ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Nástroj pro generování High Fidelity prototypů ze zdrojových Low Fidelity prototypů

Štěpán Otta

Leden 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Otta** Jméno: **Štěpán** Osobní číslo: **465867**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Nástroj pro generování High Fidelity prototypů ze zdrojových Low Fidelity prototypů

Název bakalářské práce anglicky:

High Fidelity prototype generation from Low Fidelity prototype sources

Pokyny pro vypracování:

Proveďte analýzu procesu tvorby aplikací s využitím Low Fidelity a High Fidelity prototypů. Zaměřte se zejména na požadavky na funkcionality jednotlivých typů prototypů a na typy komponent, které jsou v jednotlivých typech prototypů používány. Také srovnejte prototypy ve fázi návrhu a ve fázi jejich vyhodnocování. Dále analyzujte již existující řešení, které podporují migraci mezi jednotlivými fázemi prototypů, jako je např. Framer nebo Supernova Studio.
Navrhněte prototyp desktopové aplikace, která bude transformovat zdrojové Low Fidelity prototypy do High Fidelity varianty. Po dohodě s vedoucím zvolte formát vstupních souborů, např. Balsamiq Mockups. Návrh aplikace ověřte na alespoň dvou Low Fidelity prototypech. Aplikaci koncipujte jako otevřenou pro možnost zapojení dalších funkcí, které podpoří např. testování, generování obsahu nebo napojení datových zdrojů.
Implementujte desktopovou aplikaci pro transformaci prototypů. Její funkčnost ověřte na alespoň 4 Low Fidelity prototypech.

Seznam doporučené literatury:

[1] Warfel, T. Z. (2009). Prototyping: a practitioner's guide. Rosenfeld media.
[2] T. Lowdermilk, User-Centered Design, O'Reilly Media, 2013.
[3] Rivero, J. M., Rossi, G., Grigera, J., Burella, J., Luna, E. R., & Gordillo, S. (2010, July). From mockups to user interface models: an extensible model driven approach. In International Conference on Web Engineering (pp. 13-24). Springer, Berlin, Heidelberg.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Ivo Malý, Ph.D., katedra počítačové grafiky a interakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **09.07.2019** Termín odevzdání bakalářské práce: **07.01.2020**

Platnost zadání bakalářské práce: **19.02.2021**

Ing. Ivo Malý, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____

Datum převzetí zadání Podpis studenta

Poděkovaní / Prohlášení

Zde bych chtěl poděkovat vedoucímu mé bakalářské práce Ing. Ivovi Malému, Ph.D., za příjemnou spolupráci a veškeré užitečné rady, které mi během psaní této práce poskytl.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 3. 1. 2020

.....

Abstrakt / Abstract

V mé bakalářské práci se budu zabývat prototypy grafických uživatelských rozhraní, jejich věrností a transformacemi. Popíšu rozdíly mezi jednotlivými prototypy dle jejich věrnosti, komponenty používaných v těchto prototypech. Popíšu současné možnosti transformace prototypů mezi různými úrovněmi věrnosti. Nakonec implementuji program, který bude transformovat prototypy nízké věrnosti vytvořené v programu Balsamiq Mockups do kódu pro univerzální platformu Windows. Jako vstup pro transformaci použiji soubory formátu BMML, do kterého umožňuje program Balsamiq Mockups export. Tyto soubory budu načítat a transformovat do výsledných souborů ve formátu XAML a cs. Tento program otestuji na čtyřech připravených prototypech.

Klíčová slova: transformace, věrnost, prototyp, Balsamiq Mockups, UWP

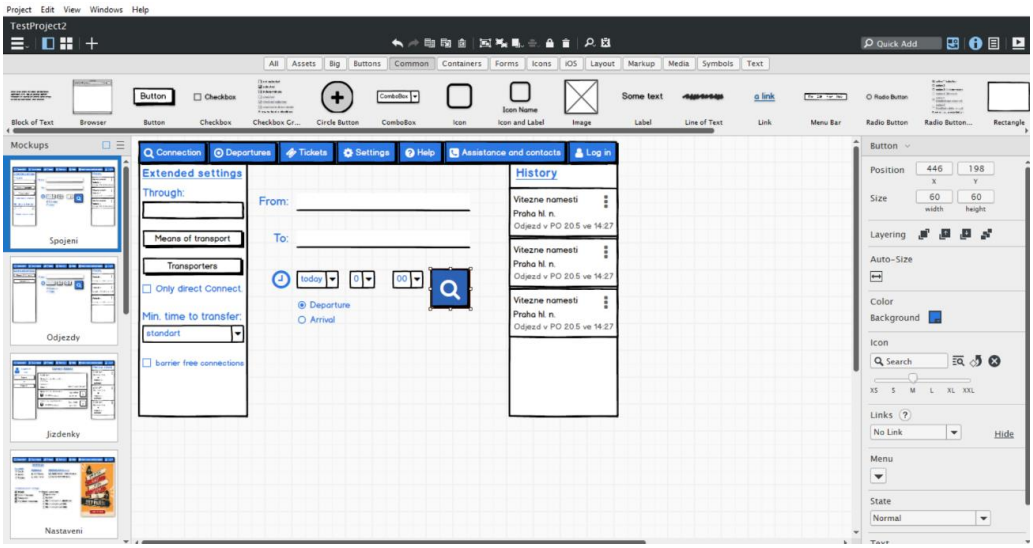
In my bachelor's thesis I will be dealing with graphical user prototypes, fidelity of prototypes and transformation of prototypes. I will describe differences between prototypes of different fidelities and components used in these prototypes. I will describe current possibilities for prototype transformation between prototypes of different fidelity. Finally, I will implement program that will transform low fidelity prototypes, made in Balsamiq Mockups, to code for Universal Windows Platform. As input for transformation I will use BMML files exported from Balsamiq Mockups. These files I will load and transform to XAML and cs files. This program I will test this program using four different prototypes.

Key words: transformation, fidelity, prototype, Balsamiq Mockups, UWP

Obsah /

1. Kapitola
Úvod

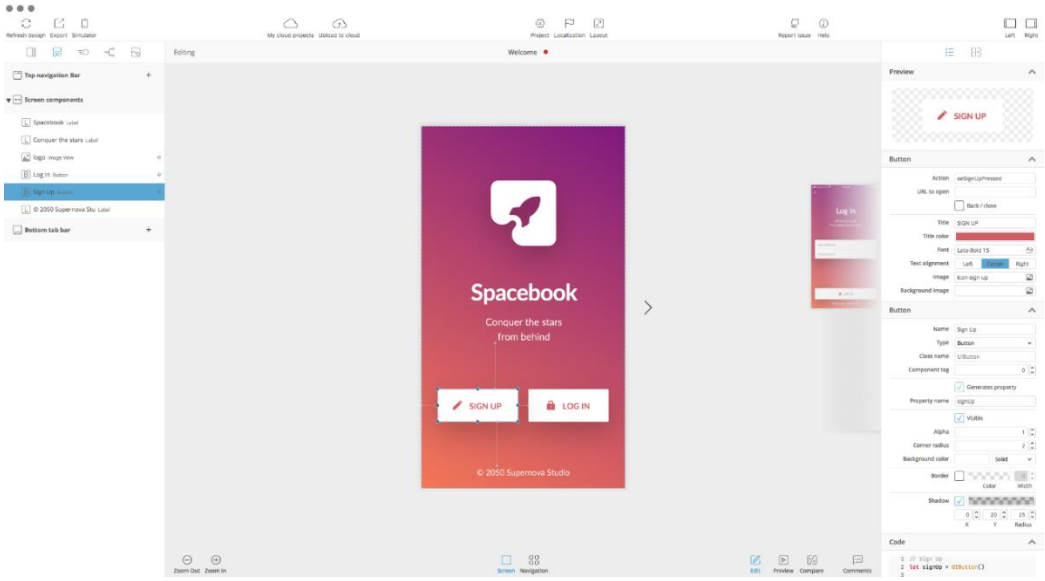
Tvorba prototypů uživatelských rozhraní provází naprostou většinu softwarových projektů. Během tohoto procesu, se však často opakuje začínání od nuly. Tvorba uživatelského rozhraní začíná u jednoduchých prototypů, které jsou vytvářeny na papír nebo pomocí specializovaných nástrojů. Jedním z nich je například program Balsamiq Mockups 3, viz Obrázek 1.Tyto prototypy nevyžadují velké množství funkcionalit a jsou zaměřeny především na rozvržení jednotlivých ovládacích prvků.



Obrázek 1 : Ukázka programu Balsamiq Mockups 3.

Při posunu ke složitějším a detailnějším prototypům začíná tvorba prototypu znovu od začátku. Kdyby byla možnost, jak využít jednoduché prototypy z počátku vývoje, při tvorbě složitějších prototypů nebo při vývoji finálního uživatelského prostředí, znamenalo by to úsporu času i zvýšení efektivity práce.

Nabídka programů, které nabízejí tuto možnost, je však velmi omezená. Programy, které realizují tento převod jsou v České republice vyvíjený program Supernova Studio, viz Obrázek 2, a program Framer X. Nevýhodou obou těchto programů může být, že jsou k dispozici pouze pro Mac OS.



Obrázek 2 : Ukázka programu Supernova Studio.

Jsou možné různé přístupy k řešení toho problému. Prvním přístupem je, že se prototyp vyvíjí v dané aplikaci již od začátku a aplikace pokrývá tvorbu prototypu od těch nejjednodušších až po ty složité. Zároveň s tvorbou může být i generován kód ve vybraném jazyce. Druhý možný přístup je ve spolupráci s jiným programem, který je určen na tvorbu jednoduchých prototypů. Například Supernova Studio spolupracuje s programem Sketch.

Cílem této práce tedy je vytvořit program, který uživatelům, kterými by měli být primárně návrháři uživatelských rozhraní, transformovat prototypy vytvořené v programu Balsamic Mockups 3 do zdrojové kódu v jazyce C# určeného pro platformu Univerzální platforma Windows. Na rozdíl, od již existujících programů bude tento program určen pro platformu Microsoft Windows. Aplikace načte vyexportovaný prototyp pro jednotlivých souborech ve formátu bmml a vytvoří jednotlivé soubory ve formátu cs a xaml, které budou obsahovat definice jednotlivých obrazovek prototypu.

