VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

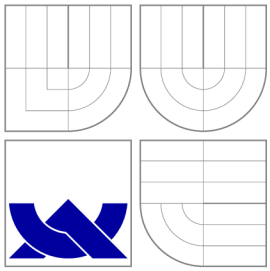
PLÁNOVÁNÍ A SLEDOVÁNÍ ORGANIZACE ČASU PRO WINDOWS PHONE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

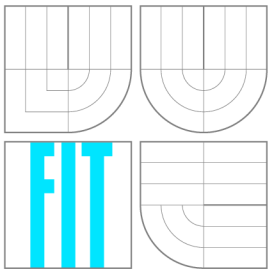
AUTOR PRÁCE
AUTHOR

KAREL POPELKA

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PLÁNOVÁNÍ A SLEDOVÁNÍ ORGANIZACE ČASU PRO WINDOWS PHONE

PLANNING AND MONITORING OF TIME MANAGEMENT FOR WINDOWS PHONE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

KAREL POPELKA

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. ADAM HEROUT, Ph.D.

BRNO 2014

Abstrakt

Cílem práce je popsat vývoj aplikace pro plánování a sledování organizace času. Vývoj je cílen na platformu Windows Phone 8 se zaměřením na intuitivní uživatelské rozhraní. Implementace aplikace je řešena pomocí návrhového vzoru MVVM a platformních doporučení implementované v jazyce C#. Uživatelské rozhraní je navrženo pomocí jazyka XAML. Výsledkem je uživatelsky přívětivá aplikace, kterou lze využít při organizaci běžných aktivit. Hlavním přínosem práce je pochopení platformních doporučení Windows Phone 8 a principů přívětivého uživatelského rozhraní. Na závěr je rozebráno testování aplikace a vyhodnocení reakce respondentů.

Abstract

This bachelor thesis describes the development of an application for planning and monitoring of time management. As the target architecture, Windows Phone 8 was chosen with aiming at intuitive user interface. The application is implemented using MVVM design pattern by platform-recommended C# language. The user interface is built by using the XAML language. The result is a user friendly application suitable for organizing common activities. The main feature of this thesis is understanding of platform recommendations for Windows Phone 8 as well as friendly user interface principles. The conclusion includes a detailed description of application testing and an assessment of respondent reactions.

Klíčová slova

Windows Phone 8, WP8, LINQ, MVVM, Organizace času, Zaznamenávání času, XAML, C#, WPF, Mobilní aplikace, Uživatelské rozhraní

Keywords

Windows Phone 8, WP8, LINQ, MVVM, Time management, Time record, XAML, C#, WPF, Mobile application, User interface

Citace

Karel Popelka: Plánování a sledování organizace času pro Windows Phone, bakalářská práce, Brno, FIT VUT v Brně, 2014

Plánování a sledování organizace času pro Windows Phone

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Doc. Ing. Adam Herout, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Karel Popelka
16. května 2014

Poděkování

Děkuji Doc. Ing. Adam Heroutovi, Ph.D. za odborné vedení práce a cenné rady při řešení návrhu aplikace. Dále bych chtěl poděkovat rodičům a všem ostatním, kteří mě při psaní práce podpořili.

Obsah

Kapitola 1

Úvod

Tato bakalářská práce popisuje úplný návrh aplikace pro sledování a monitorování času pro platformu Windows Phone 8. Práce se zaměřuje na rychlost a přívětivost aplikace.

Nejdříve bude obecně popsán pohled na platformu. Budou se zde řešit minimální specifikace mobilních přístrojů, dostupné vývojové prostředí, možnost emulování aplikace, použitelné programovací jazyky a různé způsoby přístupu k programovaní. Na závěr úvodu do platformy bude popsán teoreticky návrhový vzor MVVM, možnost přidání knihoven třetích stran a možností Windows Phone Store.

Následující kapitola se bude zabývat úvodem do problematiky plánování a monitorování času. Bude zde také probrána analýza již existujících řešení.

Po analýze existujících řešení se bude práce zabývat obecným návrhem aplikace a problémům spojených s přívětivým uživatelským rozhraním. Bude také řešit problémy zjištěné při návrhu již existujících řešení.

Po návrhu aplikace následuje samotné implementace pomocí návrhového vzoru MVVM. Je zde popsáno rozdělení projektu do přehledné struktury a vytvoření jednotlivých vrstev pomocí předefinovaných bázových tříd. Dále je v této kapitole obsažená globalizace a lokalizace stránek do určitých jazyků.

Předposlední kapitola je zaměřena na popisu testování aplikace v průběhu vývoje. Následuje rozbor hodnocení a názor respondentů na výslednou aplikaci.

V závěru bude zhodnocení výsledků práce a návrhy možné pokračování projektu.

Kapitola 2

Platforma Windows Phone

* Mobilní platforma Windows Phone (dále jen WP), která byla poprvé uvedená na trh v říjnu roku 2010 již prošla velkým vývojem. Její první verze WP7 nebyla z počátku uživatelsky oblíbená. Kopírování souborů přes USB kabel pomocí protokolu USB Mass Storage nebo nastavení vlastní melodie nebylo podporováno. Naproti tomu byl vývoj WP směřován na stabilitu systému, díky které měl WP i své fanoušky.

Naštěstí si Microsoft uvědomil, že tímto směrem se dále ubírat nelze a ve verzi WP8 se již věci pohnuly k lepšímu. Microsoft předělal kompletně jádro systému a WP8 splňoval všechny základní ovládací prvky mobilního operačního systému (dále jen OS), na které uživatelé byli zvyklí. To však mělo kritický následek na uživatele mobilních telefonů s WP7. Mobilní telefony s WP7 nebylo možné kvůli změně jádra aktualizovat na novější WP8 a proto vznikl update WP7.8 u kterého bohužel vývoj WP7 končil.

Platforma je také známa pro její designový styl nazvaný Metro. Tento styl je velice střídmý a základní inspirace byla objevena v letištních tabulích či metrech, které lidi často potkávají a jsou jim velice blízké. Mezi hlavní rysy tohoto stylu patří rychlost, čitelnost a jednoduchost. Při návrhu je dobré dodržovat zásady stylu Metro. Pokud uživatel otevře aplikaci a objeví se úplně jiný styl musí si navyknout na nový styl aplikace. Pokud se aplikace přizpůsobí uživatelskému tématu a bude dodržovat designové zásady, uživatel téměř nepozná rozdíl a aplikaci bude ovládat intuitivně. Z tohoto důvodu jsem kladl velký důraz při vývoji na programátorská platformní doporučení.

2.1 Podporovaný hardware přístrojů Windows Phone 8

Microsoft chytře zavedl minimální konfiguraci pro mobilní telefony. Tímto krokem si zajistil svižnost OS na mobilních přístrojích nižších tříd. Programátor po zvolení platformy již má obrázek o minimální konfiguraci zařízení. Nicméně při vývoji aplikace je dobré mít základní pojem o přístrojích, pro které bude jeho aplikace určena. Například pokud vyvíjí programátor aplikaci v emulátoru na výkonném počítači a poté spustí na přístroji reakce budou u náročných operací někdy i výrazně pomalejší. Minimální uvedená konfigurace WP8:

- Qualcomm Snapdragon S4 dual-core processor
- minimálně 512MB RAM pro WVGA modely
- minimálně 1GB RAM pro 720p a WXGA modely

*Kapitola popisuje teoretické znalosti, které vychází převážně z literatury [6], [11], [8], [12] a [9]. Hlavní myšlenka není mým vlastním dílem nýbrž interpretací studovaného problému.

- minimálně 4GB flash paměti
- GPS a A-GNSS
- podpora micro-USB 2.0
- senzor polohy, senzor okolního světla (magnetometr a gyroskop jsou volitelné)
- 802.11b/g a Bluetooth
- podpora hardwarové akcelerace DirectX s hardwarovou akcelerací Direct3D a užitím programovatelného GPU
- více dotykový kapacitní displej minimálně se čtyřmi dotykovými body současně

Z hlediska vývoje aplikace pro záznam a organizace času jsem se zaměřil na podporované rozlišení displejů u WP8. U předchozí verze bylo rozlišení podporováno pouze jedno a to 480 x 800 px. S nástupem WP8 se podpora rozlišení displeje rozšířila. Podporované rozlišení lze vidět v tabulce 2.1. I když Layout WP8 klade velký důraz na nezávislost velikosti mezi různými rozlišeními, je přesto dobré při návrhu myslet na různá rozlišení a layout aplikace tomu přizpůsobit.

Tabulka 2.1: Přehled podporovaných rozlišení pro WP8

Název rozlišení	Rozlišení [px]	Poměr
WVGA	480 x 800	15:9
WXGA	768 x 1280	15:9
720p	720 x 1280	16:9

2.2 Vývojové prostředí a emulátor

Další důležitou částí při studiu platformy bylo seznámení s programovacím rozhraním platformy WP8.

Aplikace pro WP8 jsou vyvíjeny v prostředí Visual Studio. Pro vývoj aplikací a jejich publikaci je nejdříve nutné se u firmy Microsoft zaregistrovat jako vývojář. Microsoft také nabízí studentům některých fakult registraci zdarma. Po registraci je možné odblokovat telefon a aplikaci testovat na něm. Dále je možné využít emulátor. Po příchodu WP8 nebylo možné spustit emulátor na jakékoliv sestavě. Microsoft si uvědomil, že vývojáři chtějí svižný a stabilní emulátor. Proto postavil emulátor na technologii Hyper-V, která vyžaduje HW podporu SLAT v procesoru. Emulátor je díky této technologii velice svižný a jeho používání je velmi pohodlné.

Jak jsem již zmiňoval, je nutné si uvědomit cílovou konfiguraci zařízení, pro které je aplikace určena. Proto Microsoft přišel s pomocí programátorům formou různých emulátorů. Emulátory běžící na Hyper-V jsou dostupné ve všech rozlišeních a různých RAM modifikacích. Proto je dobré při vývoji vyzkoušet aplikace na emulátoru s jiným rozlišením a omezenou RAM pamětí. Dále je možné také vyzkoušet aplikaci na emulátoru různých verzí WP8.

Po úspěšné instalaci emulátoru a Visual Studia jsem se zaměřil na nástroje, které Visual Studio nabízí. Jednou z velmi užitečných funkcí je interaktivní debugger. Díky interpretovanému přístupu platformy .NET je možné aplikaci zastavit a přímo zadat příkaz, který se

v danou chvíli provede. Je ale důležité mít na paměti to, že se provedený příkaz promítne do aktuálního stavu aplikace. Například při změně proměnné bude po pokračování aplikace proměnná nabývat nové hodnoty. Další výhodou je zastavení v cyklu až po několikáté iteraci. Viděl jsem již mono přístupů ladění aplikace, ale z praxe jsem zjistil, že je výhodné se nejdříve dobře seznámit s nabízenými možnostmi vývoje. Mnoho lidí ztratí drahocenný čas procházením ladících výpisů a přitom mají k dispozici tak mocný nástroj.

2.3 Programovací jazyky dostupné při vývoji Windows Phone

Další důležitou součástí je pochopení a prostudování možností nabízených jazyků pro psaní kódu. Mnoho programátorů má tendenci psát co nejvíce kódů sami a nakonec zjistí, že již někdo předním něco podobného udělal a ještě efektivněji. Tohoto přístupu jsem se chtěl vyvarovat a nejdříve prostudoval nabízené možnosti a dostupné knihovny.

Jako programovací jazyky lze využít XAML, který rozeberu vzápětí. Dále jsou dostupné programovací jazyky jako je C/C++, C# nebo Visual Basic .NET. Obrázek 2.1 ukazuje využití jazyků podle způsobu programování.

2.3.1 XAML

Při programování uživatelského rozhraní (dále jen UI) je hlavním stavebním kamenem XAML, což je anglická zkratka pro Extensible Application Markup Language. Tento jazyk není nic víc, než deklarativní jazyk založený na XML. Jeho použití je jednoduché a využívá také jmenné prostory jako XML.

Příklad definování elementu v XAML:

```
<Button Content="Popisek"/>
```

Jednotlivým elementům pak lze definovat pomocí XML atributů jejich vlastnosti. Pomocí XAML lze definovat jednoduché chování objektů jako jsou například animace.

Při vytváření UI pomocí XAML se vytvoří dva soubory s koncovkami .xaml a .xaml.cs. Soubor .xaml slouží pro definici elementů a soubor .xaml.cs slouží k psaní funkcionality ve vybraném programovacím jazyce o kterém si povíme něco v následující podkapitole. Obsah souboru .xaml.cs je také v literatuře nazýván code behind. V tomto souboru je definované chování UI a díky provázanosti je možné přistupovat k elementům definovaným v .xaml souboru.

Dalším důležitým prvkem jazyka XAML je mechanismus zvaný binding. Pomocí tohoto mechanismu je XAML schopno načítat z její property DataContext konkrétní data, s kterými manipuluje. Binding umí provázat hodnoty logické vrstvy a vrstvy UI. Také zjišťuje, jestli nebyli hodnoty změněny a pomocí dalších mechanismů je schopen promítat hodnoty z jedné vrstvy do druhé bez povšimnutí programátora. Další výhodou mechanismu binding je možnost svázání elementu s příkazem. Na tuto problematiku se prakticky zaměřím v kapitole 5. Pro správnou funkcionalitu je nutné nastavit DataContex přímo v XAML nebo při inicializaci XAML komponent v code behind.

