Razvoj Windows 10 aplikacija

Kedveš, Josip

Master's thesis / Diplomski rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Department of Mathematics / Sveučilište Josipa Jurja Strossmayera u Osijeku, Odjel za matematiku**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:126:685147

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Repository of School of Applied Mathematics and Computer Science

Sveučilište J.J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike, smjer: Matematika i računarstvo

Josip Kedveš

Razvoj Windows 10 aplikacije

Diplomski rad

Osijek, 2017.

Sveučilište J.J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike, smjer: Matematika i računarstvo

Josip Kedveš

Razvoj Windows 10 aplikacije

Diplomski rad

Voditelj: izv. prof. dr. sc. Domagoj Matijević
Suvoditelj: doc. dr. sc. Alfonzo Baumgartner

Osijek, 2017.

Sadržaj

1 Uvod **1**

2 Univerzalne aplikacije **2**
2.1 Povijest Windows univerzalnih aplikacija 2

3 UWP - univerzalna Windows platforma **4**
3.1 Windows 10 4
3.2 Universal Windows Platform 5
3.2.1 Adaptivni kod 6
3.2.2 Adaptivni dizajn 9

4 UWP aplikacije **12**
4.1 Životni ciklus aplikacije 12
4.2 Produljeno izvođenje 13
4.3 Pozadinski zadaci 14
4.4 Pohrana podataka 15
4.5 Ostale značajke 15
4.5.1 Package.appxmanifest 15
4.5.2 Međuaplikacijska komunikacija 17
4.5.3 Continuum 18

5 Izrada aplikacije **19**
5.1 MVVM 19
5.1.1 Data binding 20
5.2 Arhitektura i konfiguriranje aplikacije 22
5.3 Videozapisi 25
5.3.1 Reprodukcija 26
5.4 Korištenje geolokacije 27
5.4.1 Pristup geolokacijskim resursima 27
5.4.2 Korištenje adaptivnog dizajna 29
5.5 Postavke 31

Zaključak **32**

Literatura **33**

Sažetak **34**

Summary **35**

Životopis **36**

1 Uvod

Ovaj je rad za zadatak imao predstaviti univerzalnu Windows platformu za razvoj Windows 10 aplikacija. Osim opisivanja platforme, predstavljene su neke njene bitne karakteristike te je napravljena usporedba s prijašnjim razvojnim okruženjima. Također, u sklopu rada su opisane aplikacije zasnovane na ovoj platformi te je izrađena jedna Windows 10 aplikacija.

U drugom poglavlju je predstavljena ideja koja leži iza koncepta univerzalnih aplikacija i na koje su ga načine različiti proizvođači pokušali implementirati. Uz to je dan kratak povijesni pregled ideje univerzalnih aplikacija u Windows operativnim sustavima.

Tematika sljedećeg poglavlja je univerzalna Windows platforma. Predstavljena je ideja adaptivnosti kroz adaptivnost programskog koda i korisničkog sučelja. Također, opisane su bitne karakteristike operativnog sustava Windows 10.

Četvrto poglavlje za cilj ima usporedbu aplikacija zasnovanih na ovoj platformi s klasičnim Windows programima. Najviše je pažnje posvećeno posebnostima aplikacije te načinima prilagodbe na iste.

Posljednje poglavlje predstavlja aplikaciju za snimanje prometa koja je izrađena u sklopu ovog rada. U njemu je objašnjena arhitektura aplikacija te je na primjeru nekoliko funkcionalnosti prikazano kako razvoj na univerzalnoj Windows platformi izgleda u praksi.

2 Univerzalne aplikacije

Suvremeni čovjek u svakodnevnom životu koristi brojne uređaje za brže i efikasnije rješavanje zadataka, konzumiranje digitalnog sadržaja, zabavu te učenje. To su najčešće stolna i prijenosna računala, pametni telefoni te tableti. U danom trenutku, sukladno potrebama i okolnostima odabrat ćemo najprikladniji od njih – pametni telefon za brz pristup nekoj informaciji ili stolno računalo za pisanje dugačkog dokumenta. Uređaji kojima se služimo najčešće ne koriste isti operativni sustav niti isto korisničko sučelje te nam pružaju drugačije korisničko iskustvo. Kako nadići te razlike i dovesti našu interakciju s računalima na novu razinu?

Prvi, a ujedno i najrašireniji pristup se oslanjanje na web aplikacije. Danas moderne web aplikacije korisnicima pružaju puno bolje iskustvo i više funkcionalnosti nego ikad, a neke su dosegle i razinu svojih klasičnih ekvivalenata. Razlog toga su porast kvalitete internetske veze i razvojnih alata, ali i težnje korisnika za jednostavnijim i bržim korisničkim sučeljima. Mobilni internet i responzivni dizajn nam omogućuju da te web aplikacije koristimo i na mobilnim uređajima, a da pritom ne žrtvujemo funkcionalnost ili korisničko iskustvo. Puni potencijal ovakvog pristupa dobivamo ako pridodamo i spremanje podatka u cloudu[1].

Iako prethodni pristup ima brojne prednosti, neki pružatelji softvera su se odlučili za drugačiji pristup. Uzmimo za primjer Google i Apple. Apple na svojim uređajima koristi prvenstveno dva operativna sustava: iOS i OS X. Iako su oni namijenjeni za različite skupine uređaja, ova dva sustava mogu međusobno komunicirati te se tako nadopunjavati, a neke aplikacije omogućavaju i dijeljenje podatka između OS X i iOS verzije. Istim pristupom se vodi i Google kada su u pitanju Chrome OS i Android. Iako je Chrome OS prvenstveno bio zamisljen kao sustav oko web preglednika orijentiran na Googlove web servise, on sada podržava instalaciju određenih Android aplikacija. Iako broj tih aplikacija trenutno nije velik, nazire se ideja stvaranje jedinstvene platforme na različitim računalnim arhitekturama.

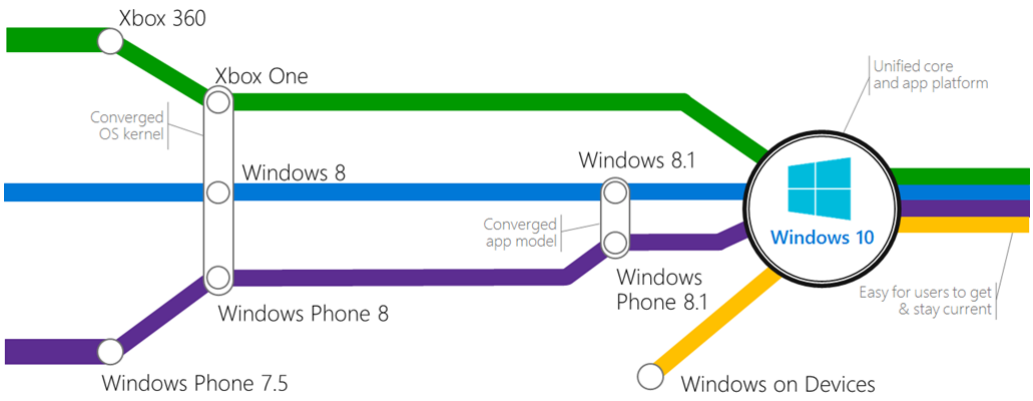
2.1 Povijest Windows univerzalnih aplikacija

Microsoft je do izlaska Windowsa 8 bio poprilično konzervativan što se tiče svojih operativnih sustava. Windows, Windows Server, Windows Mobile/Phone i Windows Embedded su bili zasnovani na drugačijim arhitekturama i namijenjeni za prilično različite zadatke. Windows 8 je korisnicima donio novo korisničko sučelje, prvenstveno prilagođeno uređajima s ekranima osjetljivom na dodir. Osim vidljivih, ovaj sustav je donio mnogo promjena „ispod haube“.

Prilikom kreiranja novog sustava Microsoft je odlučio prilagoditi hibridni NT kernel kojeg je dosad koristio za desktop i server verzije Windowsa za korištenje i u drugim platformama. Novi NT kernel, verzija 6.2, je bio ugrađen u dva operativna sustava koji su na tržište izašla 2012. godine: Windows 8 i Windows Phone 8, ali i u operativni sustav igraće konzole Xbox One koja je izašla 2 godine kasnije. Ovo je bio prvi korak pri stvaranju jedinstvene platforme.

Osim podrške za klasične WPF i Silverlight aplikacije zasnovane na dobro poznatom .NET Frameworku, novi sustavi su podržavali i Windows Store (Metro) aplikacije. Metro aplikacije su bile zasnovane na novoj platformi Windows Runtime (WinRT). WinRT nije bio

[1] računarstvo u oblaku



Slika 2.1: Razvoj Windows operativnih sustava - [6]

zamišljen kao zamjena za klasični .NET Framework, već kao usporedna platforma koja će za cilj imati olakšati razvoj novih aplikacija koje će biti duboko integrirane u novi sustav i biti prilagođena novom korisničkom sučelju. Aplikacije za desktop i mobilnu platformu su i dalje pisane zasebno, no oko 30% funkcionalnosti je bilo identično. Za razliku od prijašnjih aplikacija, ove se aplikacije izvršavaju u sandboxu[2] te moraju biti distribuirane putem odgovarajuće verzije Windows Storea.

Windows 8.1 i Windows Phone 8.1 koji su izašli krajem 2013. i početkom 2014. godine donijeli su brojna poboljšanja i nove funkcionalnosti. Ova izdanja Windowsa su donijela i nešto puno bitnije – univerzalne aplikacije. Zahvaljujući dijeljenju 95% API-ja između dva operativna sustava i korištenju istog životnog ciklusa aplikacija napokon je bilo moguće napisati jednu aplikaciju koja će raditi na oba sustava. Iako je ovo bio veliki napredak u odnosu na prethodne verzije, developeri su i dalje morali pisati dva prezentacijska sloja ako su željeli izraditi aplikaciju za obje platforme te gotovu aplikaciju distribuirati putem dviju trgovina aplikacija. Prezentacijski slojevi aplikacije su se vezali za jedinstveni sloj u kojem se vršila poslovna logika aplikacija. Ako bi u nekom slučaju morali koristiti neki od 5% API-ja koji nisu bili zajednički to bi napravili korištenjem predprocesorskih naredbi.

[2]skup sigurnosnih mehanizama koji nadgledaju program i korigiraju njegovo ponašanje

3 UWP - univerzalna Windows platforma

U ovom će poglavlju biti predstavljena univerzalna Windows platforma kao zadnji korak u nastojanjima Microsofta da korisnicima pruži jedinstveno iskustvo. No predstavimo prvo operativni sustav Windows 10 - prvi Microsoftov sustav koji je od početka stvaran s idejom pokretanja univezalnih aplikacija.

3.1 Windows 10

Windows 10 je zadnja verzija Windowsa na tržištu, a po tvrdnjama Microsofta, ujedno i posljednja verzija Windowsa ikad. Windows 10 je postao dostupan korisnicima 29. srpnja 2015. godine kao besplatna nadogradnja s Windowsa 7 i 8.1, a u Mobile varijanti 15. studenoga iste godine. Stavimo li na stranu nove funkcionalosti i izgled, ono što ove Windowse čini dugačijim od prethodnika su prvenstveno njihova sveobuhvatnost, način nadograđivanja i nova platforma za razvoj aplikacija.

Slika 3.1: Tipovi uređaja koje pokreće Windows 10 - [5]

Stolno računalo, prijenosnik, mobitel, tablet, igraća konzola, sat za praćenje aktivnosti, Raspberry Pi, HoloLens, Surface Hub – sve ove uređaje pokreće Windows 10. Valja istaknuti da se tu ipak ne radi o jedinstvenom operativnom sustavu, ali jezgru svakog od tih sustava čini jedinstaveni Windows Core. Tako primjerice imamo Windows 10 Mobile koji može pokretati pametne telefone i tablete ili Windows IoT Core namijenjen za razvoje pločice poput Raspberry Pi-ja ili DragonBoard-a. Svaki od operacijskih sustava je osmišljen s idejom da najbolje iskoristi mogućnosti uređaja kojem je namijenjen, ali da i dalje ostane dio Windowsa 10.

