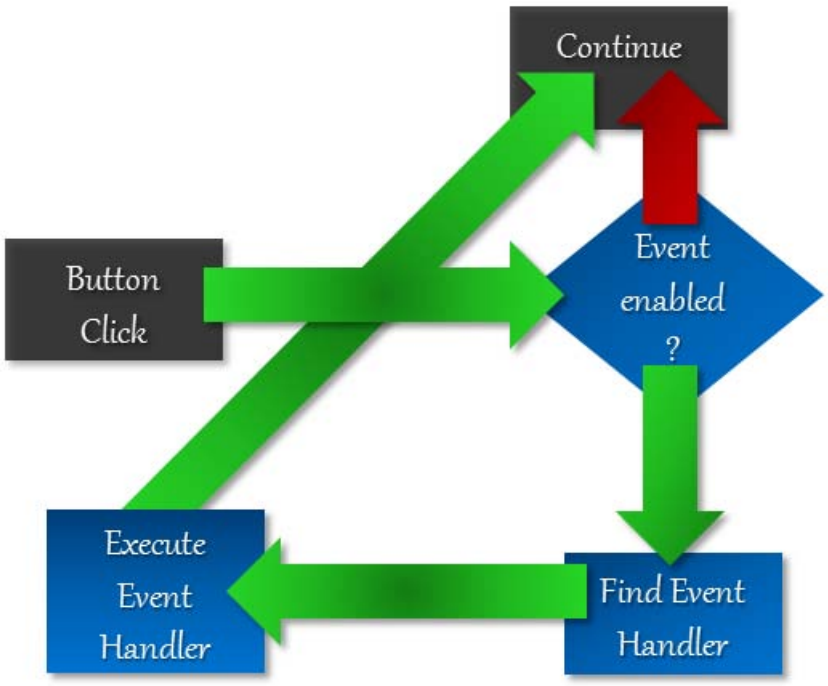
Slika 22: Stiliran naziv aplikacije na glavni strani

4.8 Obravnava dogodkov

Programski koncept **dogodkov** ali **obravnave dogodkov** (angl. *events*, *event handling* [11]) je znan že od samih začetkov programiranja v ogrodju .NET in je nepogrešljiv pri programiranju aplikacij za Windows Store. Pod dogodke smatramo interakcijo uporabnika z aplikacijo (npr. *DoubleTapped*, *DragEnter*, *Drop*, *KeyUp*, itd.) in dogodke,

ki so posledica neke akcije (npr. *Load*, *Unload*, *SizeChanged*, itd.). Nekatere dogodke lahko za določeno kontrolo tudi onemogočimo. To naredimo preprosto tako, da v XAML deklaracijo kontrole, za katero želimo onemogočiti določeni dogodek, postavimo lastnost za ta dogodek na vrednost *False*. Če npr. želimo onemogočiti dogodek *DoubleTapped* za kontrolo *TextBlock*, potem v njeno XAML deklaracijo dodamo zapis *IsDoubleTappedEnabled="False"*. To pomeni, da kljub temu da je odzivna metoda (**event handler**) za dogodek morda deklarirana v *code-behind* datoteki pripadajoče XAML datoteke, in se dogodek zgodi, se odzivna metoda npr. *TextBlock_DoubleTapped* ne bo izvedla.

Primer prožitve in obravnave dogodka za kontrolo *Button* nam ponazarja slika 23.



Slika 23: Primer prožitve in obravnave dogodka

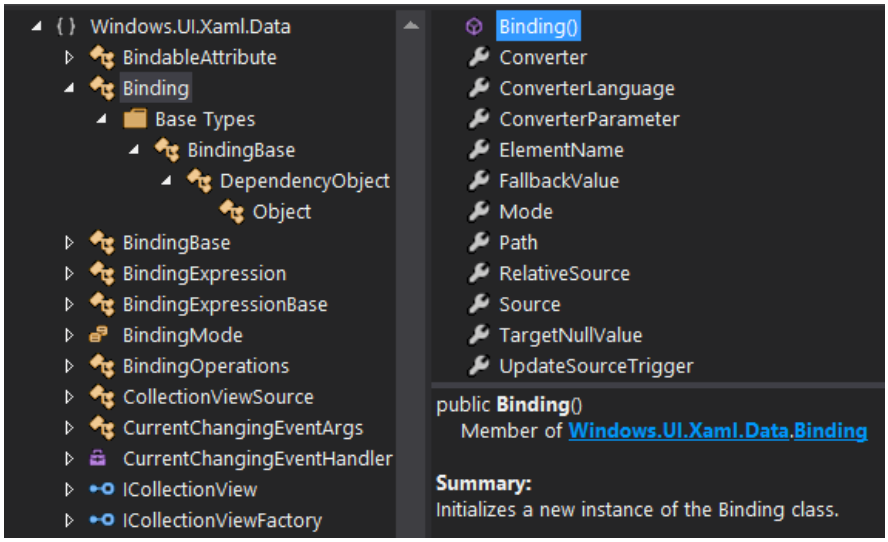
V prototipu uporabljamo storitev *SkyDrive*, vendar se mora uporabnik predhodno prijaviti v svoj račun na *Windows Live*. Postopek prijave je precej preprost - uporabnik v aplikacijski vrstici klikne na gumb *SkyDrive* in ob prijavi se izvede odzivna metoda *AbbSkyDriveClick* za dogodek *Click*, kot kaže slika 24. Kot lahko vidimo, koda najprej preveri vrednost javne statične spremenljivke *LiveStatus*, če je uporabnik že uspešno prijavljen in če je, ne naredi nič. V nasprotnem primeru pa spremenljivki priredi vrednost *Connected*. LiveStatus je deklarirana v glavnem aplikacijskem razredu *App*, s čimer vsem modulom aplikacije zagotovimo globalni dostop do statusa prijave uporabnika.

```
private async void AbbSkyDriveClick(object sender, RoutedEventArgs e)
{
    try
    {
        if (App.LiveStatus != LiveConnectSessionStatus.Connected)
        {
            var auth = new LiveAuthClient();
            var loginResult = await auth.LoginAsync(new string[] { "wl.basic" });
            if (loginResult.Status == LiveConnectSessionStatus.Connected)
            {
                App.LiveStatus = LiveConnectSessionStatus.Connected;
                tbSkyDriveStatus.Text = "Signed in.";
            }
        }
    }
    catch (LiveAuthException exception)
    {
        tbSkyDriveStatus.Text = "Error signing in: " + exception.Message;
    }
}
```

Slika 24: Odzivna metoda za prijavo na *Windows Live*

4.9 Povezovanje podatkov

Poleg oblikovanja uporabniškega grafičnega vmesnika je povezovanje podatkov verjetno eden najpomembnejših aspektov razvoja aplikacij Windows Store, saj so le-te, kot smo že omenili, ponavadi podatkovno vodene. Pri povezovanju podatkov mislimo na kontrole in kako podatke povezati z le-temi. Glavni cilj vezave podatkov je deklarativno povezati podatke določeni kontroli. V primeru trivialnih kontrol, kot npr. *TextBox*, je zadeva enostavna, saj ta kontrola definira eno samo lastnost tj. *Text* za manipulacijo s podatki. Problem se pojavi v primeru kontrol, kot npr. *ComboBox*, ki dedujejo od razreda *ItemsControl*, ki predstavlja kontrolo z množico elementov, od katerih ima vsak element lastnost manipulacije podatkov. V primeru kontrole *ComboBox* je to množica elementov *ComboBoxItem*, ki imajo lastnost *Content*, kamor lahko zapišemo podatek tipa *niz* (angl. *String*). Za ta namen nam XAML omogoča deklarativno vezavo podatkov in sicer z uporabo lastnosti **Binding**, ki jo moramo dodati deklaraciji kontrole. Lastnost je tipa *Binding* in lastnosti tega razreda so prikazane na sliki 25. Najbolj pogosto uporabljana lastnost razreda je **Path**, ki predstavlja pot do izvora podatkov.