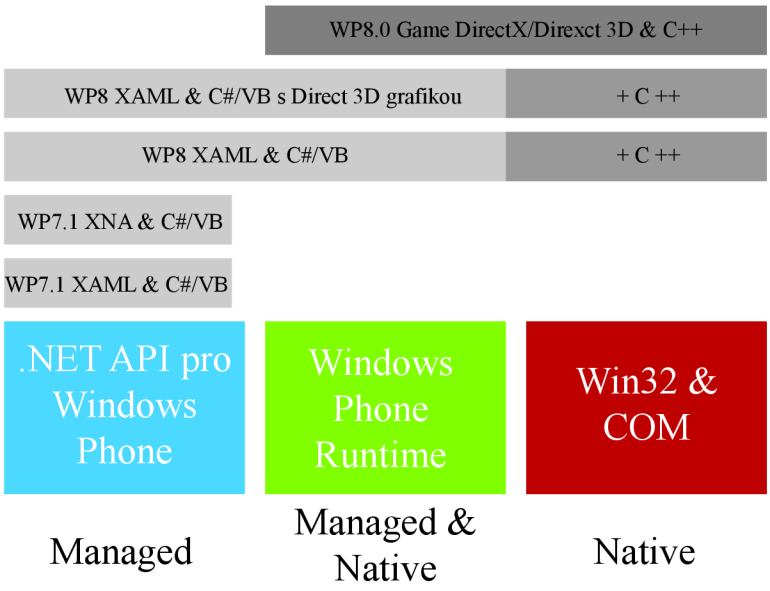
2.3.2 Způsoby programování

Platforma WP8 podporuje různé způsoby programování aplikace. Programátor si musí zvolit správný způsob podle toho, co požaduje od aplikace. U každého způsobu jsou programátorovi nabídnuty různá programovací rozhraní. Programování aplikace se dělí na základní tři způsoby dle vývoje:

- **Manage code** - Také v češtině organizovaný kód. Protože se tento název neujal budu používat managed. Jedná se o nejobvyklejší způsob vyvíjení aplikací. UI je definováno pomocí XAML a logiku je možné psát pomocí C# nebo Visual Basic .NET. Tento způsob umožňuje při vývoji užít .NET API pro WP a Windows Phone Runetime API. .NET API je primární rozhraní, které v sobě zahrnuje základní třídy, jako jsou například Microsoft.Phone.

- **Native code** - V češtivně jako nativní taktéž budu dále užívat native. Tento způsob je nejběžnější pro vývoj her. Při použití native code je možné vytvářet Direct3D aplikace. V tomto režimu není programátorovi nabídnuto vyvíjení UI v XAML. Dále režim podporuje Windows Runtime API a Win32/COM API. Tímto způsobem je aplikace vyvíjena v jazyce C++. Proto je možné vytvářet čistou aplikaci, v které je možné získat nejvyšší výkon mobilu.

- **Manage & Native** - Jako poslední možnost je využití obou způsobů současně. Programátor má možnost psát aplikační UI v XAML, logiku aplikace v C# a zároveň si napsat funkce pomocí C++ s přístupem k Direct3D. K těmto funkcím má přístup v logice aplikace. V podstatě lze využít při vyvíjení všechna možná programátorská rozhraní. Tento způsob je nejlepší, pokud programátor chce vyvíjet aplikaci, které mají klasické UI a potřebuje přístup ke grafickému adaptéru například pro úpravu obrázku.



Obrázek 2.1: Obrázek ukazuje hlavní tři možnosti vývoje aplikace pro WP platformu a dostupná API pro zvolenou možnost vývoje [4].

2.4 Životní cyklus, model aplikace a procesy na pozadí

Při vývoji WP si návrháři uvědomili, že uživatelé mobilních telefonů chtějí, aby zároveň jejich aplikace prováděli zadané úkony na pozadí jako je například nahrávání dat na internet, stahování aplikací a upozornění na novinky pomocí živých dlaždic. Paradoxně naproti tomu

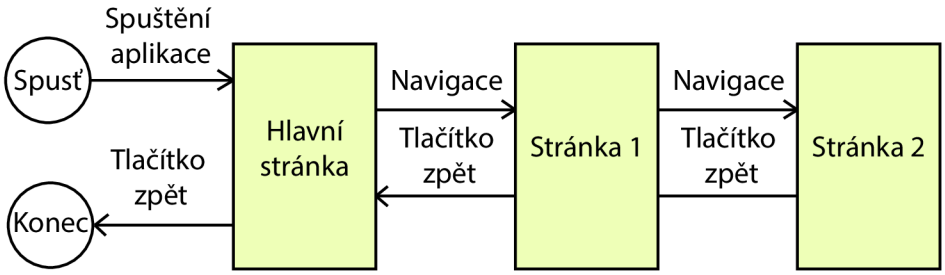
vyžadují, aby výdrž mobilu byla co nejdelší. Cíl vývojářů v tomto ohledu byl takový, že aplikace nebudou přímo spuštěné na pozadí. Proto vznikl životní cyklus aplikace.

Životní cyklus aplikace je jedním z důležitých faktorů ovlivňujících rychlost běhu mobilního zařízení. Životní cyklus WP8 se může nalézat ve stavech Running, Dormant a Tombstone. Vzhledem k tomu, že ani v české dostupné literatuře nejsou tyto stavy přeloženy, budu používat anglické názvy. Popis jednotlivých stavů životního cyklu aplikace:

- **Closed** - Aplikace je kompletně ukončena a všechny informace o stavu aplikace jsou zrušeny. Z running stavu přejde do stavu closed pomocí tlačítka zpět, nebo ukončení aplikace ve správci úloh.
- **Running** - Stav spuštěné aplikace. Aplikace má dostupné všechny zdroje.
- **Dormant** - Tento stav nastane pokud se uživatel naviguje z aplikace například stiskem tlačítka start nebo spuštěním jiné aplikace. Systém uvede aplikaci do režimu spánku a zastaví všechny vlákna aplikace. Nicméně aplikace zůstává stále v paměti. Při navigaci do stavu dormant je možné základní nastavení aplikace do objektu State.
- **Tombstoned** - Stav tombstone ovlivňuje operační systém. Pokud je nedostatek zdrojů systém vybere, které uspané aplikace budou uvedeny do tohoto stavu. Následně je aplikace odstraněna z paměti, ale systém uchová objekt State. Stav je téměř schodný se stavem closed, ale díky uložení objektu State je možný rychlejší start aplikace.

Oproti některé konkurenci, u které aplikace stále volně běží na pozadí se všemi zdroji, je tento přístup o něco složitější na programování, ale jeho výsledek se projeví na stabilitě a rychlosti systému.

Základní model aplikace je sestaven z jednotlivých stránek, mezi kterými se přechází. Stránku lze pochopit jako aktuální zobrazenou část na mobilním přístroji. Princip lze vidět na obrázku 2.2. Navigace mezi stránkami probíhá pomocí navigačního servisu. Předávání informací mezi stránkami probíhá pomocí parametrů, které si stránky mohou předávat při volání navigace mezi sebou pomocí proměnné QueryString.



Obrázek 2.2: Přepínání mezi stránkami od spuštění pro ukončení aplikace pomocí navigace v WP8 platformě [12].

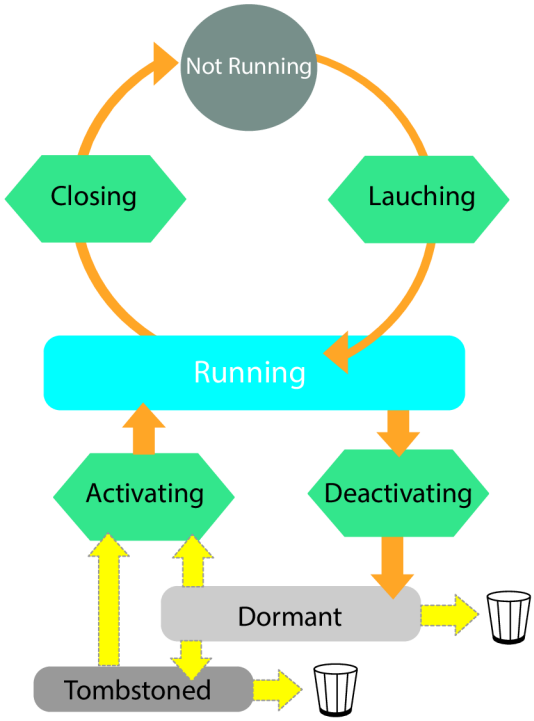
Mezi přechody z jednotlivých stavů aplikace a stránkami se vážou základní události, které jsou ve WP8 předprogramované a pro správný chod aplikace je nutné jejich obsluha.

Přehled potřebných událostí pro vývoj aplikace je v tabulce 2.2.

Protože aplikace nemůžou volně běžet na pozadí, má každý programátor přístup k servisu ovládajícímu služby na pozadí. Zároveň multitasking WP8 byl navržen tak, aby splňoval tyto potřeby a omezení. Díky tomu je WP8 schopný provádět operace na pozadí a zároveň vydržet obstojnou dobu na baterii. Podporované služby na pozadí WP8:

Tabulka 2.2: Události modelu aplikace WP8

Náze události	**Určeno pro**	**Popis**
Lauching	aplikaci	Událost se volá po spuštění aplikace
Deactivated	aplikaci	Událost se volá po navigaci z aplikace. Například spuštění jiné aplikace nebo stisk tlačítka start.
Activated	aplikaci	Událost se volá při přechodu z jakéhokoliv stavu do stavu Running.
Closing	aplikaci	Událost se volá po uzavření aplikace. Například stisk tlačítka back v hlavní stránkce.
OnNavigateFrom	jakoukoliv stránku	Událost se volá ve zdrojové stránce v případě navigace ze zdrojové stránky na jinou stránku v rámci aplikace.
OnNavigateTo	jakoukoliv stránku	Událost se volá v cílové stránce v případě navigace z jakékoliv stránky v rámci aplikace.



Obrázek 2.3: Životní cyklus aplikace. Obrázek demonstruje přecházení mezi jednotlivými stavy. Zelené události jsou vyvolávány převážně uživatelem a šedé stavy obstarává pouze jádro WP8 [7].

- **Servis přenosu** - Dovoluje aplikacím na pozadí komunikovat pomocí HTTP protokolu. Aplikace mohou přes tento servis začít stahovat nebo nahrávat soubor a přitom být uvedeny do stavu dormant.

- **Budík** - Servis pro nastavení budíku. Rozdíl oproti upomínkám je v omezenějších možnostech nastavení.
- **Servis zvuku na pozadí** - Servis využívaný pro přehrávání zvuku na pozadí i v okamžiku uspání aplikace.
- **Upomínky** - Servis dovoluje nastavení času zahájení, odkazu pomocí URI a další různá nastavení. Tento servis na pozadí je využit v aplikaci na plánování a monitorování času.

2.5 Návrhový vzor Model-View-ViewModel

Návrhový vzor Model-View-ViewModel je platformí doporučení při vývoji. Jeho zkartka je MVVM. Lze také přeložit jako Model-Pohled-PohledModel, ale toto spojení se v češtině nevyužívá. Proto dále budu využívat anglický ekvivalent. MVVM je určený pro návrh pomocí managed code. Díky tomuto vzoru se odděluje logika aplikace od dat a zároveň od UI. Pokud programátor nevyužije MVVM a designer aplikace změnil layout bylo pro programátora obtížné provést změny. MVVM je navržen tak, aby programátor nebyl vázaný na designovém návrhu aplikace. Proto při správném využití je kód čistý a je možné ho jednoduše udržovat a upravovat.

Pro správný návrh aplikace je nutné dobře pochopit binging. Binding v MVVM modelu spojuje View a ViewModel. Pro binding je nutné předem implementovat mechanismy, které umožňují odesílat notifikace o změně vlastnosti objektu.

Funkcionalita jednotlivých částí MVVM:

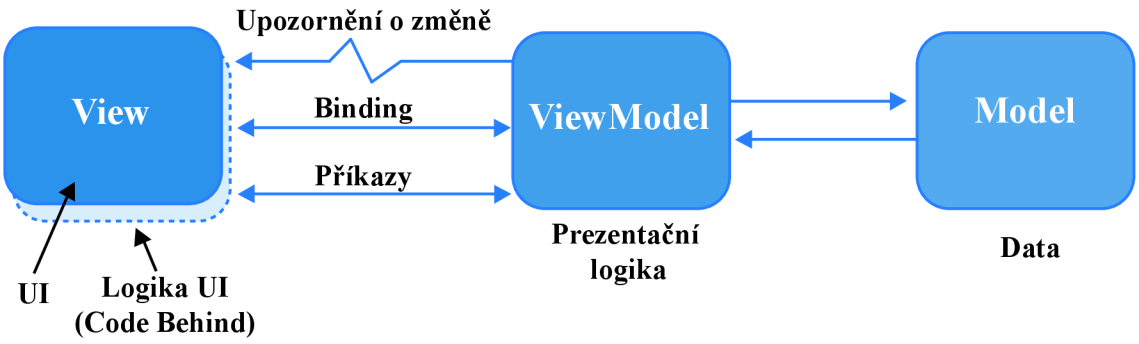
- **Model** - Model je část programu, která v sobě obsahuje základní struktury dat. Nejčastěji je model spojován s databází. Může ale také být načítán z webové stránky nebo z XML souboru.
- **View** - View slouží pro návrháře designu aplikace. V případě MVVM je view XAML kód. Návrhář má k dispozici při vývoji silnou podporu animací a pro návrh lze využít specializovaný program Blend, který je součástí vývojového SDK WP8.
- **View-Model** - Nejdůležitější část vzoru MVVM je View-Model. Tato část se stará o propojení mezí daty a designem. Má za úkol načítat a upravovat model aplikace a zároveň reagovat na události uživatelského rozhraní a vykonávat potřebné operace.