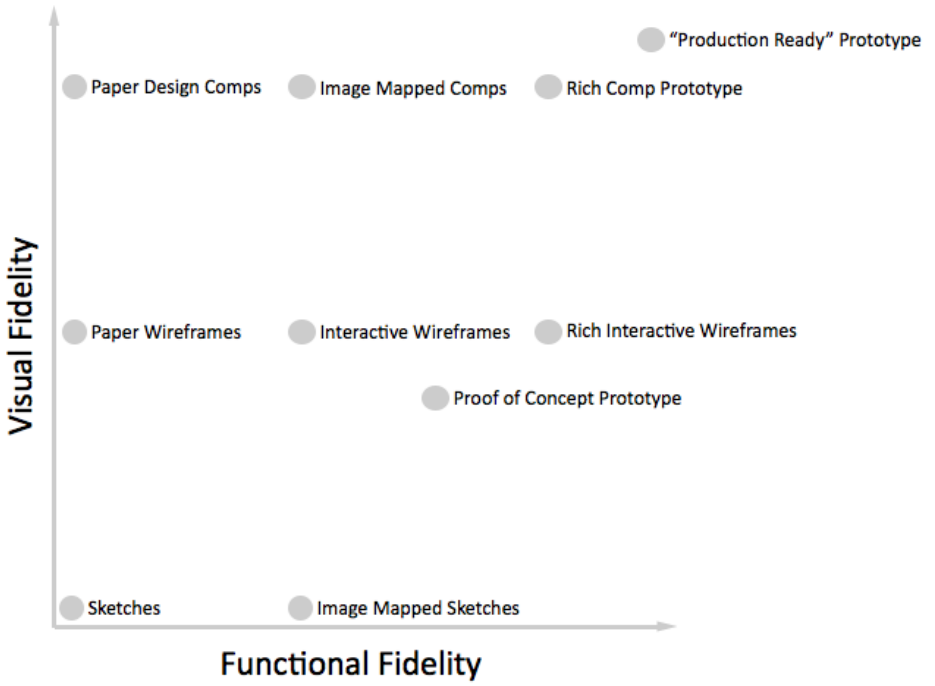
Využití je pro zvýšení efektivity vývoje procesu tvorby uživatelského rozhraní. Transformací se z tvorby uživatelského rozhraní vyřadí nutnost při posunu k detailnějšímu prototypu začínat znovu od začátku. Navíc aplikace transformací vytvoří i vstupní bod pro tvorbu finálního uživatelského rozhraní.

1. Kapitola
Věrnost prototypů

Jako první bylo třeba analyzovat co je vlastně Fidelity (dále jen věrnost) prototypů a jakou hraje roli v procesu tvorby uživatelských rozhraní, popřípadě v celých softwarových projektech. Věrnost ukazuje, jak realistický daný prototyp je nebo na jaké úrovni se podobá finálnímu produktu. Ukazuje, jaká je úroveň detailu a funkcionalit zahrnutých v prototypu. (1)

Prototypy se dle věrnosti dělí na dvě hlavní skupiny: prototypy s nízkou věrností a prototypy s vysokou věrností. Přesto však prototypy zařazené v těchto skupinách mohou být z pohledu výstižnosti velice rozdílné. Spousta prototypů se také pohybuje někde na rozhraní těchto skupin a nelze je přesně zařadit. Úroveň věrnosti může být velice široká u různých prototypů. Zároveň také platí, že ne každý prototyp se hodí pro každou situaci. Různá úroveň věrnosti prototypu je vhodná pro jiné situace.

Pro přesnější rozdělení můžeme věrnost, podle jiného kritéria rozdělit do dalších dvou skupin: vizuální věrnost a funkční věrnost. Vizuální věrnost znázorňuje, na jaké úrovni je prototyp podobný finálnímu produktu z pohledu vzhledu, grafických prvků, barev, tvarů. Funkční věrnost ukazuje, na jaké úrovni je prototyp podobný finálnímu produktu z hlediska funkcionalit, ovládání, rozmístění ovládacích prvků. V grafu, viz Obrázek 3, můžeme vidět některé ukázky různé úrovně prototypů dle úrovně funkční a vizuální věrnosti. (2)



Obrázek 3: Graf úrovně věrnosti **(2)**

Jak ukazuje graf (Obrázek 3), prototypy mohou být od ručně kreslených obrázků až po prototypy, které jsou funkcionalitou i vzhledem velmi blízko finálnímu produktu. Různé typy prototypů se hodí k jinému účelu a vytváří se v jiné fázi vývoje a také mají různé výhody a nevýhody.

1.1 Prototypy nízké věrnosti

Prototypy s nízkou věrností jsou vhodné především v brzkých fázích vývoje, ve fázi analýzy. Jsou vhodné k dolování požadavků. Můžeme je použít k základnímu testování rozvržení ovládacích prvků a průchodu aplikací.

Hlavní využití prototypů s nízkou věrností je zodpovězení základních otázek týkajících se konstrukce celé aplikace. Můžeme si pomocí nich ověřit, zda aplikace obsahuje všechny potřebné prvky a funkce, zda průchod aplikací je přehledný a plynulý, vyjasnění si pohledu na věc se všemi zúčastněnými(stakeholders). (2)

Náklady na jejich tvorbu jsou nízké, protože k jejich realizaci můžeme použít i například tužku a začít jednoduchými nákresy na papír. Existují i různé nástroje na tvorbu těchto prototypů, které jsou často k dispozici zdarma a práce s nimi je jednoduchá.

Prototypy s nízkou věrností také umožňují snadné a rychlé změny. V počátcích vývoje je nutnost reagovat na poznatky z testování častými změnami prototypů, proto jsou v počátku vývoje prototypy s nízkou věrností vhodné, protože tyto změny bude možné rychle a levně provádět.

Další výhodou je, že každý účastník projektu může s těmito prototypy snadno pracovat, a to nejen je prohlížet, ale i upravovat, protože práce s nimi nevyžaduje žádnou znalost programování, ani zkušenosti s používáním komplikovaného softwaru. (1)

Nevýhody prototypů s nízkou věrností jsou naopak například to, že není možné realizovat žádné složité animace nebo přechody. Během testování také může nastávat problém s tím, že není přesně jasné, co má a co nemá fungovat, testování tedy vyžaduje po subjektech určitou míru představivosti, a proto často mohou z testů vyplývat závěry, které souvisí s touto omezenou funkčností samotného prototypu. (3)

1.1.1 Prototypování na papír

Varianta testování použitelnosti, kde reprezentativní skupina uživatelů plní realistické úkoly interagováním s papírovou verzí rozhraní, které je ovládáno osobou, která simuluje počítač, ale nevysvětluje, jak by mělo rozhraní fungovat. (4)

Prototyp uživatelského rozhraní se kreslí na papír po jednotlivých obrazovkách a částech. Při testování se poté využívá osoba, která zastupuje roli počítače v reakcích na uživatelské akce.

Výhody prototypování na papír:

- Rychlý a levný způsob prototypování
- Každý se může prototypování účastnit, protože kreslení papírových prototypů nevyžaduje, žádné zvláštní schopnosti
- Umožňuje otestovat a na základě testů přizpůsobit uživatelské rozhraní, aniž by bylo potřeba cokoliv programovat, dokonce lze pohodlně měnit prototyp bezprostředně během testování

- Podporuje kreativitu všech účastníku vývoje
- Může být využité jako dokumentace pro finální produkt (4)

Nevýhody prototypování na papír:

- Nemůže identifikovat všechny problémy s rozhraním
- Neprodukuje žádný kód, ani další elektronický materiál, který by se dál použít k transformaci na prototypy vyšší úrovně, proto pro nás není tento způsob prototypování nijak dále užitečný.
- Během testování je vždy potřeba navíc osoba, která zastupuje funkci počítače (4)

1.1.2 Ovladatelné propojované modely (Clickable wireframes)

Propojovaný model je reprezentace obrazovky grafického uživatelského rozhraní, návrhář na tento model rozmísťuje jednotlivé komponenty uživatelského rozhraní. Tyto statické modely poté návrhář mezi sebou propojuje. Tento typ prototypování je realizován pomocí specializovaných nástrojů jako je například AdobeXD nebo Balsamiq Mockups. Tento typ prototypů je pro nás zásadní. Je ideální pro možnost transformace na prototypy vyšších věrností. Nabízí nám elektronický materiál, s kterým bude možné automaticky pracovat a transformovat ho. Jedinou nevýhodou je, že tento materiál se výrazně liší dle použitého nástroje.

Výhody ovladatelných propojovaných modelů:

- Rychlý a levný způsob prototypování
- Umožňuje otestovat a na základě testů přizpůsobit uživatelské rozhraní, aniž by bylo potřeba cokoliv programovat
- Oproti prototypování na papír není potřeba osoba navíc při testování, protože prototypy jsou interaktivní a odezvu na uživatelské akce zajišťuje počítač

Nevýhody ovladatelných propojovaných modelů:

- Neprodukuje žádný kód
- Nemůže identifikovat všechny problémy s rozhraním

1.2 Prototypy vysoké věrnosti

Prototypy s vysokou věrností jsou vhodné především v pozdějších fázích vývoje. Tuto skupinu je však praktičtější detailněji rozčlenit. Můžeme je rozčlenit podle vizuální a funkční věrnosti. Rozdělujeme je na 3 „podtypy“. (2)

1.2.1 Nízká vizuální a vysoká funkční věrnost

Tyto prototypy se vyznačují jednoduchým grafickým zpracováním, vše černobílé a jednoduché základní tvary. Interakce naproti tomu, jsou takřka totožné s finálním produktem. Lze ho využít na kontrolu, zda jsou splněny všechny požadované funkcionality, zda je návrh systému dobře použitelný, zda je průchod systémem plynulý a přehledný, může být také použit jako velice kvalitní dokumentace pro vývojáře a je ideální pro remote testing (Testování, kde se pomocí online softwaru nahrává obrazovku účastníka testu, který pracuje ve svém přirozeném prostředí nebo ve specificky vybrané lokaci.). (2)

Pro tvorbu těchto prototypů lze použít dva způsoby specializovaný software nebo vytvářet prototyp pomocí programování. Existuje množství specializovaných programů, které jsou určeny na tvorbu prototypů. S různými programy lze dosáhnout různé úrovně věrnosti. Nevýhodou oproti programování prototypu je, že jej nelze použít jako výchozí bod pro programování finálního produktu. Výhodou je, že jsou tyto programy přístupné i lidem bez znalosti programování.

Druhým způsobem je přímo programování prototypu. Výhodou toho je, že můžeme výsledný prototyp použít jako vstupní bod pro tvorbu výsledného uživatelského rozhraní. Pomocí programování můžeme realizovat všechny prvky funkčnosti, které budeme potřebovat. Velkou nevýhodou je, že člověk, který ho bude vytvářet musí být programátor. Změny v prototypu mohou být často složité a časově náročné.