Slika 25: Razred *Binding* in njegove lastnosti

V poglavju Asinhrono programiranje smo pokazali, kako pridobimo podatke za vse biljard torbice iz lokalne JSON datoteke. Pokazali bomo, kako smo vezali podatke s kontrolami na strani *FeaturedPage*, ki predstavlja galerijo slik biljard torbic. Ko podatke inicialno pridobimo z asinhrono metodo **GetGroupsAsync**, jih moramo povezati še s kontrolami na XAML strani *FeaturedPage*, ki je glavna predstavitvena stran aplikacije in prikazuje vse torbice kot listo elementov. To naredimo tako, kot kaže slika 26. Torej, v XAML datoteki deklarativno priredimo spremenljivki **DataContext** vrednost **{Binding DefaultViewModel, RelativeSource={RelativeSource Self}}**.

```
<Page
    x:Name="pageFeatured"
    x:Class="CSharp.Pages.FeaturedPage"
    DataContext="{
        Binding DefaultViewModel,
        RelativeSource={RelativeSource Self}
    }"
```

Slika 26: Prireditev podatkovnega modela lastnosti *DataContext* strani *FeaturedPage*

To pomeni, da vežemo podatkovni kontekst *DataContext* XAML strani s spremenljivko *DefaultViewModel*. Le-ta je tipa *ObservableDictionary*, ki je implementacija vmesnika *IObservableMap* in služi kot mapa za shranjevanje ključev in njihovih vrednosti. Lastnost *Path* razreda *Binding* je pozicijski parameter in ga ni potrebno eksplicitno specificirati, npr. namesto **{Binding Path=ImagePath}** lahko preprosto napišemo **{Binding ImagePath}**, kot smo prikazali na sliki 27. Tu smo naredili večkratno vezavo podatkov in

sicer smo na objekt *Image* vezali lastnost *Source* na *ImagePath* iz izvora podatkov in lastnost *Name* na *Title*.

```
<GridView.ItemTemplate>
    <DataTemplate>
        <Grid x:Name="gridFeatured" HorizontalAlignment="Left" Width="340" Height="255">
            <Border Background="{ThemeResource ListViewItemPlaceholderBackgroundThemeBrush}">
                <Image Source="{Binding ImagePath}"
                       Stretch="UniformToFill"
                       AutomationProperties.Name="{Binding Title}"/>
            </Border>
            <StackPanel x:Name="stackPanelFeatured" RightTapped="FeaturedRightTapped" Vertical
                <TextBlock Text="{Binding Title}" Foreground="{ThemeResource ListViewItemOverl
                <TextBlock Text="{Binding Subtitle}" Foreground="{ThemeResource ListViewItemOv
            </StackPanel>
        </Grid>
    </DataTemplate>
</GridView.ItemTemplate>
```

Slika 27: Primer povezovanja podatkov na več lastnosti objekta

Da bo vezava podatkov resnično delovala, moramo opraviti še nekaj korakov. Najprej moramo v *code-behind* C# datoteki v konstruktorju razreda *FeaturedPage* registrirati metodo, ki se bo odzvala na dogodek *LoadState* in bo ob dogodku *LoadState* populirala stran z vsebino, ki jo vežemo. Metoda je prikazana na sliki 28 in se kliče asinhrono.

```
private async void navigationHelperLoadState(object sender, LoadStateEventArgs e)
{
    object navigationParameter;
    if (e.PageState != null && e.PageState.ContainsKey("SelectedItem"))
    {
        navigationParameter = e.PageState["SelectedItem"];
    }

    var dataGroups = await VolturiCueCasesDataSource.GetGroupsAsync();
    this.DefaultViewModel["Groups"] = dataGroups;
}
```

Slika 28: Metoda *LoadState* strani *FeaturedPage*

Na analogen način smo naredili vezavo podatkov na XAML strani *ItemDetailPage*. Ko v galeriji kliknemo na izbrano sliko torbice, se zgodi navigacija na stran o detajlih izbranega elementa. Ob klicu se prenese enolični identifikator torbice (*UniqueId*) in ob nalaganju strani *ItemDetailPage* se najprej izvede konstruktor strani, kjer se registrira odzivna metoda *navigationHelperLoadState*, vendar se takoj za konstruktorjem najprej

izvede metoda *OnNavigatedTo*, ki prebere identifikator izbrane torbice in si ga zapomni. Nato se izvede še odzivna metoda *navigationHelperLoadState*, ki ključ torbice pridobi iz parametra *LoadStateEventArgs*, kot lahko razberemo iz slike 29.

```
private async void navigationHelperLoadState(object sender, LoadStateEventArgs e)
{
    var item = await VolturiCueCasesDataSource.
        GetItemAsync((String)e.NavigationParameter);
    this.DefaultViewModel["Item"] = item;
}
```

Slika 29: Metoda *LoadStateItem* strani *ItemDetailPage*

Metoda *navigationHelperLoadState* najprej pokliče asinhrono metodo *GetItemAsync* (slika 30) in nato poiskani element priredi podatkovnemu modelu strani. Analogno prejšnjemu primeru naredimo podatkovno vezavo na XAML strani *ItemDetailPage* in s tem je vezava podatkov zaključena.

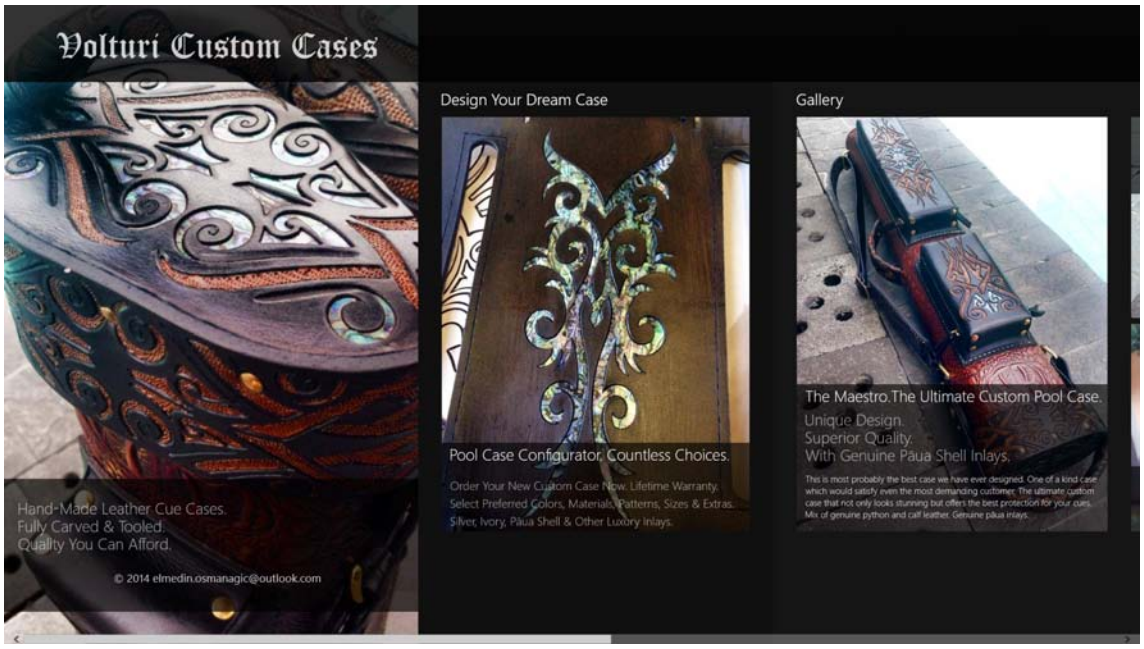
```
public static async Task<VolturiCueCasesDataItem> GetItemAsync(string uniqueId)
{
    try
    {
        if (string.IsNullOrEmpty(uniqueId))
        {
            throw new Exception("uniqueId cannot be empty.");
        }

        await _volturiCueCasesDataSource.GetDataAsync();
        return (_volturiCueCasesDataSource.Groups
            .SelectMany(group => group.Items)
            .Where((item) => item.UniqueId.Equals(uniqueId))).FirstOrDefault();
    }
    catch (Exception e)
    {
        Debug.WriteLine("GetItemAsync> exception: " + e);
    }

    return null;
}
```

Slika 30: Metoda *GetItemAsync* za asinhrono iskanje elementa

4.10 Oblikovanje grafičnega vmesnika

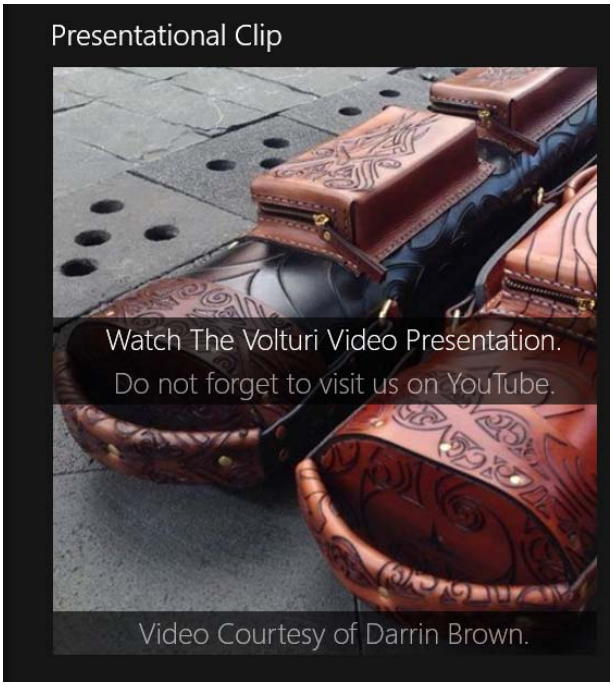
Pri razvoju grafičnega vmesnika smo uporabili nekatere skupne koncepte, na nekaterih pa smo aplicirali analogijo. Primer prvega je zasnova uporabniškega grafičnega vmesnika, saj je bil v obeh primerih cilj razviti navzven identičen grafični vmesnik. Uporabljeni sta bili projektni predlogi *Hub App (XAML) Visual C#* ter *Hub App JavaScript*, ki sta si med seboj neodvisni in katerih nadgradnja se temeljito razlikuje.