Provázanost vrstev a jejich funkcionalitu lze také vidět na obrázku 2.4.

2.6 NuGet manager

Existuje mnoho projektů, které vyvíjejí užitečné komponenty pro WP platformu. Ovšem jejich užívání v jiných programovacích jazycích je poněkud obtížné. Pokud se programátor rozhodne využít nějakou knihovnu od vývojářů třetích stran musí načíst všechny její reference a knihovny, které vyžaduje. Pokud je k dispozici nová verze knihovny programátor musí celý proces zahrnutí knihovny opakovat, proto vznikl NuGet manager. NuGet Package Manager je rozšíření pro Visual Studio, které se stará o vkládání knihoven třetích stran.

Pří jeho používání je práce s dodatečnými knihovnami jednoduchá. Uživatel si vybere dostupnou knihovnu, kterou chce mít zahrnutou ve svém projektu a NuGet mu vytvoří všechny



Obrázek 2.4: Ukázka návrhového vzoru MVVM, která zobrazuje jednotlivé mechanismy využívané pro propojení vrstev [5].

potřebné reference. Dále stáhne knihovnu do projektu se všemi ostatními knihovnami, které daná knihovna potřebuje. NuGet také hlídá verze používaných knihoven a v případě nové verze obstará její stažení.

2.7 Windows Phone Store

Protože mým cílem bylo výslednou aplikaci nahrát na oficiální obchod pro WP musel jsem prostudovat i možnosti nahrávání aplikací. WP Store po publikaci aplikace umožňuje zpřístupnit aplikaci do různých zemích. Z tohoto důvodu bych chtěl aplikaci vyvíjet tak, aby se dali přidávat postupně jazykové lokalizace.

Po publikaci musí aplikace projít schvalovacím procesem. Po schválení aplikace je provedená certifikace a zároveň přidaná a i ochrana od Microsoftu proti pirátství. Díky tomu se programátor nemusí zabývat ochranou programů a jeho program je automaticky po schválení chráněn právě dostupnou ochranou.

Aplikace může být publikována hned několika způsoby. Každý způsob má svoje výhody a omezení. Pokud při vývoji aplikace existuje skupina testerů, je zbytečné, aby každému posílali novou verzi. Místo toho lze využít WP Store, který nabízí možnost instalace aplikace testerům a OS automaticky hlídá její aktualizace. Protože pracuji samostatně žádnou z těchto možností jsem nevyužil kromě finální publikace.

Výpis možností publikace na WP Store:

- **Beta aplikace** - aplikace je přístupná pouze určité skupině lidí
- **Publikovaná a skrytá aplikace** - aplikace nelze najít při klasickém vyhledávání a počet stažení je omezen
- **Publikovaná a viditelná aplikace** - aplikace je přístupná pro všechny uživatele
- **Firemní publikace** - aplikace je viditelná v rámci konkrétní firmy

Pokud je aplikace umístěna na WP Store nabízí se programátorovi možnost procházet Crash Report. Díky Crash Reportu jsou veškeré pády aplikace zaznamenávány i když se jedná o testera nebo o uživatele. Pokud aplikace zhavaruje zaznamená OS pád aplikace a při první možnosti odešle data na WP Store. Díky tomu jsou pády zaznamenány i od běžných uživatelů, kteří se nemusí report pádu aplikace starat. Po přihlášení na WP Store má pak

programátor možnost zkontrolovat v grafické formě, jestli nenastal nějakému uživateli pád aplikace. Pokud ano je možné stáhnout podrobný výpis o pádech aplikace.

Ve výpisu jsou pády řazeny do kategorií. Je možné zjistit kolikrát nastala stejná výjimka u všech uživatelů. Dokonce je možné si zobrazit Stack trace výjimky totožný s výpisem ve Visual Studiu. Díky tomu je programátor schopný analyzovat chybu a aplikaci do další aktualizace opravit.

Kapitola 3

Úvod do problematiky plánování a monitorování času

Na organizaci času existuje mnoho programů, jak na Windows, Linux tak i Android či iOS. Každý program má své pro a proti. Existuje také mnoho metod, které se touto problematikou zabývají.

Například metoda GTD (zkratka pro Getting Things Done) popisovaná v knize [1] a [2]. Hlavní přínos metody GTD spočívá v tom, že se člověk naučí nezatěžovat mysl nad přemýšlením, co má udělat. Místo toho si navykne na externí systém, při jehož vytváření bude pouze posuzovat co je důležité, co méně důležité a kdy to má vykonat. Podle autora není lidský mozek uzpůsoben na pamatování spousty věcí, které má jedinec vykonat. Místo toho se metoda snaží jedince více vést k uspořádání si aktivit a umožňuje větší soustředěnost na jejich vykonávání. Tato metoda je velice propracovaná a pro někoho zbytečně moc komplikovaná. Vím o lidech, kteří přestali GTD po chvíli používat, protože to nevyhovovalo jejich stylu.

Dále existují velmi jednoduché metody, kde si uživatel sepisuje aktivity na další den a následně je odškrtává. U těchto metod je nepraktické to, že se uživatel nemůže zpětně podívat na svůj den a provést celkovou analýzu.

Uvědomil jsem si, že každý má svůj vlastní systém, který mu vyhovuje. Mým úkolem proto bylo navrhnout jednoduchou aplikaci, která by dělala kompromisy mezi těmito metodami.

3.1 Analýza již existujících řešení

Jak už jsem zde uvedl existuje velký výběr aplikací pro záznam a organizaci času. Mnoho aplikací je dobrých, ale použitelných nebo opravdu promyšlených je jen pár. Proto jsem nejdříve musel analyzovat existující řešení na všech různých platformách.

Vzhledem k tomu, že se jednalo o krátkodobé testování omezil jsem hledané aplikace jen na aplikace zdarma nebo na třicetidenní verze. Protože jsem chtěl analyzovat různé přístupy z různých platforem dal jsem si za úkol analyzovat aplikace alespoň tří různých platforem. Protože jsem vlastníkem Windows Phone, musel jsem řešit jak otestovat aplikace na jiných platformách. Podařilo se mi zapůjčit zařízení s OS Android po dobu testování. Bohužel jsem nesehnal iOS na delší dobu propůjčení. Proto jsem si nejdříve vyhledal a prostudoval recenze na různé aplikace platformy iOS a následně provedl krátkou analýzu na přístroji.

Nejdříve před samotnou analýzou jsem si vytyčil požadavky od aplikace. Zaměřil jsem

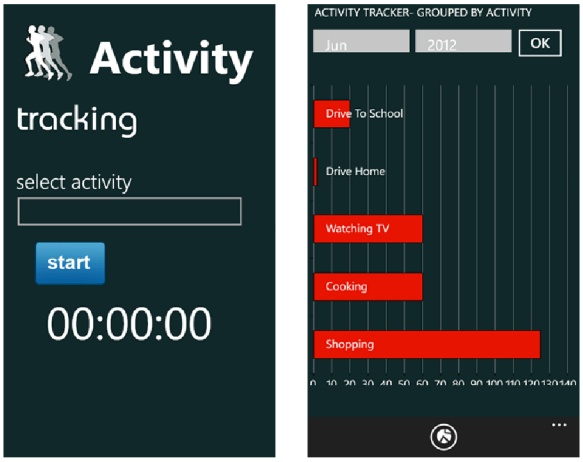
se na to co aplikace nesmí a naopak co by měla dělat. Díky těmto požadavkům jsem se mohl zaměřit na konkrétní prvky.

- aplikace musí být intuitivní (pozorovat jak rychle ji začnu ovládat)
- dodržování stylu WP (zaměřit se na prvky vyhovující stylu WP)
- na jedné obrazovce nesmí být mnoho ovládacích prvků
- ovládací prvky musí být pohodlné i pro uživatelé s většími prsty
- méně je někdy více (zaměřit se na odezvu aplikace)
- dobrá orientace (zaměřit se na způsob vyhledávání u více položek)

Při analýze jsem narazil na aplikace, které mě oslovili a naopak u kterých jsem nebyl moc nadšený. Zde bych chtě uvést některé z nich.

3.1.1 Activiy Tracker

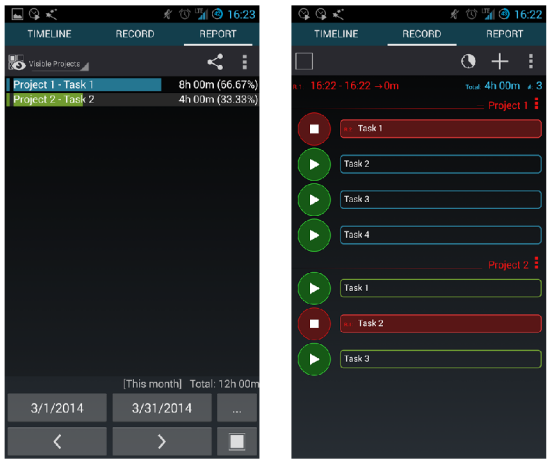
Activity Tracker, který můžete vidět na obrázku 3.1 je dostupný pro platformu WP. Jeho zpracování působí velmi minimalisticky. Líbilo se mi jeho řešení grafického vyobrazení statistik. Bohužel Porušování typografických zásad jako je například náhodné odsazování prvků, kazilo jeho celkový dojem.



Obrázek 3.1: ScreenShot z aplikace Activity Tracker

3.1.2 Gleo Time Tracker

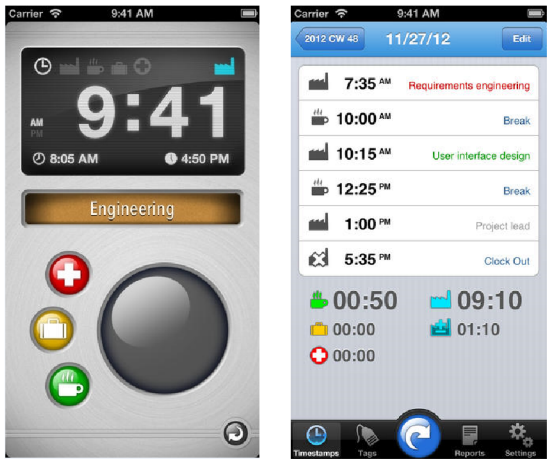
Gleo Time Tracker, který můžete vidět na obrázku 3.2 je určen pro platforu android. Úvodní obrazovka působila také minimalisticky, avšak nabízela mnoho možností. Bylo možné přidávat nové projekty nebo aktivity hned z hlavní obrazovky a byli zde zobrazeny i základní časy. Ve statistikách bylo možné vidět graficky čas strávený na jednotlivých účelech a zároveň bylo možné hned vedle zjistit přesný čas a procentuální podíl času strávený na projektu v rámci všech projektů. Na aplikaci se mi zdála zvláštní možnost spuštění více aktivit zároveň, jak lze vidět na obrázku 3.2. Dále bych neuváděl u statistik procentuální čas strávený na účelu v rámci všech účelů.



Obrázek 3.2: ScreenShot z aplikace Gleo Time Tracker

3.1.3 Time Stamps

Jako poslední z aplikací, které bych tu chtěl zmínit je Time Stamp. Aplikace je dostupná na zařízení iOS. Zdálo se mi, že autor použil programovací doporučení a aplikace zapadá do prostředí iOS. I když je aplikace trochu jinak zaměřená než aplikace pro plánování a monitorování času, působila aplikace na mě přehledným dojmem. Na aplikaci se mi líbilo obrázkové menu. Uživatele nejdříve vede text pod obrázky a pokud si zvykne používá menu intuitivně a rychle bez čtení textu.



Obrázek 3.3: ScreenShot z aplikace Time Stamps

Kapitola 4

Návrh aplikace aplikace Chytrý Organizátor

4.1 Případy užití aplikace

Po analýze existujících řešení jsem začal tvořit Use Case diagram, jehož celkový výsledek je vidět na obrázku 4.1 a obrázku 4.2. Pro rychlou orientaci v programu a intuitivnost aplikace bylo důležité správně rozvrhnout rozmístění ovládacích prvků a layout stránek, které se odvíjí od tohoto diagramu.