1.2.2 Vysoká vizuální a nízká funkční věrnost

Tento typ prototypu nebude tak častý, ale v některých případech je vzhled výsledku důležitější než funkcionalita a jsou typy testů, na které je právě takovýto prototyp vhodný. Například „slap and map“ testování, což znamená, že se pomocí html propojí jednotlivé obrázky mezi sebou, čímž se vytvoří prototyp.

1.2.3 Vysoká vizuální i funkční věrnost

Tyto prototypy vyžadují hodně času, více prostředků i více zkušeností na výrobu. Nejsou vhodné pro většinu případů, právě proto že jsou zatížené těmito problémy. Výhodou je, že jsou velmi blízko finálního produktu, a proto je možné pomocí nich testovat téměř vše, také je možné pomocí nich prezentovat, jak bude finální produkt vypadat.

1.3 Vhodná věrnost

Tvorby prototypu začíná vždy tím, že musíme určit potřeby. Určíme si, co pomocí tohoto prototypu chceme ověřit, co chceme pomocí něj testovat. Poté na základě těchto potřeb zvolíme takový prototyp, který nám umožní tyto potřeby uspokojit, tedy prototyp vhodné věrnosti. Prototypy různých věrností můžeme dosáhnout jiných závěrů. Fáze určení potřeb je proto při tvorbě prototypů zásadní. (2)

2. Kapitola
Komponenty

Před návrhem a implementací je také třeba popsat jaké jednotlivé komponenty se při tvorbě prototypů používají, jaké nabízejí možnosti a jak se s nimi pracuje. Různé prototypy vyžadují použití různých komponent. Je to proto, že každý prototyp vyžaduje od použitých komponent různé funkcionality. Bylo by možné používat všude komponenty, které umí vše, ale to by v mnohých případech velmi komplikovalo práci.

Prototypy nízké věrnosti nevyžadují velké množství funkcionalit. Z uživatelských reakcí si tyto prototypy v naprosté většině případů vystačí pouze s kliknutím na danou komponentu. Při použití některých nástrojů, které jsou zaměřené vyloženě graficky nemusí být žádná funkcionalita elementů. Je dokonce možné, že zde nebudou, že nebude k dispozici žádná knihovna konkrétních ovládacích prvků, ale budou k dispozici pouze tvary jako například obdélník nebo elipsa. Návrhář si z těchto tvarů bude požadované ovládací prvky vytvářet sám, přesně podle svých představ. Co se týče vlastností jednotlivých komponent, tak je toho samozřejmě potřeba více, ale stejně se většinou pohybujeme ve velmi základních nastavení. Lze měnit text, barva text, podtržení a podobně. Nepoužívají se animace, složitý vzhled ani komplikované tvary. Důležité je, aby se s komponentami snadno a rychle pracovalo. Nepracuje se zde s kódem, ale úpravy jsou realizovány výhradně pomocí poskytnutého uživatelského rozhraní. Tyto

komponenty nejsou založené na platformě specifických komponentách, které jsou součástí operačních systémů.

Speciální kategorií je prototypování na papír, kde v podstatě o žádných konkrétních komponentách nelze mluvit. Návrhář je však omezen tím, co se dá na papír nakreslit. Nelze na papír nijak nakreslit animace ani složitější grafický vzhled. Problém je i s většinou funkcionalit, ty se při testování simulují člověkem, který test řídí.

U prototypů vysoké věrnosti je situace jiná. Komponenty musí mít vysokou úroveň modifikovatelnosti a zároveň také větší množství funkcionalit. Jako funkcionality mohou být vyžadovány například reakce na přejetí myší, kliknutí pravým tlačítky myši nebo táhnout a pustit. Zde bude vždy k dispozici knihovna komponent, situace, že by byly k dispozici pouze tvary by byla nedostačující pro prototypy vysoké věrnosti. Prototypy vysoké věrnosti se snaží být velmi blízko finálnímu produktu, je proto dobré využívat nativní komponenty dané platformy. Tyto komponenty dávají veškeré dostupné možnosti. Proto existují programy, které jsou na tvorbu prototypů vysoké věrnosti uzpůsobené.

3. Kapitola
Transformace prototypů

Je třeba analyzovat, jak se v současné době s tímto problémem zachází a jak se dá řešit. Problém, se kterým se mohou návrháři grafických uživatelských rozhraní potýkat, spočívá v tom, že pokud tým návrhářů vytvoří v počátcích vývoje prototyp nízké věrnosti pomocí některé z nabízených aplikací a poté se chce posunout dále a využít již vytvořený prototyp k vytvoření prototypu s vyšší úrovní věrnosti, není zde možnost, jak toho dosáhnout a musí proto původní prototyp opustit a začít například přímo pomocí kódu vytvářet prototyp s vyšší věrností. To snižuje efektivitu práce a využití času.

V současné době se mi podařilo najít pouze dvě aplikace, které se pokoušejí realizovat transformaci mezi prototypy nižších věrností a platformě specifickým kódem. Aplikace, které mohou návrháři využít jsou Framer a Supernova studio, nevýhoda těchto aplikací může být, pro některé návrháře a vývojáře, že tyto aplikace jsou pouze pro mac OS. Navíc jsou obě aplikace jsou zaměřené především na prototypování mobilních aplikací a webových aplikací.

3.1 Supernova Studio

Supernova studio se zaměřuje primárně na vytvoření mostu mezi prototypy nízké věrnosti a prototypy vysoké věrností nebo implementací. Umožňuje import prototypů z programu Sketch nebo Adobe XD, pro které automaticky generuje kód. Automatický lokalizuje jednotlivé ovládací prvky a převede je na nativní komponenty dle zvolené platformy. Zároveň během transformace optimalizuje dané uživatelské rozhraní. Platformy, na které je možné vyvíjet jsou iOS, Android a React Native. Zdrojový kód je k dispozici v jazycích Swift, Java, Kotlin a JavaScript. Jako výstup získáte sestavitelné projekty právě v těchto jazycích. Vzniklé nativní komponenty může uživatel dále měnit a upravovat, například je může přeměnit ve více pokročilé komponenty, aby mohl dosáhnout na kýžené možnosti. Dále také lze tyto komponenty skládat do komplexních, vlastních komponent. (5)

Supernova studio není nástrojem pro návrh prototypů uživatelských rozhraní. Návrháři mají již oblíbené nástroj a není potřeba je nutit učit se práci s novým nástrojem, proto se Supernova studio zaměřuje na transformaci prototypů, a ne na jejich tvorbu od nuly. (6)

Supernova studio dále nabízí nástroje pro testování těchto prototypů. Možnost tvorby navigačních řetězců i vzorových uživatelských průchodů. Dále také interaktivní mód pomocí, které lze testovat dané prototypy. (6)

Dalšími výhodami Supernova studio jsou například engine, který dovolí uživatelům snadno a efektivně vytvářet složité animace. Engine, který se stará o automatickou tvorbu reagujícího rozhraní, tedy zajišťuje to, aby bylo rozhraní použitelné na různých zařízeních různé velikosti, výsledek si může uživatel přímo v programu zobrazit, otestovat a případně upravit. (6)

3.2 Framer X

Framer X se oproti Supernova studiu zaměřuje na to, aby nabídl uživatelům možnost zůstat od začátku až do konce vývoje prototypů v tomto programu. Framer nabízí rozsáhlý editor prototypů přímo v sobě. Program svou nabídky rozděluje do tří oblastí design, prototypování a vývoj. Uživatelem tedy může být návrhář i programátor. Navíc je zde pro jejich práci zajištěno propojení. Nenastává tedy situace, že by návrhář vytvořil něco, co nelze naprogramovat. (7) Uživatel tedy vytváří prototyp od začátků přímo v

programu Framer. Může si vytvářet vlastní elementy grafického uživatelského rozhraní nebo využít ty, které jsou k dispozici v zabudované knihovně. Tato knihovna nabízí i ovládací prvky vytvořené jinými uživateli, každý uživatel může přidat svůj vytvořený ovládací prvek do nabídky. Pomocí Frameru může vývojář zároveň vytvářet design pomocí grafických komponent, pro tyto komponenty je automaticky generován kód. Pokud je uživatel programátor může zároveň psát a editovat kód. Platformy, na které je možné pomocí Framer-u vyvíjet jsou React a HTML. Lze také programovat pomocí Javascriptu. (8)

Stejně jako Supernova, však Framer X nabízí i možnost transformace prototypu vytvořeného v oblíbeném programu Sketch. Je možné použít i prototypy vytvořené v jiných programech, pokud jsou ve formátu SVG, protože Framer podporuje vkládání SVG souborů. (7)

4. Kapitola
Formát vstupních souborů

Na začátku bylo nutné zvolit program pro tvorbu prototypů nízké věrnosti, v němž vytvořené prototypy budeme používat jako vstup pro transformaci. Zvolil jsem program Balsamiq Mockups 3. Tento program splňuje požadavek, že je k dispozici pro Microsoft Windows.

Další velkou výhodou je možnost exportu v něm vytvořených prototypů ve formát bmml, to znamená Balsamiq Mockups Markup Language. Tento formát vychází je stejný jako formát xml. (9) Výhoda tohoto formátu je, že ho může člověk přečíst, dobře se tedy kontrolovalo, jak bude potřeba jednotlivé parametry transformovat. Zároveň je formát vhodný i pro automatické zpracování. Prototyp se exportuje jako zip obsahující soubory formátu bmml pro jednotlivé obrazovky prototypu, adresář s obrázky a souborem obsahujícím symboly. Pro načítání a rozbor tohoto formátu jsou k dispozici velice dobré knihovny. Dokument pro jednu obrazovku prototypu může vypadat takto:

```
<mockup version="1.0" skin="sketch" fontFace="Balsamiq Sans"
measuredW="537" measuredH="327" mockupW="280" mockupH="191">
  <controls>
    <control controlID="0"
controlTypeID="com.balsamiq.mockups::Button" x="302" y="136" w="213"
h="94" measuredW="134" measuredH="27" zOrder="0" locked="false">
      <controlProperties>
        <align>left</align>
        <bold>true</bold>
        <color>16750848</color>
```

```
        <icon>edge%7Csmall</icon>
        <italic>true</italic>
        <menuIcon>true</menuIcon>
        <size>18</size>
        <state>disabled</state>
        <text>Button</text>
        <underline>true</underline>
      </controlProperties>
    </control>
  </controls>
</mockup>
```

Tato obrazovka obsahuje pouze tlačítko s nastaveným množstvím různých parametrů. Každá obrazovka je exportována do vlastního souboru. Obrazovka je reprezentována elementem `<mockup …> …<\mockup>`, který obsahuje element `<controls>`, který je v podstatě seznam jednotlivých ovládacích prvků, každý tento ovládací prvek je reprezentován elementem `<control…> …<\control>`. Tento element ještě obsahuje `<controlProperties>`, což je seznam jednotlivých atributů, které je možné u daného ovládacího prvku nastavit.