Slika 31 nam kaže grafični vmesnik izhodiščne tj. glavne strani aplikacije. Pri oblikovanju glavne strani in nasploh celotnega prototipa smo upoštevali *principe stilskega oblikovanja*, katere smo opisali v poglavju 2.



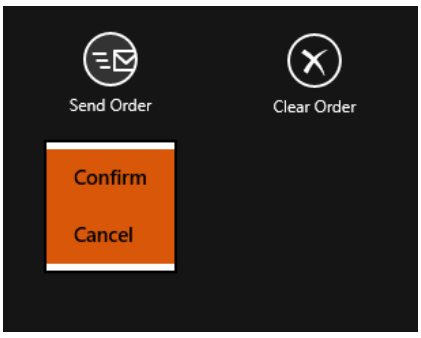
Slika 31: Izhodiščna stran aplikacije

Anatomija glavne strani, kot to prikazuje slika 31, vključuje glavno sliko, kratek opis proizvajalca *Volturi Custom Cases* in njegov moto, nato pa sledijo horizontalno postavljene sekcije (*Windows.UI.Xaml.Controls.HubSection* ali *WinJS.UI.HubSection*) *Design Your Dream Case*, *Gallery*, *My Favorite Cases* in *Presentational Clip* kot zadnja (slika 32).



Slika 32: Sekcija *Presentational Clip*

S klikom miške ali dotikom prsta na eno od sekcij se sproži obravnava dogodka (*Clicked* oz. *Tapped*), kar povzroči navigacijo na zahtevano stran. V primeru, da uporabnik zahteva npr. stran *Design Your Dream Case*, se odpre vnosna forma za vnos podatkov o naročilu biljard torbice (delni prikaz forme vidimo na sliki 34). Pod imeni sekcij smo postavili gumbe *Save to SkyDrive*, *Save*, *Send Order* in *Clear Order*. Vsak gumb ima opcijo potrditve (angl. *Confirm*) ali preklica akcije (angl. *Cancel*) akcije. Posebnost je v tem, da se na klik gumba prikaže posebna kontrola ti. **Flyout**, ki zahteva interakcijo uporabnika za nadaljevanje akcije, kot je razvidno iz slike 33.

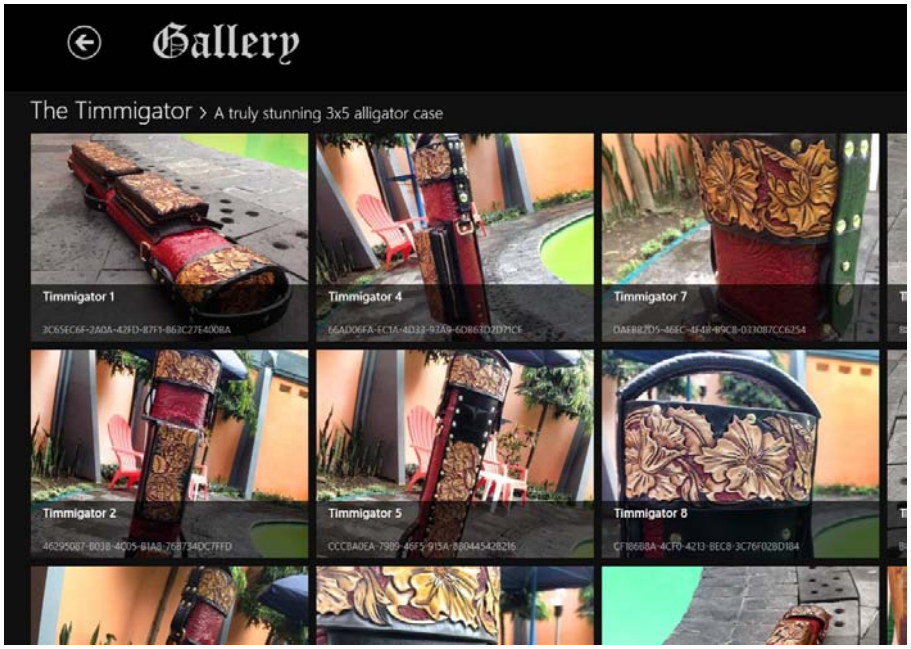


Slika 33: Opciji *Confirm* in *Cancel* pri *Flyout* kontroli gumba *Send Order*

Vnosna forma s slike 34 je zgrajena na osnovi tehnike *preklapljanja plošč*. Forma je sestavljena iz šestih sekcij (*Customer Info*, *Case Model*, itd.) od katerih ima vsaka svojo ploščo na desni strani, najprej se pa pokaže *Customer Info* plošča, medtem ko so ostale plošče skrite (XAML : *Visibility=Collapsed*, CSS : *visibility=hidden*). Če uporabnik preklopi na neko drugo sekcijo, postavimo vrednost atributa vidljivosti njene plošče na *vidno* ter vse ostale na *skrito*. Ta problem se da rešiti tudi z uporabo atributov *ZIndex* (XAML) in *z-index* (CSS) in sicer tako, da programabilno spreminjamo Z indeks vsem ploščam (Z indeks določa da bo (v primeru, ko se več elementov prekriva v vsaj eni točki) viden vedno tisti element, ki ima večji Z indeks od ostalih), vendar je takšna rešitev veliko manj elegantna. S tem se izognemo tudi problematičnemu računanju ti. *margin* vrednosti elementov, kajti vse koordinate elementov postavljamo relativno in ne absolutno glede na starševski element (CSS : *position=relative* in ne *position=absolute*).

Slika 34: Vnosna forma za naročilo biljard torbe

Pokažimo še eno podstran, ki se odpre, ko uporabnik odpre sekcijo *Gallery* (slika 35). V galeriji slik prikažemo vse podatke, ki smo jih predhodno prebrali iz *JSON* datoteke ter hkrati opravimo vezavo podatkov. Po definiciji hierarhičnega vzorca smo sedaj na drugi od treh strani tj. ko uporabnik izbere sekcijo.



Slika 35: Podstran *Gallery*

Če uporabnik želi videti detajle določenega elementa, s klikom ali dotikom na izbrani element povzroči navigacijo na tretjo, zadnjo stran hierarhičnega vzorca. Del strani lahko vidimo na sliki 36. Uporabnik lahko pregleda vse slike ali s klikom na posebne gumbe za pomik desno ali levo oz. to naredi s tipkami.

Slika 36: Podstran *Details*

5 Razvoj prototipa aplikacije Windows Store z uporabo tehnologij HTML, CSS in JavaScript

Ta del diplomske naloge je namenjen specifikam razvoja aplikacij Windows Store z uporabo tehnologij HTML, CSS in JavaScript. Tudi tu bomo najprej na kratko opisali tehnologije, ki smo jih uporabili pri razvoju prototipa, nato pa bomo predstavili posebnosti, ki so vezane za razvoj JavaScript aplikacij za Windows Store.

5.1 Tehnologije

Jezik za označevanje nadbesedila HTML

HTML je primarni jezik za izdelavo in urejanje spletnih strani. **HTML5** predstavlja peto revizijo jezika HTML in uvaja nekatere nove HTML elemente kot recimo <*audio*>, <*video*>, <*source*>, <*embed*>, <*keygen*>, itd. ter JavaScript API spremembe, kot npr. *Canvas 2D Context*, *Media Capture*, *Contacts*, *XMLHttpRequest2*, itd.

Kaskadne stilske podloge CSS

CSS je jezik kaskadnih podlog, ki se uporablja za stiliranje dokumentov, napisanih v jeziku za označevanje nadbesedila. CSS omogoča dobro delitev vsebine dokumenta in njegovo predstavitev. CSS se lahko uporablja na več načinov, npr. vgnezdeno v samem dokumentu, vključi se s pomočjo značke *link* ali pa se aplicira na HTML elementu. CSS specifikacije vzdržuje mednarodna organizacija **World Wide Web Consortium (W3C**), trenutno zadnja uradna variacija je *CSS3*, medtem ko za variacijo *CSS4* obstaja specificiranih že nekaj modulov (*Image Values*, *Backgrounds & Borders*, itd).

Programski jezik JavaScript

JavaScript je dinamičen skriptni programski jezik, ki se vglavnem uporablja za spletno programiranje na strani odjemalcev, kar omogoča interakcijo z uporabnikom, manipulacijo objektov DOM, asinhrone operacije, itd. JavaScript je tudi prototipni, multi paradigmatični in funkcionalni jezik ter je implementacija skriptnega jezike **ECMAScript**, ki je standardiziran pod oznako *ECMA-262* [10].

