Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Softwarového inženýrství

Bakalářská práce

Desktopový MacOS klient pro systém RTM

David Ungurean

Vedoucí práce: Ing. Robert Pergl, Ph.D.

12. května 2015

Poděkování

Rád bych poděkoval především vedoucímu práce Ing. Robertu Perglovi, PhD. za vedení a za jeho cenné rady a připomínky. Rovněž bych chtěl poděkovat rodině a přátelům za podporu při studiu a tvorbě práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 12. května 2015 .....

Odkaz na tuto práci

Ungurean, David. *Desktopový MacOS klient pro systém RTM*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato práce se zabývá vývojem klientské Mac OS X aplikace pro systém Remember the Milk, který poskytuje služby spojené se správou úkolů. Teoretická část práce obsahuje rešerši nejlepší praxe návrhu a implementace architektury a grafického uživatelského rozhraní OS X aplikací. Část praktická se věnuje vývoji za použití technologií, které maximalizují efektivitu ovládání.

Klíčová slova Mac, OS X, správa úkolů, klientská aplikace, efektivita ovládání, produktivita

Abstract

This thesis deals with development of client Mac OS X application for Remember the Milk system, which provides services associated with task management. The theoretical part describes best practices in design and implementation of architecture and graphical user interface of OS X applications. The practical part focuses on development, using technologies that maximaze the efficiency of use.

Keywords Mac, OS X, task management, client application, efficiency of use, productivity

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Dobré plánování úkolů zvyšuje produktivitu a efektivnost využití času jedince. Není tedy divu, že se mnohé společnosti snaží prorazit na trh s produktem, který dokáže plánování a správu úkolů zjednodušit. Jednou z nich je i společnost Remember the Milk (dále jen RTM), která k dnešnímu dni zaznamenává více než pět milionů registrovaných uživatelelských účtů.

RTM poskytuje uživateli rozsáhlý ekosystém aplikací, pomocí kterých může organizovat a třídit svoje úkoly. Historickým středobodem RTM je webová aplikace, ve které jsou dostupné všechny hlavní funkce. S postupem času začala přibývat rozšíření pro významné služby jako je GMail, Outlook a klienti pro hlavní mobilní operační systémy. Desktopový klient pro operační systém Mac OS X zatím chybí. Cílem této práce je tento nedostatek napravit.

Toto téma jsem si zvolil, protože se zabývám tvorbou aplikací pro operační sytém iOS, který je také vyvíjen firmou Apple. Knihovny pro vývoj jsou velice podobné, jedná se však o diametrálně odlišný systém pro zcela jiný druh zařízení. Tento fakt mě natolik zaujal, že jsem se rozhodl zmíněné rozdíly doučit a nově nabyté znalosti aplikovat v praxi právě na klientu pro službu Remember the Milk.

Pro neoficiální RTM-aplikace je zakázáno používat v názvu zkratku RTM nebo spojení *Remember the Milk*. Rozhodl jsem se proto po vzoru aplikací *MilkMaid* a *MilkBar* pojmenovat mého klienta podobným způsobem. Vybral jsem název *Buttermilk*.

Kapitola 1

Cíl práce

1.1 Rešeršní část

Cílem teoretické části práce je provést rešerši nejlepší praxe návrhu a implementace grafického rozhraní a architektury aplikací pro operační systém Mac OS X 10.9 a vyšší. K tomu patří popis použitých technologií, postupu vývoje a rozbor grafického prostředí.

1.2 Praktická část

Cílem části praktické je analýza, návrh a implementace klientské desktopové aplikace pro systém Remember the Milk na platformě Mac OS X. U těchto úkonů budu využívat informací získaných z rešerše. V implementační části budu demonstrovat správné postupy tvorby podobných aplikací, využívání návrhových vzorů a dodržování doporučených konvencí. Výsledkem by měla být funkční aplikace, která bude schopna vykonávat většinu hlavních funkcí webové verze.

Kapitola 2

Rešerše tvorby aplikací pro OS X

V následující kapitole se pokusím stručně shrnout dosavadní přístup k tvorbě aplikací pro operační systém OS X. Jaké technologie se pro vývoj používají, jak se navrhuje architektura a jakým způsobem se navrhuje grafické uživatelské rozhraní.

2.1 Cocoa

Cocoa je sada objektově orientovaných frameworků, které programátorovi nabízí základní prvky potřebné k tvorbě aplikací pro systémy iOS a OS X a současně těmto aplikacím poskytuje runtime prostředí.

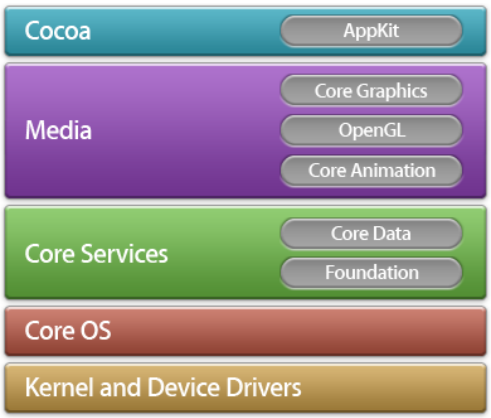
Runtime část Cocoa zajišťuje zobrazování uživatelského rozhraní a volání nižších procedur operačního systému [1]. Knihovní část obsahuje třídy, které poskytují široké spektrum běžných funkcí, od prvků pro práci s řetězci až po objekty uživatelského prostředí. V případech, kdy poskytnuté rozhraní nepokryje všechny potřeby našeho programu, není problém třídu oddědit nebo rozšířit pomocí kategorií.

Pro vývoj aplikací s využitím frameworku Cocoa lze použít několik programovacích jazyků, doporučovaným je Objective-C, ve kterém je většina Cocoa napsaná.

V polovině roku 2014 uvedl Apple nový programovací jazyk Swift. Tento jazyk byl v době vypracování praktické části práce stále ještě ve vývoji, neposkytoval možnost privátních proměnných a vývojové prostředí při práci s ním nebylo dostatečně odladěné. Proto jsem se rozhodl pro implementaci v jazyku Objective-C.

2.1.1 Vrstvy a příslušné frameworky

Každý framework z Cocoa koresponduje s jednou vrstvou operačního systému. Jednotlivé vrstvy staví na vrstvách předchozích [2], obalují je a rozšiřují jejich rozhraní. Knihovny z vyšších vrstev poskytují objetově-orientované abstrakce pro nízkoúrovňové konstrukty. Tyto abstrakce redukují objem kódu, který by jinak musel programátor napsat. Přestože se o Cocoa hovoří jako o sadě objektově-orientovaných knihoven, musím zmínit, že některé frameworky z nižších vrstev jsou napsány převážně v jazyce C. S nimi je zapotřebí zacházet s větší opatrností, a to hlavně při správě paměti. Kvůli těmto dvěma zmíněným faktům je dobrým zvykem upřednostňovat frameworky z horních vrstev před těmi z vrstev nižších. V nich by měl programátor hledat, až pokud nenajde řešení v předchozích vrstvách.



Obrázek 2.1: Jednotlivé vrstvy Cocoa [3]

2.1.1.1 Application (Cocoa)

Aplikační vrstva Cocoa je nejvrchnější vrstvou. Zajišťuje vzhled aplikací a jejich odezvu k akcím uživatele [4]. V této vrstvě by měl vývojář hledat a najít většinu základních funkcí, které potřebuje pro tvorbu aplikací. Autolayout, gesta, notifikační centrum nebo funkce napojené na vyhledávací systém spotlight.

Mezi hlavní frameworky této vrstvy patří:

- **AppKit** – Appkit je klíčový framework k vývoji aplikací pro OS X. Implementuje uživatelské rozhraní včetně oken, dialogů, control prvků a správy událostí.
- **Preferences Pane** – Umožňuje přidat plugin s nastavením aplikace do Předvoleb systému.

2.1.1.2 Media

V Media vrstvě se nachází frameworky pro práci s grafikou, videem a zvukem. Poskytuje funkce pro úpravu obrázků, ruční kreslení tvarů, linií a bezierových křivek. Obsahuje knihovnu OpenGL ES pro rendrování 2D a 3D grafiky s pomocí hardwarové akcelerace a známý framework Core Graphics (také známý jako Quartz 2D).

- **CoreGraphics** – CoreGraphics je srdcem dvojrozměrné grafiky v OS X. Poskytuje podporu pro rendrování vektorové grafiky a úpravu obrázků.
- **CoreAnimation** – Knihovna pro tvorbu uživatelských animací, které se vykreslují pomocí CoreGraphics.
- **AV Kit** – AV Kit umožňuje přehrávání audiovizuálního obsahu s využitím standartních systémových prvků.