Výhodou Balsamiq Mockups byla také má předchozí zkušenost s programem a velice jednoduché a intuitivní ovládání.

4.1 Symboly

Balsamiq Mockups umožňuje návrhářům vytvářet vlastní ovládací prvky pro jejich prototypy. Tyto ovládací prvky se nazývají symboly a vytvářejí se skládání z již existujících ovládacích prvků. Tyto prvky pak lze opakovaně používat a pro jednotlivé instance lze měnit atributy jednotlivých základních ovládacích prvků, z kterých se symbol skládá. Všechny vytvořené symboly se exportují do jednoho souboru, který vypadá jako jedna obrazovka obsahující všechny tyto symboly. Takový soubor může vypadat například takto:

```
<mockup version="1.0" skin="sketch" fontFace="Balsamiq Sans"
measuredW="1596" measuredH="1180" mockupW="1590" mockupH="1174">
  <controls>
    <control controlID="0" controlTypeID="__group__" x="6" y="6"
w="334" h="157" measuredW="0" measuredH="0" zOrder="0" locked="false"
isInGroup="-1">
      <controlProperties>
        <controlName>TextBlockSym</controlName>
      </controlProperties>
      <groupChildrenDescriptors>
        <control controlID="0"
controlTypeID="com.balsamiq.mockups::Canvas" x="0" y="0" w="334"
```

```
h="157" measuredW="0" measuredH="0" zOrder="0" locked="false"
isInGroup="0"/>
        <control controlID="1"
controlTypeID="com.balsamiq.mockups::Canvas" x="31" y="16" w="273"
h="124" measuredW="0" measuredH="0" zOrder="1" locked="false"
isInGroup="0"/>
        <control controlID="2"
controlTypeID="com.balsamiq.mockups::Label" x="130" y="68" w="-1" h="-
1" measuredW="0" measuredH="0" zOrder="2" locked="false"
isInGroup="0">
          <controlProperties>
            <text>Puvodni%20text</text>
          </controlProperties>
        </control>
      </groupChildrenDescriptors>
    </control>
  </controls>
</mockup>
```

Každý symbol je reprezentován elementem `<control…>` `…<\control>`. Tento element je typu group, což znamená, že je to skupina, která obsahuje další ovládací prvky.

Instance symbolu se exportuje jako součást obrazovky a je identifikována jménem symbolu. Tato instance může vypadat například takto:

```
<control controlID="5" controlTypeID="com.balsamiq.mockups::Component"
x="476" y="50" w="-1" h="-1" measuredW="0" measuredH="0" zOrder="1"
locked="false" isInGroup="-1">
      <controlProperties>
        <override controlID="0" x="0" y="0" w="334" h="157"/>
        <override controlID="1" x="31" y="16" w="273" h="124"/>
        <override controlID="2" x="61" y="35" w="-1" h="-1">
          <align>center</align>
          <bold>true</bold>
          <color>13576743</color>
          <italic>true</italic>
        </override>
        <src>./assets/Project%20Symbols.bmml#TextBlockSym</src>
    </controlProperties>
</control>
```

Každá instance symbolu je označena atributem `controlTypeID` s hodnotou `com.balsamiq.mockups::Component"`. Element `<override…/>` označuje ovládací prvek jehož atributy byly změněny pro tuto instanci. O který symbol se jedná označuje element `<src>`, jeho hodnota je cesta k souboru se symboly a název symbolu.

5. Kapitola
Výstup programu

Program transformuje prototypy do nativního kódu pro Univerzální platformu Windows. Tato cílová platforma byla zvolena pro šíři jejího užití. Aplikace, které jsou vytvořeny pro Univerzální Platformu Windows lze spouštět a použít na stolních počítačích a nootebocích s operačním systémem Windows, na konzolích xBox, Hololens cože je headset pro virtuální realitu, mobilních telefonech, a dokonce na zařízeních pro internet věcí. (10) Výstupem programu budou soubory ve formátech cs a XAML. Tyto soubory obsahují kód v jazyce C# a formátu XAML.

Formát XAML je deklarativní značkovací jazyk založený na široce využívaném formátu XML. V tomto formátu jsou vytvořeny viditelné ovladatelné obrazovky s viditelnými ovládacími prvky. V jazyce C# je kód, který zajišťuje fungování těchto obrazovek za běhu aplikace. Větší část transformace bude tedy probíhat do souborů ve formátu XAML, kam se transformují všechny ovládací prvky a jejich parametry. Do jazyka C# jsou transformovány především přechody obrazovek při uživatelském klikání na jednotlivé ovládací prvky. Do jazyka C# jsou také transformovány součásti symbolů, které jsou nutné pro jejich různé instance na různých obrazovkách, což je popsáno níže v kapitole 7.

6. Kapitola
Načítání a transformace vstupu

Pro načítání a rozbor souborů využívám třídy a metody z jmenného prostoru *System.Xml*. Rozbor těchto souborů realizuje třída *XMLReader.cs*. Načítám jednotlivé ovládací prvky a ty přímo transformuji do mnou připravených tříd reprezentující tyto ovládací prvky. Pro každý ovládací prvek jsem našel protějšek, který nabízí UWP. Data z bmml souborů jsou rovnou transformovány do podoby vhodné k serializaci do formátu xaml. Tedy pro načtenou hodnotu určité vlastnosti komponenty si uložím odpovídající hodnotu, která vytváří stejný výsledek v UWP. Pokud prototyp obsahuje symboly, nejdříve jsou načteny ty, jejich načítání provází mnoho odlišností, proto o nich píšu samostatně. Element `<control…>` `…<\control>` reprezentující daný ovládací prvek se předá odpovídající metodě.

Tato metoda vytvoří objekt dané třídy dle typu ovládacího prvku a načte a transformuje hodnoty jednotlivých vlastností této komponenty. Takto jsou postupně transformovány všechny ovládací prvky každé obrazovky. Všechny načtené prvky jsou uchovány ve třídě zastupující obrazovku, pro UWP stránku. Poté se přechází k serializaci. Během transformace mohou ještě probíhat dodatečné transformace, které nebylo z určitých důvodů možné provést hned při načítání.

6.1 Transformace symbolů

Výše popsané symboly se nabízelo transformovat dvěma způsoby. První způsob je opustit logiku nového ovládacího prvku a na každou stránku vložit všechny jednotlivé komponenty znovu s danými změnami. Druhá možnost byla držet se logiky používané v Balsamiq Mockups a transformovat tyto symboly opět jako nové ovládací prvky. Pro tuto možnost nabízí UWP třídu *User Control*.

První možnost by byla výrazně jednodušší na transformaci. Znamenali by to jen na každé stránce, kde se symbol vyskytuje ho znovu serializovat. Jediné, co by se muselo při každém výskytu provést by byli změny zadané pro danou instanci toho symbolu. Tato možnost, by však při dalším pokračováním ve vývoji daného uživatelského rozhraní znamenala, že vývojář nebude moci pracovat s celým symbolem, který se několikrát opakuje, na jednom místě, ale při každé změně by musel provést tuto změnu na všech stránkách, kde se *User Control* vyskytuje. To podle mě nedává tedy příliš smysl, protože již v původním prototypu dal jeho návrhář signál, že se toto seskupení ovládacích prvků často opakuje, a proto z něho vytvořil symbol, aby při případných úpravách nemusel upravovat všechny obrazovky.

Mnou zvolená druhá možnost předchází nevýhody té první. Druhá možnost si také nesla své nevýhody. Transformovat symbol do *User Control* se ukázalo jako poměrně obtížné. Problémy působily především změny vlastností jednotlivých ovládacích prvků pro různé instance. Na to, aby toto bylo možné u *User Control* využívá UWP konstrukt *dependency property*. Dle mého názoru, však výhody, které toto řešení nabízí pro následný vývoj uživatelského rozhraní převážily komplikace při implementaci.

Jak již bylo dříve zmíněno, každá instance symbolu je vždy součástí stránky jako jeden z elementů. Tento element obsahuje všechny informace o změnách pro tuto konkrétní instanci. Pro každý prvek, na kterém jsou nějaké změny je zde element `<override…/>` tento element obsahuje id ovládacího prvku, ke kterému patří a výčet

jednotlivých změn, které jsou zastoupeny pomocí dalších XML elementů pojmenovaných dle změněných vlastností. Třída *UIComponentComparator* se stará o nalezení odpovídající třídy, která zastupuje daný symbol a o nalezení odpovídajícího ovládacího prvku. Pro tento prvek pak porovná jednotlivé změny a uloží informaci o tom, že daná vlastnost musí být zviditelněna pomocí dependency property a také dané třídě zastupující instanci toho symbolu uloží změněné hodnoty. Zbytek procesu pak již probíhá při serializaci.

Komplikace nastává u třech ovládacích prvků těmi jsou CheckBoxGroup RadioButtonGroup. První dva ovládací nemají v UWP přímý protějšek, jsou transformovány do jednotlivých radio tlačítek a zaškrtávacích tlačítek. Problém byl, že v XAML souboru jsou tyto tři ovládací prvky reprezentovány pouze jako text, který obsahuje některé speciální symboly. Je tedy možné v nich provádět poměrně zásadní změny. Například některé zaškrtávací tlačítko ze skupiny odstranit. Odstranění právě působí zásadní problém, protože nelze snadno určit, které z tlačítek bylo odstraněno a jak toto odstranění provést v UWP. Řešení jsem zvolil tak, že pro odstranění daného řádku je nutné vyřešit tím, že místo uživatel napíše „DELETED“, aby šlo poznat, který z řádků byl odstraněn. V cílovém UWP pak odstranění jednotlivého prvku realizuji jako nastavení jeho viditelnosti na hodnotu collapsed čímž zařídím, že není pro danou instanci vidět.

Dalším problematický elementem je comboBox, kde kvůli změnám v jeho items bylo nutné vytvořit přím v jazyce C# pole těchto itemů a propojit s ním comboBox. Není jiná možnost, jak v jednotlivých instancích měnit množství těchto itemů, v Balsamiq Mockups je totiž možné tyto itemy odstraňovat.