V první řadě jsem se zaměřil na to, co uživatel od aplikace očekává a jaké úkony by chtěl nejčastěji vykonat. Pro přehlednost a jednoduchost návrhu jsem provedl dekompozici problému na čtyři hlavní části:

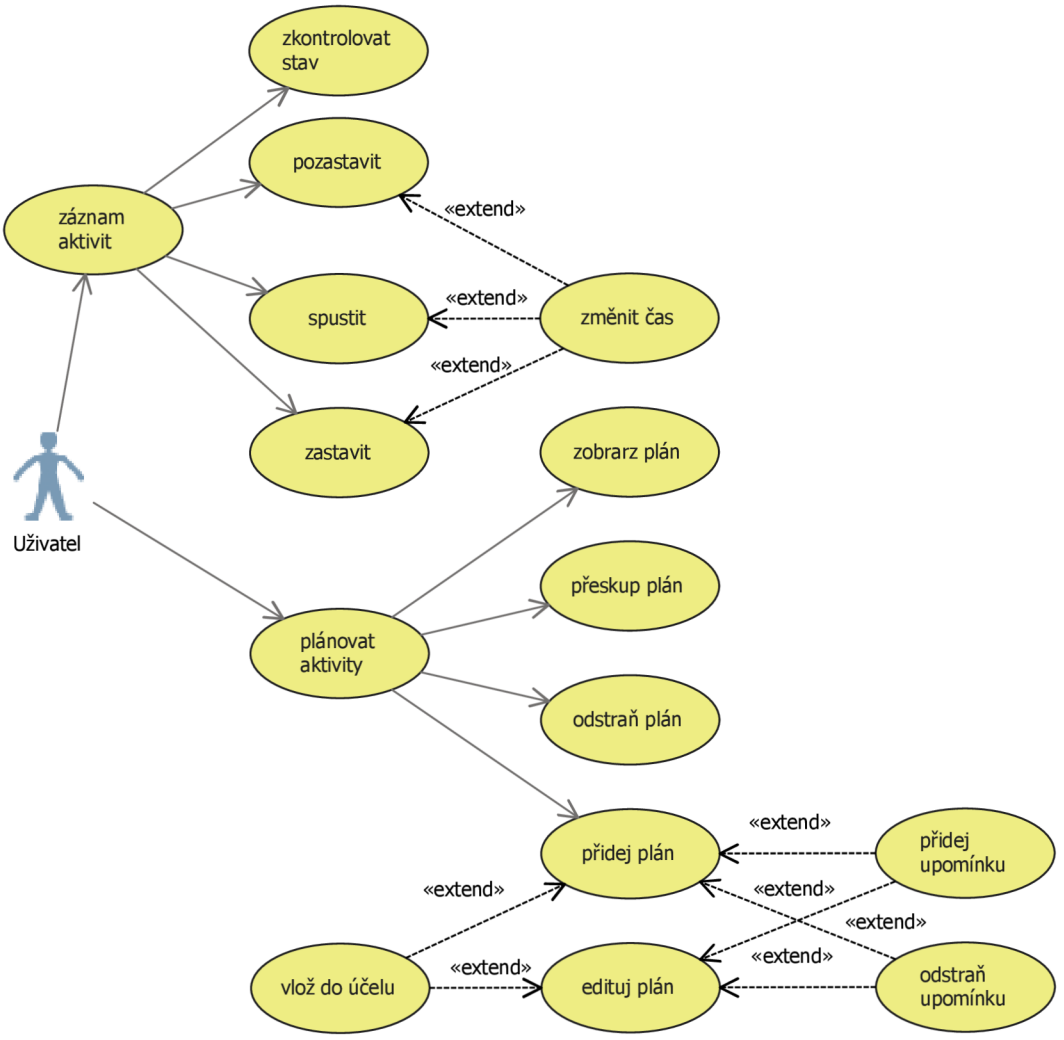
- záznam aktivity
- plánování aktivit
- kontrolování statistik
- zobrazení účelů

Podstatné bylo vyhodnotit, která případy užití mají největší počet výskytů a naopak. Například prvek ohodnotit aplikaci, který je zřídka kdy využit by se neměl vyskytovat na jedné z nejvíce užívaných stránek.

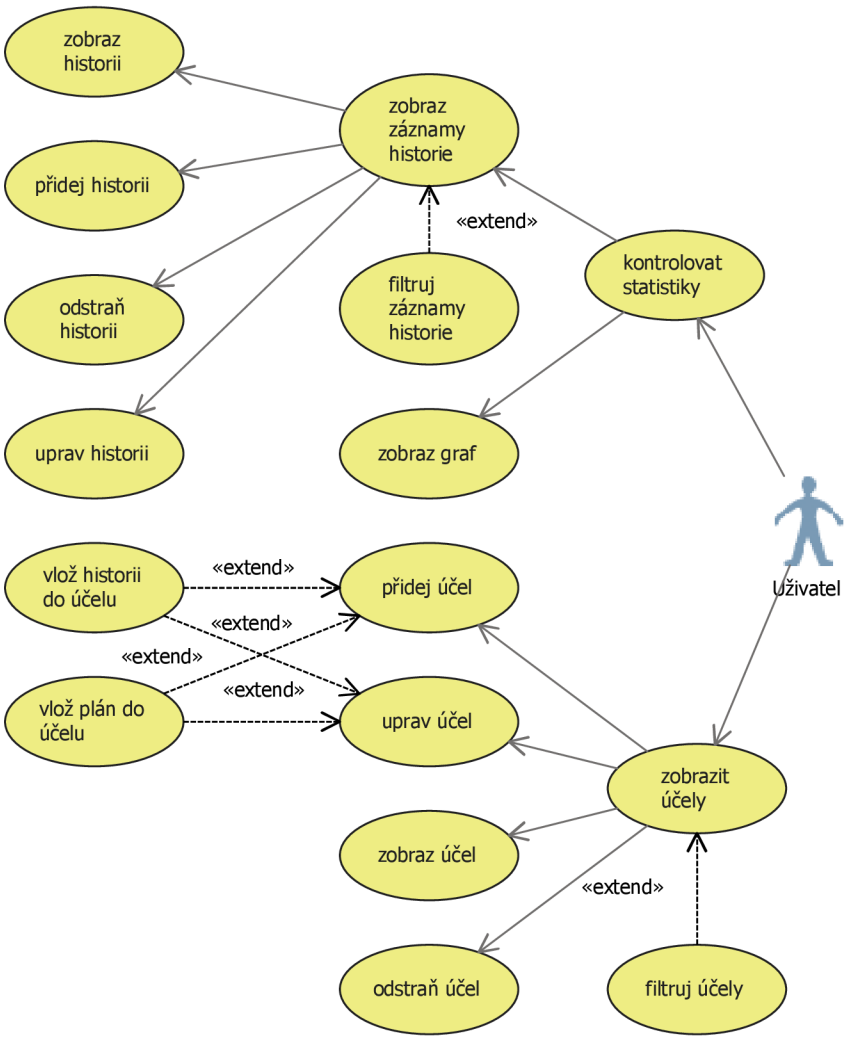
Jako další krok bylo rozebrat jednotlivé problémy a určit další úkony, které se budou pod těmito problémy vyskytovat. Při přemýšlení nad užitím aplikace jsem se snažil seřadit tyto čtyři podproblémy podle výskytu. Zde je budu uvádět od nejdůležitějšího.

Záznam aktivit je činnost, kde uživatel chce zkontrolovat, která aktivita právě probíhá. Dále chce spustit, zastavit nebo pozastavit aktivitu. V mnoho případech bude chtít uživatel pouze zkontrolovat, či spustil správnou aktivitu. Proto po načtení úvodní obrazovky bude první úvodní stránka zobrazovat právě tyto informace.

Plánování aktivity bude probíhat nejspíše jednou za den nebo za týden podle uživatelových potřeb. Proto bude organizace a plánování další nejčastější případ, který si uživatel zvolí. Protože aplikace není primárně určena jako kalendář, není zde možné zadávat časy k jednotlivým plánům. Nicméně je vhodné, aby si uživatel mohl zvolit jednoduchou upomínku, která mu připomene jeho důležité aktivity v plánu. Po úvaze jsem zahrnul možnost nastavení upomínky při úpravě nebo vytváření plánu, kterou uživatel může nebo nemusí



Obrázek 4.1: Use Case diagram aplikace část 1. Diagram vyobrazuje dva nejčastější úkony uživatele, kterými jsou záznam aktivit a plánování aktivit. Dále je možné vidět, jaké další úkony lze provádět pod těmito hlavními úkony.



Obrázek 4.2: Use Case diagram aplikace část 2. Diagram vyobrazuje další dva nejčastější úkony uživatele, kterými jsou kontrola statistik s zobrazení účelů. Dále je možné vidět, jaké další úkony lze provádět pod těmito hlavními úkony.

využít. Dále je důležité, aby uživatel mohl po naplánování aktivity zadat její účel. Uživateli musí být nabídnuto i přeskládat plány podle jeho priorit.

Jednou za pár cyklů plánování a zaznamenávání aktivit se uživatel rozhodne **prohlédnout statistiky**. Pro rychlý přehled bude sloužit jednoduchý textový rozpis historií, ve které si může uživatel v rychlosti zjistit informace o uběhlých aktivitách. Rozpis historií bude možné filtrovat podle data. Dále zde budou graficky vyjádřené statistiky, které bude uživatel pravděpodobně využívat méně než rozpis historie. V grafickém zobrazení by statistiky měli umět vypočítat celkový čas strávený na různých účelech. Případ musí také zahrnovat zobrazení detailů, vložení, odstranění a úpravu historie.

Posledních z případů, které uživatel bude využívat je **zobrazení účelů**. Protože si uživatel vytvoří účely na začátku a poté už je bude upravovat jednou za čas, zvolil jsem organizaci účelů dostupnou přes menu aplikace. Protože účelů může být mnoho musí aplikace podporovat rychlou navigaci nebo nějaký filtr účelů. Dále se zde musí nacházet přidání, odstranění, úprava a zobrazení detailů účelu. Jednou z podstatných věcí které jsem zahrnul do Use Case diagramu bylo možnost přidání plánu nebo historie do plánu.

4.2 Obecné problémy při návrhu

Při analýze existujících aplikací jsem narazil na problém, který se zdá být jako nepodstatný, ale při jeho vyřešení usnadní uživatelům práci navíc. Nastával v situaci, když uživatel začal provádět aktivitu a zapomněl odkliknout ikonku start aktivity. Když si to po určité době uvědomí, musí počkat na konec aktivity, aby věděl jaké časy přidat do historie. V případě, že pak uživatel zapomene úplně je pro něj stresující, že musí zpětně zadávat více aktivit.

Po této úvaze jsem se zamyslel nad tím, jak uživateli ovládání aplikace v tomto směru ulehčit. Uživatel nemusí být pak v podvědomém stresu, že musí někde potvrdit tlačítko. Uživatel musí aplikaci pochopit jako doplněk místo podvědomí, že užívání aplikace je činnost, kterou provádí navíc. Provedl jsem pár návrhů a nejvíce se mi líbil způsob, kde uživatel odklikne checkBox s popisem rozdílný čas a následně mu bude nabídnut výběr rozdílu času. Poté jsem objevil další problém. Musel jsem vyřešit, jak uživateli intuitivně nabídnout změnu času.

Po chvíli přemýšlení jsem si uvědomil, že uživatel může znát přesný čas, kdy aktivitu ukončil nebo vědět dobu od ukončení. Pokud by uživatel zrovna chtěl zadat čas jedním způsobem a aplikace mu nabídla druhý způsob, nutilo by ho to přepočítávat čas a kontrolovat, jaký je aktuální čas. Jedním ze způsobů, který mě napadl, bylo nabídnou uživateli více tlačítek, ale to by vedlo k nepřehlednosti aplikace.

Konečný způsob návrhu byl takový, že nabídnu uživateli pouze jediný checkBox, který není implicitně zaškrtnutý. Po zaškrtnutí a použití zadání různého času se checkBox musí automaticky odškrtnout. Pokud uživatel zatrhne tento checkBox musí se mu pod ním objevit políčko s výběrem metody. Mety zadávání budou čas nebo uplynutá doba. Aby uživatel pochopil lépe funkcionalitu musí být vedle checkBoxu informační tlačítko s otazníkem a po kliknutí se zobrazí informace ohledně funkcionality těchto prvků.

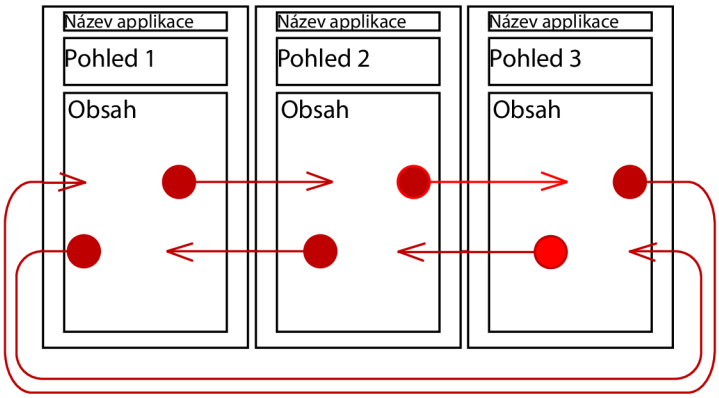
Podobný problém nastává v případě, když se vyskytne neplánovaná aktivita. Uživatel najednou potřebuje odstartovat aktivitu, kterou nemá v plánu. Než by vytvořil nový plán a spustil ho musel by provést mnoho úkonů a to by v naléhavých případech uživatele odradilo. I s takovým problémem by si měl aplikace poradit. Na úvodní obrazovce by mělo být tlačítko pro rychlou aktivitu. Po spuštění aktivity se sama naplánuje a po ukončení aktivity bude uživatel vyzván, aby zadal její jméno. Díky tomu je uživatel během jednoho kliknutí schopný spustit neplánovanou aktivitu a informace zadat v klidu později.

4.3 Návrh layoutu aplikace

Nejdříve jsem si nakreslil více variant aplikace. Abych dostal inovativní přístup k návrhu aplikace bylo mým cílem navrhnout minimálně 6 variant. Při návrhu prvních variant jsem se držel podvědomě prvotního návrhu. Poslední varianty už začali být odlišné a zajímavé. Jednotlivé varianty jsem ukazoval cílovým uživatelům a podle jejich reakce jsem vybíral užitečné prvky. Po základní představě jsem se pustil do konečného návrhu aplikace.

Prvotní myšlenka designu aplikace bylo navrhnout úvodní obrazovku, kterou je možné tahem zprava doleva nebo zleva doprava přepínat mezi pohledy. Ideální prvek, kterým jsem se při návrhu inspiroval se jmenuje pivot a je vidět na obrázku 4.3.