2.1.1.3 Core Services

Knihovny vrstvy Core Service poskytují základní služby aplikacím bez jakéholiv dopadu na jejich vzled. Spravují například službu Grand Central Dispatch (GCD), která díky svému jednoduchému rozhraní mnohonásobně ulehčuje práci s vlákny. Dále se zde nalézá framework Social poskytující API pro zasílání požadavků sociálním médiím, které má uživatelem spojené se svým účtem.

- **Foundation** – Spolu s AppKitem z aplikační vrstvy se jedná o dva nejpodstatnější frameworky ze sady Cocoa. Obsahuje třídy reprezentující řetězce, čísla, kolekce a další.
- **Core Data** – Core Data je framework využívaný pro perzistenci model objektů aplikace. Pro ukládání dat využívá zabudované technologie SQLite.
- **WebKit** – Umožňuje aplikaci zobrazovat obsah HTML. Je schopen zpracovávat HTML i JavaScript kód.

2.1.1.4 Core OS

Technologie a frameworky z vrstvy Core OS poskytují nízkoúrovňové služby spjaté s ovládáním hardwaru a sítě. Tyto služby jsou přímo závislé na vrstvě Kernel.

- **System Configuration** – Poskytuje aplikaci informace o nastavení síťového rozhraní a dostupnosti připojení.

2.1.1.5 Kernel and Device Drivers (Darwin)

Darwin (Kernel and Device Drivers Layer) je nejspodnější vrstva OS X. Zahrnuje kernel, ovladače a upravenou BSD část systému. Tato vrstva je postavena převážně na open source technologiích.

- **BSD** – Jedná se o upravenou verzi operačního systému BSD. Tato implementace poskytuje mimo jiné API POSIX vláken a socketů BSD.
- **I/O Kit** – Knihovna pro vývoj ovladačů pro OS X.

2.2 Objective-C

Jazyk Objective-C se v současnosti používá primárně k vývoji aplikací pro operační systémy firmy Apple, a to iOS a Mac OS X. Jedná se o objektově orientovaný jazyk, implementovaný jako nadstavba jazyka ANSI C, do které byl přidán systém zasílání zpráv známý z jazyka Smalltalk. Díky této vlastnosti je možné přeložit libovolný program napsaný v jazyce C kompilátorem pro Objective-C. Stejně jako C++ i Objective-C přidává jazyku C objektově orientované rysy. Obj-C je však narozdíl od C++ značně více dynamické a odkládá většinu rozhodnutí do runtimu.

V následujících několika odstavcích se pokusím stručně shrnout další podstatné vlastnosti tohoto jazyka, které ovlivňují způsob, jakým se v něm vyvíjí aplikace.

2.2.1 Posílání zpráv

V Objective-C se stejně jako ve Smalltalku posílají zprávy místo přímého volání metod. Každá zpráva má definovaný selektor, což je řetězec popisující její název. Když pak za běhu programu dostane objekt zprávu, přistoupí se do tabulky selektorů, kde se přes přiřazený ukazatel dostaneme na místo v paměti, kde se nachází implementace dané metody. Díky této vlastnosti lze při runtimu zaměňovat ukazatele selektorů a měnit tak chování bez nutnosti znovuspuštění programu. Dokonce se můžeme objektů dotazovat, zda umí reagovat na selektor metodou `- (BOOL)respondsToSelector:(SEL)aSelector`, a na základě výsledku se patřičně zachovat. Této vlastnosti se využívá například u protokolů.

2.2.2 Protokoly

Protokoly deklarují metody, které by třídy, co splňují daný protokol, měly implementovat. Daná třída buďto musí implementovat danou metodu, toto se vyjadřuje direktivem `@required` nebo jen může implementovat – direktivum `@optional`. Přes protokoly se v Cocoa často realizuje návrhový vzor delegát. Dobrý příklad je buňka tabulky, které deleguje příslušný kontroler. Při

akci kliknutí na buňku, přepošle buňka zprávu svému delegátovi, který ji dále zpracuje.

2.2.3 Bloky

Bloky jsou místní verze anonymních funkcí, známých též jako lambda výrazy. Při tvorbě iOS a OS X aplikací mají hojné využití. Typickým příkladem je tzv. completion block, který se volá po dokončení akce s předem neznámou délkou. Dále je bloky možné využít i jako callback a posílat si pomocí nich informace. Za zmínku stojí také kombinace bloků s technologií Grand Central Dispatch, kterou Apple vyvinul pro jednoduché spouštění paralelních operací na vícejádrových procesorech. Díky ní je možné vykonávat některé, většinou časově náročné, bloky kódu na pozadí, a vyhnout se tak zástavám v grafickém prostředí. Tohoto jsem hojně využil při obalování metod knihovny rtm2cocoa do bloků (detailněji popsáno v sekci 4.6), jelikož byla celá synchronní a docházelo kvůli tomu k výše zmíněným zásekům.

2.2.4 Správa paměti

Zde si můžeme vybrat mezi dvěma hlavními druhy správy paměti:

- **ARC** (automatické počítání referencí) – Dostupné od verze Xcode 4.2 a OS X v10.6. Automatické počítání referencí nechává správu paměti plně na překladači. Ten vhodně doplní do kódu příkazy `retain` a `release`, aby nedocházelo k únikům paměti. Jediný způsob, jak můžeme do jisté míry ovlivnit uvolňování objektů, je pomocí atributů `strong` a `weak` u tzv. properties (2.2.6).

- **MRC** (manuální počítání referencí) – U tohoto systému zůstává počítání referencí na programátorovi.

Třídy, které využívají ARC, nelze v projektu bez zásahu používat s třídami využívající MRC. Tento problém může nastat například při snaze použít starší knihovnu. Osobně jsem při implementaci na tuto překážku narazil. A to při snaze využít framework rtm2cocoa, který slouží jako wrapper pro API služby RTM. Musel jsem všechny rtm2cocoa třídy vložit do jednoho balíčku, vytvořit nový cíl pro kompilaci a nastavit kompilátoru pro tento cíl příznak `-fno-objc-arc`.

2.2.5 Kategorie

Kategorie slouží jako rozšíření stávajících tříd. Pomocí kategorií můžeme přidat do již existující třídy atributy i metody, aniž bychom museli využít dědičnosti. Dokonce lze metody i přepisovat, tady v tomto případě musíme jednat

opatrně. Obzvláště u systémových tříd, kde si nejsme přesně jisti jejich implementací. Díky kategoriím můžeme například přidat do třídy `NSDate` pomocné metody jako `isToday`, `isTomorrow`, kterých jsem využil při třídění úkolů v implementační části práce.

2.2.6 Properties

Properties jsou nadstavbou nad instančními proměnnými. Pokud definujeme proměnnou jako property a neřekneme jinak, vygeneruje se nám automaticky getter a setter. Vyhneme se tím psaní zbytečného kódu a současně se zamezí přímému přístupu k proměnným. U properties můžeme definovat několik vlastností, a to práva zápisu, typ reference a atomicitu.

```
1 @property (readonly, nonatomic, strong) NSString *jmeno;
```

Tento příkaz definuje instanční proměnnou typu ukazatel na objekt třídy `NSString`. Instance naší třídy si na ni bude držet silnou referenci (atribut strong), implikující vlastnictví objektu, vygeneruje se pouze getter a příznak nonatomic zajistí, že se nebude zbytečně generovat kód, který by zajistil atomicitu operací nad tímto objektem.

2.2.7 Hlavičkové soubory

Třídy v Obj-C narozdíl od moderních jazyků jako je Java nebo C# vyžadují rozhraní rozdělené do dvou souborů. Hlavičkový soubor s koncovkou `.h` a implementační s koncovkou `.m`. Důvodem je zpětná kompatibilita s jazykem C. Nevýhodou je dvojnásobný počet souborů se zdrojovým kódem. Často zapomínanou výhodou je například možnost lokálních konstant v implementačních souborech, které pak nejsou viditelné nikde jinde.

Jelikož Objective-C nemá stejnou možnost privátních a protected metod jako je tomu v C++, využívají se hlavičkové soubory i pro řešení tohoto problému.

2.2.8 Jmenné prostory

Objective-C nemá jmenné prostory. Toto je dle mého názoru jeden z největších nedostatků. Důsledkem je nutnost pojmenovávat všechny třídy unikátně. Konvence doporučuje přiřadit všem třídám projektu určitý prefix. V práci jsem použil prefix `BM` vytvořený zkrácením názvu Buttermilk.