Další komplikaci přinesly symboly v tom, že kvůli změnám pro jednotlivé instance a k tomu potřebnému konstruktu dependency property je nutné hodnoty vlastností ovládacích prvků transformovat nejen do formát XAML, ale také pří do kódu jazyce C#. Formát těchto hodnot se v jazyce C# a ve formátu XAML se velmi liší. Musela tedy přibýt třída *Utils,* která obsahuje metodu *PrepareAttributeForCSSerialization* realizující transformaci těchto hodnot do jazyka C#.

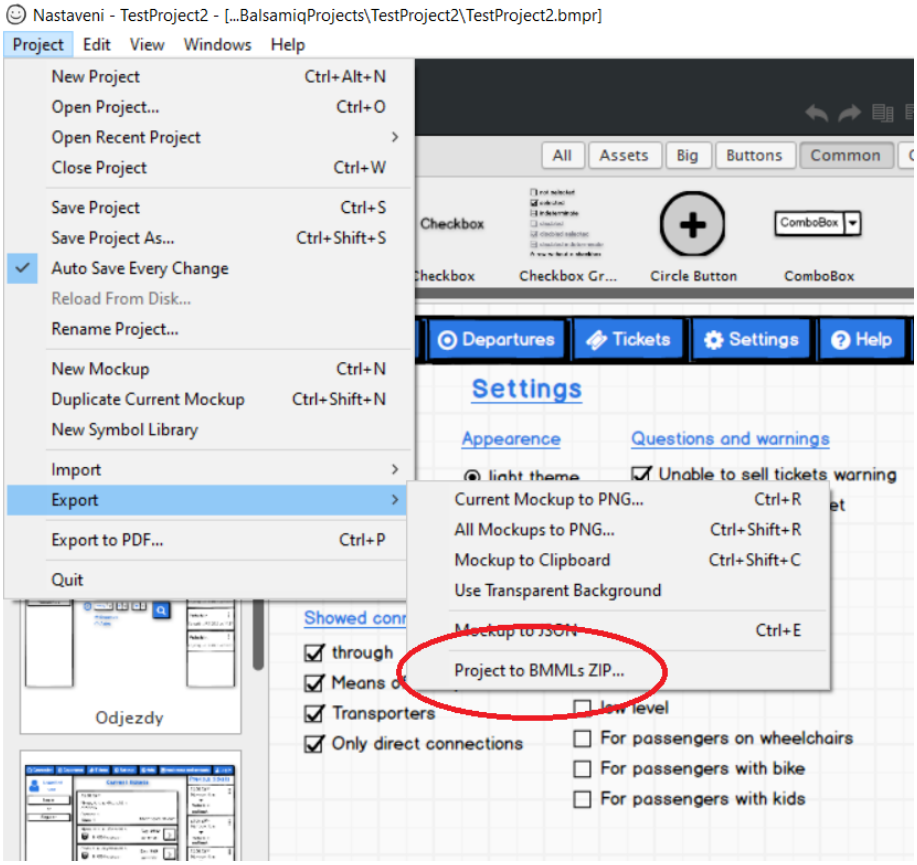
6.2 Serializace

Načtený prototyp je poté serializován do formátu Xaml a formátu cs. Na to je využívána třída *XmlWriter* ze jmenného prostoru *System.Xml.* Tato třída se ukázala jako

nejlepší pro mé potřeby, problémy nastávají pouze u některých vlastností, některých elementů a také u bindingu. Každá třída, která zastupuje ovládací prvek má vlastní metodu *serialize*, která provádí serializaci do formátu Xaml. V těchto metodách mohou probíhat dodatečné transformace, které nebylo možné provést hned při načítání. Tato metoda je volána z metody *Serialize*, která přísluší třídě *Page*, která reprezentuje danou obrazovku. Tato metoda získává od jednotlivých objektů, zastupujících jednotlivé ovládací prvky, *string* se serializovanými daty, které spojuje dohromady, aby vytvořila kompletní obrazovku.

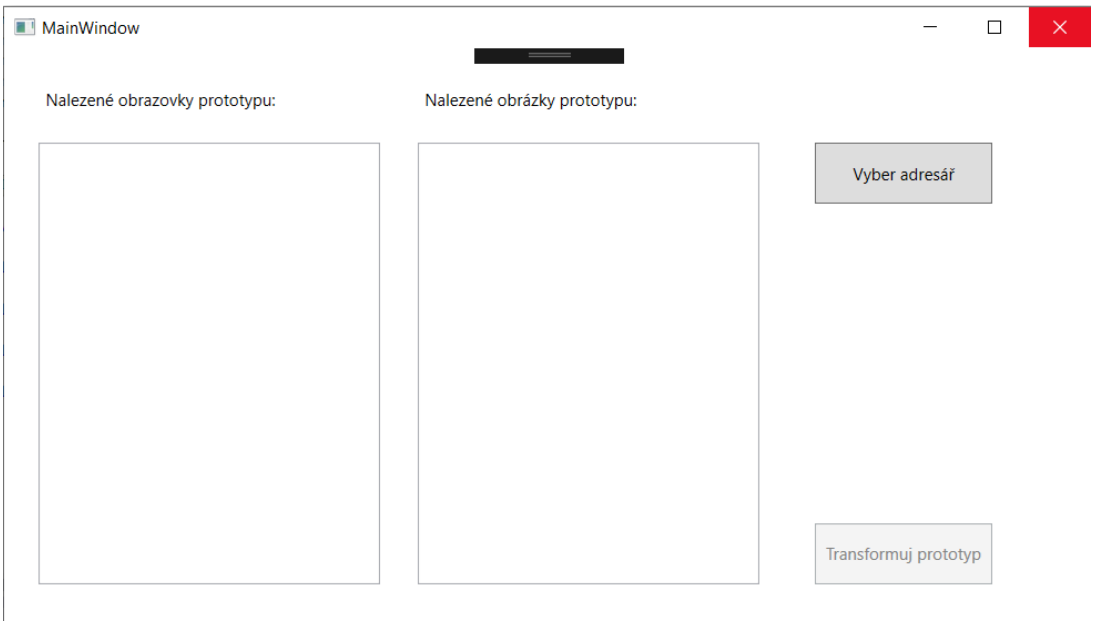
7. Kapitola
Návrh a implementace programu

Pro použití programu je nutné mít vstup ve správném formátu. Jako vstup slouží prototypy vytvořené v programu Balsamiq Mockups. Tento prototyp je nutné z programu správně vyexportovat. Na to slouží možnost „Project to BMMLs ZIP ...“. Tuto variantu exportu najdete v záložce „Project“ pod možností „Export“, viz Obrázek 4.



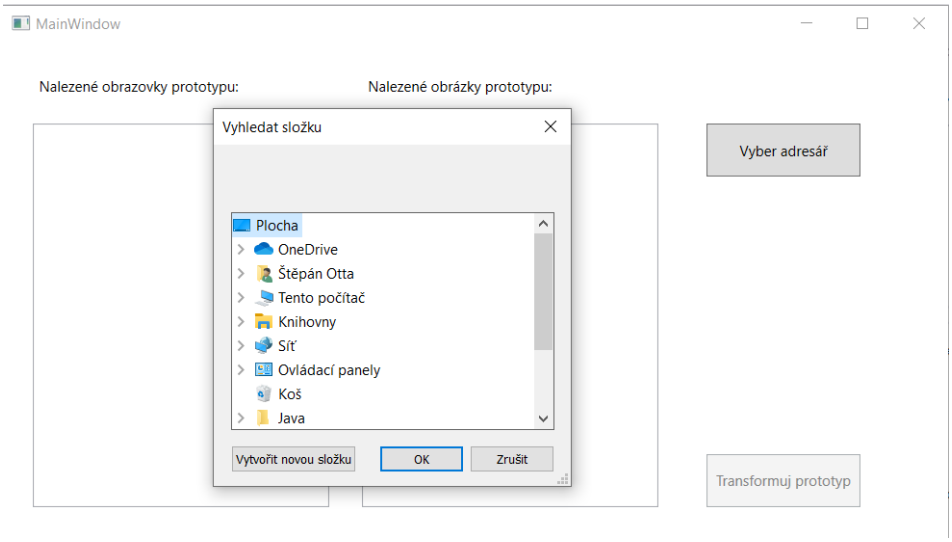
Obrázek 4 : *Export prototypu z Balsamiq Mockups*

Program provede konverzi a zobrazí vám dialogové okno, ve kterém zvolíte kam chcete výsledek uložit. Poté je nutné tento soubor zip extrahovat do složky. Tak aby byly všechny extrahované soubory ve složce se stejným názvem jako je název souboru zip. Poté již můžeme přejít k transformaci pomocí našeho programu.



Obrázek 5 : *Uživatelské rozhraní programu*

Program je ovládán jednoduchým uživatelským rozhraním, viz Obrázek 5. Uživatelské rozhraní sestává pouze ze dvou tlačítek a dvou seznamů.

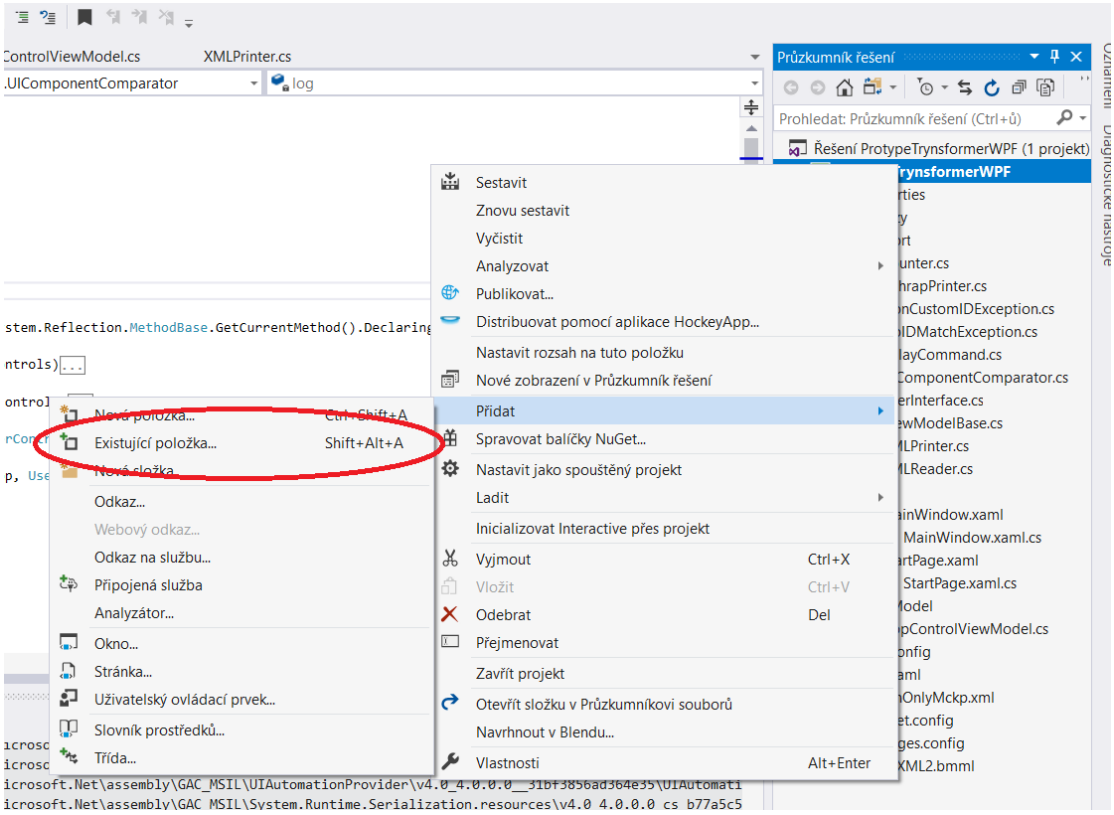


Obrázek 6 : *Výběr složky s prototypem pro transformaci*

První tlačítkem uživatel zvolí složku, ve které se nachází soubory transformovaného prototypu. Po stisknutí daného tlačítka se uživateli zobrazí dialogové okno pro zvolení složky, viz Obrázek 6.