Kvůli přehlednosti programu jsem zvolil maximálně tři takovéto pohledy v úvodní stránce. Některé analyzované aplikace používali tyto pohledy na různá nastavení nebo na pohledy s odkazy na hodnocení. Tohoto přístupu jsem se chtěl vyvarovat, protože uživatel nastaví nebo ohodnotí aplikaci pouze jednou a již tyto prvky bude využívat zřídka kdy. Dále jsem se chtěl vyvarovat přepínání mezi více jak třemi pohledy. Průměrně inteligentní uživatel při mnoho pohledech na stránce sice pochopí na jakém pohledu je a na jaký se může přepnou, ale intuitivně ho bude ovládání brzdit a bude nucen více přemýšlet.



Obrázek 4.3: Inspirace designu úvodní obrazovky podle prvku pivot navrženého firmou Microsoft.

Návrh hlavní obrazovky

V prvním pohledu by se měli nacházet prvky pro záznam aktivit. Rozhodl jsem se, že se zde bude dále nacházet zvýrazněný text, který bude vypisovat jestli probíhá či neprobíhá aktivita. Dále se bude úvodní pohled hlavní obrazovky přepínat mezi třemi základními stavy a to spuštěná aktivita, zastavená aktivita a neprobíhá aktivita. Výsledný návrh je možno vidět na obrázku 4.4.

Ve stavu kdy neprobíhá žádná aktivita je uživateli nabídnuto spustit rychlou aktivitu nebo zaškrtnout výše zmiňovanou volbu s rozdílným časem. Další úvaha byla, aby uživatel nemusel složitě přecházet na plán a vybírat aktivity z plánu. Při analýze existujících řešení jsem narazil na řešení, kde uživatel může začít psát do pole a aplikace mu začne nabízet schodné aktivity se zadaným řetězcem. Myšlenka byla zajímavá, ale vadilo mi, že je uživatel nucen přemýšlet jak si pojmenoval jeho naplánované aktivity nebo se musí navigovat do

(a) Neprobíhá aktivita

(b) Spuštěná aktivita

(c) Zastavená aktivita

Obrázek 4.4: Ukázka třech základních stavů úvodní stránky a jejich rozmístění prvků.

plánování a vybrat si aktivitu tam. Nechal jsem se vést základní myšlenkou. Pokud půjde všechno podle plánu uživatel zvolí jednu z prvních naplánovaných aktivit nebo v opačném případě bude chtít spustit rychlou aktivitu. Vylepšil jsem předchozí myšlenku a místo našeptávače uživateli nabídnu seznam právě tří prvních naplánovaných aktivit, kde u každé bude tlačítko spustit.

Další pohled úvodní obrazovky bude obsahovat seřazený rozpis podle priorit naplánovaných aktivit. Pokud chce uživatel zkontrolovat pouze plán, mohl by si omylem kliknout na přesun naplánované aktivity nebo její vymazání. Proto aplikace nabízí dva režimy editace a prohlížení. Tato změna se bude provádět pomocí menu, kde budou dvě rychlá obrázková tlačítka pro přidání nové aktivity a přepínání mezi režimy. Implicitně po zapnutí aplikace je nastaven režim prohlížení.

V režimu prohlížení bude uživateli nabídnuto pouze jedno tlačítko a to na spuštění naplánované aktivity. Tím se omezí obrazovka s mnoho tlačítky, které je možné omylem stisknout.

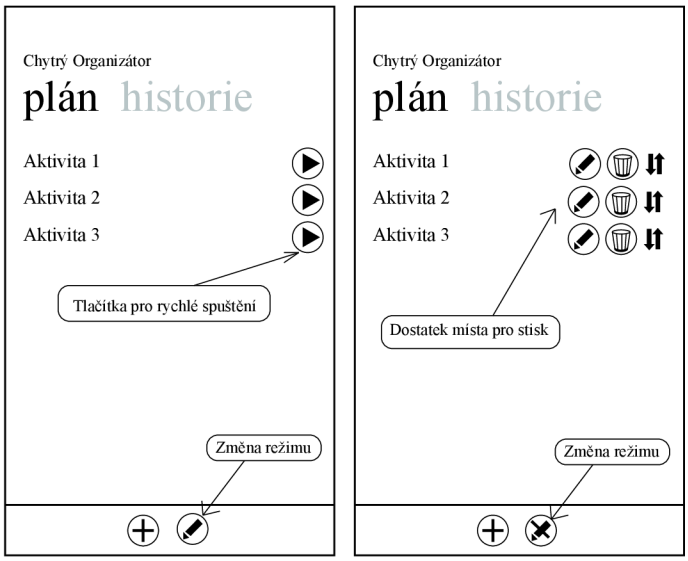
V režimu editace je možné naplánovanou aktivity přesunou, editovat, odstranit nebo po kliknutí na text zobrazit její detaily.

Výsledný návrh je vidět na obrázku 4.5.

Jako poslední prvek na druhém pohledu hlavní stránky je vyhledávání. Nahoře obsahu bude tlačítko s lupou. Po jeho stisku vyjede textBox, do kterého se zadá řetězec. Po zadání řetězce se provede filtr, který prohledá názvy naplánované aktivity a pokud najde schodu zvýrazní ji a aktivity bez schody odebere ze seznamu.

Poslední pohled hlavní obrazovky bude rozpis historií. Protože historií může být mnoho a navigace na tuto stránku by velice zpomalila rychlost aplikace bude nahoře nabídnuty prvky s možností zadávání počátečního a koncového data filtrování historie. Po spuštění aplikace je implicitně nastaven rozsah týden před aktuálním dnem a aktuální den.

U rychlé rekapitulace historie je důležité vidět název aktivity, před jakou dobou byla provedena a zkrácený čas trvání (hodiny, minuty). Při návrhu zobrazení náhledů historií jsem



(a) Režim prohlížení (b) Režim editace

Obrázek 4.5: Výsledný návrh plánu, ve kterém je vidět rozmístění jednotlivých ovládacích prvků plánu v jednotlivých režimech.

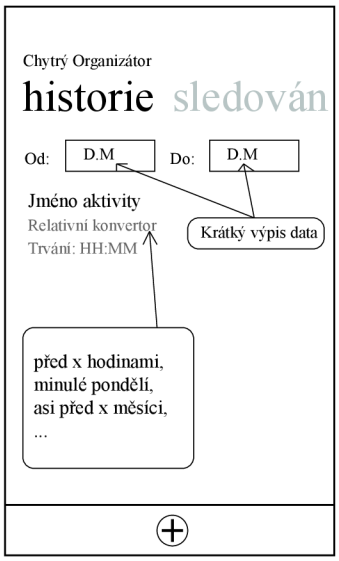
se inspiroval prvky sociálních sítí a ostatních promyšlených aplikací. Místo data započetí aktivity zde bude zobrazen relativní konvertor času. Tzn. místo konkrétního data zde bude napsáno asi před hodinou nebo minulý pátek atd. a uživatel nebude nucen přemýšlet, jaký je datum. Pokud vezmu v potaz, že historie se budou editovat zřídka kdy je zbytečné uživatele zatěžovat tlačítky smazání nebo editace historie vedle záznamu. Místo toho po delším stisku položky je uživateli nabídnuto kontextové menu, na které jsou uživatelé zvyklí z rozhraní WP. Kontextové menu bude obsahovat smazání nebo editaci. Dále po krátkém stisku se zobrazí stránka s podrobnými údaji o historii. Jako posledním prvkem je obrázkové tlačítko v menu pro vytvoření nové historie. Výsledný návrh je možné vidět na obrázku 4.6.

Nyní se dostáváme k návrhu stránek, které jsou mimo hlavní stránku. Vznikla zde otázka, jak nejlépe a odkud se na tyto stránky odkazovat.

Hlavní stránka na každém svém pohledu obsahuje menu a v něm rychlá obrázková tlačítka, která jsou definovaná pro každý pohled zvlášť. Menu bude možné otevřít a zde se budou nacházet textové položky menu. Tyto položky budou pro všechny pohledy stejné. Budou se zde nacházet odkazy na stránky organizovaní účelů, statistiky a informace o aplikaci.

Po stisku odkazu o aplikaci, který se nachází v menu, se objeví informace o aplikaci. Zde se bude nacházet název aplikace, její verze, krátký popisek a odkazy na ohodnocení aplikace nebo možnost odeslání emailu s připomínkami nebo návrhy.

Další odkazovaná stránka v menu je organizace účelů. Po spuštění této stránky jsou nabídnuty všechny účely. Protože účelů může být více jsou voleny dvě varianty zobrazení, které uživatel nemůže ovlivnit. Při zobrazení do deseti účelů je zvolen klasický list seřazený podle abecedy. Po deseti účelech a výše jsou účely seskupovány do skupin podle písmen abecedy. V tomto zobrazení je před skupinou zobrazeno písmeno po jehož stisknutí se dá rychle navigovat na jinou skupinu. V této nabídce se nepoužívané skupiny nemohou dát stisknout a pro přehlednost aplikace musí být jinak barevně odlišeny. Jednotlivé položky účelu mají k dispozici dvě tlačítka editace nebo smazání.



Obrázek 4.6: Výsledný návrh historie, ve kterém je vidět relativní konvertor času a rozvržení prvku umožňující filtrování historie.

Návrh obrazovky statistik

U návrhu statistiky jsem začal od stránky zobrazení grafu. Mnoho již existujících aplikací zobrazovalo grafy jak autory napadlo. Ale pro každodenní používání aplikace ztrácely význam. Například koláčový graf. Pokud se uživateli podařilo vykonat spoustu aktivit byl graf poměrně nepřehledný. Další nevýhodou je, že ukazuje jednotlivé časy aktivit v poměru mezi sebou. Na co je uživateli, že graficky může vidět například dvě aktivity, které mu zabrali přibližně stejně času když se na časy jednotlivých aktivit musí stejně koukat do textového výstupu legendy? Přemýšlel jsem proto nad tím, jak do grafu zobrazit jednotlivé časy. Moje řešení bylo takové, že jsem použil sloupcový graf orientovaný horizontálně. Na ose x je vynesená mřížka pro čas a svisle jsou zobrazeny aktivity s časem trvání. Před vykreslením grafu jsou aktivity seřazené od nejdéle trvající po nejméně trvající aktivitu. Tento typ grafu lze dobře aplikovat na mobilní platformy, protože jeho orientace je řešena převážně směrem nahoru a dolu. Graf proto pojme velké množství položek. Poslední Prvek statistik bylo zobrazení celkového času historií. V případě mnoho aktivit v účelu musí být čas schopný zobrazit i několik desítek dní. Celkový čas se pak zobrazí pod hlavním titulkem grafu.

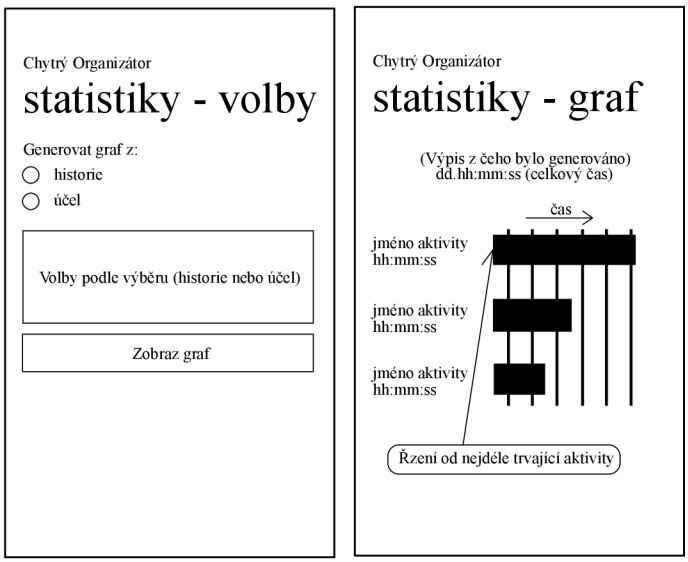
Nyní se dostáváme ke složitější části statistik. Jak uživateli nabídnou zadávání hodnot, z kterých chce graf generovat. V první řadě je požadavek shlukování více aktivit se stejným jménem do sebe.

Za předpokladu, že uživatel zvolí zobrazení celého dne, kde jednou vykonával aktivitu x pro účel y a následně samou x pro účel z. Lze předpokládat, že chce zobrazit čas aktivity x jako jediný. Pokud by chtěl vidět práci pro jeden účel, nezvolil by si zobrazení z historie, ale z účelu.

Z toho vyplývá základní návrh layoutu pro možnosti statistik. Konečné řešení jsem navrhl takové, že po stisku statistik z menu hlavní stránky je uživateli nabídnuta stránka s nastavením generování grafu. Tato stránka musí po vygenerování dat zobrazit graf v jiné stránce. Po odchodu ze stránky s grafem nesmí být již stránky s nastavením zobrazena.

Nyní už zbývá jen rozmyslet samotnou stránku s nastavením grafu. Pomocí rolovacího výběru si uživatel zvolí jestli chce generovat graf z účelu nebo z historie. Po zvolení mu

je nabídnut výběr z účelů nebo možnost zvolení rozmezí pro historie. Poslední chybějící prvek je tlačítko na zobrazení grafu. Protože bude docházet k hledání a shlukování záznamů může být operace časově náročná. Z hlediska návrhu to znamená, že musí být zobrazen progressBar v době generování grafu, aby uživatel neměl pocit zamrznutí aplikace. Výsledek možností statistik je vidět na obrázku 4.7.



(a) Volby statistik. (b) Výsledný graf.