2.3 Struktura a architektura OS X aplikací

Aplikační soubory můžeme rozdělit do tří částí: zdrojové kódy, resources a metadata.

Mezi zdrojové kódy řadíme hlavičkové a implementační soubory aplikace. Tyto soubory definují chování aplikace. V hlavičkových souborech deklarujeme veřejné rozhraní tříd, jejich metody a atributy. V implementačních souborech se pak nalézá příslušná implementace a privátní metody spolu s privátními atributy.

Resources tvoří soubory, které se dále nekompilují. Typicky se jedná o obrázky, videa, ikony a lokalizovatelné textové řetězce. Tyto soubory jsou přibaleny do archivu aplikace, která je dále využívá.

Metadata se ukládají do souboru formátu XML a popisují obecné informace o aplikaci. Její název, v jakém SDK byla vytvořena, jaká je minimální kompatibilní verze systému, na kterém je aplikaci možné spustit a další informace potřebné k archivaci.

2.3.1 Model-View-Controller

MVC je architekturní vzor, který každému objektu v aplikaci přiřazuje jednu ze tří rolí: model, view nebo controller. Nedefinuje pouze toto rozdělení, ale i způsob, jakým mezi sebou jednotlivé objekty mohou komunikovat. Každý z těchto objektů je od ostatních striktně oddělen [5]. Toto rozdělení umožňuje větší flexibilitu a znovupoužitelnost tříd. Jedním z příkladů, na kterém lze demonstrovat výhody MVC, je jednoduchý port aplikace z iOS na OS X, kde stačí pouze vyměnit zobrazovací vrstvu view, jelikož model a controller zůstavají v podstatě stejné.

2.3.1.1 Model

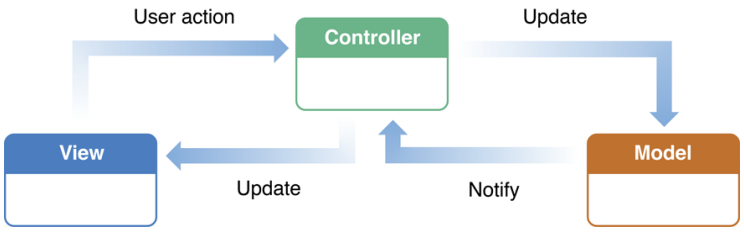
Objekty modelu obsahují data problémové domény aplikace a definují výpočetní logiku, která je zpracovává. Modelové objekty mohou kompozičně obsahovat další modelové objekty, komunikovat s nimi, nebo na ně delegovat práci. Při korektním návrhu by objekty modelu neměly nikdy komunikovat s objekty view. Pro zprostředkování této komunikace je zde kontroler. Když model změní stav, například získá přes síť nová data, notifikuje kontroler, který dále provede překreslení view objektů.

2.3.1.2 View

View objekty jsou prvky grafického rozhraní, které může uživatel přímo vidět a interagovat s nimi. Akce uživatele je zpracována, odchycena controllerem a dále se promítá do změny v modelu.

2.3.1.3 Controller

Controller vystupuje jako prostředník mezi view a modelem. Řídí logiku aplikace, změny a překreslování. Dále spravuje životní cyklus jiných objektů, především view.



Obrázek 2.2: Model-View-Controller [5]

2.4 Návrh uživatelského rozhraní

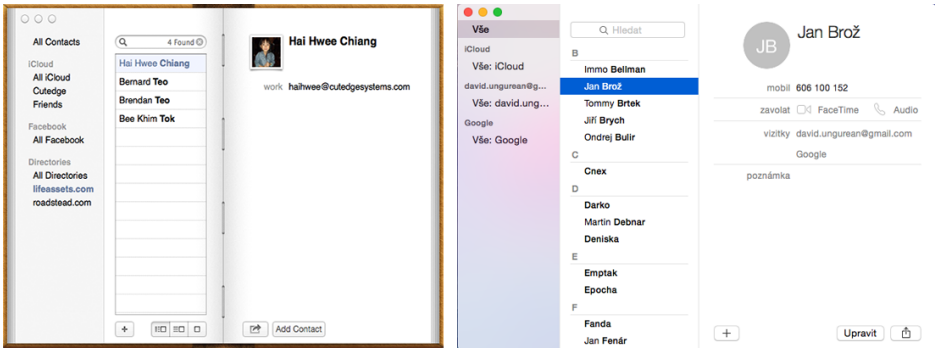
Grafické rozhraní by mělo být navrženo tak, aby uživatel musel co nejméně přemýšlet před každou akcí, kterou chce provést. Hlavní prvky by měly být dobře viditelné a jejich účel by měl být na první pohled zřejmý.

2.4.1 Prostředí operačního systému

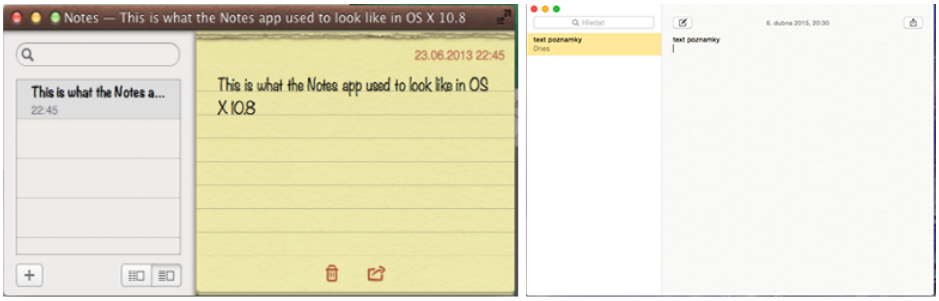
Při tvorbě grafického rozhraní je nutné brát ohled také na vzhled opračního systému. Jaké používá fonty, zdali má prostředí ve flat stylu, jestli využívá například prvků průhlednosti nebo jiných grafických elementů. V této oblasti prošel OS X za poslední dva roky znatelným vývojem, který jej přiblížil více grafickému rozhraní operačního systému iOS.

Změny ve vzhledu systémových prvků z Mavericks na Yosemite

Jednou z nejpodstatnějších změn je upuštění od starého vzhledu, který využíval přírodní prvky a materiály, jako je kůže nebo papír. Tyto prvky se nahradily světlejším a převážně jednoodstínovým vzhledem s minimem rušivých elementů. Pro porovnání jsem připojil snímky 2.3 a 2.4, kde na levé straně můžete vidět vzhled z verzí před a na pravé straně z verzí po uvedení OS X Yosemite. Konkrétně se jedná o systémové aplikace Kontakty a Poznámky.



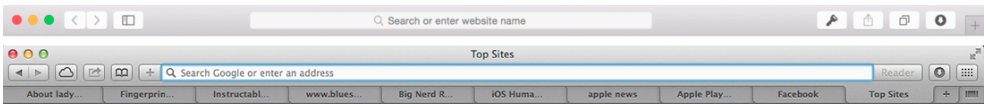
Obrázek 2.3: Nová a stará verze aplikace Kontakty [6]



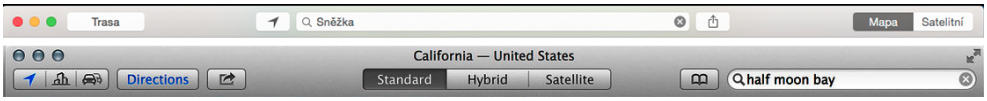
Obrázek 2.4: Nová a stará verze aplikace Poznámky [7]

Druhou hlavní změnou je přidání prvků průhlednosti a tzv. vibrancy. Vibrancy je sofistikovaný model míšení barev, který umožňuje elementům UI absorbovat barvy z obsahu, který se aktuálně nachází pod nimi [8]. Díky tomu i jednoduchá akce, jako je změna pozadí obrazovky, dokáže změnit vzled většiny programů. Systém je tímto možné více přizpůsobit vlastním preferencím.

Úpravou si prošla i záhlaví oken. S cílem co největšího zjedodušení GUI a přidání prostoru pro obsah uživatele se musela záhlaví zákonitě zmenšit. Na snímcích 2.5 a 2.6 je vidět rozdíl mezi horní (Yosemite) a spodní (Mavericks) verzí v systémových aplikacích Safari a Mapy. Původní dvě úrovně se slily do jedné a ubylo přebytečných ovládacích prvků. U těchto záhlaví je navíc možné nastavit mód Vibrancy, který jim umožní na sebe brát vzhled obsahu aplikace.