Po zvolení složky se v prvním seznamu zobrazí jednotlivé soubory formátu bmml, které program ve složce detekoval. Ve druhém seznamu se zobrazí obrázky, které k prototypu patří a nacházejí se v podsložce Assets. Druhé tlačítko je neaktivní, dokud není načten prototyp pro transformaci. Druhým tlačítkem se zahájí transformace, když je transformace dokončená zmizí ze seznamů soubory i obrázky a tlačítko pro transformaci bude opět neaktivní.

Program vytvoří jednotlivé soubory se zdrojovým kódem. Tyto soubory jsou uloženy ve složce, ve které byl uložen prototyp. Pro spuštění prototypu je nutné poté soubory přidat do projektu v programu Visual Studio, viz Obrázek 7.



Obrázek 7 : *Přidat položku do projektu ve Visual Studiu*

7.1 Implementace

Program byl vytvořen v jazyce C#, ve frameworku .NET pro platformu UWP. Implementoval jsem několik tříd, které zastupují určité úkony v rámci celé transformace, kterou program vykonává.

Základním kamenem aplikace je třída *AppControlViewModel*, která je spojena s uživatelským rozhraním a obstarává reakce na vstup uživatele. Tato třída plní požadavky uživatele ve spolupráci s dalšími třídami. Po načtení prototypu a poté co uživatel spustí transformaci předá tato třída načtené soubory prototypu třídě *XMLReader*. Třída *XMLReader* se stará o načítání a transformaci jednotlivých souborů prototypu. Postupně po jednotlivých souborech zastupující jednotlivé obrazovky načte a transformuje soubory do tříd, které zastupují jednotlivé obrazovky a ovládací prvky. Pokud se v prototypu používají symboly, pak se pro načtení a transformaci symbolů využije třída *UIComponentComparator*. Tato třídy slouží k vyhledání symbolů podle id a názvu a poté do nich transformuje data pro jednotlivé instance těchto symbolů.

Třída *Page* zastupuje obrazovku prototypu. Pro jednotlivé ovládací prvky jsou to jednotlivé třídy, všechny dědí z abstraktní třídy *PrototypeGUIElement*. Třída *UserControl* zastupuje symboly, které jsou do ní transformovány a serializovány jako uživatelské ovládací prvky. Po dokončení transformace je provedena serializace, kterou provádí třídy *XMLPrinter* a *CShrapPrinter*. Jedna třída zajišťuje serializaci souborů ve formátu cs, tedy kód v jazyce C# a druhá třída zajišťuje serializaci souborů ve formátu XAML.

Pomocná třída *Counter* slouží ke generování unikátních id pro jednotlivé načítané ovládací prvky.

8. Kapitola
Testování programu

Program jsem testoval na 4 prototypech. Zvolil jsem takové 4 typy těchto prototypů. První prototyp jsem koncipoval jako prototyp uživatelského rozhraní mobilní aplikace určené pro cestovatele. Tato aplikace by uživateli nabízela hledání míst na mapě, plánování cest, informace o počasí v zvolených destinacích a další funkce. Druhý prototyp je prototyp programu pro stolní počítač, aplikace je určená pro vyhledávání

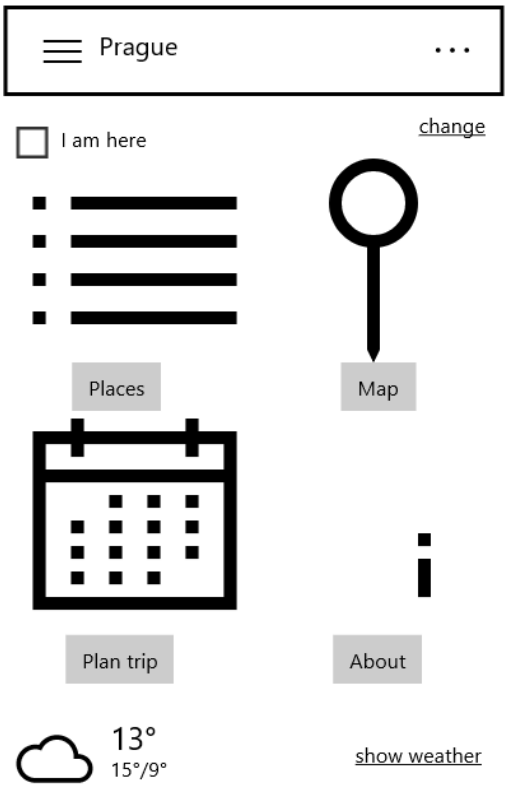
dopravních spojení a nákup jízdenek. Třetí prototyp není prototyp žádné konkrétní, smysluplné aplikace, ale je vytvořen tak, aby co nejvíce otestoval a ukázal možnosti tohoto programu. Jeho obrazovky tedy obsahují všechny ovládací prvky, které je program schopen transformovat a také co největší množství variací těchto prvků, tak aby byly otestovány všechny funkce programu. Obsahuje také obrazovky zaměřené na testování transformace skupin a symbolů. Čtvrtý prototyp je zaměřen na použití symbolů. Program jsem testoval zároveň i v průběhu vývoje na prototypech vytvořených pro situace, které bylo třeba testovat. Tyto prototypy jsem vytvářel přímo pro konkrétní situace, které jsem se snažil v danou chvíli otestovat, nejsou proto prototypem žádné smysluplné aplikace, ale sloužili pouze pro velice konkrétní testovací situace.

Finální testovaní na čtyřech uvedených prototypech probíhalo tak, že jsem daný prototyp exportoval z Balsamiq Mockups, načetl do programu, transformoval a poté jsem porovnával původní obrazovky vytvořené v Balsamiq Mockups a mé transformované obrazovky po spuštění. Většina rozdílů je minimální a je způsobená rozdílností elementů, který nabízí Balsamiq Mockups, a které nabízí UWP.

8.1 Porovnání výsledků

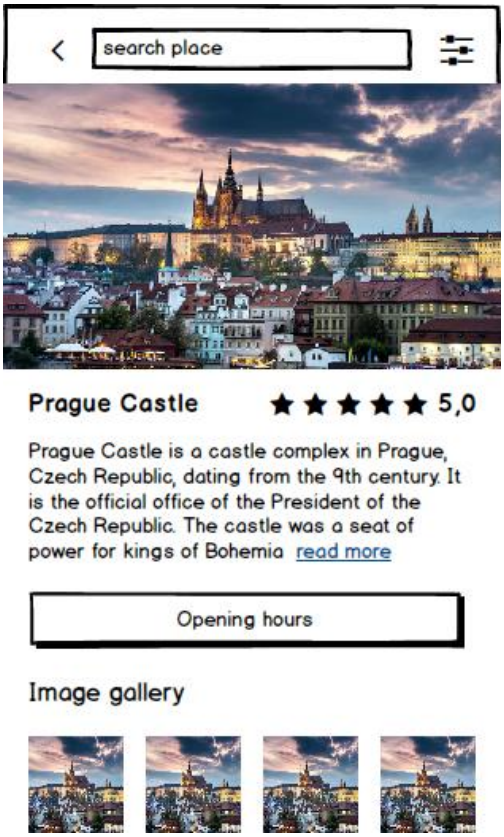
Zde uvedu jednotlivé příklady porovnání obrazovek vstupních prototypů z programu Balsamiq Mockups a výsledných transformovaných prototypů v UWP. Prvním z finálních testovacích prototypů je prototyp mobilní aplikace určené pro cestování. Tento prototyp otestoval schopnost transformovat rozložení elementů vhodné pro mobilní telefony. Dále dobře testuje i většinu základních elementů, které program dokáže transformovat.

Obrázek 9 : *Homepage – UWP*

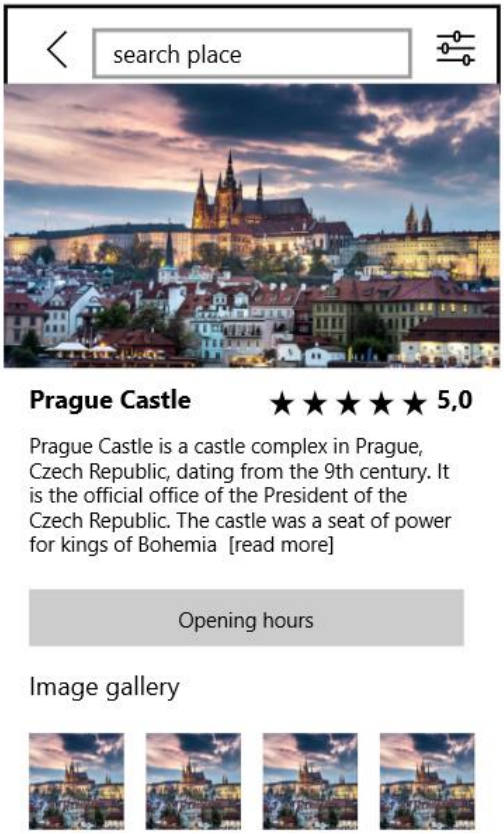


Obrázek 8 : *Homepage – Balsamiq*

Lze pozorovat rozdíl v ikonách, protože jsou tu použité dva různé fonty. Ikony obsažené v programu Balsamiq Mockups nelze bohužel transformovat jedna ku jedné. Další malý rozdíl je ve vzhledu tlačítek, nativní tlačítka pro univerzální platformu Windows mají defaultní vzhled jiný, něž tlačítka z Balsamiq Mockups, udržet však úplně přesný vzhled všech komponent nebylo účelem, ani by to nemělo příliš smysl. Transformace drží přesně rozložení jednotlivých ovládacích prvků i jejich rozměry, obsah a grafické zpracování. Viz Obrázek 8 a Obrázek 9.