5.2 Asinhrono programiranje

Programiranje aplikacij Windows Store z uporabo jezika JavaScript na abstraktnem nivoju definira iste potrebe po asinhronem programiranju in akcije, ki naj bi se izvajale asinhrono, kot razvoj z uporabo jezika C#. WinJS zajema tudi implementacijo predloga JavaScript paradigme za asinhrono programiranje Common JS Promises/A [18]. Paradigma skozi ti. obljube ponuja vmesnik za interakcijo z objekti, ki predstavljajo rezultat asinhronih akcij, ki se ali pa tudi ne zaključijo v kateremkoli času v prihodnosti. Vmesnik s tem ponuja način, kako obvladovati obljube, kot produkte asinhronih akcij, na predvidljiv način. Med asinhrone akcije spadajo npr. prikaz raznih sporočil uporabniku, delo z datotekami, pošiljanje in prejemanje podatkov z interneta, interakcija s servisi in napravami, itd. Asinhrono programiranje se izvaja na način kot ga definira paradigma *Common JS Promises*, tj. da asinhrone metode vračajo objekt obljube (angl. *promise object*). Konkretno to pomeni, da je Microsoft implementiral Common JS Promises vmesnik skozi WinJS knjižnico. Na sliki 37 vidimo primer uporabe obljube. Z uporabo **WinJS.xhr** funkcije [22], ki zavije klic za *XMLHttpRequest* objekt v obljubo, najprej pridobimo podatke, nato (angl. *then*) jih z uporabo funkcije *JSON.parse* razčlenimo in jih v zanki *forEach* enega za drugim dodamo kontroli *WinJS.Binding.List*, ki jo potem lahko uporabimo.

```
var list = new WinJS.Binding.List();
var jsonUrl = "model/VolturiCueCasesData.json";

WinJS.xhr({ url: jsonUrl })
    .then(function (xhr) {
        var items = JSON.parse(xhr.responseText);

        items.forEach(function (item) {
            list.push(item);
        });
    });
```

Slika 37: Primer asinhronega klica funkcije *WinJS.xhr* z uporabo obljube

Common JS Promises/A kot WinJS.Promise

WinJS JavaScript objekt, ki definira obljubo in implementira paradigmo *Common JS Promises/A* [7] se nahaja v samem vrhu imenskega prostora WinJS in sicer kot razred **WinJS.Promise**. Obljubo lahko definiramo kot mehanizem, ki načrtuje nadaljne delo na

vrednosti, ki še ni bila izračunana oziroma prirejena spremenljivki. Z drugimi besedami je obljuba abstrakcija za upravljanje interakcij z asinhronim API. Obljuba je torej objekt, ki vrne vrednost v prihodnosti in ne takoj. Obljube zavijajo (angl. **wrap**) Windows Store Windows Runtime API metode in s tem pripomorejo k asinhronem programiranju aplikacije Windows Store. Obljube se lahko tudi verižijo, kar bomo pokazali na konkretnih primerih v naslednjih podpoglavjih.

Funkcije razreda WinJS.Promise

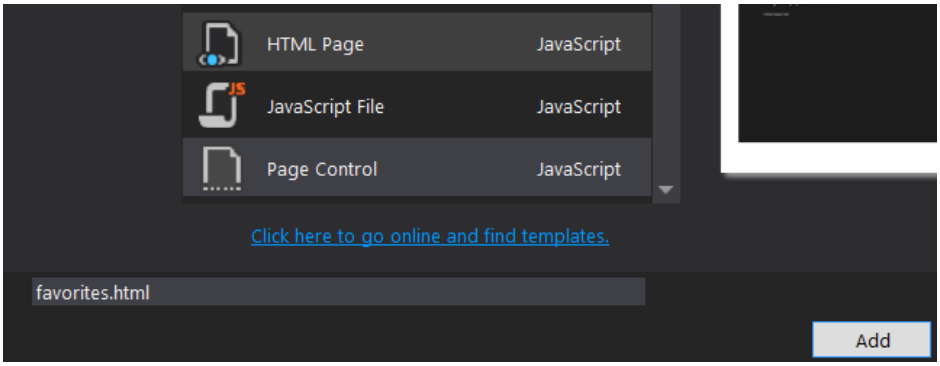
Razložili bomo nekatere nepogrešljive funkcije razreda WinJS ter primernost in nujnost njihove uporabe v implementaciji aplikacij Windows Store na asinhron način.

Razred WinJS.Promise definira naslednje funkcije [1]:

- **addEventListener** - Kontroli doda funkcijo, ki čaka na dogodek.
- **any** - Vrne obljubo, ki je izpolnjena, ko je ena od vhodnih obljub izpolnjena.
- **as** - Če je argument že obljuba, vrne klicatelju objekt obljube, če ni, potem vrne objekt, zavit v obljubo (angl. *wrapped object*).
- **cancel** - Poskuša preklicati zapolnitev obljubljene vrednosti. Če obljuba še ni bila izpolnjena in je preklic omogočen, potem obljuba zapade v stanje napake z vrednostjo *Error(Canceled)*.
- **is** - Ugotovi, če je objekt obljuba ali ne.
- **join** - Ustvari obljubo, ki je izpolnjena, ko so vse vrednosti izpolnjene.
- **removeEventListener** - Inverz funkcije *addEventListener*. Kontroli odstrani funkcijo, ki čaka na dogodek.
- **then** - Specificira, kaj vse je potrebno narediti, da se določena obljuba izpolni. Nepogrešljiva funkcija pri veriženju obljub.
- **thenEach** - Izvede operacijo na vseh vhodnih obljubah in vrne obljubo, ki vsebuje rezultate vseh vhodnih obljub.
- **timeout** - Definira maksimalen čas, v katerem se obljuba mora izpolniti. V nasprotnem primeru prekliče obljubo.
- **wrap** - Zavije vrednost, ki ni obljuba, v obljubo.
- **wrapError** - Podobno kot funkcija *wrap*, le da slednja zavije vrednost napake.

5.3 Modularne enote

V C# sekciji smo govorili o parcialnih razredih in principu *code-behind*, ki ločijo deklarativni od programskega dela. Princip code-behind v JavaScript varianti ne obstaja kot tak, zato WinJS API ponuja analogijo skozi kontrolo **WinJS.UI.Pages.PageControl**. Objekt **PageControl** predstavlja modularno enoto HTML, CSS in JavaScript datotek, na katero lahko navigiramo [23], v VS rešitvi pa ga kreiramo tako, da ga preprosto dodamo z uporabo VS funkcije *Add / New Item / PageControl* (slika 38).



Slika 38: Dodajanje objekta *PageControl*

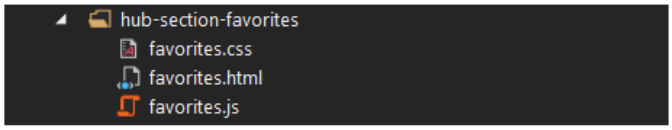
Drugače povedano *predstavlja PageControl HTML stran*, ki ima definirani CSS in JavaScript datoteki z istim imenom. Posebnost modularne enote *PageControl* je tudi v tem, da je edina WinJS kontrola, ki nima nobenih privzetih lastnosti, temveč samo tiste, ki jih definira razvijalec. Objekta te kontrole ne moremo instancirati niti z uporabo ključne besede **new**, ampak to lahko naredimo le z uporabo funkcije *WinJS.UI.Pages.define*, kateri podamo URI datoteke HTML, ki definira kontrolo PageControl in objekt, ki definira njene člane (slika 39).

```
var pageControl = WinJS.UI.Pages.define(uri, members);
```

Slika 39: Kreiranje instance objekta *PageControl*

Primer že dodanega objekta lahko vidimo na sliki 40, kjer v zgornjem delu vidimo JavaScript datoteko *favorites.js*, v spodnjem pa tudi pripadajoči HTML in CSS datoteki. Na analogen način dodamo tudi vse preostale strani, ki jih definiramo v projektu. Vidimo, da nam *favorites.js* že ponuja deklaracijo odzivnih funkcij *ready*, *unload* in *updateLayout*, v katere dodamo ustrezno kodo, ki se bo izvedla ob teh dogodkih.

```
(function () {
    "use strict";

    WinJS.UI.Pages.define("/pages/hub-section-favorites/favorites.html", {
        ready: function (element, options) {
        },

        unload: function () {
        },

        updateLayout: function (element) {
        }
    });
})();
```



Slika 40: PageControl *favorites* objekt

Omenimo še, da modularno enoto PageControl prikažemo tako, da pokličemo funkcijo **render**, ki ima naslednjo sintakso:

WinJS.UI.Pages.render(uri, element, options, parentedPromise).done(/ success and error handlers */);*

Parameter *uri* predstavlja URI vsebino, ki definira stran, *element* predstavlja **DOM** element, kateremu dodamo PageControl enoto, medtem ko sta parametra *options* in *parentedPromise* neobvezna parametra.

5.4 Aplikacijski model

Kot smo že omenili, Windows Store aplikacija definira vstopni razred ali točko v **package.appxmanifest** datoteki. V JavaScript prototipu je to HTML stran **default.html**, ki vključuje JavaScript skripto **default.js**, ki pokliče API funkcijo *WinJS.Application.start* ter tako poskrbi za začetek procesiranja dogodkov ob zagonu aplikacije. Registracijo odzivnih funkcij naredimo tako, da pokličemo WinJS.Application funkcijo *addEventListener* [1], kot argumenta pa podamo ime oziroma tip dogodka ter objekt, ki posluša na dogodek. Objekt je ponavadi anonimna funkcija s pripadajočo kodo, ki se bo v primeru dogodka izvedla ob notifikaciji o dogodku. Celotno izvajanje se dogaja tako, da ko uporabnik zažene Windows Store JavaScript aplikacijo, OS ustvari ti. zaboj (angl. *con-*

tainer) in gostiteljski proces aplikacije **WWAHost.exe**, ki naloži aplikacijo v lokalnem kontekstu (angl. *local context*). Proces WWAHost.exe ima isti ti. **rendering engine** in **JavaScript engine** kot *Internet Explorer 10*, pa vendar zaradi varnosti ne uporablja spletnega konteksta (angl. *web context*). Lokalni kontekst je v tem smislu limitiran glede na spletni kontekst in do modulov operacijskega sistema dostopa prek uporabe **WinJS** API funkcij. Kontekst izvajanja JavaScript aplikacij za Windows Store ponazarja slika 41.