Do ove verzije, Microsoft je bio poprilično konzervativan kada su u pitanju nadogradnje. Nove verzije operativnog sustava su bile izdavane približno svake tri godine, a Service Packovi[3] su često bili samo paketi zakrpa izdanih u istom tom razdoblju. Ovakvim pristupom korisnici nove funkcionalnosti i poboljšanja dobivaju tek sa svakom novom verzijom operativnog sustava – trenutni se sustav samo održava, a nove funkcionalosti se ugrađuju u sljedeću iteraciju. Windows 10 koristi potpuno drugačiji pristup. Prema riječima Terrya

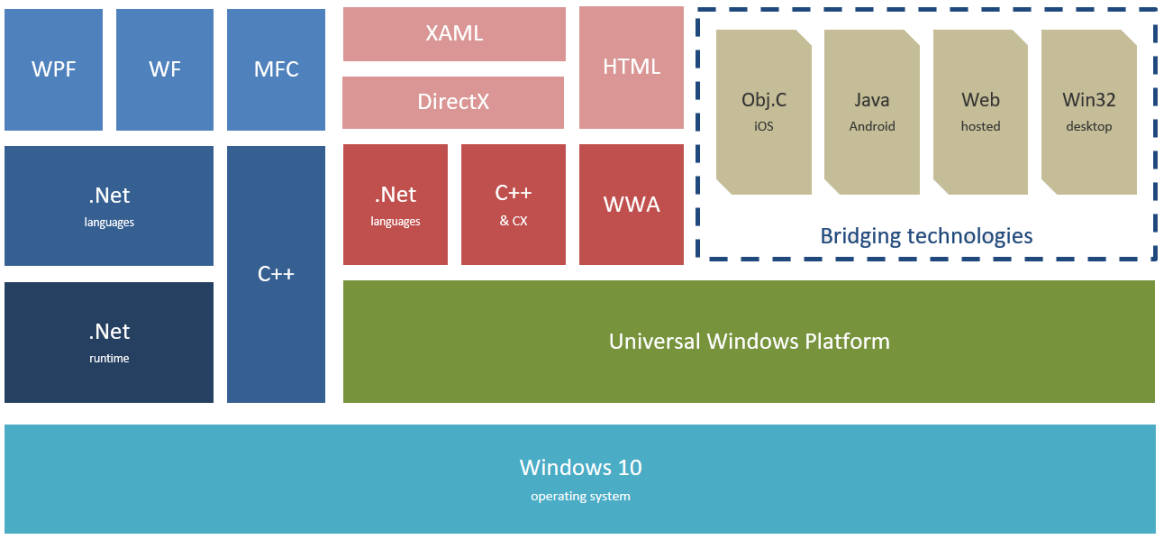
[3]veći paketi nadogradnji koji su bili izdavani u većim vremenskim razmacima

Myersona Windowsi 10 predstavljaju operativni sustav kao uslugu, te će stoga i nadogradnje biti konstantne. Osim sigurnosnih ažuriranja i zakrpa koje pristižu svaki tjedan, svakih nekoliko mjeseci stiže i velika nadogradnja koja donosi i nove funkcionalnosti. Već su izašle dvije takve nadogradnje: November Update (studeni 2015.) i Anniversary Update (kolovoz 2016.), a treća, imena Creators Update, je najavljena za proljeće 2017. godine. Ovim pristupom korisnici na nove funkcionalnosti ne moraju čekati dugi vremenski period, a razvojni tim konstantno dobiva povratne informacije. Naravno, operativni sustav na ovaj način dulje vremena ostaje „svjež“ te samim time produljuje životni ciklus i bolje drži korak s konkurencijom.

Veliku ulogu u dodavanju novih funkcionalosti i procesu nadogradnje imaju i tehnički potkovaniji korisnici i entuzijasti uključeni u Windows Insider program. Ovaj program nudi korisnicima da se na nekoliko načina uključe u razvoj sustava kojeg koriste. Prva opcija nudi mogućnost isprobavanja nove verzije operativnog sustava prije nego one budu dostupne svim korisnicima dok druga omogućuje korisnicima da isprobavaju nove mogućnosti za vrijeme samog razvoja pa samim time korisnici ujedno postaju i beta testeri. Ovakav pristup se pokazao vrlo dobrim te je usmjerio razvoj Windowsa 10 u skladu s korisničkim kritikama i sugestijama.

3.2 Universal Windows Platform

Što je zapravo univerzalna Windows platforma? Prema Microsoftu, to je jedinstvena API platforma koja omogućuje razvoj aplikacija namijenjenih za uređaje pokretane Windowsima 10. Prilikom razvoja ovih aplikacija ne ciljamo određene verzije Windowsa 10 već isključivo verziju univerzalne platforme. Točnije, prilikom izrade aplikacije definiramo minimalnu verziju potrebnu za pokretanje aplikacije, ali i najvišu službeno podržanu verziju. Zbog novog načina nadograđivanja operativnog sustava, Microsoft je odlučio kako bi bilo bolje ne vezati razvojne grane platforme i samog sustava kako bi veći broj korisnika imao pristup najnovijim funkcionalnostima platforme.



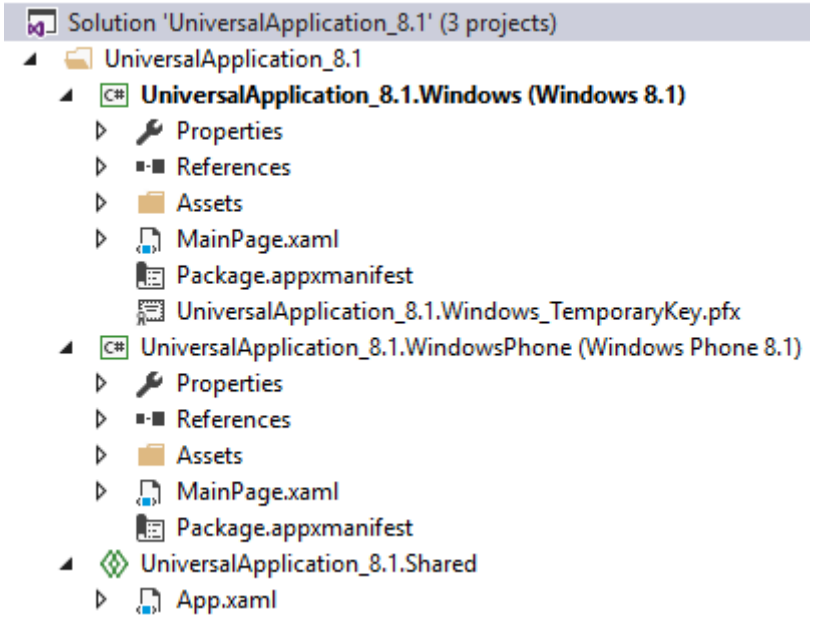
Slika 3.2: Shema nativnih tehnologija za izradu Windows aplikacija - [6]

Prethodni shematski prikaz nam daje bolju sliku odnosa UWP-a i ostalih aktualnih Microsoftovih razvojnih platforma za izradu desktop aplikacija. Dakle, UWP se nalazi odmah pored „klasičnih“ aplikacija i servisa baziranih na .NET frameworku i njena svrha nije zamjena ostalih vrsta aplikacije već izrada novih, univerzalnih aplikacija. Što se programskih jezika i razvojnih okruženja tiče, možemo izdvojiti 3 cjeline. Prvi, a ujedno i najrašireniji način izrade aplikacije se bazira na korištenju .NET jezika poput C# i Visual Basica u kombinaciji s XAML-om[4]. Developeri vješti u izradi web aplikacija univerzalne aplikacije mogu izraditi pomoću WinJS-a[5], HTML-a i CSS-a, dok oni kojima je razvoj igara bliži mogu koristiti C++/CX i DirectX.

Iako je većinu aplikacija poželjno raditi od početka, to s UWP aplikacijama nije slučaj. Prilikom predstavljanja Windowsa 10 predstavljeni su takozvani brigde alati. Njihova je zadaća već gotove Android, iOS, web ili Win32[6] aplikacije na lak način pretvoriti u UWP aplikacije. Iako su ovi alati i dalje u ranim razvojnim fazama, već je nekoliko iOS i Win32 aplikacija dobilo svoju univerzalnu inačicu na ovaj način. U sljedeća dva poglavlja će biti riječi o karakteristikama koju ovu platformu zaista čine univerzalnom.

3.2.1 Adaptivni kod

Kako bi stekli bolju predodžbu o UWP aplikacijama pogledajmo prvo kako je izgleda projekt univerzalne aplikacije namijenjene 8.1 sustavima. Na slici 3.3 je jasno vidljivo kako



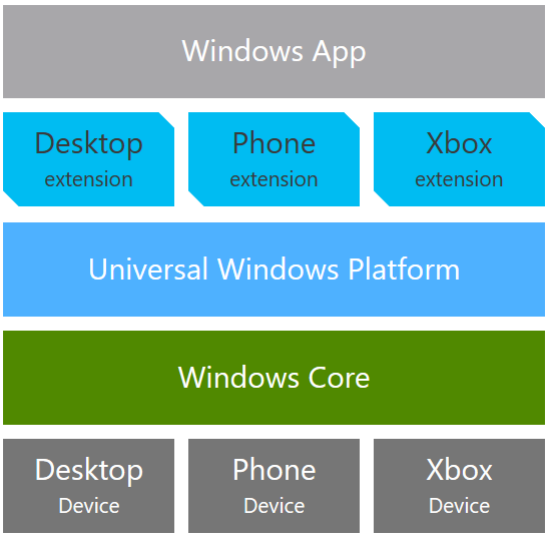
Slika 3.3: Struktura projekta univerzalne 8.1 aplikacije

u ovom slučaju imamo 3 projekta: Windows, Windows Phone i Shared. Prva dva projekta sadrže aplikacija za odgovarajuće aplikacije dok treći sadrži dijelove koda ili resurse koje te dvije aplikacije dijele. Prilikom svakog dodavanja ili izmjene podataka u tom projektu promjene se automatski reflektiraju na svaku od aplikacija. Ako u zajedničkom projektu

[4]deklarativni jezik korišten za izradu sučelja Windows aplikacija
[5]verzija JavaScripta modificirana od strane Microsofta
[6]kratica za sve neuniverzalne Windows aplikacije

želimo koristiti neku od funkcionalnosti dostupnu samo jednoj od platformi to ćemo napraviti tako da kod okružimo odgovarajućim predprocesorskim naredbama za grananje (npr. `# if WINDOWS_PHONE_APP`). Prilikom kompajliranja projekta taj će se kod nalaziti samo u jednoj od aplikacija. Svaku od aplikacija se potom distribuira u odgovarajuću trgovinu aplikacija.

Razvoj UWP aplikacije se uvelike razlikuje od prethodnog pristupa – kreiramo jednu aplikaciju iz koje radimo jedan paket kojeg distribuiramo preko jedne trgovine aplikacija. Kako to postižemo? U prethodnom je poglavlju pojašnjeno što je to univerzalna Windows platforma i koji je njen kontekst u Windows okruženju, no postoji još jedan sloj između platforme i same aplikacije (vidi sliku 3.4). Radi se od ekstenzijama platforme koje donose prošireni skup funkcionalnosti u odnosu na baznu platformu. U tim ekstenzijama se nalaze metode i funkcije namijenjene za pojedine grupe uređaja te tako imamo Desktop, Phone, IoT ili Xbox ekstenzije.



Slika 3.4: Struktura UWP aplikacije - [6]

Logično je postaviti pitanje zašto su se određene funkcionalnosti premjestile iz platforme u zasebnu ekstenziju. Uzmimo na primjer Raspberry Pi 3 – jedan IoT[7] uređaj. Kao i većina IoT uređaja, ova ploča posjeduje GPIO[8] pinove te je razumno očekivati da ćemo istim tim pinovima moći manipulirati i u našoj aplikaciji. Budući da se radi samo o jednoj obitelji uređaja, uključivanje tih funkcionalnosti bi nepotrebno povećalo kompleksnost platforme, a velika većina uređaja ne imala nikakve pogodnosti. Zato je upravljanje GPIO pinovima uključeno u IoT ekstenzije zajedno s drugim funkcionalnostima specifičnim IoT uređajima.

Prilikom izrade aplikacija, ukoliko se za tim ukaže potreba, možemo uključiti jednu ili više ekstenzija u par klikova. Dodavanjem ekstenzija ne ograničavamo na kojim će se uređajima pokretati naša aplikacije već samo otvaramo mogućnost korištenja funkcionalnosti koje nisu dostupne u baznoj platformi. Budući da se s ovakvim pristupom nismo osigurali da će uređaj koji će koristiti određenu aplikaciju zaista podržavati sve njene funkcionalnosti na neki način moramo provjeriti njegove mogućnosti. Za razliku od prethodne generacije

[7]kratica za Internet of Things - inteligentni umreženi predmet
[8]kratica General purpose input/output - dijelovi razvojne ploče namijenjeni komunikaciji dvojne prirode

aplikacija, UWP aplikacije imaju mogućnost provjeravanja funkcionalnosti prilikom izvršavanja aplikacije. Alat koji se koristi za tu provjeru je klasa `ApiInformation` iz namespacea `Windows.Foundation.Metadata` i njenih 8 metoda:

- `IsApiContractPresent`
- `IsEnumNamedValuePresent`
- `IsEventPresent`
- `IsMethodPresent`
- `IsPropertyPresent`
- `IsReadOnlyPropertyPresent`
- `IsTypePresent`
- `IsWriteablePropertyPresent`

kojima predajemo imena određenog seta funkcionalnosti. Osim provjere prisutnosti neke od ovih metoda provjeravaju i radi li se o adekvatnoj verziji funkcionalnosti. U sljedećem je primjeru pokazano kako primjenom prethodno navedene tehnike možemo sakriti statusnu traku na mobilnom uređaju.

```
if (ApiInformation.IsApiContractPresent("Windows.Phone.PhoneContract", 1, 0))
{
    await Windows.UI.ViewManagement.StatusBar.GetForCurrentView().HideAsync();
}
```

Primjer 3.5: Provjera postojanja i verzije Phone modula

Iako je poželjno pokriti što veći broj uređaja i verzija sustava, to nekad nije moguće. Zbog asinkronog ritma nadogradnji sustava, platforme i ekstenzija nekad je nužno ograničiti dostupnost aplikacije. To može napraviti na više načina, a prvi od njih je dodavanje restrikcija prilikom distribucije aplikacije putem trgovine. Drugi, puno modularniji i robusniji, način je deklariranje zahtjeva u samoj aplikaciji. Na ovaj način možemo ograničiti ne samo dostupnost aplikacije pojedinim vrstama uređaja već i pojedinim verzijama UWP-a i odgovarajućih ekstenzija. Sljedeći primjeri pokazuju tri različita scenarija primjene ovih ograničenja.

```
<Dependencies>
    <TargetDeviceFamily Name="Windows.IoT "
                minVersion="10.1.0.0" maxVersionTested="10.3.0.0" />
</Dependencies>
```