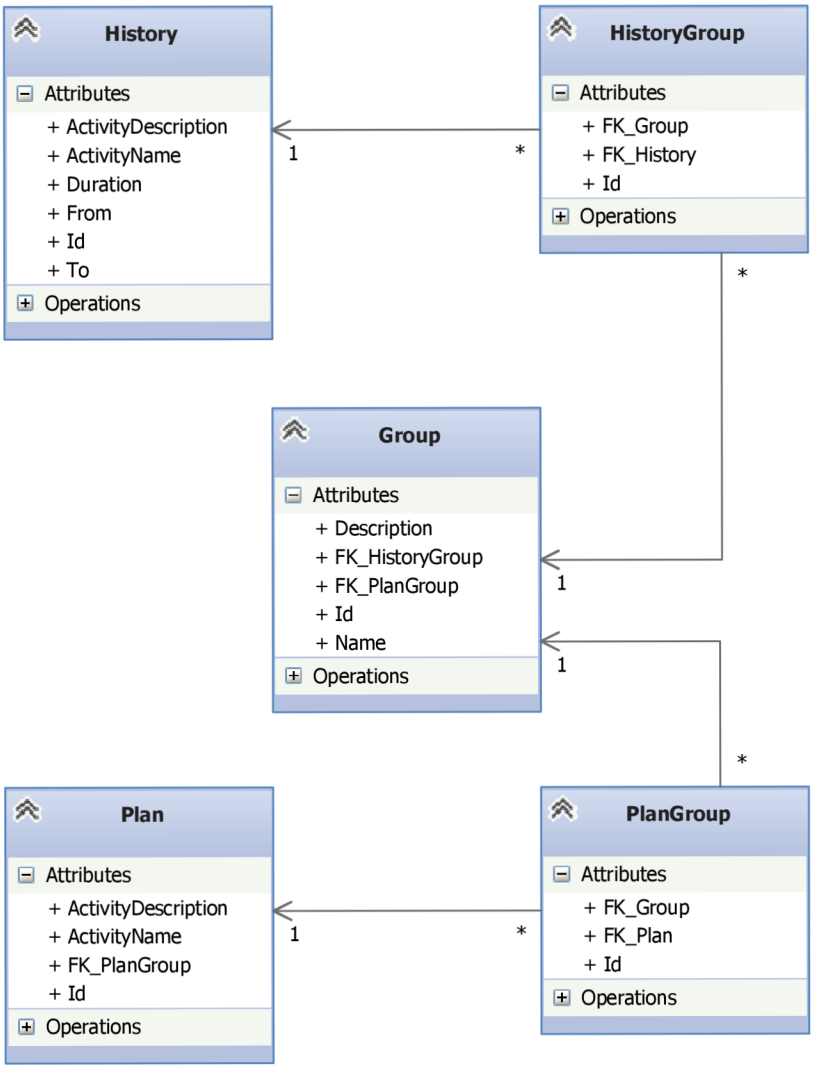
Obrázek 4.7: Ukázka návrhu statistik, která zobrazuje stránku pro možnost zadání parametrů grafu a výslednou stránku grafického vyjádření.

Návrh stránek editace, přidávání a úprav

Poslední část návrhu spočívala v navržení layoutu stránek editace, vytváření a zobrazení detailů. Pro tyto účel jsem nejdříve musel navrhnou obecnou strukturu dat a jejich propojení, které můžeme vidět v obrázku 4.8 V první řadě jsem si vyčlenil položky, které budou stálé. Těmi jsou plán, účel a historie. Nejdříve jsem chtěl zde i zařadit aktivitu, ale to se ukázalo být nevhodné. Místo toho jsem navrhl, že v historii a plánu bude jméno aktivity a její popis. Díky tomu bude uživatel schopen řadit aktivitu stejného názvu pokaždé do jiného účelu. Účely a plány budou ve vztahu M:N stejně jako účely a historie.

Pro úplnost bych chtěl jen podotknout změnu pojmenování účelu. Původně byl účel pojmenován skupina. Po provedení analýzy vyhovujících slov před publikací se slovo skupina zdálo nevystihující správný význam. Proto došlo ke změně. Díky návrhu, který bude popsán v sekci 5 jsem provedl přepsání pouze v textu aplikace, ale programově jsou v aplikaci účely pojmenovány group.

Ve stránce zobrazení detailů se nabízí možnost smazání položky nebo její editace. Dále je tato stránka tvořena více pohledy, kde první obsahuje detaily a další pohledy seznam přiřazených položek. Například účel bude obsahovat v jeho úvodním pohledu jméno a jeho popis. V dalším pohledu seznam plánů patřících do účelu a v posledním pohledu seznam historií patřících do účelu. Editaci položek bude obdobná zobrazení detailů a musí obsahovat v menu tlačítko zrušení, po kterém se neuloží provedené změny a tlačítko uložení, které změny promítne do databáze.



Obrázek 4.8: Návrh struktury dat. Hlavními datovými strukturami je historie, účel(původní pojmenování skupina) a plán. Protože historie i plány jsou ve vztahu M:N jsou potřeba struktury plán-účel a plán-historie pro jejich propojení.

Kapitola 5

Implementace aplikace Chytrý Organizátor

V této části se budu věnovat implementaci aplikace Chytrý organizátor. Nejdříve se zaměřím na rozvržení projektu a přizpůsobení návrhovému vzoru MVVM. Dále se budeme zabývat navržení databáze, posílání objektů mezi stránkami, jazykové lokaci a na závěr bude implementace kódu XML.

5.1 Knihovny třetích stran

Při vývoji jsem se rozhodl použít dostupné knihovny, které jsou vyzkoušené a optimalizované. Díky tomu jsem se mohl víc zaměřit na vývoj uživatelského rozhraní a funkcionalitu aplikace.

Pro vývoj aplikace jsem využil:

- **App Bar Utils** - Knihovna díky které je možné definovat menu pro různé pohledy stránky přímo v XAML.
- **Data Visualization Controls** - Knihovna pro zobrazení grafů definovaných v XAML.
- **Reorder List Box** - List box, u kterého je možné povolit přesouvání položek.
- **Windows Phone Toolkit** - Nejpoužívanější knihovna pro vývoj mobilních aplikací platformy WP. Knihovna zpřístupně rozsáhlou škálu komponent, které byli využity při vývoji WP platformy.
- **Coding4Fun Toolkit** - Knihovna rozšiřující funkcionalitu Windows Phone Toolkit. Díky této knihovně lze zpřístupnit další zajímavé komponenty.

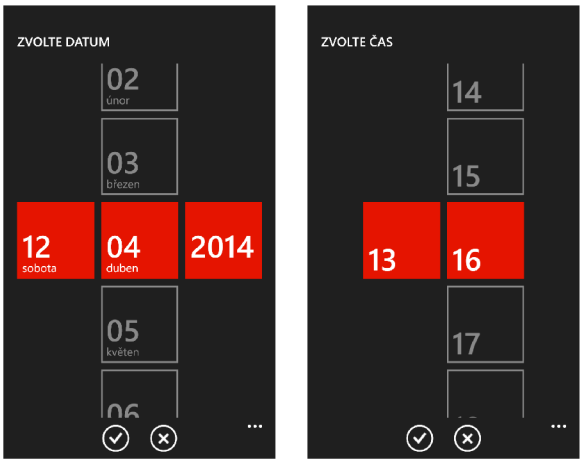
Výše uvedené knihovny jsou dostupné přes NuGet manager. Protože NuGet je založen na repozitářích projektů, je možné projekty stáhnout. Po stažení má programátor k dispozici také ukázkový program, který objasní styl a využití prvků dané knihovny.

Díky této možnosti lze pozměnit danou knihovnu a přizpůsobit jí aplikaci. Při tomto přístupu již není možné spravovat knihovnu pomocí NuGet Package Manager.

Knihovna Windows Phone Toolkit nabízí Date Picker a Time Pickers, které je možné vidět na obrázku 5.1. Prvky dovolují nastavit datum nebo čas. Po kliknutí se objeví pás

s kterým je možno pohybovat a tím lze vybrat hodnotu. Pokud uživatel chce vybrat například 30 minut a počáteční stav je 0 minut musí pásem pohybovat o 30 položek a to se mi zdálo jako zdlouhavé řešení.

Proto jsem prostudoval knihovnu Windows Phone Toolkit a poupravil chování komponent. Po úpravě pás v případě rychlejšího pohybu přejde na celé desítky čísel. Pokud chce uživatel zadat 30 minut a výchozí hodnota je 0 stačí tři krát rychleji posunout s pásem místo přejetí 30 položek. Pokud uživatel posune pásem normální rychlostí může provést nastavení času na minuty. Díky této změně by měl být uživatel schopný nastavit čas rychleji a aplikace by měla být pro něj zase o něco přívětivější.



Obrázek 5.1: Ukázka prvku Data Picker a Time Picker.

5.2 Rozvržení struktury projektu a MVVM

Nejdříve jsem si v projektu vytvořil strukturu složek, kterou budu používat pro přehlednost aplikace. Strukturu složek jsem přizpůsobil návrhovému vzoru MVVM. Struktura obsahuje Složky:

- **Icons** - obsahuje ikony pro obrázková tlačítka, která napomáhají orientaci v programu
- **Models** - obsahuje tabulky databáze a databázový kontext
- **OtherLogic** - obsahuje všechnu ostatní logiku, která se stará o běh aplikace
- **Resources** - obsahuje všechny soubory s texty, díky tomuto přístupu lze snadno přidat podporu dalších jazyků
- **ToolkitContent** - obsahuje ikony využívané knihovnami třetích stran
- **ViewModels** - složka pro logiku všech stránek
- **Views** - složka pro UI všech stránek

Návrhový vzor MVVM pro správnou funkcionalitu potřebuje implementovat mechanismy komunikující mezi vrstvami. Rozhraní *INotifyPropertyChanged* a *INotifyPropertyChanging*, slouží k upozornění o změně jednotlivých property dané třídy. Dále je potřeba implementovat rozhraní *ICommand*, které umožňuje navázání funkcí s prvky v XAML. Protože

implementace je potřeba u všech tříd používající tyto mechanismy, vytvořil jsem si bázové třídy *NotifyPropertyBase* a *CommandBase*.

Bázová třída *NotifyPropertyBase* obsahuje funkce pro upozornění o změně property. Nejčastější způsob uváděný v literatuře je volání funkce SetProperty s parametrem, který je typu string a nabývá hodnoty jména změněné property. Takto použitý přístup lze vidět v ukázce kódu 5.1.

```
private string _str = string.Empty;
public string Str
{
  get { return _str; }
  set
  {
    _str = value;
    OnPropertyChanged("Str");
  }
}
```

Zdrojový kód 5.1: Změna property pomocí řetězce

To má však nevýhodu. Pokud programátor zapomene pozměnit všechny řetězce s názvem proměnné překladač neohlásí chybu a program nebude fungovat korektně. Proto jsem prostudoval více do hloubky reflexi a zjistil jsem, že existuje delegát typu Expression, který v sobě nese různé dodatečné informace. Díky tomuto delegátu jsem byl schopný implementovat změnu property pomocí lambda výrazu z jakékoliv funkce. Dále jsem prostudoval attribut [CallerMemberName], který je schopen předat jméno property v případě volání z jejího těla. Výsledek použití třídy je vidět v ukázce kódu 5.2.

Dalším potřebným mechanismem pro správnou implementaci MVVM je implementace rozhraní *ICommand*. Implementaci jsem provedl v bázové třídě Command a Command$\langle T \rangle$. Třída Command implementuje příkazy bez parametrů a třída Command$\langle T \rangle$ implementuje příkazy s parametry pomocí šablonování. Po implementaci třídy je použití příkazů velmi jednoduché, jak lze vidět v ukázce kódu 5.3. Expression výraz umí pracovat i s lambda výrazy, proto je možné namísto definování funkcí použit i lambda výraz.

Při implementaci jsem si uvědomil, že se budou u jednotlivých ViewsModel a View vyskytovat stejné funkce pro navigaci stránek. Proto jsem vytvořil vlastní třídy *ViewBase* a *ViewModelBase*, které dovolí využívat příkazy pro navigační servis přímo z ViewModel a poskytnu základní logickou funkcionalitu navigace v XAML. Při definici XAML kódu si zvolím dědění z vytvořené třídy *ViewBase*. Následně si vytvořím vrstvu ViewModlel, která bude dědit od *ViewModelBase*. V posledním kroku si vytvořím datový kontext a poté budou dostupné všechna základní funkcionalita využívaná v MVVM. Díky tomuto přístupu je code behind krásně čistý jak je vidět v kódu 5.4.

V projektu na přiloženém mediu jsem vytvořil ViewModelTemplate.cs, ViewTemplate.xaml a ViewTemplate.xaml.cs, které jsou ukázkou a zároveň šablonou pro vytvoření nové stránky s využitím implementovaného MVVM.

Před návrhem bázových tříd jsem si nejdříve definoval rozhraní pro IViewModelBase, které musí obsahovat příkazy PageLoaded, GoBack, GoForward a NaviateTo. Dále musí obsahovat delegát Loaded. Díky této definici mohu v View pracovat s rozhraním namísto

```
public class ViewModel : NotifyPropertyBase
{
  // notifikace hodnoty v tele property
  private string _str = string.Empty;
  public string Str
  {
    get { return _str; }
    set { SetProperty(ref _str, value); }
  }

  // notifikace hodnoty z jine funkce dane tridy
  // bez pouziti SetProperty
  private void SetStr(string str)
  {
    _str = str;
    OnPropertyChanged(() => Str);
  }
}
```

Zdrojový kód 5.2: Ukázka použití třídy NotifyPropertyBase

určitých ViewModel. Rozhraní IViewModelBase je implementováno ve třídě ViewModelBase. Proto každá logika, která je zděděná z bázové třídy obsahuje tyto příkazy.