Slika 41: Kontekst izvajanja JavaScript Windows Store aplikacij

5.5 Navigacija med stranmi

V JavaScript projektu smo definirali naslednje strani HTML:

- **default.html** - vstopna ali privzeta stran aplikacije, ki združuje vse sekcije
- **design.html** - stran omogoča uporabniku urejevanje naročila za biljard torbo ter shranjevanje le-tega v lokalni ali SkyDrive imenik
- **favorites.html** - stran je inicialno prazna, sicer pa služi za prikaz priljubljenih torbic
- **featured.html** - stran prikazuje različne biljard torbe, njihovo izbiro ter prikaz detajlov ter dodajanje torbic med priljubljene
- **item.html** - stran prikazuje vse detajle o izbrani torbici
- **clip.html** - stran vsebuje video prezentacijo o *Volturi Custom Cases*

Zgoraj omenjene strani so del modularnih enot *default*, *design*, *favorites*, *featured*, *item* in *clip*, kar pomeni da ima vsaka stran pripadajoči JavaScript in CSS datoteki. Vsaka modularna enota pa ima svojo mapo, kar v splošnem poenostavi projektno strukturo, še posebej, če je število modularnih enot veliko.

VS JavaScript projektni vzorci vsebujejo posebno JavaScript datoteko **navigator.js**, ki definira razred **PageControlNavigator** kot pomoč pri navigaciji. Posebna lastnost nagivatorja je, da preprečuje ti. *top-level* navigacijo. Predpostavimo, da imamo stran *A.html*, ki ima neko vsebino, kateri bomo rekli stran *C.html*. V primeru spletne navigacije bi s klikom na povezavo do strani *B.html* navigirali tako, da bi se zgodila *top-level* navigacija in bi pristali na strani *B.html* (npr. *Navigate to page B*). V primeru aplikacij Windows Store pa še vedno ostanemo na strani *A.html*, vendar se zdaj vsebina strani *B.html* naloži v stran *A.html* na mesto, kjer je prej bila vsebina strani *C.html*. Poleg omenjene navigacije PageControlNavigator ponuja tudi obravnavo dogodkov, ki se zgodijo ob navigaciji kot npr. *WinJS.Navigation.navigate* in *WinJS.Navigation.navigating* [1].

WinJS skozi imenski prostor *Navigation* razvijalcem ponuja tudi funkcionalnost osnovne navigacije med stranmi ter navigacijsko kopico, kot npr. lastnosti *canGoBack*, *history* in *location*, nato funkcije *addEventListener*, *navigate*, *forward*, itd. ter dogodke *onbeforenavigate*, *onnavigated* in *onnavigating*. Primer navigacije z uporabo funkcije **navigate** prikazuje slika 42.

```
sectionItemNavigate: util.markSupportedForProcessing(
    function (args) {
        var item = Data.getItemReference(sectionItems.getAt(args.detail.itemIndex));
        WinJS.Navigation.navigate("/pages/item/item.html", { item: item });
    }
)
```

Slika 42: Primer navigacije med stranmi

5.6 Kontrole

Za razvoj aplikacij Windows Store z uporabo HTML, CSS in JavaScript je na voljo precejšnje število UI kontrol, ki so implementirane v imenskem prostoru **WinJS.UI**. Nekatere pomembnejše in bolj uporabljane pa so **AppBar** (prikaz komand), **BackButton** (gumb za navigacijo nazaj), **DatePicker** (izbira datuma), **FlipView** (prikaz kolekcije elementov, največ enega naenkrat), **Flyout** (prikaz interaktivnega sporočila uporabniku), **ListView** (prikaz kolekcije elementov v mreži), **SearchBox** (omogoča uporabnikom razne poizvedbe), ipd. Kontrole imajo tudi svoje ti. člane (angl. **members**) in sicer kon-

struktorje (angl. **constructors**), dogodke (angl. **events**), metode (angl. **methods**) in lastnosti (angl. **properties**).

WinJS.UI kontrole lahko kreiramo programsko (JavaScript) ali deklarativno (HTML). JavaScript in HTML sintaksi kreiranja objekta *WinJS.UI.SearchBox* sta vidni na sliki 43. Poleg tega smo nastavili nekatere lastnosti v konstruktorju (JavaScript): queryText, chooseSuggestionOnEnter in focusOnKeyboardsInput. Iste lastnosti smo deklarativno nastavili HTML kontroli tako, da smo uporabili lastnost **data-win-options**, kjer smo nato vsaki lastnosti priredili njeno vrednost. Na analogen način kreiramo tudi druge kontrole in jim lahko nastavimo lastnosti, ki jih specificira WinJS API.

```
var searchBox = new WinJS.UI.SearchBox(
    element,
    {
        queryText: 'Elmedin Osmanagic',
        chooseSuggestionOnEnter: true,
        focusOnKeyboardInput : false
    });
```

```
<div data-win-control="WinJS.UI.SearchBox"
    data-win-options="{
       queryText: 'Elmedin Osmanagic',
       chooseSuggestionOnEnter: true,
       focusOnKeyboardInput : false
    }">
</div>
```

Slika 43: HTML in JavaScript sintaksi kontrole *WinJS.UI.SearchBox*

Slika 44 prikazuje način, kako smo se lotili gradnje UI kontrole *Hub*. V aplikaciji obstaja samo ena kontrola *Hub*, ki vsebuje več kontrol *HubSection*, katere predstavljajo posamezne logične module aplikacije (*Design Your Dream Case*, *Gallery*, *Favorites* in *Presentational Clip*). Vsak od modulov oz. sekcij vsebuje množico kontrol, od katerih so nekatere sestavljene (angl. *composite*), nekatere pa singularne (angl. *singular*). Kot lahko vidimo na sliki ima vsaka kontrola apliciran svoj *CSS stil* , npr. *class="main"*, *class="design"*, itd.

```
<div class="main"
     data-win-control="WinJS.UI.Hub"
     data-win-options="{ isHeaderStatic: true }">
    <div class="design" data-win-control="WinJS.UI.HubSection">
        <div class="search" data-win-control="WinJS.UI.SearchBox"/>
    </div>
    <div class="gallery" data-win-control="WinJS.UI.HubSection">
        <!-- content -->
    </div>
    <div class="favorites" data-win-control="WinJS.UI.HubSection">
        <!-- content -->
    </div>
    <div class="clip" data-win-control="WinJS.UI.HubSection">
        <!-- content -->
    </div>
</div>
```

Slika 44: Gradnja kontrole *WinJS.UI.Hub* in njenih sekcij

5.7 Stili

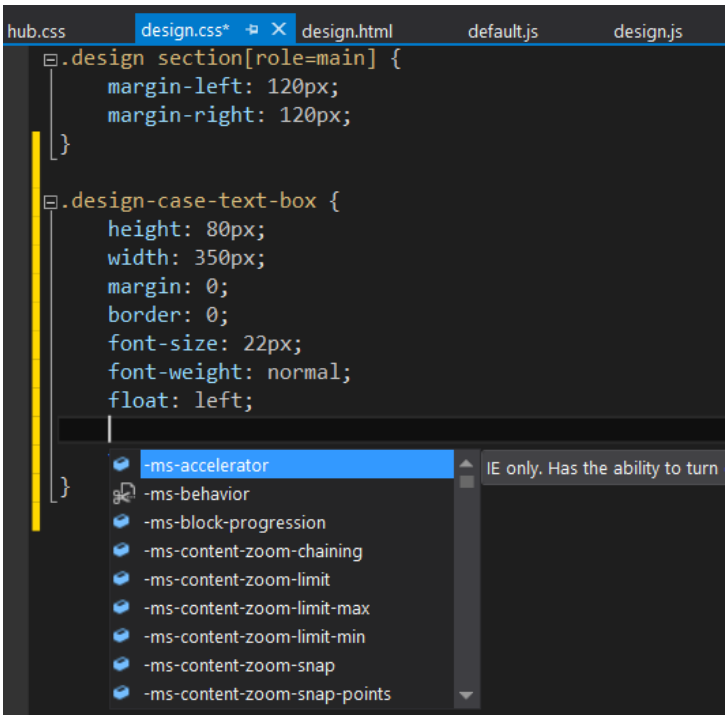
Pri razvoju aplikacij Windows Store z uporabo tehnologij HTML, CSS in JavaScript uporabljamo kaskadne stilske podloge **CSS** za definijo in uporabo stilov, s katerimi oblikujemo HTML kontrole. Stiliranje elementov s *CSS* je v JavaScript primeru analogija stiliranju z *XAML* načinom stiliranja. CSS lahko uporabljamo na več načinov in sicer na **zunanji** (angl. *external*), **notranji** (angl. *internal*) ali **v kodi** (angl. *inline*) način. Na sliki 46 smo pokazali, kako v CSS datoteki definiramo primer CSS razreda (angl. *class*), medtem ko je sliki 45 prikaz apliciranja CSS razreda HTML objektu *buttonFirstName*. To je primer *external* uporabe stiliranje elementov, moramo pa podati še URI od CSS datoteke, kjer je razred *design-case-text-box* definiran.

```
<button
        class="design-case-text-box"
        id="buttonFirstName" title="This is my custom CSS style"
        onclick="showContent"/>
```