Primjer 3.6: Ograničenje obzirom na vrstu uređaja

```
<Dependencies>
    <TargetDeviceFamily Name="Windows.Universal"
                minVersion="10.0.10240.0" maxVersionTested="10.0.10586.0" />
</Dependencies>
```

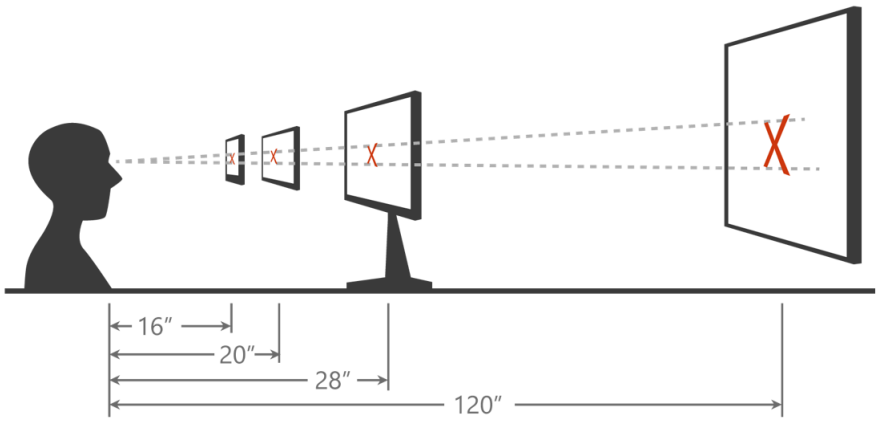
Primjer 3.7: Ograničenje obzirom na verziju platforme

```
<Dependencies>
    <TargetDeviceFamily Name="Windows.Universal"
                minVersion="10.0.10240.0" maxVersionTested="10.0.14393.0" />
    <TargetDeviceFamily Name="Windows.Mobile"
                minVersion="10.1.0.0" maxVersionTested="10.1.0.0" />
</Dependencies>
```

Primjer 3.8: Dodatno ograničenje na Mobile uređaje

3.2.2 Adaptivni dizajn

Osim prilagodbe samog programskog koda, prilikom izrade univerzalne aplikacije se mora voditi računa i o prilagodbi korisničkog sučelja različitim uređajima koji će istu pokretati. Naravno, nerazumno je očekivati da će sve aplikacije biti namijenjene svim uređajima odnosno da su svi Windows 10 uređaji pogodni za sve vrste aplikacija, no UWP aplikacija na svim uređajima za koje je namijenjena mora pružati adekvatno korisničko iskustvo. Prije svega, iz daljnjih razmatranja bi trebalo isključiti IoT i Holographic uređaje koji zbog svojih karakteristika i namjene imaju sučelje koncipirano na drugačiji način ili ga uopće nemaju. Dakle, promatrat ćemo pametne telefone, tablete te prijenosna i stolna računala. Iako isti tipovi uređaja često mogu imati ekrane različitih veličina, rezolucija ili razlučivosti, oči korisnika se nalaze na jako sličnim udaljenostima od samog ekrana, a ta udaljenost ovisi o namjeni samog uređaja kako možemo vidjeti na slici 3.9.



Slika 3.9: Optimalna udaljenost očiju od ekrana u inčima - [6]

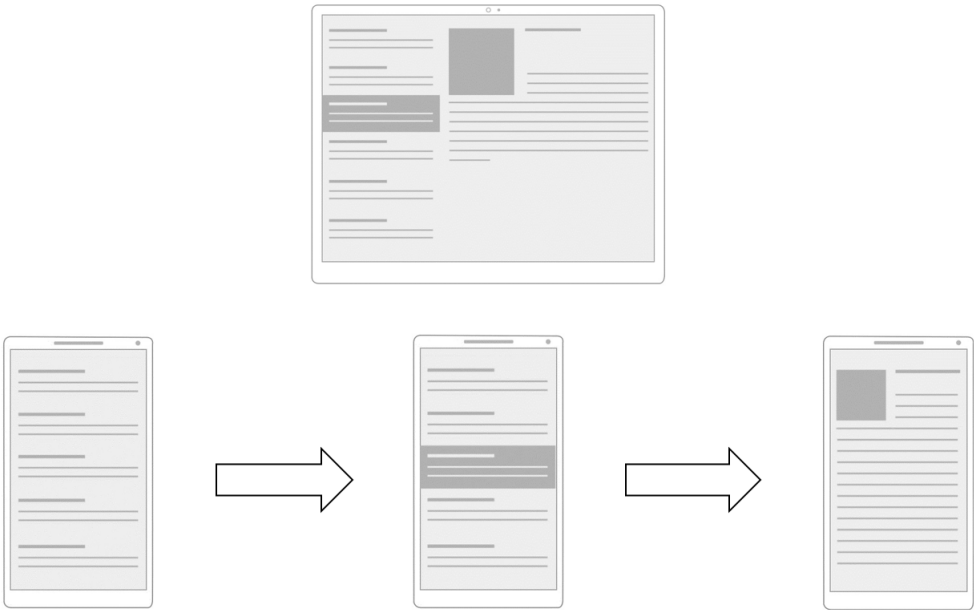
Kako korisniku pružiti najbolje iskustvo bez obzira na uređaj koji koristiti? Za početak, trebali bi odbaciti veličine zaslona i rezolucije te za referentne točke uzeti veličine elemenata na sučelju relativno u odnosu na udaljenost s koje se to sučelje promatra (kao npr. 'X' na prethodnoj slici). Zato prilikom dizajniranja i izrade sučelja univerzalne aplikacija treba voditi računa samo o efektivnim pikselima. Efektivni piksel je grupa jednog ili više piksela koja čini jedinstvenu cjelinu nastala prilikom skaliranja operativnog sustava. Algoritmi skaliranja u obzir uzimaju sve karakteristike zaslona, vrstu uređaja, ali i korisničke preferencije.

Iako je baziranje dizajna na efektivnim pikselima sasvim dovoljno za neke vrste aplikacija, praksa je pokazala kako se sučelje ipak treba prilagoditi različitim vrstama uređaja. Imajući to na umu, svaki tip uređaja bi trebao imati reprezentativni primjerak oko kojeg se gradi osnovni dizajn sučelja poput tableta sa zaslonom dijagonale 8″ ili prijenosnim računalom standardne veličine pritom uzimajući u obzir i njihove orijentacije. Za granice tih sučelja

najčešće se odabiru minimalna visina ili širina izražena u efektivnim pikselima. Tehnike koje se ovdje koriste dijele mnoge sličnosti s tehnikama koje koriste web developeri, takozvanih 6R:

- *Reposition*
- *Resize*
- *Reflow*
- *Reveal*
- *Replace*
- *Rearchitect*

Zadaća repozicioniranja sučelja je u svakom trenutku osigurati najintuitivnije sučelje prilagođeno uređaju i što je moguće efikasniji raspored kontrola. Tehnika mijenjanja veličina sadržaja koristi „višak prostora“ kako bi određenom sadržaju npr. slikama dala više prostora na velikim ekranima ili smanjila sporedni sadržaj na manjim ekranima. Treća tehnika je klasično prilagođavanje elemenata sukladno širini ekrana - npr. dva usporedna stupca teksta i slika na postaju jedan kada zaokrenemo tablet i smanjimo horizontalnu rezoluciju. Ovisno o veličini ekrana, neke se dodatne opcije mogu smjestiti u dodatne izbornike ili mogu promijeniti oblik. Ovisno o namjeni aplikacije ponekad je potrebno raspored elemenata prilagoditi uređaju kao bi se olakšala navigacija ili pridala veća pažnja nekim funkcionalnostima aplikacije. Zadnji pristup podrazumijeva izradu potpuno drugačijih korisničkih sučelja za različite ekrana. On se najčešće koristi kada je aplikacija prvotno bila zamišljena za samo jedan tip uređaja, a kasnije je dodana podrška i za druge uređaje.

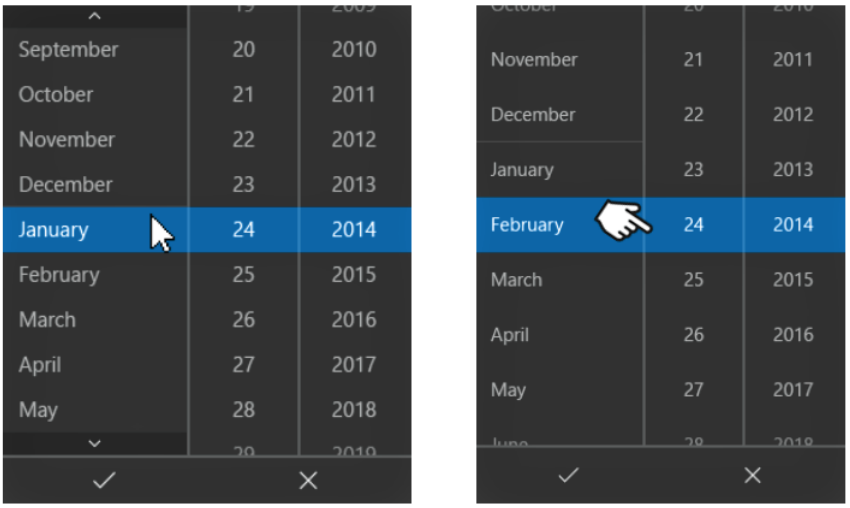


Slika 3.10: Adaptivno sučelje

Microsoftov tim zadužen za dizajniranje sučelja Windowsa 10 i aplikacija ugrađenih u sustav je na osnovnu tih tehnika predložio tri pravca prilagodbe sučelja ekranima različitih

karakteristika. To su respozivni dizajn, adaptivni dizajn te dizajn po mjeri. Respozivni dizajn se temelji na reflow i resize tehnikama, a najpogodniji je za aplikacije čija je glavna namjena konzumiranje digitalnih sadržaja. Korisničko iskustvo u ovom slučaju ostaje gotovo identično - samo se količina sadržaja mijenja. Drugi, zahtjevniji, pristup koristi tehnike reposition, reveal i replace. Zahvaljujući njima, korisnik će na sučelju uvijek imati optimalnu količinu sadržaja i broj kontrola, a sučelje će najbolje odgovarati trenutno uređaju i načinu interakcije. Adaptivni dizajn odgovara najvećem broju aplikacija zbog svoje modularnosti i jednostavnog načina navigacije kroz sučelje. Posljednja opcija je ujedno i opcija koja bi trebala biti zadnji izbor. Dizajn po mjeri znači da za svaki tip uređaja koji će aplikacija podržavati radimo zasebno sučelja za neke dijelove aplikacija. Ponekad aplikacija ima toliko funkcionalnosti da bi prilagodba ekrana rezultirala velikim utroškom vremena ili lošijim korisničkim iskustvom pa je jednostavnija varijanta sučelja razviti za svaki uređaj posebno.

Boljem korisničkom iskustvu, osim dizajna sučelja, pridonose i prilagođene kontrole. Uzmimo kao primjer kontrolu za odabir datuma na slici 3.11. Na lijevoj strani vidimo kako kontrola izgleda kada se koristi na ekranu osobnog računala. Vrijednosti su zgusnutije, svaku od vrijednosti možemo odabrati putem tipkovnice ili miša. Na desnoj strani slike se nalazi prikaz kontrole na ekranima osjetljiv na dodir. Prostor između vrijednosti je puno veći kako bi interakcija bila što lakša, a umjesto navigacijskih strelica vrijednosti mijenjamo jednostavnim povlačenjem prsta. Drugi primjer prilagodbe kontrola su polja za unos teksta. Ukoliko uređaj posjeduje ekran osjetljiv na dodir prilikom fokusiranja tog polja dolazi do podizanja virtualne tipkovnice s rasporedom prilagođenim tom polju. Dozu osobnosti svom operativnom sustavu korisnik može dati odabirom primarne boje i boje pozadine, što velik broj kontrola uzima u obzir te pridonosi vizualnoj integriranosti aplikacije.