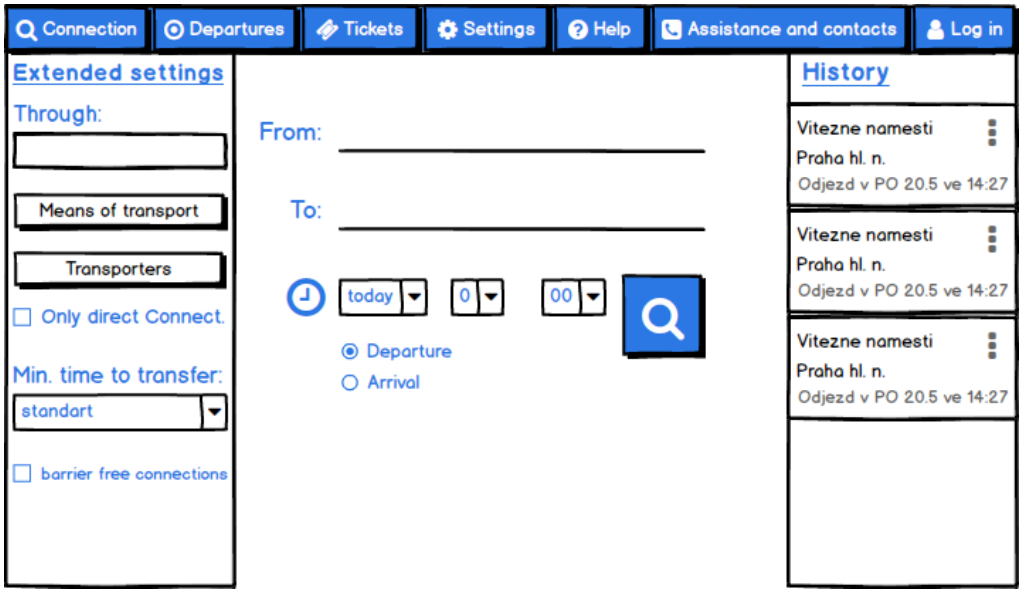
Obrázek 11 : *Place Detail – Balsamiq*



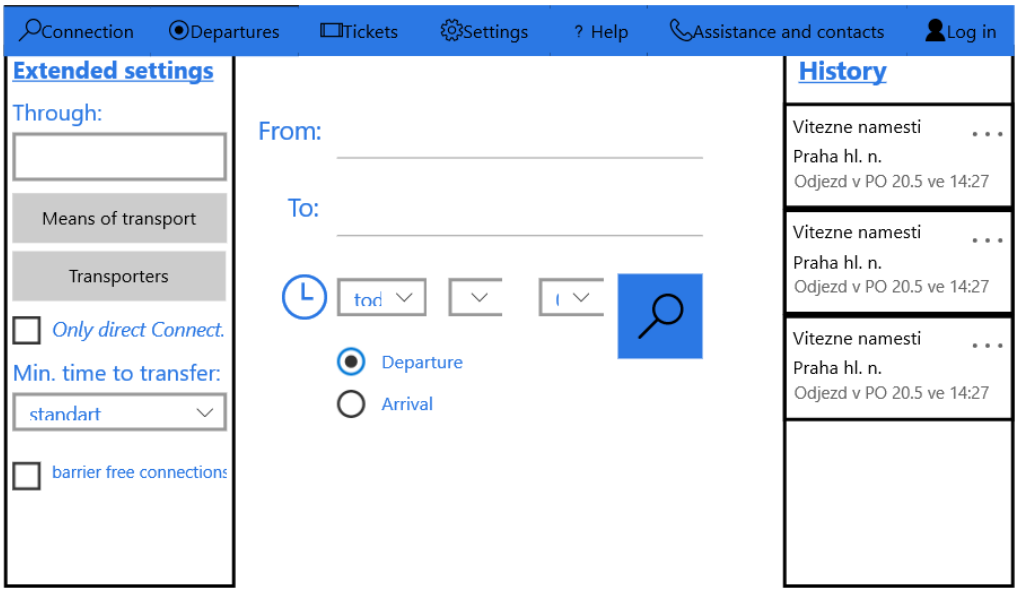
Obrázek 10 : *Place Detail – UWP*

Program bez problémů transformuje obrázky, viz Obrázek 10 a Obrázek 11.

Druhým testovaným prototypem je prototyp programu pro stolní počítač, který je určen pro nákup jízdenek a vyhledávání spojení. Tento prototyp testuje transformaci rozložení elementů pro stolní počítač a také znovu testuje většinu základních elementů, které program dokáže transformovat.



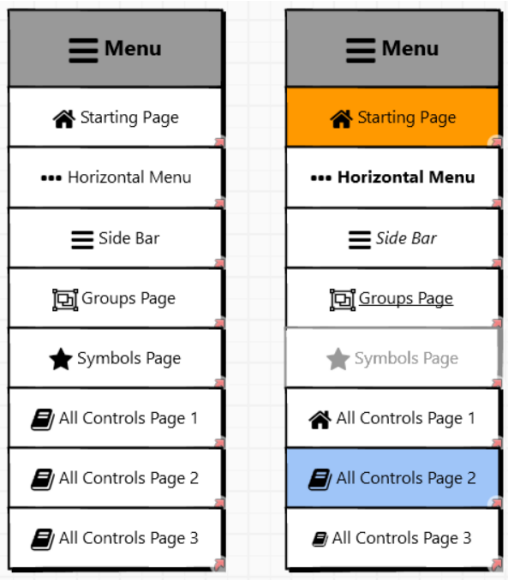
Obrázek 12 : *Spojení – Balsamiq*



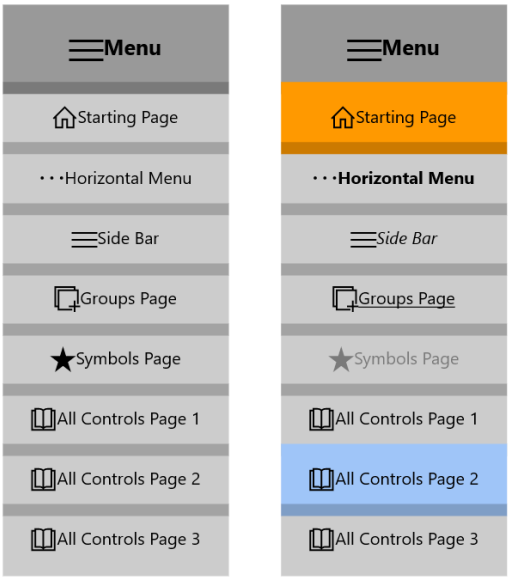
Obrázek 13 : *Spojení – UWP*

Program si poradí i s rozložením prototypu určeném pro desktop, viz Obrázek 12 a Obrázek 13.

Třetí testovaný prototyp nebyl prototyp smysluplné aplikace, ale byl to prototyp vytvořen tak, aby testoval co největší množství funkcí programu. Jsou zde proto testovány všechny ovládací prvky, které program transformuje. Jsou zde obrazovky, které testují transformaci skupin a symbolů. Také jsou zde testovány přechody mezi obrazovkami po uživatelských kliknutích.

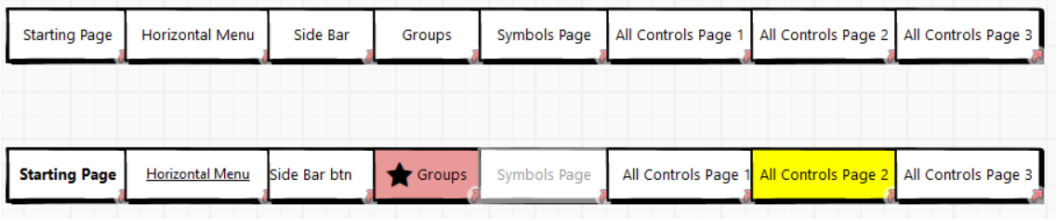


Obrázek 14 : *Postranní menu – Balsamiq*

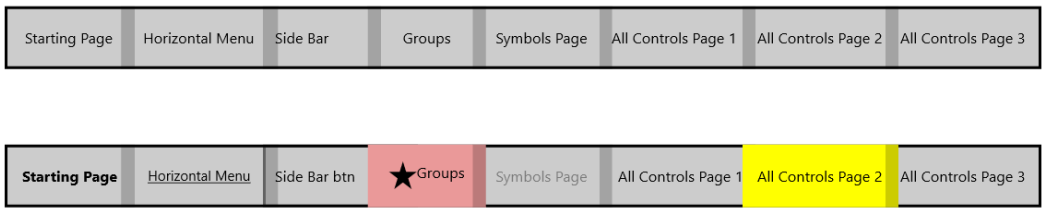


Obrázek 15 : *Postanní menu - UWP*

Na Obrázek 14 a Obrázek 15 je vidět transformace symbolu. Jako symbol bylo vytvořeno postranní menu a toto menu bylo transformováno do uživatelského ovládacího prvku. Jsou zde také dvě instance toho symbolu. První instance je nezměněná, tedy jen vložený originální symbol, druhá instance obsahuje řadu lokálních změn. Obě instance jsou správně transformovány, včetně lokálních změn pro druhou instanci.