Slika 45: Dodelitev CSS razreda HTML kontroli

Na sliki 46 lahko tudi vidimo, da nam je v pomoč Visual Studio fukcija *IntelliSense*, ki nam ob pritisku kombinacije tipk *Ctrl+Space* prikaže vse lastnosti, ki jih lahko uporabimo. Tako si ne rabimo zapomniti velike večine definirah CSS lastnosti. Na podoben način

stiliramo tudi vse ostale HTML kontrole. Če želimo uporabiti *external* način, potem v HTML datoteko vključimo CSS datoteko, tako da med začetno in zaključno HTML značko **head** dodamo značko **link**, kjer specificiramo atributa *href*, ki določa *URI* oz. lokacijo CSS datoteke, ki jo želimo referencirati ter atribut *rel*, ki opisuje relacijo trenutnega dokumenta in vrednosti specificirane z atributom *href*. V primeru, da je datoteka tipa CSS, bo vrednost *rel* atributa *stylesheet*.



Slika 46: Primer definicije CSS razreda

5.8 Obravnava dogodkov

Pokazali smo že nekaj osnov obravnave dogodkov pri razvoju aplikacij Windows Store z uporabo C# in XAML, kjer smo tudi na primeru razložili interakcijo uporabnika z aplikacijo ter procesiranje uporabnikovega pritiska na določeni gumb. Vsak *WinJS.UI* objekt ima definirane določene dogodke, ob katerih se lahko izvedejo odzivne funkcije. Na JavaScript primeru bomo pokazali registracijo in odzivne funkcije, kjer smo uporabili kontroli *WinJS.UI.Flyout* in *WinJS.UI.AppBar*. Kontroli *WinJS.UI.Flyout* definirata dogodke *onafterhide*, *onaftershow*, *onbeforehide* in *onbeforeshow*, medtem ko recimo kontrola *WinJS.UI.ListView* definira mnogo več dogodkov, med drugim tudi *onselectionchanging*,

onitemdragstart, *oncontentanimating*, itd. Razvijalci implementirajo funkcije za tiste dogodke, ob katerih želijo sprožiti ustrezno akcijo. Na primeru bomo pokazali sprožitev akcije ob kliku na gumb aplikacijske vrstice, ki sproži akcijo za odjavo uporabnika iz storitve. Uporabnik se bo lahko odjavil iz storitve ali prekinil akcijo tako, da bo izbral eno od dveh opcij, ki se prikažeta v kontroli *Flyout*, ki pa se prikaže na klik gumba za odjavo.

WinJS.UI.Flyout ima definirano funkcijo **show**, ki prikaže kontrolo *flyout*, ima pa naslednjo definicijo:

WinJS.UI.Flyout.show(anchor, placement, alignment);

Parameter *anchor* je obvezen in predstavlja DOM element, na katerega obesimo *flyout*. Parameter *placement* predstavlja pozicijo, kamor umestimo kontrolo in ima definirane vrednosti *top*, *bottom*, *left* in *right*, medtem ko ima parameter *alignment* definirane vrednosti *center*, *left* in *right*, vendar se upoštevajo samo, če je *placement* parameter enak *top* ali *bottom*.

Primer deklaracije *Flyout* in *AppBar* kontrol prikazuje slika 47.

```
<div id="flyoutLogout" data-win-control="WinJS.UI.Flyout">
    <select id="selectLogout">
        <option>Logout</option>
        <option>Cancel</option>
    </select>
    <br />
</div>

<div id="applicationBar" data-win-control="WinJS.UI.AppBar">
    <button id="buttonLogout"
            onclick="showFlyout"
            data-win-control="WinJS.UI.AppBarCommand"
            data-win-options="{
                icon:'logout',
                label:'Logout',
                type:'button'
            }"/>
</div>
```

Slika 47: Deklaracija kontrol *Flyout* in *AppBar*

Na klik gumba *Logout* (sproži se dogodek ***onclick***) se uporabniku pokaže ti. *Flyout* kontrola, ki ima opciji *Logout* in *Cancel*. Koda za prikaz *Flyout* kontrole je trivialna in je prikazana na sliki 48. Ob kliku na gumb z identifikatorjem *buttonLogout* se pokliče funkcija *showFlyout()*, ki poišče element z identifikatorjem *flyoutLogout* ter ga nato prikaže.

```
function showFlyout() {
    var flyout = document.getElementById("flyoutLogout");
    flyout.show();
}
```

Slika 48: Prikaz kontrole *Flyout*

5.9 Povezovanje podatkov

Opisali smo že, kako povezujemo podatke s kontrolami XAML, kjer smo tudi pojasnili osnove povezovanja podatkov v aplikacijah Windows Store. Pri razvoju JavaScript aplikacij Windows Store uporabljamo analogen pristop vezave podatkov. WinJS API ponuja implementacijo povezovanja podatkov v imenskem prostoru **Binding**, ki vsebuje *povezovanje podatkov (angl. data binding)* in *povezovanje vzorcev (angl. template binding)*. Proces vezave podatkov z uporabo programskega jezika JavaScript zahteva, da objekt, ki ga želimo vezati, označimo kot *opazovan* (angl. *observable*), nato pa ga moramo opazovati tako, da sprejemamo obvestila, ko se vrednost objekta spremeni. V primeru deklarativnega povezovanje podatkov, lahko z uporabo lastnosti **data-win-bind** povežemo s podatki katerokoli HTML kontrolo in jo tako označimo kot *opazovano*. Primer, kako označiti določeni objekt kot *opazovan*, smo prikazali na sliki 49. Ker je v našem primeru objekt *naročilo* (angl. *order*) sestavljen iz množice objektov in je realno takšna struktura predolga za prikaz, smo prikazali samo en del objekta in sicer elementa *firstName* in *lastName* (objekt *customerInfo*), element *caseSize* (objekt *interior*) in element *noOfPockets* (objekt *exterior*). S tem smo izpolnili prvi pogoj povezovanja podatkov.

```
<div class="item">
    First Name:
    <span data-win-bind="innerText:customerInfo.firstName"/>
</div>
<div class="item">
    Last Name:
    <span data-win-bind="innerText:customerInfo.lastName"/>
</div>
<div class="item">
    Case Size:
    <span data-win-bind="innerText:interior.caseSize"/>
</div>
<div class="item">
    No. of Pockets:
    <span data-win-bind="innerText:exterior.noOfPockets"/>
</div>
```

Slika 49: Uporaba lastnosti *data-win-bind*

Izpolnitev drugega pogoja povezovanja podatkov smo prikazali na sliki 50. Ob dogodku *ready*, ki se zgodi po dogodkih *loaded* in *activated* tj. ko je aplikacija pripravljena (angl. *ready*) za uporabo, naredimo objekt *order*, ki vsebuje objekte *customerInfo* (podatki o stranki), *interior* (podatki o notranjosti torbice) in *exterior* (podatki o zunanjosti torbice). Vsem lastnostim teh objektov priredimo neke privzete vrednosti, katere bo uporabnik lahko spreminjal ob izpolnjevanju naročila. Ko smo objekt *order* inicializirali, pokličemo funkcijo **WinJS.Binding.ProcessAll**, ki *poveže vrednosti objekta order z vrednostmi DOM elementov, ki imajo data-win-bind atribut*. Kot vemo, smo vsem *span* elementom iz slike 49 podali data-win-bind atribut. Konkretno to pomeni, da se bo lastnost *innerText* HTML elementa (*span* v našem primeru) povezala z izvorom podatkov (angl. *data source*), kjer je izvor podatkov objekt *order*.

```
(function () {
    "use strict";

    WinJS.UI.Pages.define("/pages/design/design.html", {

        ready: function (element, options) {
            var order = {
                customerInfo: {
                    firstName: null,
                    lastName: null
                },
                interior: {
                    caseSize: "3/5"
                },
                exterior: {
                    noOfPockets: 2
                }
            };

            WinJS.Binding.processAll(null, order);
        },
```

Slika 50: Primer klica funkcije *WinJS.Binding.ProcessAll* v primeru objekta *order*

Omenimo še zelo popularno JavaScript knjižnico **Knockout.js (KO)** za povezovanje podatkov. *KO* je JavaScript implementacija vzorca **Model View ViewModel (MVVM)** in tako ponuja razdelitev domenskih podatkov, podatkov za prikaz in modulov, ki te podatke prikazujejo. *KO* in podobne JS knjižnice (npr. *jQuery*) se lahko uporabljajo tudi z WinJS API. Kar zadeva arhitekturni vzorec MVVM je uporaba le-tega dobra praksa tudi pri razvoju aplikacij Windows Store, vendar se dodana vrednost vzorca MVVM pokaže vglavnem pri razvoju bolj kompleksnih aplikacij. Posebnost vzorca MVVM je, da je bil razvit posebej za XAML. MVVM je bil razvit kot specializacija arhitekturnega vzorca **Presentation Model (PM)**, katerega avtor je Martin Fowler [17]. PM se uporablja tudi v **HTML5** skozi *KO* knjižnico, ki jo je razvil Steve Sanderson.