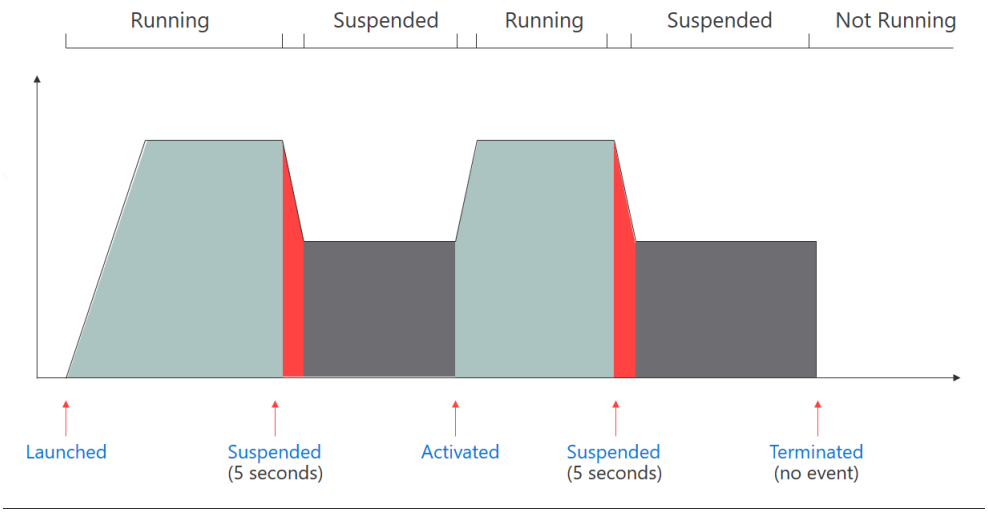
Slika 3.11: Kontrola za odabir datuma

4 UWP aplikacije

Nakon pregleda značajki platforme koja pokreće aplikacije, u ovom će poglavlju biti riječi o karakteristikama tih aplikacija. Iako će najveći broj univerzalnih aplikacija biti pokretan na stolnim i prijenosnim računalima, njihove karakteristike su puno bliže mobilnim aplikacijama.

4.1 Životni ciklus aplikacije

Desktop programi koje svakodnevno koristimo imaju dva stanja – aktivno i neaktivno. Oni iz neaktivnog stanja prelaze u aktivno nakon pokretanja i u njemu ostaju sve dok ih ne zatvorimo ili dođe do pogreške te ih sustav mora prisilno zatvoriti. Minimiziranje prozora ili gubitak fokusa u većini slučajeva nema nikakav utjecaj na njihovo ponašanje. Životni ciklus jedne UWP aplikacije je bitno drugačiji – one se mogu nalaziti u tri stanja. To su pokrenuto, suspendirano i nepokrenuto stanje, a mimo ovih stanja se nalaze akcije koje mogu vršiti u pozadini neovisno o stanju aplikacije te će o njima više riječi biti kasnije.



Slika 4.1: Životni ciklus UWP aplikacije - [6]

Aplikacija prelazi u pokrenuto stanje uz događaj `Launched`. Pri pokretanju aplikacije mogu se dobiti informacije o prethodnom stanju aplikacija te načinu na koji je aplikacija sada pokrenuta i je li pritom dobila neke podatke od sustava ili druge aplikacije. Na Windows 10 Mobile i IoT uređajima samo jedna aplikacija može biti u pokrenutom stanju, dok na ostalim uređajima može biti aktivno više aplikacija, ali samo dok su u punom, neminimiziranom, prikazu. U ovom stanju aplikacije se ponašaju kako je predviđeno, sve deklarirane funkcionalnosti su im dostupne i one su u potpunosti učitane u memoriju.

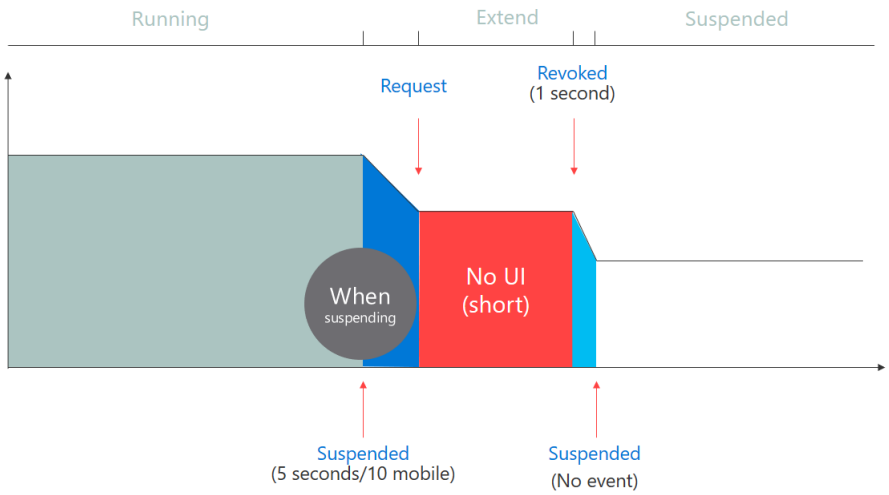
Zamislimo sad da korisnik minimizira aplikaciju ili se vrati na početni zaslon. Pritom se okida događaj `Suspending` i aplikaciji se pruža prilika da unutar 5 sekundi (10 sekundi na Windows 10 Mobileu) spremi podatke, završi započete radnje i pripremi za suspendirano stanje. U ovom intervalu sučelje aplikacije nije prikazano, no aplikacija i dalje ima punu funkcionalnost. Po isteku ovog intervala aplikacija ulazi u suspendirano stanje u kojem ne dolazi do izvršavanja nikakvog programskog koda te jednostavno miruje u memoriji i čeka na ponovnu aktivaciju.

Ako se korisnik opet vrati na ovu aplikaciju ona se opet vraća u pokrenuto stanje i dolazi do `Resuming` događaja. Pri prijelazu između ova dva stanja dobra praksa je osvježiti sve prethodno dohvaćane podatke, provjeriti dostupnost resursa te učitati navigacijsku povijest aplikacije. U slučaju da je aplikacija bila suspendirana duži vremenski period ponekad je bolje korisniku ponuditi opciju da se aplikacija pokrene "od nule".

No scenarij iz prethodnog ulomka se ne mora uvijek dogoditi. U slučaju da sustavu ponestane memorije ili baterije, on će se pokušati zaštititi terminiranjem aplikacija koje su u suspendiranom stanju. Do terminacije može doći i ako korisnik ručno zatvori aplikaciju ili ako dođe do gašenja sustava. Aplikacija nije svjesna ovog događaja, tj. informaciju da je terminirana može dobiti isključivo prilikom sljedećeg pokretanja, stoga spremanje pri suspendiranju biva još važnije. U tom slučaju bi podatke iz prošle sesije vratiti samo onda ako je sustav terminirao aplikaciju koja je bila u pozadini.

4.2 Produljeno izvođenje

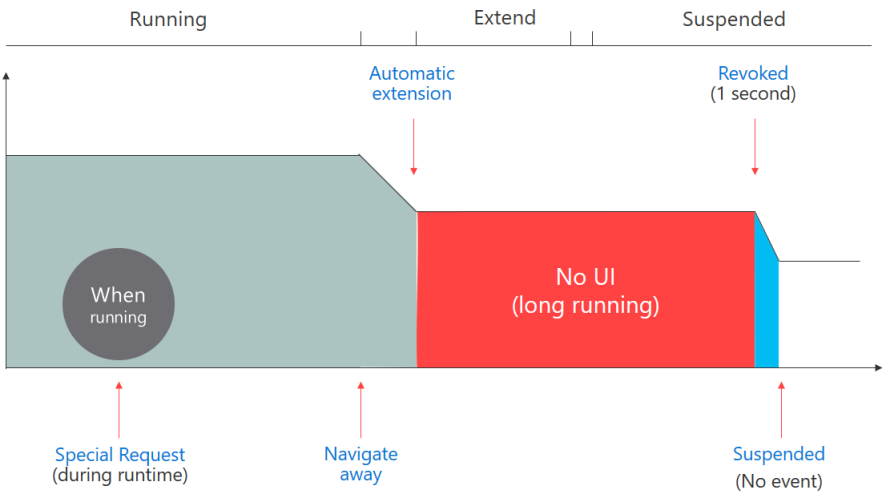
Iako većina aplikacija dobro funkcionira s prethodnim životnim ciklusom, nekima takvo ponašanje smanjuje funkcionalnosti. U tu svrhu aplikacija može koristiti koncept produljenog izvođenja. Produljeno izvođenje aplikaciji omogućava modificiranje vlastitog životnog ciklusa ako to okolnosti i trenutno stanje operacijskog sustava dopuštaju.



Slika 4.2: Produljavanje izvođenje tijekom `Suspending` događaja - [6]

Prvi način korištenja toga je produljenje vremena aktivnosti prilikom `Suspending` događaja. Neke aplikacije prilikom prelaska u pozadinski način rada moraju spremiti velike količine podataka te im predefinirani interval od 5/10 sekundi nije dovoljan za spremanje svih podataka. Aplikacija u tom slučaju od sustava može tražiti dopuštenje za produljenje tog intervala. Ako sustav dopusti aplikaciji produljeno izvođenje, ona će biti u mogućnosti spremiti sve podatke, a u suprotnom će imati 5 odnosno 10 sekundi da spremi osnovne podatke. Nije definirano koliko će produljeno izvođenje trajati jer to prvenstveno ovisi o stanju sustava, a aplikacija će dobiti notifikaciju 1 sekundu prije prelaska u suspendirano stanje.

Drugi način produljenog izvođenja je konceptualno bitno drugačiji. Neke aplikacije, poput onih navigacijskih, trebaju cijelo vrijeme biti aktivne. Takve aplikacije za vrijeme dok



Slika 4.3: Dugotrajno produljeno izvođenje - [6]

su u aktivnom stanju moraju poslati zahtjev sustavu za produljeno izvođenje koji uključuje i razlog produljenog izvođenja. Ako im sustav odobri zahtjev, one nastavljanju funkcionirati na isti način kad uđu u suspendirano stanje. Dakle, suspendirano stanje ovih aplikacija je istovjetno pokrenutom stanju, a jedina razlika je što korisnik ne vidi sučelje aplikacije. Isto kao i u prvo slučaju, produljeno izvođenje može biti ukinuto u bilo kojem trenutku uz prethodnu najavu.

4.3 Pozadinski zadaci

Ponekad aplikacije trebaju vršiti neke akcije periodički ili kao odgovor na neki specifični događaj. Te akcije najčešće trebaju biti neovisne o trenutnom stanju aplikacije te se zato odvajaju u zasebne komponente. Pozadinski zadatak (*engl. background task*) je zaokružen skup naredbi koji se odvija neovisno od aplikacije, a svaka aplikacija može imati više pozadinskih zadataka. Reprezentativni primjer toga je aplikacija vremenske prognoze i njena pločica koja se ažurira svakih pola sata kako bi prikazala aktualne podatke.

Kako dolazi do pokretanja pozadinskih zadataka? Svaki pozadinski zadatak se može aktivirati na razne načine: primitkom push notifikacije, istekom određenog vremenskog intervala, promjenom lokacije, kao reakcija na promjenu nekog od parametra uređaja, ali i direktno iz pokrenute aplikacije. Oni mogu biti jednokratni – npr. podsjetnik baziran na geolokaciji ili višekratni – dohvat podataka svakih pola sata. Pozadinski zadatak, kao i sama aplikacija, ima svoj životni ciklus. Prilikom aktiviranja poziva se `Run` metoda koja je obavezna pri implementaciji. Pritom zadatak može vidjeti koji je događaj bio uzrok njegovog pozivanja i, u skladu s time, adekvatno reagirati. Svaki se pozadinski zadatak prvo mora registrirati, a u isto vrijeme ne mogu biti registrirana dva zadataka s istim imenom. Osim okidača, pri registriranju pozadinskih zadataka možemo i definirati uvjete koji moraju biti zadovoljeni da bi se zadatak izvršio. Neki od tih uvjeta su spojenost na izvor energije, dostupnost mrežne konekcije ili zauzetost sustava.

Pozadinski zadaci imaju ograničeno vrijeme izvođenja koje iznosi 25 sekundi u idealnim uvjetima i pritom na raspolaganju imaju dovoljno resursa za normalno funkcioniranje. Ako je pak pozadinska radnja zahtjevnija i zahtijeva dulji period izvođenja ili više resursa od

trenutne norme sustava ona će se izvesti samo onda kada neće narušiti funkcioniranje sustava. Drugim riječima, ne postoji da će resursno zahtjevniji zadaci zaista izvršiti. Osim ovih ograničenja pri osmišljavanju aplikacije i pripadnih pozadinskih zadataka uvijek valja imati na umu da korisnik u svakom trenutku može aplikaciji ukinuti ili dati pravo rada u pozadini. Također, stanje uštede baterije (aktivirano od strane korisnika ili sustava) brani rad u pozadini svim aplikacijama osim onima kojima je korisnik dao eksplicitno dopuštenje za rad u svim uvjetima.

4.4 Pohrana podataka

U ovom potpoglavlju će biti riječi o svim uobičajenim načinima spremanja podataka. Za razliku od klasičnih Windows programa, UWP aplikacije rade u sandboxu što za posljedicu ima bitno smanjenu mogućnost manipulacije nad datotečnim sustavom. Ovisno o aplikaciji, podaci potrebni za njen rad se najčešće serijaliziraju te spremaju u JSON[9] odnosno XML[10] obliku ili, ako je to potrebno, u lokalnu SQLite[11] bazu. No gdje spremiti JSON datoteku s podacima ili neki multimedijski sadržaj?

Prva lokacija je direktorij u kojem se instalirao paket aplikacije – kao u klasičnim Windows aplikacijama. Ovakav se pristup rijetko koristi zbog brojnih mana poput utjecaja na performanse ili opasnosti od manipulacije podacima od strane korisnika. Sljedeća lokacija je izolirana pohrana aplikacije. Datotečni sustav aplikacije je nedostupan korisniku te je stoga pogodan za pohranu većinu podataka. Također, datoteke spremljene na ovu lokaciju se ažuriraju samo kada je to nužno. No kako dijeliti podatke ako aplikaciju koristimo na više uređaja? Ako aplikacija ne koristi nekakav REST servis, željeni učinak se može postići korištenjem Roaming Foldera. Ovo spremište se nalazi unutar izolirane pohrane aplikacije i ograničeno je na 100 KB te je stoga u njemu preporučljivo spremati samo najbitnije podatke i postavke. Sinkronizacija između više instanci aplikacije se vrši putem Azurea, a u slučaju prelaska memorijskog ograničenja sinkronizacija prestaje. Posljednja klasična opcija je korištenje privremenog direktorija koji se briše nakon zatvaranja aplikacije.