Obrázek 2.5: Nové a staré záhlaví okna aplikace Safari [9]



Obrázek 2.6: Nové a staré záhlaví okna aplikace Mapy [10]

2.4.2 Podobné aplikace

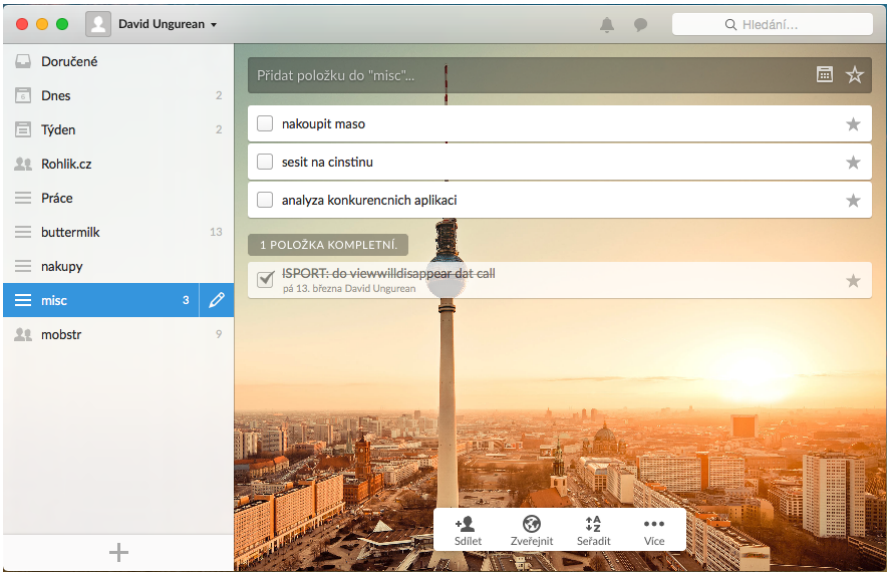
Jelikož chování uživatele je do jisté míry závislé na jeho zkušenostech s ostatními programy a návrh kvalitního GUI není jednoduchá záležitost, rozhodl jsem se provést analýzu aplikací s podobnou tématikou. Díky tomuto rozboru

se budu moci vyvarovat chybám, které ostatní vývojáři při návrhu udělali a současně získám inspiraci ke správnému využití některých grafických prvků.

Správců úkolů a obdobných aplikací existuje v dnešní době velké množství. Při rozboru těch nevýznamějších jsem se především zaměřil na rozložení hlavních prvků. Kam se obvykle pozicuje panel s úkoly, jak moc plochy by se mělo přiřadit detailu konkrétního úkolu a podobně.

2.4.2.1 Wunderlist

Wunderlist je v současnosti značně využívaná aplikace podporující širokou škálu platforem. Za cíl si klade hlavně jednoduchost a efektivitu ovládání. Na obrázku 2.7 si můžeme všimnout bočního panelu s kategoriemy. Obdobné kategorie (seznamy) má i RTM. Zbylá část obrazovky zobrazuje úkoly dané kategorie. Zde se mi nejvíce zamlouvá textové pole nad výpisem úkolů, pomocí kterého lze velmi jednoduše přidávat nové úkoly. V navigační liště je ještě vidět pole pro vyhledávání. Aplikace má minimum funkcí, je intuitivní a svůj účel splňuje velmi dobře. Bohužel zde chybí pokročilejší funkce, jako jsou tagy nebo opakování úkolů, díky kterým má RTM pro náročnější skupinu uživatelů nad ní výhodu. Aplikace je na Appstore dostupná zdarma.



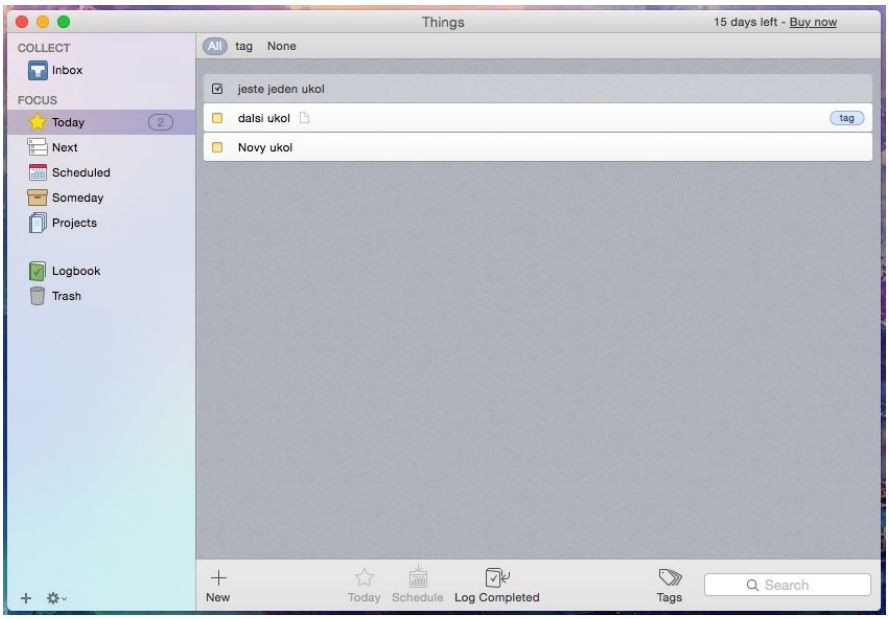
Obrázek 2.7: Wunderlist

2.4.2.2 Things

Oproti Wunderlistu je Things komerční aplikace, která stojí na App Store 49.99 €. Ke správě úkolů přidává tagy, opakovatelné úkoly a synchronizaci s kalendářem. Opět zde vidíme boční panel, kde jsou seznamy, inbox, do kterého se přidávají úkoly, jež zatím nevíme kam zařadit a položky, ve kterých

jsou úkoly roztříděny podle dne, do kdy je třeba je splnit. Na Things se mi zamlouvá využití systémového prvku sourcelist v bočním panelu. Tento prvek je hojně využíván napříč celým systémem, například v aplikaci Finder, která je denním chlebem naprosté většiny uživatelů OS X. Od verze 10.10 je pozadí sourcelistu průhledné s efektem vibrancy, který patří k stěžejním prvkům nového vzledu Yosemite.

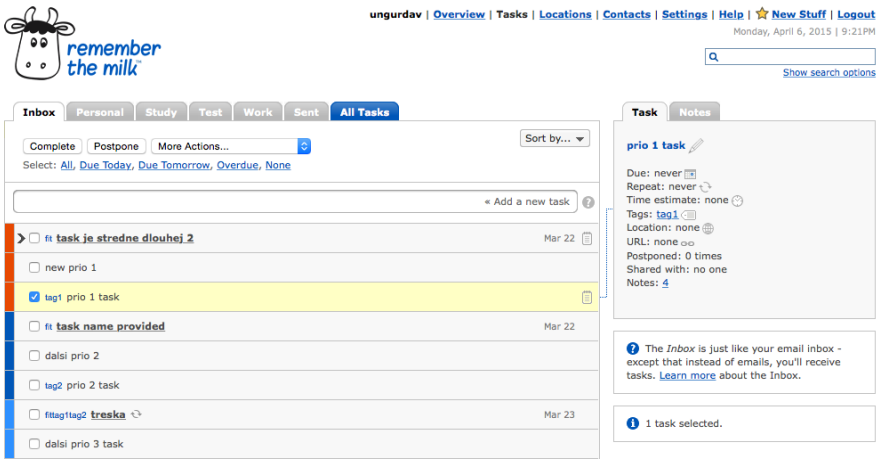
Bohužel mi při používání dělalo problém upravit již hotový úkol, a to především kvůli absenci obrazovky, která by vypisovala jeho detail. Na první pohled nebylo zřejmé, kam mám kliknout. Stejně tak i přidávání úkolu není zdaleka tak jedoduché a intuitivní, jako je tomu u Wunderlistu.