6 Primerjava uporabljenih pristopov pri razvoju prototipa aplikacije Windows Store

V poglavjih, kjer smo bolj detajlno opisali razvoj aplikacij Windows Store z uporabo obeh pristopov, smo izbrali nekatere koncepte, ki so nam služili za primerjavo. V tem poglavju smo razložili prednosti in pomanjkljivosti razvoja obeh pristopov ter za posamezni pristop podali oceno zahtevnosti pristopa, API funkcionalnosti, značilnosti tehnologij, razlik v konceptih, ipd. Oceno smo podali na osnovi lestvice od 0 do 10, kjer 0 pomeni najslabšo in 10 najboljšo oceno. Rezultati so bazirani na osebni oceni in odražajo približno oceno priporočil uporabe obeh pristopov v razvoju aplikacij Windows Store in sicer pod predpostavko, da razvijalec enako dobro pozna vse omenjene tehnologije, kajti kvantitativno primerjati oba pristopa v tem kontekstu ne bi bilo smiselno. Zadosten razlog proti je že različnost uporabljenih tehnologij na eni in drugi strani. Primer tega je recimo, splošno gledano, nesporna premoč jezika C# napram JavaScript, pa vendar so izjeme, kjer bi bilo uporabiti JavaScript bolj smiselno, lažje in nemara celo bolj učinkovito. A to so vendarle izjeme in ne pravila, vsaj kar zadeva aplikacije Windows Store. Zatorej lahko rečemo, da je bil namen diplomskega dela pokazati kvalitativne in zahtevnostne razlike v implementaciji aplikacij Windows Store z uporabo omenjenih tehnologij.

Asinhrono programiranje

C# definira več vzorcev za asinhrono programiranje (*EAP*, *APM* in *TAP*) kot JavaScript. Medtem ko je programiranje po vzorcih *EAP* in *APM* lahko dokaj težavno, pa je z vzorcem *TAP* zelo poenostavljeno. JavaScript kot programski jezik seveda ni tako močan kot C#, pa vendar je implementacija ti. *obljub* v JavaScript API vpeljala sintaktično mogoče celo enostavnejše asinhrono programiranje kot C# z uporabo TAP vzorca (*async & await*), sploh pa pri gnezdenju obljub.

Ocena za pristop (C# in XAML): 9.5
Ocena za pristop (HTML, CSS in JS): 10.0

Aplikacijski model

Aplikacijski model je zelo dobro zastavljen in ni nekih posebnih razlik med obema pristopoma razvoja, ki jasno definirata vstopno točko aplikacije, obravnavanje stanj aplikacije, shranjevanje sej ob prehodih med stanji aplikacije, itd.

Ocena za pristop (C# in XAML): 10.0
Ocena za pristop (HTML, CSS in JS): 10.0

Navigacija med stranmi

Oba pristopa razvoja privzeto ponujata razrede, ki omogočajo dodatne funkcije pri navigaciji med stranmi in tako olajšajo delo razvijalcem, pa vendar obstaja razlika in sicer v tem, da JavaScript API ponuja tudi preprečevanje ti. *top-level* navigacije. Ta funkcionalnost je podobna funkcionalnosti pri razvoju spletnih aplikacij z uporabo ogrodja *ASP.NET*, kjer se uporabljajo ti. *master pages*, kar omogoča enoličen izgled in vedenje vseh strani ali skupin strani aplikacije z uporabo predlog.

Ocena za pristop (C# in XAML): 8.5
Ocena za pristop (HTML, CSS in JS): 9.0

Kontrole

Čeprav JavaScript API definira mnoge kontrole, še vedno v razvoju pride do potrebe za drugimi kontrolami in posledično do uporabe klasičnih HTML kontrol, kajti nekatere osnovne kontrole niso definirane. Primer je recimo kontrola za gumb, ki je WinJS API ne definira, zato bi tu dali absolutno prednost jeziku XAML, kjer teh problemov ni. Po drugi strani pa v primeru potrebe po gradnji kontrol v času delovanja aplikacije le-te brez pretiranih težav instanciramo tudi v C# delu kode.

Ocena za pristop (C# in XAML): 9.0
Ocena za pristop (HTML, CSS in JS): 7.5

Stili

Preprostost kaskadnih stilskih podlog CSS z uporabo HTML5 proti funkcionalnosti in moči deklarativnega jezika XAML. Ta tema je sama po sebi verjetno dovolj velika za diplomsko nalogo. Odgovor na to vprašanje se najverjetneje skriva v nefunkcionalnih zahtevah aplikacije, ki jo želimo implementirati. V primeru razvoja prototipov diplomske naloge smo dali malo prednost kaskadnim stilskim podlogam CSS.

Ocena za pristop (C# in XAML): 9.0
Ocena za pristop (HTML, CSS in JS): 10.0

Obravnava dogodkov

Menimo, da je obravnava dogodkov z jezikom C# veliko lažja kot z jezikom JavaScript in tudi veliko bolj konsistenčna, kot smo pokazali na primeru uporabe kontrol. Visual Studio ponuja tudi hierarhijo in definicijo objektov skozi ti. brskalnik objektov (angl. *object browser*) za jezika C# in XAML, ne pa tudi za JavaScript.

Ocena za pristop (C# in XAML): 9.5
Ocena za pristop (HTML, CSS in JS): 8.5

Povezovanje podatkov

Deklarativni jezik XAML ponuja povezovanje podatkov skoraj brez omejitev, dočim je to opravilo za odtenek lažje narediti v JavaScript Windows Store aplikacijah. Prednost pristopa z uporabo jezika JavaScript je občutna tudi takrat, ko so podatki zapisani v formatu *JSON*, za kar pa v jeziku C# potrebujemo razčlenjevalnik. Za podatke v drugih formatih, je implementacija razčlenjevalnika verjetno lažje opravilo v C# kot JavaScript jeziku.

Ocena za pristop (C# in XAML): 9.0
Ocena za pristop (HTML, CSS in JS): 8.5

Oblikovanje grafičnega vmesnika

Oblikovanje grafičnega vmesnika zajema vse predhodne module in ta modul lahko primerjamo le na prototipni aplikaciji te diplomske naloge, kjer končno damo prednost C# in XAML tehnologijam. Na splošno pa menimo, da bo prednost dobil tisti pristop razvoja, ki bo lažje izpolnil zahteve za grafični vmesnik.

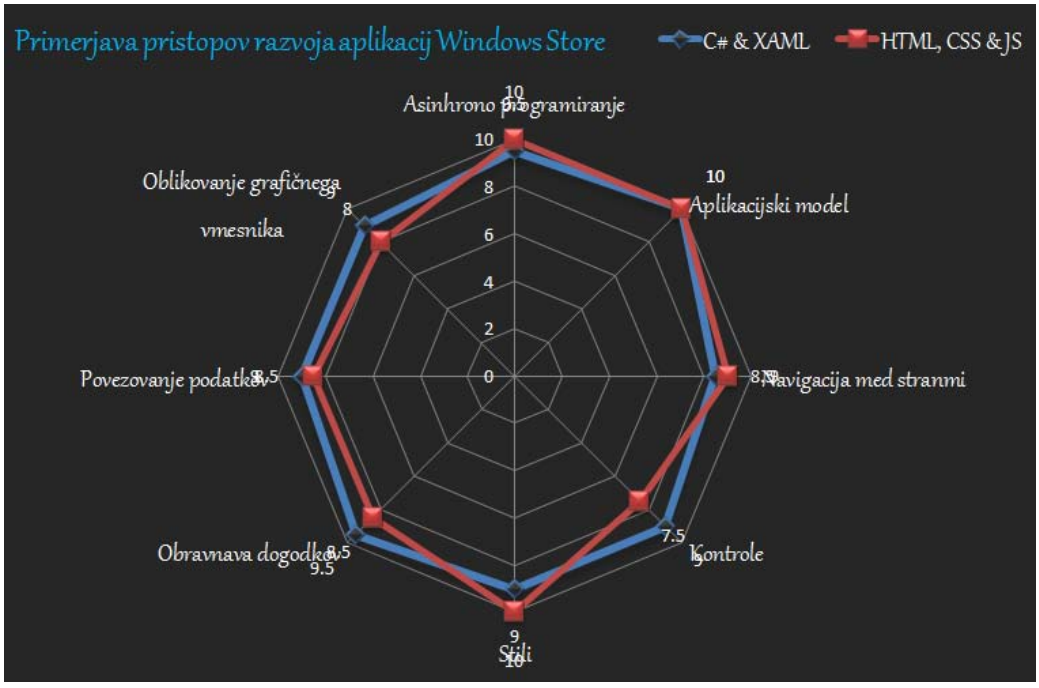
Ocena za pristop (C# in XAML): 9.0
Ocena za pristop (HTML, CSS in JS): 8.0