U nekim situacijama naša aplikacija kreira ili koristiti datoteke i direktorije koji bi trebali biti dostupni korisniku. Uz odgovarajuće deklaracije aplikacija može čitati iz direktorija „Slike“, „Glazba“ i „Videozapisi“ i pisati u istim tim direktorijima uz dodatak direktorija „Preuzimanja“. U slučaju da je uređaj spojen na mrežu, aplikacija može vršiti sve ranije navedene akcije i na udaljenim mrežnim lokacijama ovisno o pravima. Ponekad je dobro dati korisniku na izbor u koji će direktorij spremiti podatke i to se vrši preko klasičnog dijaloga za odabir direktorija/datoteka. Valja istaknuti kako pozivanje bilo kakvog dijaloga na Windows 10 Mobile sustavu rezultira suspendiranjem aplikacije, a povratna se informacija dobiva nakon vraćanja u aktivno stanje.

4.5 Ostale značajke

4.5.1 Package.appxmanifest

Jedna od najznačajnijih datoteka u cijeloj aplikaciji je `Package.appxmanifest`. U njoj su pohranjene sve najbitnije informacije o aplikacije te iz nje sustav dobiva sve konfiguracijske

[9] format za razmjenu podataka koji potječe od JavaScripta

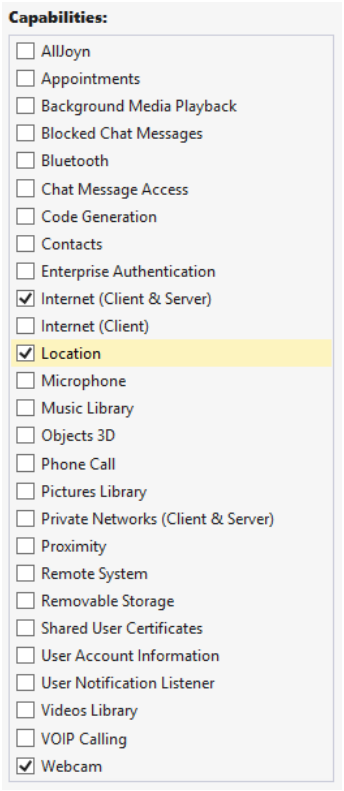
[10] format za razmjenu podataka

[11] lagani sustav za upravljanje bazom podataka često korišten pri razvoju mobilnih aplikacija

podatke. Ti su podaci pohranjeni u XML obliku, a mogu biti prikazani i putem jednostavnije grafičkog sučelja koje se sastoji od sljedećih kartica:

- *Application*
- *Visual Assests*
- *Capabilities*
- *Declarations*
- *Content URIs*
- *Packaging*

Prva kartica sadrži osnovne podatke o aplikaciji poput imena, predefinirane kulture i konfiguracije živih pločica te obavijesti na zaključanom zaslonu. Druga kartica služi za dodjeljivanje vizualnog identiteta aplikaciji putem ikone pločica, loga aplikacija, početnog zaslona kao i slika koje će se prikazati u trgovini aplikacija.



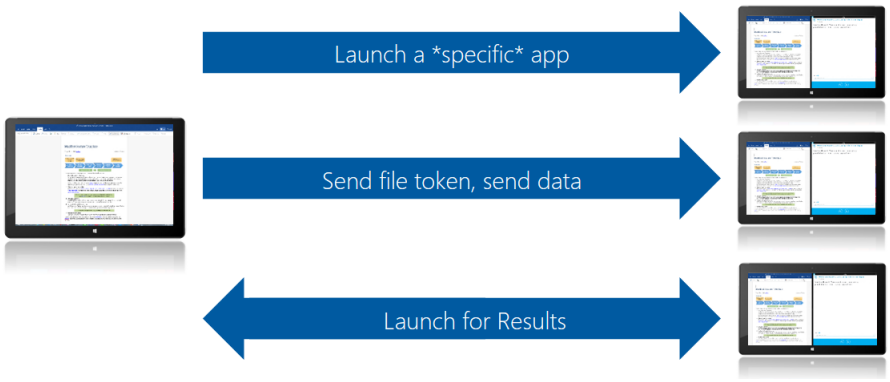
Slika 4.4: Sadržaj Capabilities kartice

U kartici Capabilities se nalazi popis svih mogućnosti sustava odnosno uređaja koje možemo koristiti. Ako želimo koristiti neku od tih mogućnosti (prikaz svih je na slici 4.4) moramo ju odabrati ovdje, a ako to ne napravimo aplikacija će se prisilno zatvoriti pri pozivanju te funkcionalnosti. Označene mogućnosti su prikazane i u detaljima svake aplikacije koja se nalazi u trgovini tako da korisnici prije preuzimanja mogu vidjeti što određena aplikacija zahtijeva. Za razliku od prijašnjih verzija Windowsa, korisnik u bilo kojem trenutku može zabraniti aplikaciji pristup nekoj od mogućnosti. Sljedeća je kartica, kao što i njeno ime kaže, zadužena za dodjeljivanja deklaracija aplikaciji. Deklaracije definiraju protokole

koje aplikacija podržava, pozadinske zadatke, načine pokretanja, podržane načine međuaplikacijske komunikacije (o kojoj će biti više riječi u potpoglavlju 4.5.2) i druge složene funkcionalnosti. Posljednje dvije sekcije sadrže popis web stranica koje mogu direktno komunicirati s aplikacijom odnosno broj verzije aplikacije, broj aplikacijskog paketa te podatke o autoru.

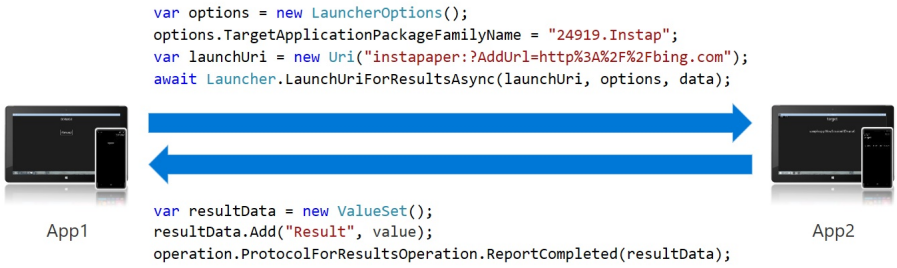
4.5.2 Međuaplikacijska komunikacija

UWP aplikacije, osim sa sustavom, mogu komunicirati i s drugim UWP aplikacijama. Prilikom razvoja većih i naprednijih aplikacija često dolazi do potrebe za izradom modula zaduženih za čitanje određenih datoteka, modula za komunikaciju ili multimediju. Osim u specifičnim slučajevima, funkcionalnosti koje nadilaze osnovnu ideju i mogućnosti aplikacije trebaju biti izvedene putem drugih aplikacija. Npr. nepotrebno je trošiti vrijeme na izradu komponente za čitanje PDF dokumenata unutar aplikacije ako je jedina nužna funkcionalnost čitanje tog dokumenta.



Slika 4.5: Dijeljenje datoteke - [6]

U tom slučaju aplikacija prvo šalje obavijest o tipu datoteke, zatim sustav pronalazi sve aplikacije koje su registrirane za obradu tog tipa datoteke. U slučaju da postoji samo jedna takva aplikacija onda se automatski pokreće, a ako postoji više takvih aplikacija pokazuje se njihov popis. Ako ne postoji takva aplikacija otvara se trgovina gdje su prikazane sve aplikacije koje podržavaju dani format. U slučaju da sadržaj želimo otvoriti sa specifičnom aplikacijom, to možemo naglasiti prilikom upita sustavu. Odredišna aplikacija ovakvim protokolom dobiva token preko kojeg dolazi do podataka koje treba učitati.



Primjer 4.6: Pokretanje aplikacije za obradu - [6]

Drugi način aktivacije je onaj u kojem se drugoj aplikacija predaje skup podataka, a ona na temelju tih podataka treba vratiti rezultat. Kao što se može vidjeti u primjeru 4.6, ovdje

se radi o direktnoj komunikaciji između dvije aplikacije. Osim ova dva načina, aplikacije istog izdavača mogu komunicirati na još jedan način – putem dijeljenog direktorija. Taj se direktorij kreira s instalacijom prve aplikacije izdavača koja podržava dijeljenje podataka, a briše pri deinstalaciji posljednje aplikacije. Ovaj se način pohrane pokazao dobrim za pohranu ikona, korisničkih postavki i konfiguracijskih parametara koji se dijele između više aplikacija.

4.5.3 Continuum

S Windowsima 10 Microsoft je predstavio Continuum – funkcionalnost koja Windows 10 Mobile uređaje približava klasičnim računalima. Uređaji koji zadovoljavaju određene hardverske zahtjeve mogu se bežično ili putem odgovarajućih postolja spojiti na vanjski ekran. Nakon spajanja uređaja Continuum može raditi na dva načina. Prvi od njih je način u kojem zaslon mobilnog uređaja služi isključivo kao pokazivač dok drugi način omogućuje korištenje dvije aplikacije istovremeno.

Kako bi prilagodili aplikaciju ovakvom načinu rada, dovoljno je slijediti principe adaptivnog dizajna, a operativni sustav će napraviti ostatak. Nažalost, ovakav pristup znači da će korisničko iskustvo biti narušeno ukoliko je sučelje zasebno dizajnirano za pojedinu obitelj uređaja budući da se aplikacija neće moći adekvatno skalirati. U slučaju da ne želimo da aplikacija funkcionira u Continuum načinu, to možemo ograničiti pri deklaraciji mogućnosti. Neke aplikacije, posebno one namijenjene reprodukciji sadržaja, mogu iskoristiti ovaj način rad za istovremeni prikaz na dva zaslona. Također, Continuum na pametne telefone donosi mogućnost otvaranja više prozora za jednu instancu aplikacije (ako aplikacija podržava tu funkcionalnost).

5 Izrada aplikacije

Nakon poglavlja koja su bavila pretežno teorijom i konceptima, u ovom poglavlju će biti prikazano kako izrada aplikacije izgleda u praksi na primjeru aplikacije DashCam. Ova je aplikacija namijenjenima vozačima automobila i prvenstveno služi za snimanje vožnje. Kamere za snimanje vožnje su postale popularne početkom ovog stoljeća u istočnoj Europi. Uzrok toga je bio velik broj pokušaja prevara auto osiguranja te iznuda odšteta. Kako bi tome stale na kraj, osiguravajuće kuće su počele s poticanjem klijenata na ugradnju kamera te su im u zamjenu nudile nižu cijene osiguranja, a neke su išle toliko daleko da su zahtijevali instalaciju kameru kod svakog klijenta.

U ostatku Europe je situacija znatno drugačija – neke zemlje poput Austrije, Luksemburga i Španjolske u potpunosti zabranjuju korištenje kamera, neke dozvoljavaju uz određena ograničenja (snimanje zvuka, položaj kamere, rezolucija, registracija), a neke poput Ujedinjenog Kraljevstva nemaju restrikcije. Stav prema kamerama za snimanje prometa je usko vezan za zakone o zaštiti osobnih podataka pojedine države. Snimke u nekim državama mogu služiti kao dokaz na sudu, dok je u nekima to slučaj samo u iznimnim situacijama. U Hrvatskoj je pravna regulativa vezana za snimanje prometa neprecizna, ali stav je da vozač koji snima promet nije u prekršaju. U slučaju prometne nesreće snimka će biti priložena kao dokazni materijal, no ne nužno i kao dokaz tijekom sudskog postupka.

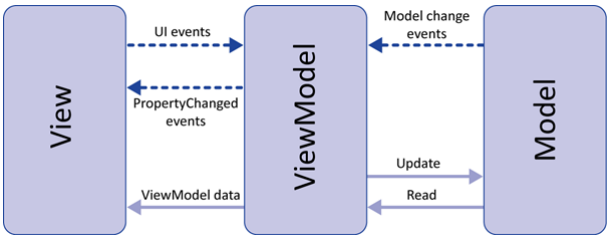
Aplikacija se sastoji od 4 sučelja, a pri dizajniranju istih je vođeno računa o različitim veličinama zaslona. Prvo sučelje omogućava snimanje videozapisa te prikazuje trenutno sučelje, a sljedeće sučelje omogućuje reprodukciju snimljenih videozapisa. Treće sučelje služi za dodavanje lokacija na kojima će snimanje automatski započeti dok posljednje sučelje nudi mogućnost izmjene postavki.

Ova je aplikacija prvenstveno namijenjena pametnim telefonima, no može biti korištena i na ostalim uređajima koje pokreće Windows 10. Prilikom razvoja aplikacija je testirana na pametnim telefonima Xiaomi Mi 4 Windows i Microsoft Lumia 950, prijenosnim računalima HP ProBook 470 i HP ProBook 650 te tabletu Lenovo Miix 3.