Obrázek 2.8: Things

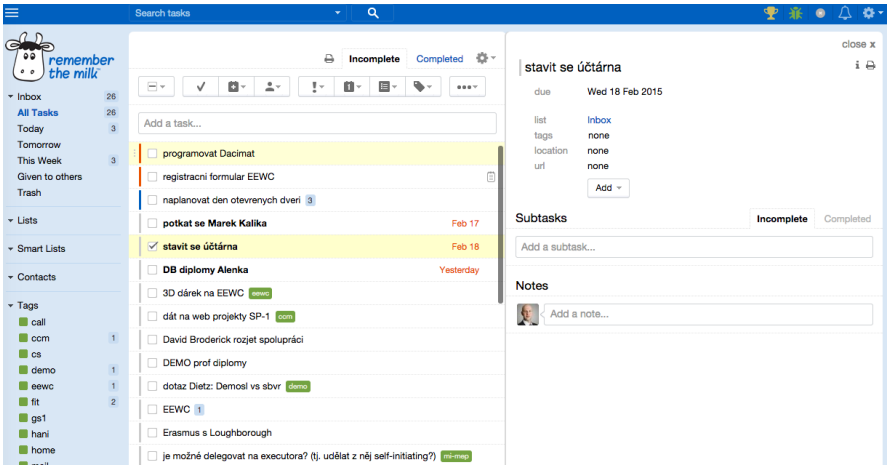
2.4.2.3 RTM Webová verze

Jako další aplikaci, kterou jsem se nechal inspirovat, uvedu webovou verzi Remember the Milk. Starší, ale zatím stále aktuální, verze je na obrázku 2.9. Zde vidíme v levé části obrazovky uložené úkoly. Nad nimi je pole pro zadávání nových a ovládání starých úkolů. Polovina funkcí pro jejich úpravu je dále na pravé straně obrazovky spolu s možností přidat k nim poznámky. Ještě si je možné všimnout seznamů/kategorií, které jsou ve starší verzi nad výpisem úkolu. Filtrování přes tagy je skoro nenalezitelné, stejně tak jako správa lokací a kontaktů.



Obrázek 2.9: Starší webová verze RTM

V novější verzi, která je zatím dostupná v beta testu pouze premium uživatelům, už tvůrci zvolili přístup podobný konkurenčním aplikacím, kde jsou seznamy a tagy na levé straně obrazovky. Prostřední část pak zabírá výpis úkolů vybraného seznamu či tagu. Pravá strana je zaměřena na zobrazení detailu konkrétního úkolu s možností přidat poznámky a dále konkrétní úkol upravovat. Pole pro přidání úkolu funguje stejně intuitivně, jako je tomu u Wunderlistu. Úprava již zadaného úkolu je zde vyřešena samostatnou pravou částí obrazovky s jeho detailem.



Obrázek 2.10: Nová webová verze RTM

2.4.2.4 Výsledek srovnání

Z napozorovaných výsledků jsem vyhodnotil, že není dobrým nápadem využívat pro aplikaci více než jedno okno. Dále se ukázalo, že boční panel s kategori-

emi je optimálním řešením dané problematiky. U Buttermilk do něj sjednotím seznamy, tagy, místa a kontakty, stejně jako je tomu u beta verze webového RTM, kterou bude Buttermilk do velké míry inspirovaný. Současní uživatelé si nové webové rozhraní vychvalují a během testování jsem si u něho nevšiml žádného výraznějšího problému. Dále je dobrým nápadem věnovat část obrazovky detailu úkolu. Panel pro zadávání úkolů se vyskytoval ve většině srovnávaných aplikací a v jediné aplikace, kde nebyl, znatelně chyběl.

Kapitola 3

Analýza a návrh

Tato kapitola se zabývá analýzou a návrhem aplikace *Buttermilk*. V první části jsou popisovány funkční a nefunkční požadavky, z kterých je navrhnut diagram případů užití a doménový model. Druhá část navazuje návrhem architektury a grafického uživatelského rozhraní na základě poznatků získaných z rešerše.

3.1 Analýza požadavků

Díky hojnému počtu vývojářů webové verze, může RTM poskytovat široké množství funkcí. Bohužel není v mých silách do práce zapracovat veškerou funkčnost webové verze. Rozhodl jsem se proto naimplementovat v prvních řadách ty nejdůležitější a nejčastěji používané funkce. Vynechal jsem pouze ty, pro které neexistuje API, nebo je uživatel provede pouze ojediněle.

3.1.1 Funkční požadavky

- zobrazení seznamů
- zobrazení úkolů v seznamech
- vytvoření úkolu
- vytvoření úkolu pomocí *Smart Add* [11]
- odstranění úkolu
- úprava úkolu (splnění, odložení, poznámka, přidání URL, přidání tagů, přiřazení lokace, změna priority)
- vyhledávání úkolů podle jména
- filtrování úkolů pomocí tagů
- procházení uložených lokací

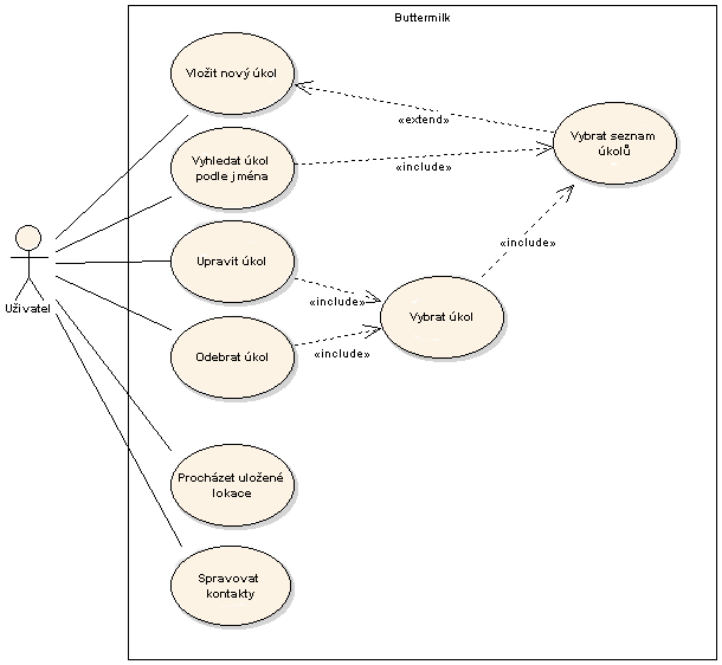
- procházení uložených kontaktů
- vytvoření úkolů přes Spotlight

3.1.2 Nefunkční požadavky

- platforma OS X 10.9 a vyšší
- využití open-source software
- efektivita a rychlost zadávání nových úkolů
- GUI v souladu s oficiálními doporučeními pro OS X

3.2 Případy užití

Při návrhu diagramu případů užití jsem vycházel z funkčních a nefunkčních požadavků.

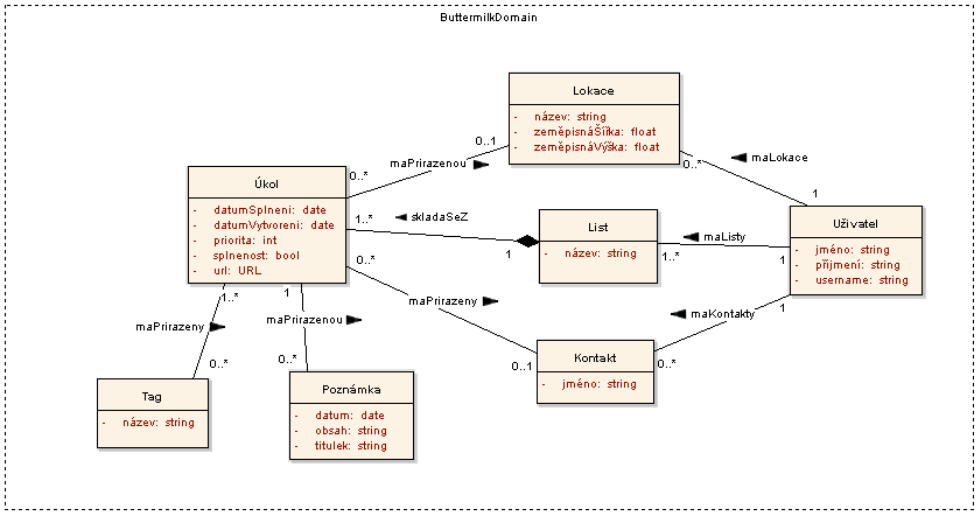


Obrázek 3.1: Diagram případů užití

Bohužel veřejné RTM API neumožňuje s lokacemi nikterak pracovat, pouze zobrazit jejich seznam. Tvorbu, úpravu a mazání kontaktů jsem do diagramu zapsal pouze jako správu kontaktů. Tyto úkony jsou zřejmé a dle mého názoru je nebylo nutné, na úkor přehlednosti diagramu, dále rozepisovat.

3.3 Doménový model

Kapitola obsahuje diagram, popis jednotlivých entit a jejich atributů.



Obrázek 3.2: Doménový model entit aplikace Buttermilk

Jak je vidět z diagramu, struktura a vztahy mezi entitami jsou následující. Přihlášený uživatel má jeden až několik seznamů s úkoly. Dále má i několik kontaktů a lokací. Ty může k úkolům přiřazovat. Příkladem by mohl být úkol *sraz s* ***Petrem*** *na* ***chatě***. K dílčím úkolům lze dále přidávat tagy a poznámky, mohou ale existovat i bez nich.