Grafični prikaz rezultatov

Na sliki 51 smo na radarskem grafu prikazali rezultate primerjave obeh pristopov razvoja aplikacij Windows Store. Vsaki od primerjanih sekcij smo dodelili vrednosti, kjer je 100 najboljša in 0 najslabša. Vhodni podatki za radarski graf in rezultati so naslednji:

	x(C# in XAML)	*x(HTML, CSS in JavaScript)*
Asinhrono programiranje	9.5	10.0
Aplikacijski model	10.0	10.0
Navigacija med stranmi	8.5	9.0
Kontrole	9.0	7.5
Stili	9.0	10.0
Obravnava dogodkov	9.5	8.5
Povezovanje podatkov	9.0	8.5
Oblikovanje grafičnega vmesnika	9.0	8.0
$\overline{x}$	**9.1875**	**8.9375**

Lahko vidimo, da je povprečna vrednost ocene razvoja z uporabo tehnologij C# in XAML **9.1875** ter HTML, CSS in JavaScript **8.9375**. Primerjava je pokazala, da so rezultati v našem primeru malenkostno bolj v prid tehnologijam C# in XAML kot HTML, CSS in JavaScript.



Slika 51: Prikaz rezultatov primerjave pristopov razvoja z uporabo radarskega grafa

7 Zaključek

V diplomskem delu smo pokazali nekatere aspekte in posebnosti razvoja aplikacij Windows Store z uporabo tehnologij C# in XAML na eni ter HTML, CSS in JavaScript na drugi strani. Kot smo lahko videli, si oba pristopa delita pomembne principe stilskega oblikovanja ti. *show pride in craftsmanship*, *do more with less*, *be fast and fluid*, *be authentically digital* in *win as one*. Prav tako si oba pristopa delita aplikacijski model na abstraktnem nivoju, vendar skozi drugačno implementacijo. Razvoj obeh prototipov je pokazal zrelost in moč jezika XAML, ki se uporablja že dolgo v mnogih drugih tehnologijah. Po drugi strani je WinJS API relativno nov in kljub temu, da ponuja obilico funkcionalnosti, še vedno ne moremo trditi, da je mogoče neko specifično funkcionalnost implementirati brez uporabe dodatnih JavaScript knjižnic.

Prav tako je vidna razlika v obsegu funkcionalnosti, ki jo ponujata *Visual Studio* in *Blend* IDE, z uporabo tehnologij C# in XAML napram tehnologijam HTML, CSS in JavaScript, kot npr. testiranje enot, razne meritve, analiza kode in ti. *IntelliSense* (ki sicer obstaja tudi za JavaScript, pa vendarle ni tako močan kot za C#). XAML vsebuje tudi ti. ločljivostno neodvisnost, kar pomeni, da vrednosti za velikost, ki jih navajamo v deklaraciji XAML kontrol, niso točke, temveč neodvisne enote naprave. JavaScript tudi ne podpira posebne funkcionalnosti za delo z množicami elementov, npr. izrazi *LINQ* in *lambda*.

Lahko tudi rečemo, da je XAML boljši za povezovanje podatkov kot JavaScript implementacija ti. *observable objects*, vendar je le-to opravilo v jeziku JavaScript nekoliko lažje. Enako bi lahko rekli za podatke, ki so v JSON formatu, saj je le-ta bil narejen specifično za podporo JavaScript (od tod tudi njegovo ime) in podatkov ni potrebno posebej razčlenjevati, za kar pa v C# potrebujemo razčlenjevalnik (angl. *parser*).

Omenili bi na kratko še razvoj aplikacij za *Windows Phone 8*. Glede na dejstvo, da je le-te možno razvijati le z uporabo tehnologij C# ali C++ in XAML, lahko rečemo, da bo prenosljivost izvorne kode veliko lažja tistim razvijalcem, ki hočejo razvijati aplikacije za *Windows Store* in *Windows Phone 8*, medtem ko bo JavaScript rešitev zahtevala čisto novo implementacijo.

Kljub vsemu menimo, da imata oba principa razvoja močne razloge za in zelo malo razlogov proti. JavaScript je bil v Windows Store aplikacije vpeljan iz razloga pridobitve množice spletnih razvijalcev, ki dobro poznajo spletne tehnologije razvoja spletnih aplikacij. Verjetno bo glede izbire pristopa na koncu odločila izbira tehnologij, v katerih se razvijalec bolje znajde. Osebno menimo, da je trenutno še vedno boljša izbira razvoja aplikacij Windows Store z uporabo tehnologij C# in XAML.

Literatura

[1] Kraig Brockschmidt. Programming Windows® 8.1 Apps with HTML, CSS, and JavaScript, str. 112, 121, 124. Založba Microsoft Press, 2012.

[2] Nick Lecrenski, Doug Holland, Allen Sanders, Kevin Ashley. Professional Windows® 8 Programming: Application Development with C# and XAML, str. 53, 54. Založba John Wiley & Sons, Inc., 2013.

[3] Adam Nathan. Windows® 8.1 Apps with XAML and C# Unleashed, str. 12, 176, 514. Založba Sams Publishing, 2013.

[4] (4.2.2014) .NET Framework 4.5. Dostopno na:
http://msdn.microsoft.com/en-us/library/vstudio/w0x726c2(v=vs.110).aspx

[5] (4.2.2014) 40+ Best Web UI Interface Framework Kits. Dostopno na:
http://www.andysowards.com/blog/2012/40-best-web-ui-interface-framework-kits/

[6] (4.2.2014) Application lifecycle (Windows Store apps) (Windows). Dostopno na:
http://msdn.microsoft.com/en-us/library/windows/apps/hh464925.aspx

[7] (4.2.2014) Asynchronous programming in JavaScript (Windows Store apps) (Windows). Dostopno na:
http://msdn.microsoft.com/en-us/library/windows/apps/hh700330.aspx

[8] (4.2.2014) C# Overview. Dostopno na:
http://tutorialspoint.com/csharp/csharp_overview

[9] (4.2.2014) ECMA C# and Common Language Infrastructure Standards. Dostopno na:
http://msdn.microsoft.com/en-us/vstudio/aa569283.aspx

[10] (4.2.2014) ECMAScript Language Specification. Dostopno na:
http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

[11] (4.2.2014) Events and routed events overview (Windows). Dostopno na:
http://msdn.microsoft.com/en-us/library/windows/apps/hh758286.aspx

[12] (4.2.2014) Getting started with the .NET Framework. Dostopno na:
http://msdn.microsoft.com/en-us/library/vstudio/hh425099(v=vs.110).aspx

[13] (4.2.2014) JavaScript API (Windows 8 and web) (Live Connect). Dostopno na:
http://msdn.microsoft.com/en-us/library/live/hh243643.aspx

[14] (4.2.2014) Portable Class Libraries. Dostopno na:
http://msdn.microsoft.com/en-us/library/vstudio/gg597391(v=vs.100).aspx

[15] (4.2.2014) Managed API (Windows 8 and Windows Phone) (Live Connect). Dostopno na:
http://msdn.microsoft.com/en-us/library/live/hh533665.aspx

[16] (4.2.2014) Navigation patterns (Windows). Dostopno na:
http://msdn.microsoft.com/en-us/library/windows/apps/hh761500.aspx

[17] (4.2.2014) Presentation Model. Dostopno na:
http://martinfowler.com/eaaDev/PresentationModel.html

[18] (4.2.2014) Promises - CommonJS Spec Wiki. Dostopno na:
http://wiki.commonjs.org/wiki/Promises

[19] (4.2.2014) REST reference (Live Connect). Dostopno na:
http://msdn.microsoft.com/en-us/library/live/hh243648.aspx

[20] (4.2.2014) Standard ECMA-335 6th Edition / June 2012 Commong Language Infrastrukture (CLI) Partitions I to VI. Dostopno na:
http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-335.pdf

[21] (4.2.2014) Task-based Asynchronous Pattern. Dostopno na:
http://download.microsoft.com/download/5/B/9/5B924336-AA5D-4903-95A0-56C6336E32C9/TAP.docx

[22] (4.2.2014) WinJS.xhr function (Windows). Dostopno na:
http://msdn.microsoft.com/en-us/library/windows/apps/br229787.aspx

[23] (4.2.2014) WinJS.UI.Pages.PageControl object (Windows). Dostopno na:
http://msdn.microsoft.com/en-us/library/windows/apps/jj126158.aspx

[24] (4.2.2014) Windows API reference for Windows Store apps (Windows). Dostopno na:
http://msdn.microsoft.com/en-us/library/windows/apps/br211377.aspx

[25] (4.2.2014) WinMain: The Application Entry Point (Windows). Dostopno na:
http://msdn.microsoft.com/en-us/library/windows/desktop/ff381406(v=vs.85).aspx