5.1 MVVM

Prilikom izrade UWP aplikacija, barem onih složenijih, koristi se Model-View-ViewModel design pattern. Prilikom razvoja bilo kakvih aplikacija pa tako i UWP aplikacija s porastom broja funkcionalnosti raste i količina programskog koda. Samim time održavanje aplikacije i dodavanje novih funkcionalnosti postaje sve kompliciranije. Zato bi trebalo podijeliti kod u manje, smislene cjeline te odvojiti poslovnu logiku i prezentacijski kod. Kako bi to postigli, prilikom razvoja većine Windows koristi se MVVM. Budući da ovaj koncept nije uvijek lako realizirati, često se koriste frameworci koji olakšavaju implementaciju, a neki od njih su Prism, MVVM Light i Caliburn.Micro (koji je korišten u izradi ove aplikacije).

Model predstavlja model podatka koji najbolje opisuje problem koji aplikacija želi riješiti. Pod tim pojmom se prvenstveno misli na sve klase predstavljaju određeni poslovni proces. Osim samih klasa, tu spadaju i repozitoriji koji sadrži te podatke, a sam bi model trebao biti neovisan o načinu na koji će podaci biti prikazani.



Slika 5.1: Shema MVVM-a - [5]

View je dio aplikacije koji prikazuje podatke koji se nalaze u modelu. Korisničko sučelje UWP aplikacija se najčešće sastoji od XAML kontrola kao što su `Page` ili `UserControl`. View uz sebe najčešće ima i code behind – C# klasu koja ima direktan pristup svim elementima XAML kontrole. Prilikom korištenja MVVM-a ona bi se trebala koristi isključivo kao podrška prikazu podataka, a ne kao sloj koji manipulira podacima[12].

ViewModel se nalazi između dva prethodno navedena sloja i pruža određenu apstrakciju između njih. Njegova je glavna zadaća podatke iz modela prilagoditi u oblik pogodan za View i reagirati na akcije koje će te podatke mijenjati. ViewModel, za razliku od code behinda, ne komunicira se kontrolama koje se nalaze na sučelju.

5.1.1 Data binding

Kako bi MVVM koncept oživio, View mora znati koje podatke u tom trenutku sadrži ViewModel, a ViewModel mora reagirati na akcije koje se događaju na Viewu. Data binding je značajka XAML-a koja omogućuje komunikaciju između podataka i XAML kontrole koja treba prikazati te podatke. Prije samog data bindinga potrebno je specificirati koji će podaci biti izvor za koju kontrolu, tj. koji će View gledati na koji ViewModel. Svaki XAML element ima property `DataContext` koji definira koji će kontekst biti korišten prilikom data bindinga. Budući da se DataContext propagira na sve ugniježđene elemente, potrebno je postaviti DataContext korijenskog elementa na klasu koja predstavlja ViewModel. Prilikom korištenja Caliburn.Micra DataContext se postavlja prema imenu: npr. MainView će automatski poprimiti DataContext MainViewModela.

```
<TextBlock Text="{Binding Speed, Mode=TwoWay}"
           FontSize="36"/>

<TextBlock Text="{Binding Settings.SpeedUnit.DisplayName, Mode=TwoWay}"
           FontSize="12"/>
```

Primjer 5.2: Jednostavni data binding

Nakon što je DataContext postavljen, XAML kontrole se mogu vezati na propertye i druge dijelove ViewModela kao što je prikazano u primjeru 5.2. Osim standardnog jednosmjernog, data binding može biti jednokratan i dvosmjeran. U slučaju da na sučelju trebamo koristiti jedan tip podataka, a u ViewModelu drugi, možemo koristiti klasu koja ima metode za pretvorbu vrijednosti poput one u primjeru 5.3.

[12] u jednostavnijim aplikacijama code behind može zamijeniti ViewModel

```
public class BoolToVisibilityConverter : IValueConverter
{

    public object Convert(object value, Type targetType, object parameter, string culture)
    {
        var val = (bool)value;
        var param = parameter as string;

        if (param == "!")
        {
            val = !val;
        }

        if (val)
        {
            return Visibility.Visible;
        }

        return Visibility.Collapsed;
    }


    public object ConvertBack(object value, Type targetType, object parameter, string culture)
    {
        var val = (Visibility)value;
        var param = (string)parameter;

        if (val == Visibility.Visible)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

Primjer 5.3: Pretvorba bool ↔ Visibility

Kako View zna da je došlo do promjene podataka u ViewModelu i da mora osvježiti prikaz? Upravo za tu namjenu je predviđen interface[13] `INotifyPropertyChanged`. Njegovom implementacijom u klase koje koristi ViewModel dobivamo mogućnost slanja notifikacije korisničkom sučelju. Po primitku te notifikacije korisničko sučelje zna da je došlo do promjene podataka te da mora ponovno izvršiti data binding.

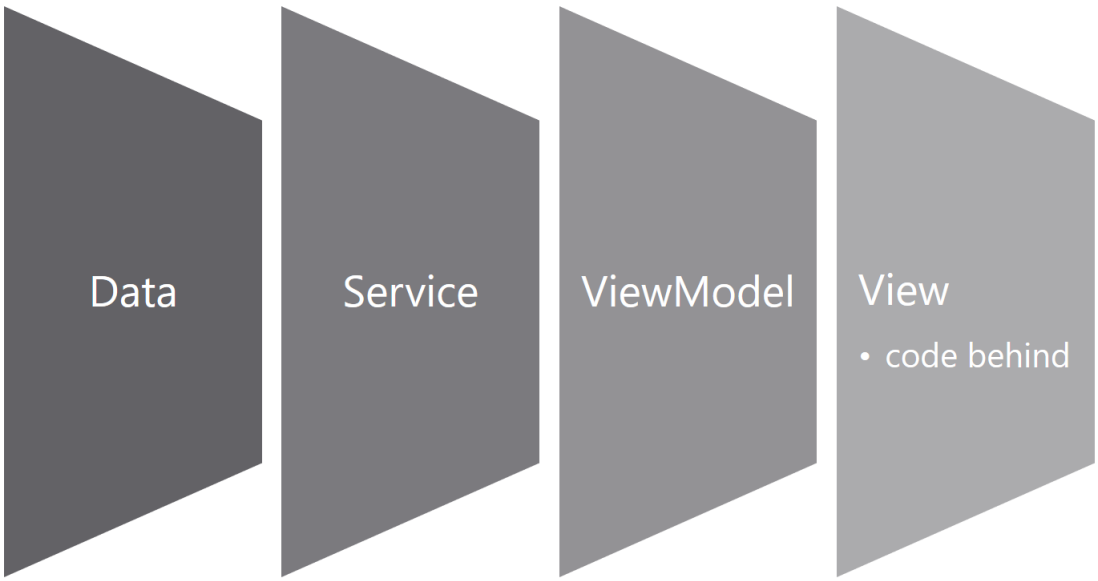
Preostaje još vidjeti kako kontrole na korisničkom sučelju mogu pozivati metode koje se nalaze u odgovarajućem ViewModelu. Svaka od kontrola u svojoj definiciji sadrži velik broj metoda koje su namijenjene praćenju različitih događaja (*engl. event handler*) poput promjene veličine, korisničke interakcije ili promjene vrijednosti. Budući da imaju direktan pristup kontrolama, metode koje se nalaze u code behindu mogu se direktno pretplatiti na te događaje. Kako bi prenijeli poruku do ViewModela poslužit ćemo se funkcionalnošću koju nam nudi Caliburn.Micro.

Recimo da želimo pozvati metodu svaku put kad netko dotakne kontrolu mape. To ćemo napraviti dodavanjem ove linije koda kontroli: `cm:Message.Attach="[Event MapTapped] = [Action MapControlTapped($source, $eventArgs)]"`. Dakle, nakon svakog dodira pozivat će se metoda `MapControlTapped` koja se nalazi u ViewModelu i prima dvije vrijednosti: XAML kontrolu koja je izvor događaja kao i argumente događaja.

[13] prototip klase ili strukture

5.2 Arhitektura i konfiguriranje aplikacije

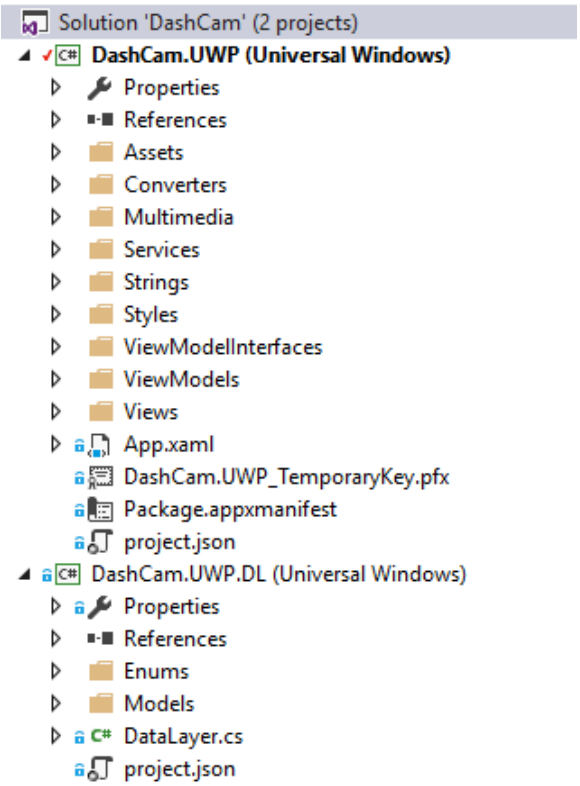
U ovom će potpoglavlju na primjeru DashCam aplikacije biti prikazani osnovni koraci pri izgradnji jedne UWP aplikacije te način implementacije prethodno navedenih koncepata. Preduvjeti za razvoj univerzalne Windows aplikacije su osobno računalo s operativnim sustavom Windows 10, Visual Studio 2015 i odgovarajuća verzija Windows SDK-a. Uz to je poželjno na računalu uključiti „Developer mode“ kako bi aplikaciju tijekom razvoja mogli testirati na računalu. U slučaju da aplikaciju želimo testirati i na Windows 10 Mobile sustavu, potrebno je priključiti pametni telefon ili preuzeti odgovarajuće emulatore.



Slika 5.4: Slojevita arhitektura

Za razliku od prethodnih verzija, sve univerzalne aplikacije se kreiraju na isti način, tj. koriste isti predložak. No prije početka same implementacije potrebno je odabrati arhitekturu najpogodniju za veličinu i namjenu aplikacije. Za aplikacije umjerene kompleksnosti s nekoliko sučelja slojevita arhitektura pruža najbolji omjer uloženog vremena i dobivenih prednosti. Na prethodnoj slici je predstavljena arhitektura DashCam aplikacije u kojoj se logika odvija kroz četiri sloja. Primijetimo kako je treći dio MVVM uzorka, Model, ovdje rastavljen na dva sloja. Prvi, Data, sloj je zadužen za čitanje i pisanje podataka iz/u memoriju uređaja. Service sloj ima malo kompleksniju zadaću – on služi kao poveznica između ViewModela i Data sloja, ali i kao privremeni repozitorij za podatke koje koristi više ViewModela. Osim toga, preko tog sloja ViewModeli najčešće pozivaju različite pomoćne funkcije ili vrše akcije koje nisu usko vezano uz poslovnu logiku. Za potrebe kompleksnijih aplikacija, ova se arhitektura može nadograditi s još slojeva kako bi bolje odvojili različite procese i podatke. Prvi od njih bi se trebao nalaziti između dva prethodno navedena sloja i služiti kao repozitorij iz kojega bi Service sloja dohvaćao i spremao podatke. U slučaju da aplikacija koristi podatke iz više izvora (npr. lokalna pohrana i podaci dobiveni pute web servisa) poželjno bi bilo odvojiti te izvore u zasebne module. Tada zadaća Data sloja postaje kombiniranje podataka u jedinstvenu cjelinu.

Prvi korak u izradi aplikacije je konfigurirati početnu klasu (budući da je i sama aplikacija XAML kontrola, `App.xaml.cs` možemo smatrati njezinim code behindom). Ovdje se nalaze



Slika 5.5: Projekt aplikacije DashCam

sve metode vezane uz životni ciklus aplikacije i konfiguraciju. U našem primjeru, budući da aplikacija koristi Caliburn.Micro, potrebno je definirati da klasa aplikacije nasljeđuje klasu `CaliburnApplication`. Osim već spomenutog automatskog postavljanja DataContexta, Caliburn.Micro nam pruža još nekoliko mogućnosti. Jedna od njih je bolja implementacija singletona – klasa koje imaju jednu instancu. Njih koristimo za klase koje poput onih koje predstavljaju Data i Service slojeve ili neke druge pomoćne klase. Time olakšavamo razvoj, ali i efikasnije koristimo resurse. Također, u ovoj klasi registriramo sve ViewModele koje ćemo koristiti u aplikaciji. Nakon toga bi trebalo kreirati posebne direktorije za ViewModele, Viewove, pomoćne klase i resurse, a dobra je praksa klase Data sloja kao i klase koje opisuju model odvojiti u zasebni projekt.

Iako ova aplikacija ima četiri sučelja, ona se sastoji od šest View-ViewModel parova. Pri pokretanju se prvo učita prazno sučelje, a zadaća odgovarajućeg ViewModela je za to vrijeme konfigurirati navigacijske servise i aktivirati sljedeće sučelje. Na njemu se nalazi lista ikona, a dodir na pojedinu ikonu aktivira odgovarajuće sučelje. Bez obzira na aktivni modul, jedan dio sučelja ostaje uvijek aktivan. Ovakvo ponašanje je postignuto korištenjem Conductora – ViewModela koji sadrži kolekciju ViewModela. Sučelje se zapravo sastoji od 5 različitih sučelja od kojih su uvijek vidljiva dva – glavno i sučelje trenutno aktivnog modula. Conductor ima mogućnost potpune kontrole nad svim ViewModelima koje sadrži, ali i trenutno aktivni ViewModel iz kolekcije može upravljati Conductorom koji ga sadrži te time dobivamo mogućnost dvosmjerne komunikacije u realnom vremenu.

Svaki ViewModel koji koristimo nasljeđuje baznu klasu imena `Screen` koja sadrži osnovne propertye i virtualne metode poput onih koje se pozivaju pri aktivaciji odnosno deaktivaciji. Sukladno potrebama, većina sadržaja bazne klase se može prilagoditi pojedinom ViewMo-