Obrázek 17 : *horizontal Menu – Balsamiq*

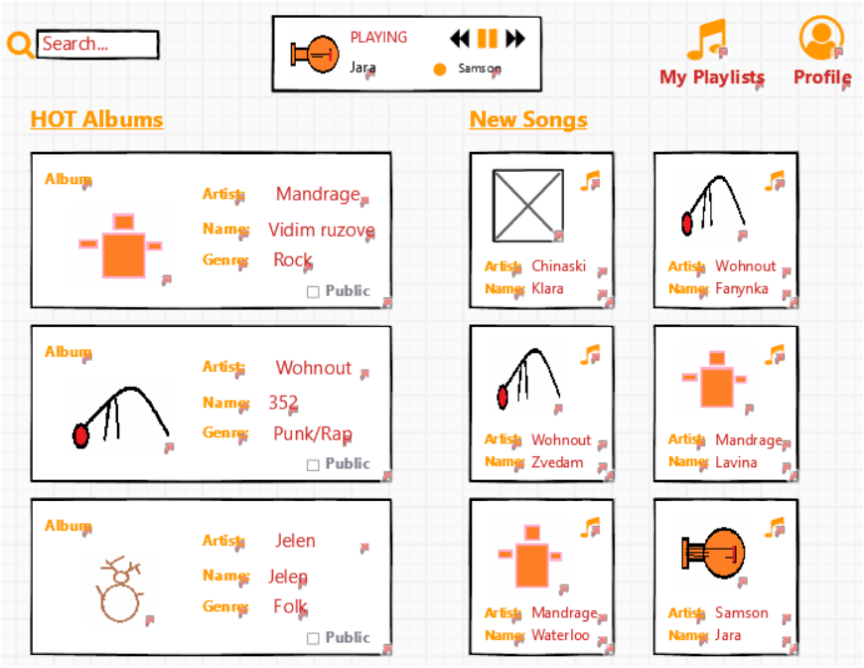


Obrázek 16 : *Horizontal Menu – UWP*

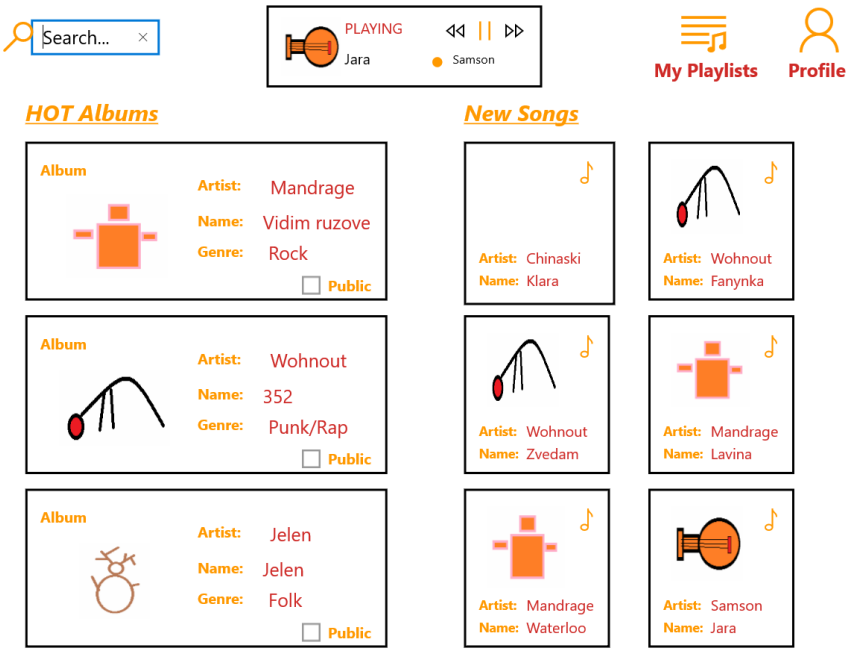
Na Obrázek 17 a Obrázek 16 je vidět druhý příklad transformace symbolu. Jako symbol bylo vytvořeno horizontální menu. Opět jsou zde vidět dvě instance, jedna pouze nezměněný symbol a druhá symbol s množstvím různých změn. I zde je vidět, že

transformace bez problémů proběhla a obě instance jsou transformovány, včetně změn pro druhou instanci.

Čtvrtý a poslední prototyp je aplikace pro přehrávání a sdílení hudby. Aplikace by primárně měla být určena pro herní konzole xBox. Na tomto prototypu se tedy testuje další poněkud odlišné rozložení ovládacích prvků. Také je tento prototyp zaměřen na použití symbolů, symboly v něm hrají zásadní roli.



Obrázek 18 : *Home Page – Balsamiq*



Obrázek 19 : *Home page – UWP*

Na obrázcích Obrázek 18 a Obrázek 19, můžete vidět transformaci obrazovky s několika instancemi několika symbolů. Všechny obrázky i text byly správně transformovány pro každou instanci každého symbolu.

9. Kapitola
Závěr

V teoretické části práce jsem se zabýval typy prototypů dle věrnosti. Popsal jsem jednotlivé typy prototypů, způsob jejich tvorby a zaměřil jsem se na jejich rozdíly, podobnosti. Také jsem se zaměřil na elementy používané v těchto prototypech a jejich specifické vlastnosti a možnosti. Dále jsem se zabýval propojením prototypů jednotlivých úrovní a také s finálním produktem. Popsal jsem aktuální situaci, která se týká možnosti přechodů mezi jednotlivými prototypy. Zaměřil jsem se na možnosti transformace prototypů nízkých věrností na prototypy vysokých věrností. Popsal jsem aplikace Framer X a Supernova Studio, které se v současné chvíli zabývají propojováním prototypů různých úrovní.

V praktické části jsem se zabýval výběrem programu, který je ideální pro tvorbu vstupních prototypů pro transformaci do nativního kódu. Poté jsem zvolil platformu, na kterou jsou výsledné transformované prototypy určeny. Hlavním výstupem práce je poté program, který transformuje prototypy nízké věrnosti vytvořené v programu Balsamiq Mockups do zdrojového kódu v jazyce C# pro univerzální platformu Windows. Aplikaci je možné využít v procesu tvorby uživatelského rozhraní pro přemostění z prototypu nízké věrnosti na prototyp vysoké věrnosti, což zvýší efektivitu práce a sníží to množství času, který by bylo nutné vynaložit při tvorbě prototypu vysoké věrnosti znovu od nuly a také to dá k dispozici výchozí bod pro vytvoření finálního grafického uživatelského rozhraní.

Hlavní výhodou je, že program dokáže velice dobře transformovat prototypy vytvořené v Balsamiq Mockups, aniž by výsledek byl zatížen nějakými významnými rozdíly. Program si velmi dobře poradí i se Symboly, což jsou uživateli vytvořené vlastní ovládací prvky. Jediné rozdíly, které se občas objevují jsou problémy s velikostí některých ovládacích prvků, kvůli tomu, že rozměry těchto prvků pro univerzální platformu Windows zahrnují větší pěvné okraje, což občas zapříčiní, že je část textu uříznutá a není vidět.

Do budoucna by program mohl dostat některá vylepšení. Prvním vylepšením by bylo doplnit transformaci pro všechny ovládací prvky, které Balsamiq Mockups nabízí. Dalším vylepšením by mohlo být více intuitivní a příjemné uživatelské rozhraní, které by bylo třeba testovat na potencionálních uživatelích. Současné uživatelské rozhraní jsem netestoval, proto nemohu určit, zda je plně dostačující.

Literatura

1. **Esposito, Emily.** Low-fidelity vs. high-fidelity prototyping. *invisionapp.com.* [Online] 29. květen 2018. https://www.invisionapp.com/inside-design/low-fi-vs-hi-fi-prototyping/.

2. **Beecher, Fred.** boxesandarrows.com. *boxesandarrows.* [Online] září 22, 2009. http://boxesandarrows.com/integrating-prototyping-into-your-design-process/.

3. **Babich, Nick.** Adobe Blog. *Prototyping 101: The Difference between Low-Fidelity and High-Fidelity Prototypes and When to Use Each.* [Online] 29. Listopad 2017. https://theblog.adobe.com/prototyping-difference-low-fidelity-high-fidelity-prototypes-use/.

4. **Snyder, Carolyn.** *Paper Prototyping.* místo neznámé : Morgan Kaufmann, 2003.

5. **Supernova Studio.** Supernova. [Online] 2019. https://supernova.io/.

6. **Supernova.** *Prototypr.io.* [Online] 13. Března 2018. https://blog.prototypr.io/introducing-supernova-studio-35335de5044c.

7. **To, Meng.** Design and Code in Framer X. *designcode.io.* [Online] 2019. https://designcode.io/framer-x-course.

8. **Framer BV.** *Framer.* [Online] 2019. https://www.framer.com/.

9. **Peldi.** New File Extension: BMML. *https://blog.balsamiq.com/.* [Online] 30. Srpen 2008. [Citace: 20. Duben 2019.] https://blog.balsamiq.com/new-file-extension-bmml/.

10. **Warfel, Todd Zaki.** *prototyping: a practitioner's guide.* Brookly, N.Y. : Rosenfeld Media, 2009. 1933820217.

11. **Lowdermilk, Travis.** *User-Centered Design: A Developer's Guide to Building User-Friendly Applications.* místo neznámé : O'Reilly Media, 2013. 1449359809.

12. **Whitney, Tyler , a další.** What's a Universal Windows Platform (UWP) app? *Microsoft Docs.* [Online] 7. Květen 2018. https://docs.microsoft.com/cs-cz/windows/uwp/get-started/universal-application-platform-guide.

Příloha A
Obsah přiloženého DVD

- Projekt ve Visual studiu ve složce PrototypeTransformer
- Složka TransformedPrototypes
  - Obrázky všech oken všech čtyřech testovacích prototypů jak z Programu Balsamiq Mockups, tak i výsledných transformovaných prototypů
  - Projekt obsahující všechny 4 výsledné transformované testovací prototypy
- Projekty v programu Balsamiq Mockups obsahující všechny 4 testovací prototypy ve složce TestingPrototypes
- Ve složce ReleasedTransformer se nachází výsledný, spustitelný program

Příloha B
Postup práce s programem

- Prvním krokem je vytvořit projekt v programu Balsamiq Mockups
- Když projekt máme, musíme ho exportovat, v záložce *Project* se nachází možnost *Export* a pod ní se nachází možnost *Project to BMMLs ZIP...*
- Výsledný zip si uložíme, kam potřebujeme a rozbalíme do složky, je nutné, aby se tato složka jmenoval stejně jako daný projekt
- Poté spustíme program
    - Spustit program může pomocí přiloženého exe souboru
    - Druhou možností spuštění programu je otevření přiloženého projektu ve Visual studiu a jeho spuštění tam
- V programu stiskneme tlačítko *Vyber adresář*, v dialogovém okně zvolíme složku s námi exportovaným projektem
- Načtené obrazovky a obrázky si můžeme zkontrolovat v seznamech
- Poté stiskneme tlačítko *Transformuj prototyp*
- Když je transformace dokončená seznamy se vyprázdní a nebude dále možné stisknout tlačítko *Transformuj prototyp*
- Výsledné transformované soubory nalezneme v původní složce společně s původními soubory, pro spuštění je nutné tyto soubory přidat do projektu ve Visual studiu, pozor na jmenný soubor tohoto projektu je nutné, aby byl stejný jako název projekt v Balsamiq Mockups