Protože každá stránka obsahuje virtuální metodu OnNavigatedTo a OnNavigatedFrom, která se volá z code behind. Musel jsem vyřešit, jak tuto metodu mít k dispozici ve ViewModel aniž bych využil code behind. Proto jsem vzájemně bázové třídy *ViewBase* a *ViewModelbase* provázal a po vytvoření konkrétního ViewModel jsou tyto metody dostupné bez programování v code behind.

Díky tomuto přístupu je poté možné využít virtuální metody OnNavigatedTo a OnNavigatedFrom z vrstvy ViewModel.

Využil jsem možností návrhu a přidal i proměnné, které obsahují odkaz na View nebo queriString s parametry stánky.

Po implementaci bázových tříd jsem mohl využívat návrhový vzor MVVM s rozšířenou funkcionalitou o výše uvedené prvky.

5.3 Databáze pomocí LINQ

Microsoft podporuje mnoho způsobů, jak ukládat statická data po vypnutí aplikace. Pro svou implementaci jsem si zvolil databázi s využitím LINQ. Použil jsem přístup code first, ve kterém nejdříve navrhnu třídu s patřičnými properties a pomocí kontextu bude databáze vytvořena sama.

Ve složce model jsem si proto vytvořil třídy Group, History, HistoryGroup, Plan a PlanGroup, ve kterých je definice schématu databáze. V modelu jsem také vytvořil třídu DatabaseContext, která obsahuje schéma tabulek a cestu k uložení databáze v proměnné ConnectioonString.

Při přístupu k databázi je jeden řádek tabulky z hlediska programování přístupný jako

Použití příkazu v XAML:

```
<Button
  Content="Prikaz s parametrem"
  Command="{Binding CommandWithParameter}"
  CommandParameter="{Binding Name}"/>

<Button
  Content="Prikaz bez parametru"
  Command="{Binding CommandWithoutParameter}"/>
```

Definice příkazu v ViewModel:

```
public Command<string> CommandWithParameter { get; set; }
public Command CommandWithoutParameter { get; set; }

private void ActionCmdPar(string str)
{
  // telo funkce s parametrem
}

private void ActionCmd()
{
  // telo funkce bez parametru
}

//konstruktor
public ViewModel()
{
  CommandWithParameter = new Command<string>(ActionCmdPar);
  CommandWithoutParameter = new Command(ActionCmd);

  //prikazy lze volat i z ViewModel
  CommandWithParameter.Execute("Hodnota parametru");
  CommandWithoutParameter.Execute();
}
```

Zdrojový kód 5.3: Ukázka použití třídy Command

objekt. Nebo více záznamů jako pole objektů. LINQ také umožňuje pomocí tříd EntityRef$\langle T \rangle$ a EntitySet$\langle T \rangle$ nastavovat vztahy mezi tabulkami. Po definici schématu je potom možné přecházet z aktuálního záznamu do záznamů jiných tabulek svázaných cizím klíčem pomocí tečkové notace.

Pro databázi mám vytvořenou třídu, která obstarává kompletní řízení přístupu do databáze a odstiňuje samotnou logiku databáze od logiky aplikace. Třída pracuje s objekty modelu a všechny databázové dotazy jsou prováděny uvnitř. K dispozici jsou pak listy, které obsahují například první tři plány nebo všechny plány. Tyto listy jsou pak využity v aplikaci. Dále se třída automaticky stará o konzistenci dat. Například pro vytvoření nové položky stačí vytvořit novou položku a předá jí metodě Create, která provede inicializaci primárního

```
public partial class NamePage : ViewBase
{
  public NamePage()
  {
    InitializeComponent();
    DataContext = new ViewModelTemplate();
  }
}
```

Zdrojový kód 5.4: Ukázka code behind s přítupem MVVM

klíče, vztahů a poté promítne položku do databáze.

5.4 Posílání objektů mezi stránkami

Protože posílání parametrů mezi stránkami je řešeno přes queryStrinq a nenašel jsem jednoduchý způsob pro předávání objektů vytvořil jsem si třídu *SendObject*, která odesílá objekty do jiných stránek. Metoda je statická a vytváří se po spuštění aplikace, proto je možné předávat parametry při přechodu na jednotlivé stránky. Při návrhu této třídy jsem využil boxování objektů, které nabízí C#. Třída má jednu property Object, do které lze veřejně pouze zapisovat. Dále také obsahuje metodu TryGetObject, která vrací booleovskou hodnotu podle úspěchu. Jejím vstupním parametrem je reference na objekt, který chceme načítat. Metoda nejdříve zjistí, jestli je objekt stejného typu a pokud ano vrátí úspěch a u vstupní proměnné provede inicializaci na hodnotu odeslaného objektu. V opačném případě vrátí neúspěch a u vstupní proměnné provede inicializaci na implicitní hodnotu jejího typu. Odeslaný parametr je možné úspěšně přečíst pouze jednou.

5.5 Jazyková lokace

Přístup k psaní lokalizace je možné u WP psát více způsoby. Jeden ze způsobů je vkládat veškeré texty do aplikace na místa, kde jsou potřeba. Následkem toho ale je velmi složitá úprava textu a následné přidání jazykových lokací je nemožné. Druhý způsob je vytvoření AppResources souborů, které oddělují aplikace od textů.

Takový přístup je velmi výhodný. Například při prvotním návrhu jsem pojmenoval účel jako skupinu. Při vývoji jsem znovu zhodnotil odpovídající slova a skupinu jsem přejmenoval na účel. V aplikaci je stále využívané slovo Group, ale díky změně AppRessources se vše zobrazuje korektně.

Protože WP nemá kompletní podporu všech světových jazyků, lze jen vyvíjet aplikace v podporovaných lokacích. Nejprve jsem si musel nastavil vyžadované podporované lokace v Properties aplikace. Po zadání vyžadovaných lokací se v projektu vytvořily AppRescouces pro přidané jazyky. Přístup k textu je pak jednoduchý. Lze k jednotlivým textům přistupovat přes třídu AppResources, kterou má na starosti jádro systému. WP vybere pak správnou lokaci podle nastavení systému uživatele.

Další nastavení týkající se jazyků je podle Microsoftu nazváno globalizace. Globalizace slouží ke korektnímu pojmenování aplikace ve stavu, kdy není spuštěná. Například jméno v menu nebo text na živé dlaždici. Proto jsem si vytvořil pomocí C++ .dll knihovnu, která

```
<TextBlock Text="text">
  <i:Interaction.Triggers>
    <i:EventTrigger EventName="Tap">
      <i:InvokeCommandAction Command="{Binding NazevPrikazu}"/>
    </i:EventTrigger>
  </i:Interaction.Triggers>
</TextBlock>
```

Zdrojový kód 5.5: Ukázka volání příkazu pomocí interakce

v sobě obsahovala tyto texty. Pro různé jazyky je potřeba dodržovat přesné pojmenování souborů. Obecný soubor s globalizační texty se jmenuje AppResLib.dll a pro český globalizační soubory jsem našel podle manuálu jméno AppResLib.dll.0405.mui.

5.6 Implementace vzhledu stránek

Vzhled jednotlivých stránek jsem uspořádal v Blend for Visual Studio podle návrhu aplikace. Základní chování elementů a animace jsem doprogramovával zápisem do XAML. Aby bylo možné svázat jednotlivé události s příkazy vrstvy ViewModel musel jsem využít knihovnu *Microsoft.Expression.Interactions*, která svázání umožňuje. Použití je možné vidět v ukázce kódu 5.5.

Jediné v čem jsem se odlišil od návrhu byla vyhledávací funkce u plánů. Protože se muselo provést pokaždé nové prohledávání při změně jména aktivity, také při přidání či odebrání plánu byla operace náročná jak na procesorový čas, tak i na baterii. V průběhu testování jsem téměř funkci vyhledávání v plánech nevyužil, protože plánovaných položek nebylo nikdy mnoho. Z tohoto důvodů jsem usoudil, že funkce provádí více škody než užitku. Proto ve finální verzi bylo prohledávání odstraněno. Výsledný vzhled aplikace je možné vidět v příloze A.

Kapitola 6

Testování a názor respondentů

Testování aplikace bylo prováděno iterativně. Nejdříve jsem provedl analýzu již existujících řešení a poté jsem navrhl více návrhů aplikace, které jsem rozkreslil na papír. Návrhy jsem předložil cílovým uživatelům a konzultoval prvky, které jsou podle uživatelů užitečné.

Po základním návrhu aplikace jsem provedl implementaci designu úvodní obrazovky pro WP platformu. Připravil jsem také implementace v kterých bylo odlišné rozmístění prvků nebo použity jiné ikony tlačítek. Podle reakce respondentů jsem vytvořil výsledný vzhled aplikace a provedl jsem další interakci s uživateli.

Když byla aplikace vytvořená využil jsem Store Test Kit nabízený jako doplněk Visual Studia. Store Test Kit spustí aplikaci v emulátoru a zaznamenává využité zdroje, událostí a nabízí mnoho dalších zaznamenávaných informací. Microsoft má vypsané doporučené scénáře k otestování. Všechny tyto scénáře jsem otestoval a analyzoval běh aplikace. Dále jsem provedl další testy, které jsem usoudil jako kritické body běhu aplikace.

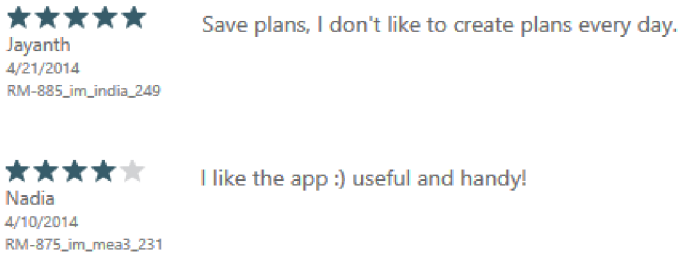
Konečný test aplikace proběhl na reálném využití, ve kterém jsem se zaměřoval na svižnost běhu aplikace, jednoduchost ovládáni a u toho jsem si psal poznámky. Program jsem využíval týden a poté provedl poslední úpravy dle poznámek a připravil k publikaci.

Po publikaci aplikace díky WP Store a sekci Crash Reports jsem objevil chybu, která nenastala v průběhu testování. Protože jsem si mohl stáhnou podrobné výsledky o chybě, analyzoval jsem příčinu a hned ten den vydal aktualizaci. Podle Crash Reports již žádná chyba v aplikaci od té doby nenastala.

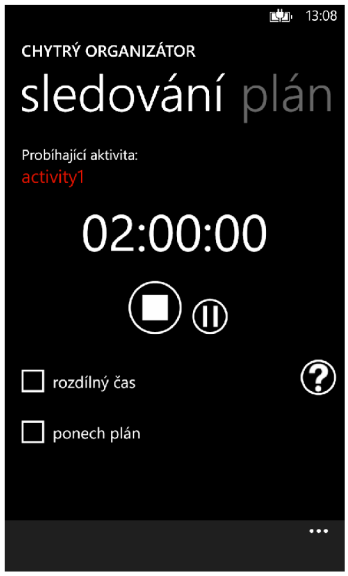
Dále jsem se zaměřil na připomínky od uživatelů z WP Store. Hodnocení aplikace se objevilo z Indie a ze Spojených Států. Protože hodnocení, jak je možné vidět na obrázku 6.1, obsahovalo návrh na dodatečnou funkcionalitu začal jsem pracovat na nové verzi podle přání respondenta.

Protože respondent pravděpodobně vykonává aktivity, které jsou obdobné, přidal jsem do pohledu s běžící aktivitou možnost zaškrtnout ponechání plánu. V případě zaškrtnutého políčka a ukončení dané aktivity se aktivita nachází jednak v historii, ale také v plánu připravená pro další využití. Výsledek je možné vidět na obrázku 6.2.