```
private Geopoint _currentLocation;
public Geopoint CurrentLocation
{
    get
    {
        return _currentLocation;
    }
    set
    {
        if (_currentLocation != value)
        {
            _currentLocation = value;
            NotifyOfPropertyChange(() => CurrentLocation);
        }
    }
}
```

Primjer 5.6: Implementacija notifikacija o promijeni

delu. Potom je potrebno dodati vlastite metode, propertye i fieldove te, u našem slučaju, pokazivač na klasu koja predstavlja Service sloj. Kako bi podržali data binding svaki property koji koristimo bi trebali implementirati način sličan u primjeru 5.6. Ako u Viewu koristimo kolekcije objekata, obično polje ili lista nisu dovoljni za normalno funkcioniranje MVVM-a. U tu svrhu koristimo `ObservableCollection` – kolekciju koja može prepoznati kada joj se doda objekt, promijeni objekt ili ako se iz nje ukloni objekt. Kako bi cijeli koncept data bindinga funkcionirao, kako je već spomenuto, potrebno je u sve klase modela koje smo generirali implementirati funkcionalnost obavještavanja o promjenama. To se najlakše može napraviti kreiranjem bazne klase kao u primjeru 5.7. Potom tu klasu trebaju naslijediti sve klase modela i implementirati obavješćivanje na način sličan onom prikazanom u primjeru 5.6.

```
public class BaseModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    public void NotifyPropertyChanged<T>(Expression<Func<T>> property)
    {
        if (PropertyChanged != null)
        {
            var memberExpression = property.Body as MemberExpression;
            PropertyChanged(this, new PropertyChangedEventArgs(memberExpression.Member.Name));
        }
    }
}
```

Primjer 5.7: Bazna klasa

Posvetimo sad malo vremena korisničkom sučelju. Poželjno je sve boje, fontove i stilove odvojiti u zasebni XAML rječnik. Istu stvar bi trebali napraviti i s vlastitim predlošcima i modifikacijama ugrađenih kontrola. Na taj način tim kod činimo urednijim, a u slučaju promjene, npr. osnovne boje ikona i teksta, to radimo samo na jednom mjestu. Prilikom kreiranja aplikacije stvorio se i Assets direktorij i on je pravo mjesto za pohranu slika i ikona koje naša aplikacija koristi.

U slučaju da želimo da naša aplikacija podržava najlakši način je da kreiramo direktorij imena „Strings“ i u njemu poddirektorije za svaki jezik, odnosno regiju koju želimo podržati. Unutar tih direktorija treba smjestiti rječnike (najbolje jedan za svako sučelje) i u njima de-

finirati ključeve (ključevi moraju biti jednaki u svim kulturama) i odgovarajuće vrijednosti. Aplikacija će tada, na osnovu jezika definiranog u sustavu, prikazivati tekst na odgovarajućem jeziku.

```
<TextBlock x:Uid="/Locations/LocationsTB"
           Text=""
           FontSize="24"
           Margin="0,0,0,12"/>
```

Primjer 5.8: Dohvat teksta iz rječnika

Potom dolazi do potrebe za kreiranjem raznih pomoćnih funkcija i izrada Data sloja. Nakon toga dolazi sama tehnička implementacija svih funkcionalnosti i kreiranje korisničkih sučelja. Opis svih procesa prilikom kreiranja ove aplikacije izlazi iz okvira rada, pa će stoga u ovom poglavlju biti prikazano samo nekoliko funkcionalnosti. Te su funkcionalnosti izdvojene kako bi se prikazali određeni koncepti koji su prethodno obrađeni ili zbog posebnog načina implementacije.

5.3 Videozapisi

Primarna funkcionalnost ove aplikacije je snimanje videozapisa tijekom vožnje. Prvi preduvjet za nju je deklariranje korištenja mikrofona i kamere uređaja. Za snimanje će biti korištena `CaptureElement` kontrola koja osim snimanja videozapisa omogućuje i slikanje. Nakon učitavanja kontrole prvo se mora inicijalizirati objekt koji će nadzirati kontrolu, pretplatiti se na odgovarajuće događaje te pokrenuti prikaz tražila kamere. Također, kako bi se osigurao nesmetan rad aplikacije, prilikom snimanja videa ekran ostaje stalno upaljen te se gasi jedino na korisničku akciju.

Videozapisi koje će aplikacija snimati bit će pohranjeni u poddirektoriju imena „DashCam“ u video galeriji. Na taj će način korisnici moći manipulirati videozapisima bez potrebe za korištenjem aplikacije. Klikom na gumb za snimanje aplikacija započinje s pripremama za snimanje videozapisa. Prvo se mora stvoriti nova datoteka s .mp4 ekstenzijom

Slika 5.9: Sučelje za snimanje

u koju će se spremati novi zapis. Potom dohvaćamo trenutne postavke aplikacije (o kojima će biti više riječi u poglavlju 5.5) i vršimo inicijalizaciju snimača imajući u vidu trenutnu orijentaciju uređaja. Kako bi aplikacija bolje radila na mobilnim uređajima, za snimanje se koristi metoda koja podatke sprema direktno u datoteku velikom brzinom, ali pri tome radi blago komprimiranje sadržaja – neznatan kompromis obzirom na namjenu videozapisa.

Tijekom snimanja, isto se može pauzirati, nastaviti ili zaustaviti. Prilikom izvršavanja posljednje akcije, datoteka biva dovršena i vidljiva u pregledniku datoteka. Poslije svakog zaustavljanja snimanja je potrebno otpustiti sve alocirane resurse koji su korišteni za snimanje i vratiti kameru u stanje prikazivanja tražila. Isti postupak, samo sveobuhvatniji, treba napraviti i kada ovo sučelje prestane biti aktivno. Na taj će način operativni sustav i ostale aplikacije imati puni pristup kameri i svim njenim funkcionalnostima, a aplikacija će na raspolaganju imati više resursa.

Osim putem klasičnih gumba, korisnici ovom aplikacijom mogu upravljati i putem glasovnih naredbi na engleskom jeziku. Ova aplikaciju za tu namjenu koristi osnovno prepoznavanje glasa, dok one naprednije mogu koristiti Cortanu[14]. Tekst glasovnih naredbi je potrebnu upisati u odgovarajući rječnik i zatim pokrenuti prepoznavanje. Nakon događaja prepoznavanja, u slučaju zadovoljavajuće pouzdanosti, dolazi do pozivanja odgovarajuće naredbe.

5.3.1 Reprodukcija

Slika 5.10: Sučelje za pregledavanje videozapisa

Snimljeni videozapisi mogu se i reproducirati unutar same aplikacije na sučelju prikazanom na slici 5.10. Reprodukcija se vrši pomoću multimedijskog elementa. Taj element je zapravo identičan kontroli za reprodukciju koja se nalazi unutar službene aplikacije za reprodukciju videozapisa koja dolazi s operativnim sustavom. Stoga u ovom slučaju nije bilo potrebno otvarati drugu aplikaciju kako bi reproducirali sadržaj. Sučelje je izrađeno od mreže elemenata koja se automatski prilagođava širini ekrana, a svaki element te mreže koristi predložak vidljiv u primjeru 5.11.

[14]virtualna pomoćnica ugrađena u Windows 10

```
<DataTemplate x:Key="VideoTemplate">

    <Grid Height="180"
          Width="320"
          Margin="4,6,4,6">

        <Grid.Background>
            <ImageBrush ImageSource="{Binding Thumbnail}" />
        </Grid.Background>

        <RelativePanel x:Name="Stripe"
                       Background="Black"
                       Opacity="0.9"
                       Height="32"
                       VerticalAlignment="Bottom"
                       HorizontalAlignment="Stretch">

            <TextBlock Text="{Binding FileName}"
                       Foreground="White"
                       RelativePanel.AlignLeftWithPanel="True"
                       RelativePanel.AlignVerticalCenterWithPanel="True"
                       Width="240"
                       TextTrimming="CharacterEllipsis"
                       Margin="4,0,0,0"/>

            <TextBlock Text="{Binding Duration}"
                       Foreground="White"
                       RelativePanel.AlignRightWithPanel="True"
                       RelativePanel.AlignVerticalCenterWithPanel="True"
                       Margin="0,0,4,0"/>

        </RelativePanel>

    </Grid>

</DataTemplate>
```

Primjer 5.11: Predložak videozapisa

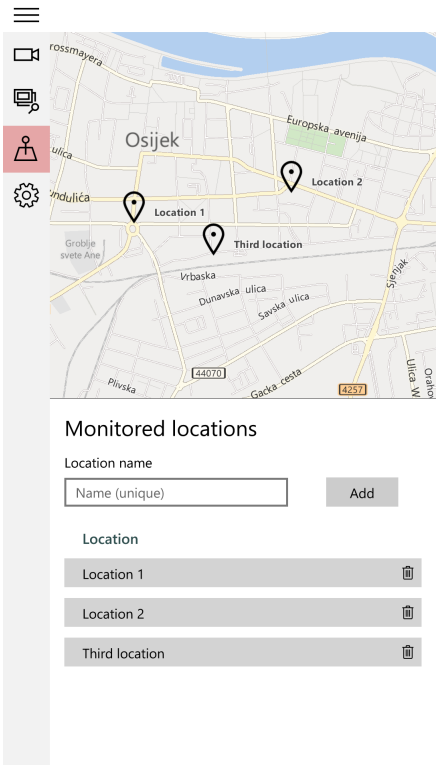
5.4 Korištenje geolokacije

Osim načina opisanih u prethodnom poglavlju, snimanje videa može započeti i na osnovu trenutne lokacije korisnika. Ovisno o stanju aplikacije, svaki ulazak odnosno izlazak korisnika iz geolokacijski ograđenog prostora (*engl. geofence*) rezultirat će početkom odnosno završetkom snimanja. Ovakav način početka snimanja može biti pogodan za korištenje na nepreglednim raskrižjima ili drugim kritičnim mjestima na prometnicama.

Sam unos tih lokacija se vrši pomoću sučelja prikazanog na slici 5.12. Korisnik na karti odabire neku točku te joj potom daje jedinstveni naziv. Nakon akcije dodavanja, kreira se nova geolokacijska ograda radijusa 1000 metara, a događaji koji će se pratiti su: ulazak, izlazak i brisanje geolokacije. Podatke o ovim ogradama ne pohranjujemo nigdje unutar aplikacije već tu zadaću prepuštamo operativnom sustavu. Razlog tomu je to što se na razini sustava koristi jedinstveni mehanizam za nadzor gelokacijskih ograda kako bi više aplikacija istovremeno moglo koristi tu mogućnost.

5.4.1 Pristup geolokacijskim resursima

Osnovni preduvjet za korištenje geolokacijskih usluga unutar aplikacije je deklariranje korištenja istih. Prilikom prvog pokretanja aplikacije sustav korisniku prikazuje obavijest o korištenju geolokacije, a korisnik može odlučiti želi li aplikaciji dati prava ili ne. Prilikom svakog pokretanja aplikacije potrebno je zatražiti prava za korištenje geolokacije i, u slučaju dopuštenja, konfigurirati postavke i pretplatiti se na događaje koje želimo pratiti na način



Slika 5.12: Sučelje lokacija

prikazan u primjeru 5.13.

U našem slučaju želimo se pretplatiti na događaj promjene lokacije i događaje promjene stanja geolokacijske ograde. Promjenu lokaciju pratimo kako bi na glavnom sučelju mogli prikazati trenutnu brzinu kretanja. No kako te događaje proslijediti do odgovarajućeg ViewModela? Načelno, prosljeđivanje u realnom vremenu između dva ViewModela ili ViewModela i neke pomoćne klase nije moguća. Zbog toga je gotovo svaki framework, pa tako i Caliburn.Micro, razvio neki oblik EventAggregatora - klase zadužene za komunikaciju u ovakvim slučajevima. EventAggregator mora imati samo jednu instancu na razini aplikaciji, a ona se koristi na svim mjestima gdje je to potrebno.