Uživatel

Entita uživatel nese informaci o celém jméně a username přihlášeného uživatele, který Buttermilk právě používá.

Tabulka 3.1: Atributy entity Uživatel

Atributy	Poznámky
jméno	Křestní jméno uživatele.
příjmení	Příjmení uživatele.
username	Přihlašovací jméno do RTM.

Kontakt

Jeden dílčí kontakt z adresáře uživatele.

Tabulka 3.2: Atributy entity Kontakt

Atributy	Poznámky
jméno	Přiřazené jméno kontaktu.

Lokace

Entita reprezentující zaměpisnou polohu a její příslušné pojmenování v aplikaci.

Tabulka 3.3: Atributy entity Lokace

Atributy	Poznámky
název	Uživatelské pojmenování lokace.
zeměpisnáŠířka	Zeměpisná šířka polohy dané lokace.
zeměpisnáVýška	Zeměpisná výška polohy dané lokace.

List (seznam)

Seznam je pojmenovaná skupina úkolů. Samostatný úkol nemůže bez seznamu existovat.

Tabulka 3.4: Atributy entity List

Atributy	Poznámky
název	Název seznamu. (vybraný uživatelem)

Úkol

Entita úkol nese informace o datu vytvoření a datu pro splnění. Dále má každý úkol svojí prioritu, kde nejprioritnější úkoly mají číslo 1. Atribut *splněnost* je příznak, který určuje, jestli už daný úkol uživatel splnil.

Tabulka 3.5: Atributy entity Úkol

Atributy	Poznámky
datumSplnění	Datum do kdy musí uživatel úkol splnit.
datumVytvoření	Datum vytvoření úkolu.
priorita	Číselná priorita 1, 2, 3.
splněnost	Příznak o tom, jestli je úkol již splněný.
url	Odkaz např. na webovou stránku.

Tag

Označení, které lze přiřadit k úkolu. Podle tagů je možné úkoly později filtrovat.

Tabulka 3.6: Atributy entity Tag

Atributy	Poznámky
název	Název (hodnota) tagu.

Poznámka

Textová poznámka k úkolu.

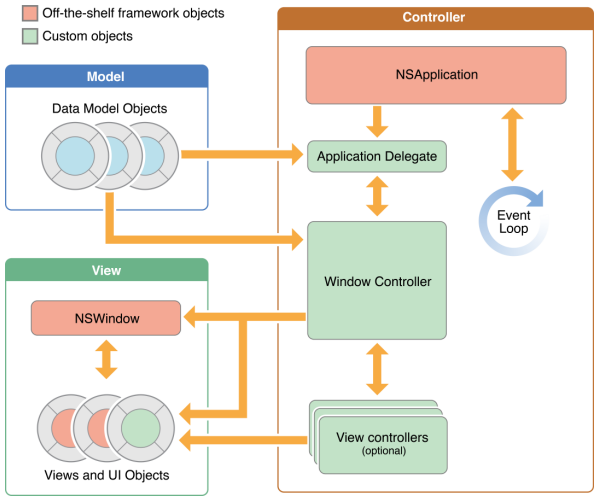
Tabulka 3.7: Atributy entity Poznámka

Atributy	Poznámky
datum	Datum přidání poznámky k úkolu.
obsah	Text poznámky.
titulek	Název poznámky.

3.4 Návrh architektury aplikace Buttermilk

Pro návrh architektury jsem zvolil Model-View-Controller, který je i podle informací z rešerše 2.3.1 nejvhodnější volbou k tvorbě aplikací pro operační systém OS X.

Framework Cocoa podporuje tvorbu aplikací s jedním, ale i více okny [12]. Návrh a práce s těmito druhy aplikací se do jisté míry odlišuje. Jednookení verze bývají ve většině případů preferované, neboť produkují více usměrněnou uživatelskou zkušenost a umožňují jednoduchou maximalizaci okna beze ztráty ostatních informací. Z výše zmíněných důvodů jsem pro aplikaci Buttermilk zvolil jednookení styl (také známý jako *shoebox*).



Obrázek 3.3: Hlavní objekty u aplikací typu shoebox [12]

Obrázek 3.3 ukazuje vztahy mezi klíčovými objekty jádra shoebox aplikací a jejich rozdělení do vrstev model, view a controller. V následující části popisuji, jak jednotlivé objekty pracují a kterými třídami je budu v aplikaci reprezentovat.

NSApplication
Vyhodnocuje příchozí události a zajišťuje interakci mezi aplikací a systémem. V naprosté většině případů se do ní nepíše vlastní kód, k tomu se používá `AppDelegate`.

AppDelegate
Objekt, který nám poskytuje možnost reagovat na změny stavu aplikace (minimalizace apod.). V Buttermilk ho budu muset rozšířit o podporu otevírání URL s vlastním schématem `bm://`.

Data Model
Objekty, které obsahují data o RTM-údajích uživatele. V aplikaci použiji `NSRTMTask`, `NSRTMList` a další. Modelovou vrstvu z naprosté většiny zajistí knihovna rtm2cocoa 4.2.

Window controller
Window controller je zodpovědný za správu jednoho okna aplikace. Při implementaci budu muset hlavní window controller oddědit na `BMMainWindowController` a přidat do něj funkce, které budou reagovat na změny ve vyhledávání, jelikož vlastníkem vyhledávacího pole je v případě Buttermilk právě window controller.

Window objekt
Reprezentuje jedno konkrétní okno. Lze mu nastavit odlišné styly, vlastnosti a výška záhlaví. Toho v aplikaci dosáhnu pomocí `BMMainWindowControlleru`.

View controller
Objekty typu view controller se používají pro rozdělení práce window controlleru. Spravují jedno konkrétní view a jeho životní cyklus. Buttermilk bude používat `BMMainSplitViewController`, `BMSideMenuViewController`, `BMLocationsViewController`, `BMTaskSelectionViewController`, `BMTaskDetailViewController` a mnoho dalších. Všechny budou dědit od předka `BMViewController`, který dědí od `NSViewController` (základní objekt z Cocoa). Význam těchto objektů je zřejmý z názvu. Více informací a náhled rozdělení controller objektů se nachází v sekci 4.1.1.

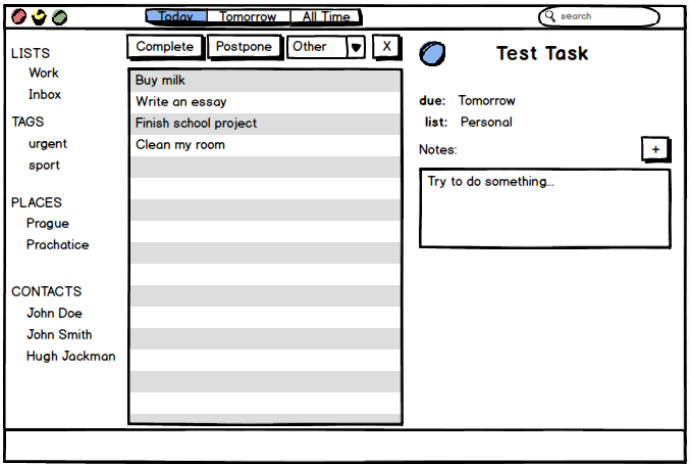
View objekt
Objekty view se vykreslují v obdélníkovém regionu window. Zobrazují hlavní ovládací a zobrazovací objekty OS X aplikací. V Buttermilk bude

zapotřebí oddědit `NSTableCellView` (view, které zobrazuje obsah buňky v tabulce), abych pomocí odděděného `BMTaskTableCellView` mohl docílit vzhledu buňky, který je známý z webového klienta.

Control objekt
Standartní systémové ovládací prvky jako je textové pole nebo tlačítko.

3.5 Návrh GUI

Při navrhování jsem využil informacích získaných z rešerše. Největším informačním přínosem pro mne bylo srovnání konkurenčních aplikací, jejich kladů a záporů. Hlavní předlohou pro vzhled aplikace bude nová verze webového klienta RTM. Jedním z mála prvků, kterými se bude má práce od webové verze lišit, je oddělení shlukovacích kategorií `Due Today`, `Due Tomorrow` a `All Tasks`, které přesunu do lišty okna a vytvořím takto další filtr. Umožní se tím třídění úkolů podle několika kritérií najednou, což u webové verze není možné. Pro vizuální přiblížení přikládám návrh rozhraní před implementací.