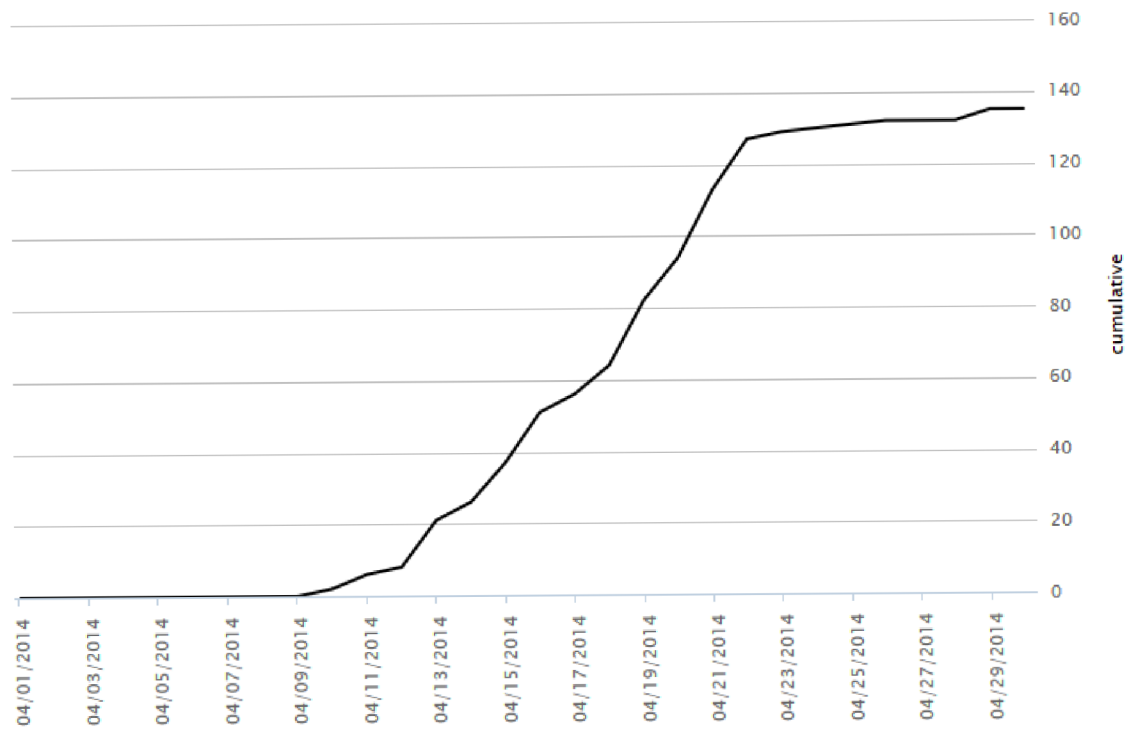
Podle statistik stahovanosti si aplikace doposud nevede špatně. Podle aktuálně dostupných výsledků (od 9.4 do 30.4) byla aplikace stažena 135x. Celkový graf je možné vidět na obrázku 6.3.



Obrázek 6.1: ScreenShot z WP Store, který ukazuje hodnocení aplikace z jiných státu s připomínkou k dodatečné funkcionalitě.



Obrázek 6.2: ScreenShot z aplikace Chytrý Organizátor po úpravě podle přání respondenta.



Obrázek 6.3: ScreenShot z WP Store, který ukazuje statistiky stažení aplikace od 9.4 do 30.4.

Kapitola 7

Závěr

Cílem práce bylo vytvořit rychlou a intuitivní aplikaci pro monitorování a organizaci času. Při vývoji jsem se snažil obsáhnou všechny cíle aplikace. I když první návrhy byli odlišné od finálního vzhledu a aplikace prošla dlouhým vývojem výsledek práce obsahuje všechny body zadání. Dále jsem splnil svůj hlavní cíl, aby se aplikace přizpůsobila stylu metro.

Výsledná aplikace podporuje angličtinu a češtinu. Její jméno v anglickém jazyce je Smart Time a v českém Chytrý organizátor. Aplikace umí organizovat účely, plánovat aktivity a zaznamenávat proběhlé aktivity. Oproti konkurenci má možnost zadávání rychlé aktivity a možnost zadávání opožděného času. Věřím, že tyto komponenty usnadní uživatelů práci s aplikací a v budoucnu si jí oblíbí. Dále aplikace umí přidat plán nebo historii do 0 až N různých účelů. Díky tomu je pak možné dosáhnout podrobnějších a přehlednějších statistik pro analýzu. Výslednou aplikaci je možné stáhnou z WP store na adrese http://windowsphone.com/s?appId=855eb2d2-a2e1-4990-91bf-804f8a474b9f.

Pro aplikaci mám vymyšlených mnoho rozšíření jako je přidání lokace k danému účelu. Dále bych se chtěl zaměřit na vylepšení grafického zobrazení. Uživatel by mohl vybrat více účelů a poté by se mu graficky zobrazili podle času stráveného na jednotlivých účelech. Další aktualizací bych chtěl vylepšit uložení plánu, který navrhl respondent. Vylepšení by spočívalo v přidání možnosti opakujícího plánu do vlastností plánu. Uživatel by pouze při vytváření nebo editaci zadal, zda se má plán opakovat nebo ne a pak by nemusel před každým ukončení zaškrtávat ponechání plánu. Dále bych chtěl reagovat na podněty respondentů.

Zda vývoj bude pokračovat rozhodnou samy uživatelé. Pokud aplikace bude stále stahovaná a dosáhne vysokého počtu uživatelů, kteří budou aplikace kladně hodnotit a využívat chtěl bych na projektu pokračovat a dostat se do podvědomí uživatelů jako programátor WP platformy. V opačném případě se raději zaměřím na nový projekt, který začnu s novým pohledem na vývoj aplikace díky nasbíraným cenným zkušenostem.

Pro mě byl projekt velice přínosný. Jakožto začátečník platformy WP jsem prozkoumal mnoho přístupů a možností, jak programovat správně pro tuto platformu. Dále jsem si oblíbil programovací jazyk C# .NET, který nabízí velmi bohaté zázemí již připravených a odladěných knihoven. Díky vizi Microsoftu přiblížení mobilní platformy a stolní platformy Windows jsem s dosaženými znalostmi velmi spokojen a do budoucna bych se chtěl dále této platformě věnovat.

Literatura

[1] Allen, D.: *Mít vše hotovo*. Jan Melvil Publishing, 2008, ISBN 978-80-903912-8-4.

[2] Allen, D.: *Aby vše klapalo*. Jan Melvil Publishing, 2009, ISBN 978-80-87270-00-4.

[3] Ferracchiati, F. C.: *LINQ for Visual C# 2008*. Apress, 2008, ISBN 978-1-4302-1580-6.

[4] JeffKoch; Lieberman, L.: Building Apps for Windows Phone 8 Jump Start. `http://channel9.msdn.com` , 2012, [cit. 2014-04-30].

[5] Kolektiv autorů: Developer's Guide to Microsoft Prism Library 5.0 for WPF. `http://msdn.microsoft.com` , [cit. 2014-04-30].

[6] MacVittie, L. A.: *XAML in a Nutshell*. O'Reilly Media, 2006, ISBN 978-0-596-52673-3.

[7] Mitra, A.: Windows Phone 8 Application Lifecycle. `http://iamabhik.wordpress.com/2012/12/01/windows-phone-8-application-lifecycle/` , [cit. 2014-04-30].

[8] Petzold, C.: *Mistrovství ve Windows Presentation Foundation*. COMPUTER PRESS, 2008, ISBN 978-80-251-2141-2.

[9] Russo, M.; Pialorsi, P.: *Microsoft LINQ Kompletní průvodce programátora*. COMPUTER PRESS, 2009, ISBN 978-80-251-2735-3.

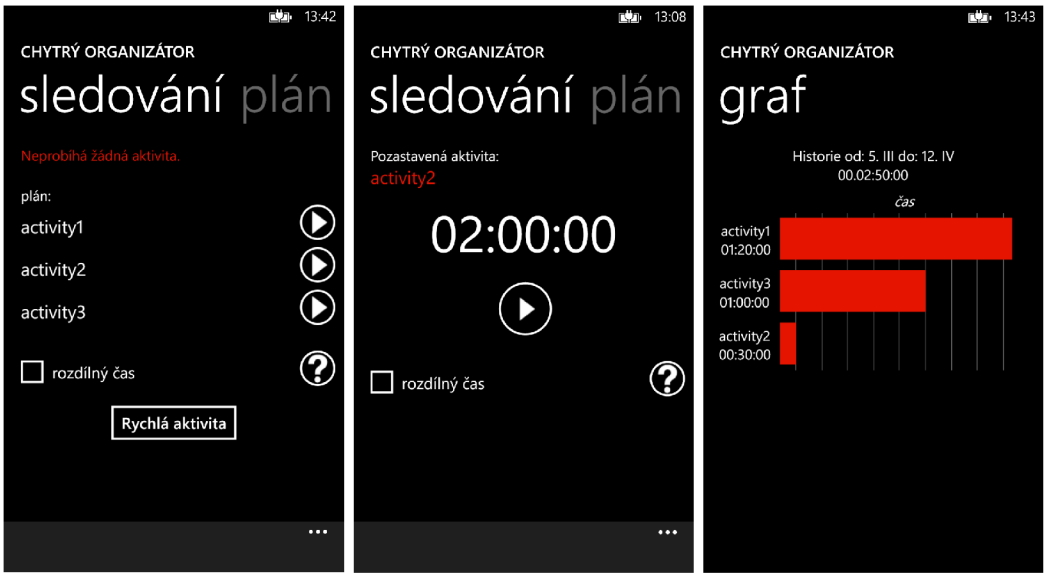
[10] Stephens, R.: *WPF Programmer's Reference*. Wrox, 2010, ISBN 978-0-470-47722-9.

[11] Szostak, T.: *Title Windows Phone 8 Application Development Essentials*. Packt Publishing, 2013, ISBN 978-1-84969-677-7.

[12] Whitechapel, A.; McKenna, S.: *Windows Phone 8 Development Internals*. O'Reilly Media, 2013, ISBN 978-0-7356-7623-7.

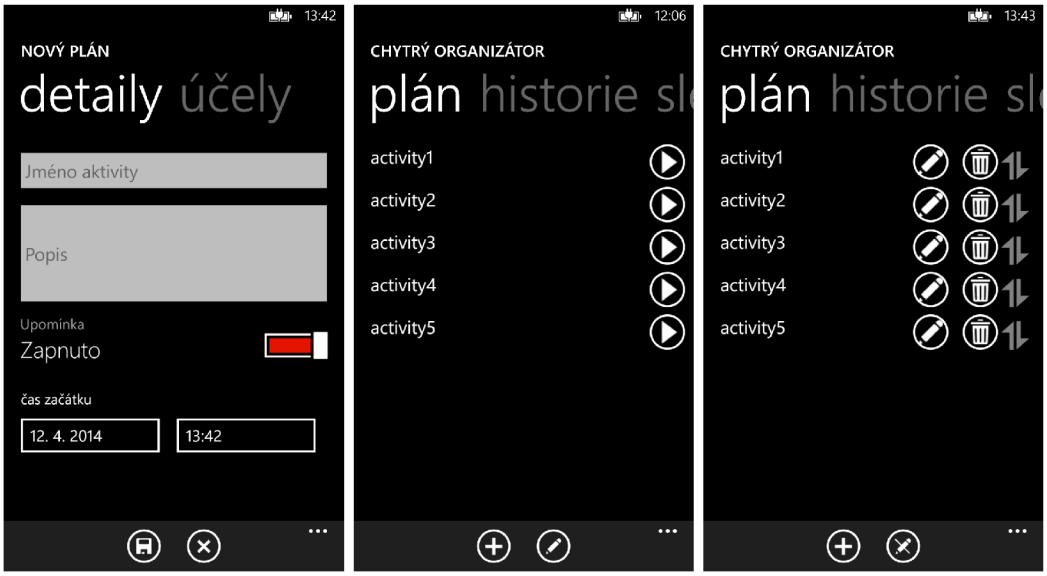
Příloha A

Výsledný vzhled aplikace



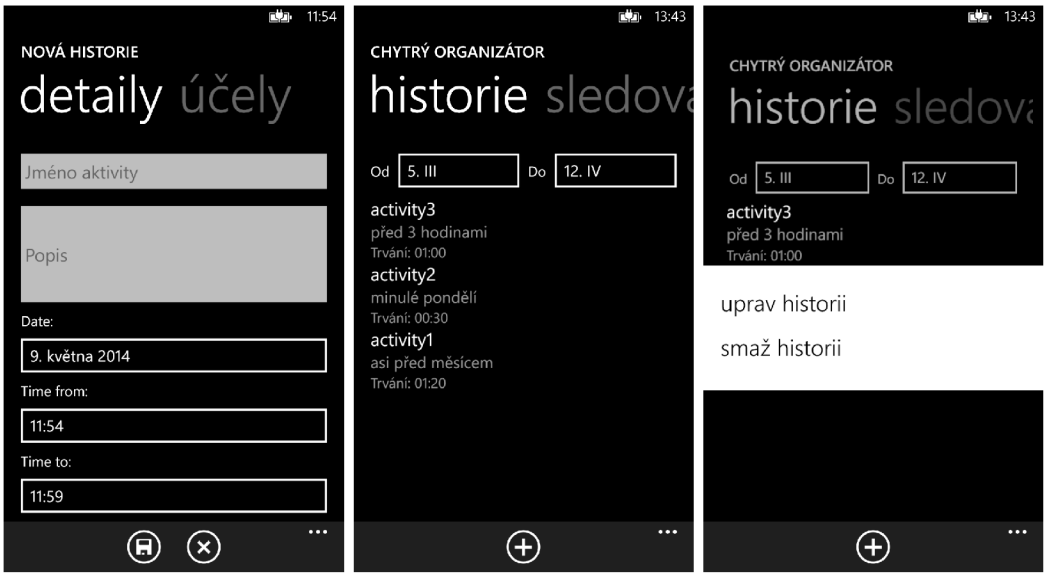
(a) Úvodní obrazovka (b) Pozastavená aktivita (c) Grafické zobrazení historií

Obrázek A.1: První sada obrázků ukazuje úvodní obrazovku aplikace a režim, ve kterém je pozastavená aktivita. Dále je zde vidět grafické zobrazení historií a umístění jednotlivých časů v grafu.



(a) Nový plán (b) Režim prohlížení (c) Režim editace

Obrázek A.2: Druhá sada obrázků ukazuje práci s plánem. Můžeme zde vidět vytvoření nového plánu a jeho volby. Dále jsou zde zobrazeny dva základní režimy pro plánování.



(a) Nová historie (b) Výpis historií (c) Kontextové menu

Obrázek A.3: poslední sada obrázků ukazuje práci s historií. Je zde možné vidět možnosti vytvoření nové historie. Dále je vidět relativní konvertor času při výpisu historií a nakonec kontextové menu historie.