```
private async void Configure()
{
    var accessStatus = await Geolocator.RequestAccessAsync();

    switch (accessStatus)
    {
        case GeolocationAccessStatus.Allowed:
            _geolocator = new Geolocator { ReportInterval = 1000 };
            _geolocator.PositionChanged += OnPositionChanged;
            _geofenceMonitor = GeofenceMonitor.Current;
            _geofenceMonitor.GeofenceStateChanged += OnGeofenceStateChanged;
            break;

        case GeolocationAccessStatus.Denied:
        case GeolocationAccessStatus.Unspecified:
            break;
    }
}
```

Primjer 5.13: Konfiguracija pristupa geolokaciji

```
private async void OnPositionChanged(Geolocator sender, PositionChangedEventArgs e)
{
    await CoreApplication.MainView.CoreWindow.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, async () =>
    {
        await _eventAggregator.PublishOnUIThreadAsync(e.Position.Coordinate);
    });
}
```

```
public void Handle(Geocoordinate message)
{
    if (message?.Speed?.ToString() != "NaN")
    {
        Speed = (int)Math.Floor((decimal)message?.Speed * Settings.SpeedUnit.Multiplier);
    }
    else
    {
        Speed = 0;
    }
}
```

Primjer 5.14: Slanje (gore) i primanje (dolje) podataka putem EventAggregatora

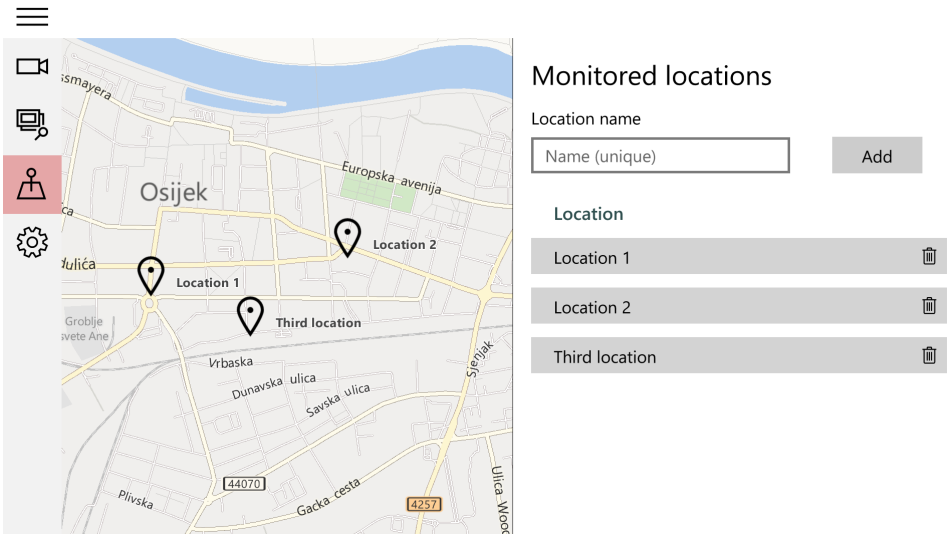
Objekt koji prima poruku se pri svakoj svojoj aktivaciji mora pretplatiti, a pri deaktivaciji odjaviti te uz to moramo implementirati sučelje za primanje tipa poruke. S druge pak strane, objekt koji šalje poruku to čini korištenjem primarne niti (one zadužene za prikazivanje sučelja) putem aplikacijskog dispečera. Mana ovog pristupa je to što poruke koje sadrže određeni tip podataka primaju svi objekti koji implementiraju sučelje za taj tip podataka.

Bez obzira na odobrenje korištenja geolokacije prilikom konfiguriranja, isto se mora zatražiti prilikom svakog sljedećeg korištenja. Novo odobrenje je nužno jer je u međuvremenu možda došlo do ulaska u izvanmrežni način rada ili je korisnik aplikaciji oduzeo prava. U slučajevima promjene prava proces konfiguracije se treba ponovno provesti kao i pretplaćivanje odnosno odjavljivanje na određene događaje.

5.4.2 Korištenje adaptivnog dizajna

Vratimo se na trenutak na sliku 5.12. Odmah je vidljivo da je to sučelje namijenjeno uspravnom položaju pametnog telefona. Ako bi sučelje ostalo isto i u vodoravnom položaju karta bi imala premalu visinu u odnosu na širinu te bi manipuliranje njome i dodavanje novih oznaka bilo znatno otežano. Također, na sučelje bi stalo tek par spremljenih lokacija i bila bi nužna dodatna akcija kako bi korisnik vidio sve lokacije. Slijedeći smjernice adaptivnog dizajna, to bi sučelje trebalo izgledati kako je vidljivo na slici 5.15.

Ovakvo ponašanje sučelja je postignuto korištenjem `VisualStateManager`-a – klase koja omogućava olakšano prilagođavanje različitih XAML sučelja. U ovom slučaju smo promatrali uvjet minimalne širine sučelja te smo u skladu s njom napravili dva prilagođena stanja. Ovisno o stanju, karta i lista lokacija zauzimaju različite retke odnosno stupce mreže u koju su smješteni.



Slika 5.15: Sučelje lokacija

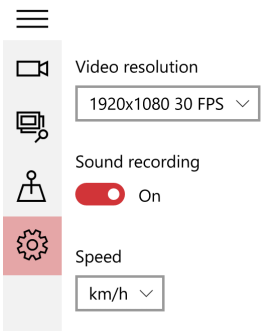
```
<VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="MinimumWidth">
            <VisualState x:Name="Landscape">
                <VisualState.StateTriggers>
                    <AdaptiveTrigger MinWindowWidth="548" />
                </VisualState.StateTriggers>
            <VisualState.Setters>
                <Setter Target="LocationsMap.(Grid.Column)" Value="0"/>
                <Setter Target="LocationsMap.(Grid.ColumnSpan)" Value="1"/>
                <Setter Target="LocationsMap.(Grid.Row)" Value="0"/>
                <Setter Target="LocationsMap.(Grid.RowSpan)" Value="2"/>
                <Setter Target="LocationData.(Grid.Column)" Value="1"/>
                <Setter Target="LocationData.(Grid.ColumnSpan)" Value="1"/>
                <Setter Target="LocationData.(Grid.Row)" Value="0"/>
                <Setter Target="LocationData.(Grid.RowSpan)" Value="2"/>
            </VisualState.Setters>
            </VisualState>

            <VisualState x:Name="Portrait">
                <VisualState.StateTriggers>
                    <AdaptiveTrigger MinWindowWidth="240" />
                </VisualState.StateTriggers>
            <VisualState.Setters>
                <Setter Target="LocationsMap.(Grid.Column)" Value="0"/>
                <Setter Target="LocationsMap.(Grid.ColumnSpan)" Value="2"/>
                <Setter Target="LocationsMap.(Grid.Row)" Value="0"/>
                <Setter Target="LocationsMap.(Grid.RowSpan)" Value="1"/>
                <Setter Target="LocationData.(Grid.Column)" Value="0"/>
                <Setter Target="LocationData.(Grid.ColumnSpan)" Value="2"/>
                <Setter Target="LocationData.(Grid.Row)" Value="1"/>
                <Setter Target="LocationData.(Grid.RowSpan)" Value="1"/>
            </VisualState.Setters>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
```

Primjer 5.16: Prilagodba sučelja

5.5 Postavke

Kao što je bilo spomenuto u poglavlju o videozapisima, aplikacija za snimanje istih koristi par jednostavnih postavki. Te postavke se nalaze na posljednjem sučelju, a osim njih tu je izbor mjerne jedinice za brzinu. Zašto su postavke uopće potrebne kod ovako jednostavne aplikacije? Kao prvo, ona će biti pokretana na različitim uređajima koji imaju kamere različitih mogućnosti te postoji mogućnost da osnovna rezolucija koju definiramo neće biti dostupna. Ovakvim pristupom korisniku je dana mogućnost odabira bilo koje rezolucije i broja sličica u sekundi koji su podržani na uređaju.



Slika 5.17: Sučelje postavki

Podržane oblike snimanja nije moguće dobiti jednostavnim čitanjem konfiguracije uređaja već do njih moramo doći na drugačiji način. Ti se podaci mogu dobiti direktno jedino iz kontrole za snimanje nakon njezine inicijalizacije. Budući da još nema podataka o postavkama ona se prvi put pokreće s predefinirano rezolucijom i snima u istoj toj rezoluciji (kontroli nije moguće promijeniti rezoluciju dok je aktivna). Nakon toga se postavke prosljeđuju ostalim dijelovima aplikacije.

Kako bi olakšali manipulaciju s postavkama njih serijaliziramo korišetenjem Newtonsoft.Json biblioteke i spremamo u JSON obliku u zaštićeni direktorij aplikacije. Prilikom svakog sljedećeg pokretanja aplikacije kontrola kamere se inicijalizira s korisničkim postavkama, ali se vrši i proces dohvaćanja svih podržanih načina snimanja i, po potrebi, ažurira postojeći popis. Na taj će način korisnik imati pristup svim načinima snimanja budući da se oni mogu promijeniti uslijed nadogradnje sustava.

Zaključak

Univerzalne aplikacije su tijekom posljednjih godina postale novi smjer u razvoju softvera. Radilo se o kombiniranju mobilnih i desktop aplikacija ili o responzivnim web aplikacijama, zadatak im je isti – pružiti korisnicima kvalitetno iskustvo gdje god se nalazili i koji god uređaj koristili. Najveći proizvođači softvera, Microsoft, Google i Apple, taj koncept pokušavaju oživjeti na različite načine.

Kroz nekoliko verzija sustava i platformi, Microsoft je ideju univerzalne aplikacije doveo na najviši nivo. Univerzalna Windows platforma svojom fleksibilnošću cilja na širok spektar uređaja pokretanih Windowsima 10. Načelo adaptivnosti koje stoji iza ove platforme, osim omogućavanja rada na brojim uređajima, nudi i olakšan razvoj aplikacija. Bez obzira na začetke u kojima se radilo o mobilnoj platformi, univerzalna Windows platforma svakom nadogradnjom dobiva sve više mogućnosti te sa tržišta izbacuje klasične Windows programe.

Unatoč sve manjem broju Windows 10 Mobile uređaja, sveukupan broj Windows 10 uređaja je u konstantnom rastu i predstavlja veliku bazu korisnika. Uskoro možemo očekivati pojačane napore i ostalih proizvođača softvera na ovom polju, a neki, poput Googlea s projektom Andromeda, na tome rade već danas.

Literatura

[1] J. Garland, Windows Store Apps Succinctly, Syncfusion, Morrisville, 2013.

[2] E. Moemeka, E. Moemeka, Real World Windows 10 Development, Springer Science+Business Media, New York, 2015.

[3] M. Pagani, More Windows 8.1 Succinctly, Syncfusion, Morrisville, 2015.

[4] M. Pagani, Windows 8.1 Succinctly, Syncfusion, Morrisville, 2015.

[5] https://msdn.microsoft.com/ - Microsoft Developer Network, prosinac 2016.

[6] https://mva.microsoft.com/ - Microsoft Virtual Academy, prosinac 2016.

Sažetak

Široki spektar i sveprisutnost digitalnih uređaja u životu suvremenog čovjeka dovela je ideje razvoja univerzalnih aplikacija. Njihova je zadaća korisniku pružiti što sličnije iskustvo korištenja neovisno o uređaju. Od svih proizvođača softvera, Microsoft je trenutno došao najdalje u ovom polju razvoja aplikativnog softvera. Koncept koji je započeo 2013. godine doveden je novu razinu predstavljanjem operativnog sustava Windows 10 i univerzalne Windows platforme. Glavna ideja univerzalne Windows platforme je korištenjem adaptivnog koda i dizajna korisnicima pružiti jedinstveno iskustvo na uređajima pokretanima Windowsima 10. Do tog se cilja dolazi orijentacijom na pojedine funkcionalnosti uređaja i sustava stavljajući samu vrstu uređaja u drugi plan. Za razliku od klasičnih Windows programa, Windows 10 aplikacije su svojim značajkama puno bliže mobilnim aplikacijama. Stoga njihov razvoj treba biti okrenut prema integraciji s operativnim sustavom i ostalim univerzalnim aplikacijama.

Ključne riječi:
univerzalne aplikacije, univerzalna Windows platforma, Windows 10, adaptivni kod, adaptivni dizajn, životni ciklus aplikacije, kamera za snimanje prometa

Summary

Windows 10 application development

Great presence an diversity of digital devices in the life of modern man has led to the development of the idea of universal application. Their mission is to provide user experience regardless of device. Of all the software vendors, Microsoft has now come furthest in the field of universal applications. The concept that started in 2013 is raised to a new level with launch of Windows 10 and Windows Universal Platform. Universal Windows Platform relies on adaptive code and adaptive design to give users a unified experience on devices powered by Windows 10. This goal is accomplished by orientating towards functionalities of the device and the system, and putting aside device type. Unlike traditional Windows programs, characteristics and functionality of Windows 10 applications are much closer to mobile applications. Therefore, their development should be oriented towards integration with the operating system and communication with other universal applications.

Key words:
universal applications, Universal Windows Platform, Windows 10, adaptive code, adaptive design, application lifecycle, dashcam

Životopis

Moje ime je Josip Kedveš i rođen sam 24. ožujka 1993. godine u Osijeku. Pohađao sam Osnovnu školu Frana Krste Frankopana u Osijeku. Po završetku osnovnoškolskog obrazovanja školovanje nastavljam u 3. gimnaziji Osijek gdje sam sudjelovao na županijskim natjecanjima iz matematike. 2011. godine upisujem preddiplomski studij matematike Odjela za matematiku Sveučilišta J. J. Strossmayera u Osijeku koji završavam 2014. godine uz završni rad "Razvojna platforma Arduino" pod vodstvom izv. prof. dr. sc. Domagoja Matijevića. Iste te godine upisujem diplomski studij matematike, smjer Matematika i računarstvo.