Obrázek 3.4: Návrh rozhraní

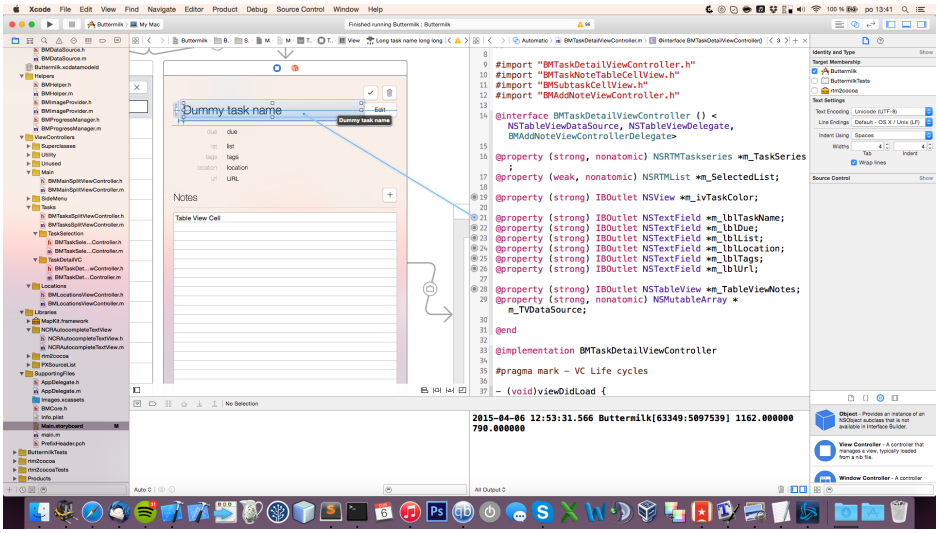
Kapitola 4

Realizace

Tato kapitola bude věnována popisu implementace aplikace Buttermilk. Zejména bych se v ní chtěl zaměřit na problémy a výzvy, na které jsem při realizaci narazil.

4.1 Vývojové prostředí Xcode

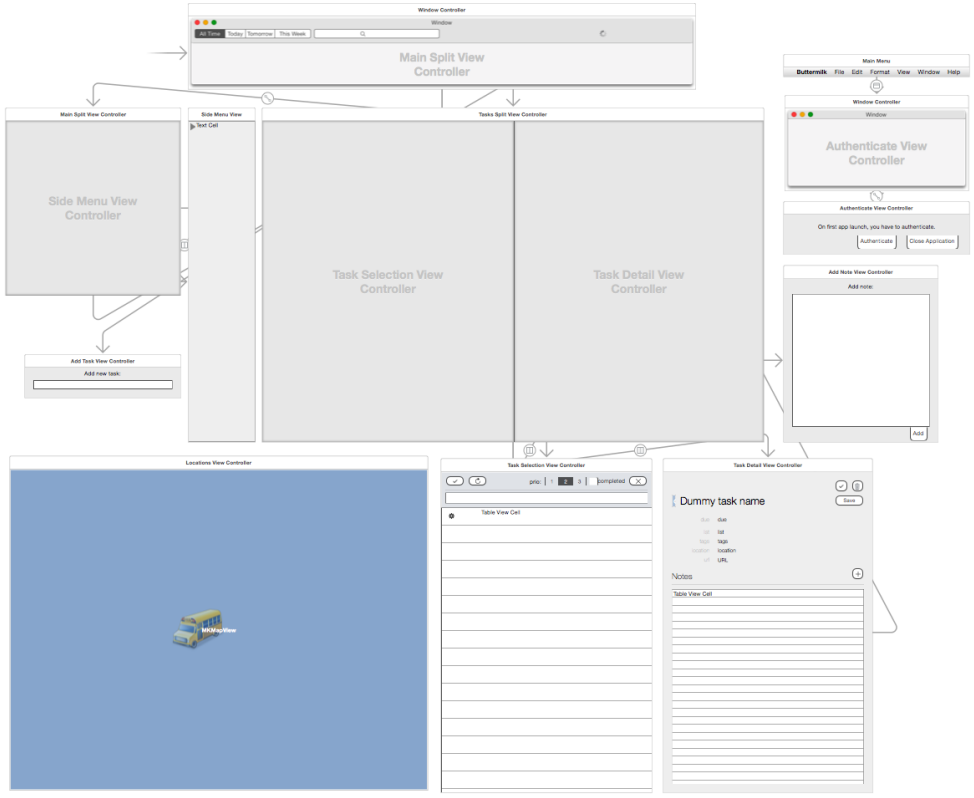
Při vývoji jsem používal vývojové prostředí Xcode, která je zdarma ke stažení na stránkách společnosti Apple. Xcode kromě editoru zdrojového kódu obsahuje i Interface Builder editor, ve kterém může vývojář jednoduše systémem WYSIWYG navrhovat uživatelské rozhraní jednotlivých obrazovek. Grafické prvky z IB je možné pomocí tzv. IBAction a IBOutlet propojit se zdrojovým kódem, jako je tomu na obrázku 4.1.



Obrázek 4.1: Demonstrace IBOutlet

4.1.1 Storyboard

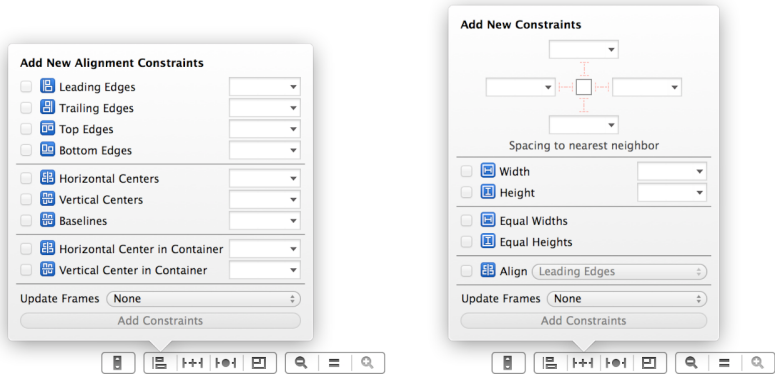
Aplikace jsou tvořeny obrazovkami, okny a přechody mezi nimi. Tyto přechody jsou většinou napojeny na stisknutí tlačítka nebo vykonání jiné podobné akce. Do verze Xcode 6 bylo možné jednotlivé obrazovky namodelovat zvlášť pomocí editoru Interface Builder a přechody mezi nimi ručně naprogramovat v kódu. Od SDK 10.10 má vývojář možnost tyto vztahy a přechody namodelovat ve Storyboard, kde vidí všechny obrazovky najednou a může mezi nimi tvořit zmíněné závislosti a upravovat současně jejich vzhled. Vše se poté uloží do souboru se strukturou XML. Tento přístup značně zjednodušuje tvorbu moderních aplikací a ubírá vývojáři nutnost psaní generovatelného kódu. V náhledu souboru Storyboard je dále na první pohled zřejmý průchod aplikací, což například zjednodušuje zapracování se novým kolegům do projektu. Bohužel u větších projektů bývají soubory Storyboard často nepřehledné, ale to lze dnes vyřešit jejich rozdělením do menších celků. Při tvorbě aplikace Buttermilk jsem si vystačil s jedním souborem, jeho náhled je na obrázku 4.2.



Obrázek 4.2: Náhled souboru Main.storyboard

4.1.2 Autolayout

Autolayout je systém, který vznikl za účelem vyřešení problému, kde fixně napozicované prvky uživatelského rozhraní vypadaly na různých zařízeních se systémem iOS odlišně. S tímto novým přístupem má každý prvek dvě a více omezujících vlastností nazývaných constraint, které určují pozici, šířku a výšku daného prvku v závislosti na ostatních prvcích obrazovky. Autolayout je od novějších verzí Xcode zabudovaný přímo v editoru Interface Builder, ale lze jej používat i v kódu, a to s pomocí objektů třídy `NSOutolayoutConstraint`.



Obrázek 4.3: Nabídky s nastavením podmínek pro Autolayout [13]

Aplikace Buttermilk podporuje různé velikosti okna. Je možné ji používat ve stavu, při kterém zabírá nejméně místa nebo maximalizovanou, ale i ve všech stavech mezi těmito dvěma. Pro správné překreslování okna jsem použil právě Autolayout.

Menu s výběrem seznamů, tagů a lokací má nastavenou fixní šířku, stejně tak jako pravá třetina zobrazující detail vybraného úkolu. Tyto části obrazovky by nikterak nebenefitovaly z většího prostoru, a tak je možné tento prostor přidělit ostatním prvkům, především tabulce s výpisem úkolů. Ta může přidaný prostor využít v případě delšího názvu a většího počtu tagů u jednotlivých položek.

Dále si okno vynucuje minimální rozměry. S rozlišením `1162 x 768` se vejde i na display nejmenšího Macbooku Air 11. Omezení pro minimální velikost jsem zavedl, protože při měnění velikosti chci většinou obrazovku zmenšit na co nejmenší možnou velikost, při které je ale zároveň schopna vypisovat veškeré potřebné informace.

4.2 Knihovny třetích stran

Při tvorbě aplikací pro systémy iOS a OS X je zcela běžné používat knihovny třetích stran. S jejich pomocí se vyhneme znovuvynalézání kola nebo nám jen

poskytnou určitý wrapper nad některými zkostnatělými konstrukty Cocoa-frameworku, čímž nám zpříjemní práci. Typickým příkladem bývají kategorie pro třídy z knihoven AppKit a Foundation. Ve své práci jich používám hnedka několik. Zde jsem sepsal seznam těch nejpodstatnějších.

- **rtm2cocoa** – Největší knihovnou, kterou jsem při implementaci Buttermilk využil, je bezesporu rtm2cocoa [14]. Jedná se o implementaci RTM API v programovacím jazyku Objective-C a to včetně autentifikačního systému a všech metod pro práci s objekty RTM.
- **PXSourceList** – PXSourcelist je wrapper pro systémový prvek source-list (boční panel) známý z aplikací jako je Finder nebo iTunes.
- **NCRAutocompleteTextView** – Jak již název napovídá, jde o podtřídu `NSTextView`, která mimo jiné přidává běžnému textovému poli možnost automatického doplňování textu. Detaily implementace naleznete v kapitole 4.7.
- **Excelsior** – Zajišťuje maršalování příchozích dat z formátu XML na Objective-C objekty.
- **NSDate+Utilities** – Kategorie pro systémovou třídu `NSDate`, která rozšiřuje její rozhraní o funkce spojené s prací s časem.
- **NSString+PercentEscape** – Kategorie pro třídu `NSString`. Umožňuje z řetězce UTF-8 vytvořit řetězec, který je možné použít jako parametr v adresách URL.

4.3 Dynamická výška řádků v tabulce

Od základu jsou všechny buňky v tabulce stejně velké. Pokud bychom chtěli, aby každá buňka měla jinou výšku, dostaneme se do problému, který je podobný otázce o vejci a slepici. Tabulka chce znát výšku řádky, protože to rozhoduje, kde bude daná buňka napozicovaná, ale abychom mohli spočítat výšku dané buňky, musela by buňka už existovat.

Dynamickou výšku buněk využívám v aplikaci jen na jednom místě, a to ve výpisu komentářů v detailech úkolu. Jelikož se v těchto buňkách nachází pouze text, stačilo ke spočítání výšky celé buňky použít funkci z listingu 4.1.

Listing 4.1: Funkce pro přibližné spočítání velikosti buňky

```
1 - (CGSize)frameForText:(NSString *)text
2           sizeWithFont:(NSFont *)font
3     constrainedToSize:(CGSize)size {
4    /* nejrive vytvorit mapu s atributy pro vypisovany text */
5     NSDictionary *attrDict = @{NSFontAttributeName : font};
6
7    /* tato funkce z knihovny AppKit spocita na zaklade rozmeru size, parametru
```

```
8       pro vykreslovani retezcu a nasich atributu, jaka je minimalni velikost
9       obdelniku, kterym je mozne tento retezec "obalit" */
10     CGRect frame =
11     [text boundingRectWithSize:size
12                        options:(NSStringDrawingUsesLineFragmentOrigin |
13                                 NSStringDrawingUsesFontLeading)
14                     attributes:attrDict];
15    /* CGRect ma size a origin */
16     return frame.size;
17 }
```

Ta dokáže spočítat výšku obdélníku, který je potřeba pro obalení textu v závislosti na šířce daného obdélníku, fontu a délce zadaného textu.

Spočítanou výšku vrátím spolu s odsazením z metody delegáta tabulky `tableView: heightOfRow:` z listingu 4.2.

Listing 4.2: Metoda delegáta tabulky určující výšku buňky

```
1  - (CGFloat)tableView:(NSTableView *)tableView heightOfRow:(NSInteger)row {
2     /* pretypovani je potrebne - NSArray vraci vzdy objekty typu NSObject */
3      NSString *text = [((NSRTMNote *)self.m_TVDataSource[row]) getValue];
4
5     /* spocitani obdelniku potrebneho pro obaleni textu */
6      CGSize size = [self frameForText:text
7                          sizeWithFont:nil
8                     constrainedToSize:CGSizeMake(CELL_WIDTH, CGFLOAT_MAX)];
9
10    /* vzdy vratit alespon minimalni velikost radky */
11     return MAX(size.height + 2*CELL_PADDING, MIN_CELL_HEIGHT);
12 }
```

Tabulka bude nyní vědět výšku dané buňky, podle ní ji vytvoří, a obsah se pomocí autolayoutu uvnitř správně rozmístí.

4.4 Příklady využití a dodržování návrhových vzorů

Návrhový vzor je prostředek abstrakce používaný při objektově orientovaném vývoji softwaru [5]. Jedná se o šablony s postupem řešení často se opakujících problémů a situací, které mohou nastat při vývoji software. Adaptováním těchto návrhových vzorů v kódu přispíváme rozšiřitelnosti a znovupoužitelnosti našich a knihovních tříd.

4.4.1 MVC a Delegát

Principu a výhodám architekturního vzoru Model-View-Controller se detailněji věnuji v kapitole 2.3.1. Jak jeho implementace vypadá v praxi nám pomohou objasnit následující kusy zdrojového kódu.

Listing 4.3: Demonstrace MVC v Cocoa

```
1  /* soubor BMTaskDetailViewController.m */
2  #import "BMTaskDetailViewController.h"
3  /* zkracena verze deklarovaneho rozhrani tridy BMTaskDetailViewController */
4  @interface BMTaskDetailViewController () <NSTableViewDataSource,
5                                            NSTableViewDelegate>
6      @property (strong, nonatomic) IBOutlet NSTableView * m_TableViewNotes;
7      @property (strong, nonatomic) NSMutableArray *       m_TVDataSource;
8  @end
9
10 #pragma mark - TableView datasource
11 - (NSView *)tableView:(NSTableView *)tableView
12    viewForTableColumn:(NSTableColumn *)tableColumn
13                   row:(NSInteger)row {
14    /* NSRTMNote je modelovy objekt nesouci informace o poznamce k ukolu */
15     NSRTMNote *note = [self.m_TVDataSource objectAtIndex:row];
16     NSString *idntfr = NSStringFromClass([BMTaskNoteTableCellView class]);
17
18    /* vytvoreni view objektu bunky tabulky */
19     BMTaskNoteTableCellView *cell =  [tableView makeViewWithIdentifier:idntfr
20                                                                  owner:self];
21    /* nastaveni obsahu bunky z modeloveho objektu note */
22     [cell setNote:note];
23     return cell;
24 }
25
26 #pragma mark - TableView delegate
27 - (void)tableViewSelectionDidChange:(NSNotification *)notification {
28     NSInteger row = [[notification object] selectedRow];
29     NSLog(@"selected row %zd", row);
30 }
31
32 /* soubor BMTaskNoteTableCellView.m */
33 - (void)setNote:(NSRTMNote *)note {
34    /* nastaveni obsahu GUI prvkum z hodnot model objektu note */
35     self.m_tvText.string = [note getValue];
36     self.m_lblTime.stringValue = [[note getCreated] shortDateString];
37    /* metoda skonci a bunka (view) si neulozila referenci na note (model) */
38 }
```

Zde třída BMTaskDetailViewController (controller) vlastní (ř. 6) tabulku NSTableView (objekt view). Controller implementuje dva protokoly, `NSTableViewDataSource` (ř. 4) a `NSTableViewDelegate` (ř. 5). V editoru Interface Builder má tabulka nastavené, že její dataSource a delegát je právě tento controller. Tímto mechanismem controller tabulce poskytuje modelové objekty (pole `m_TVDataSource`) a současně jí deleguje.

V metodě `tableView: viewForTableColumn: row:` (ř. 11) controller vytváří buňky tabulky za použití modelových objektů. Jejich vzhled je pak na základě těchto modelových objektů upraven voláním `[cell setNote:note];` (ř. 22). Jak je vidět, objekty view si nedrží referenci na objekty modelu a naopak. Veškerou komunikaci mezi nimi zajišťuje controller.

4.4.2 Observer

Pro správné doržování architekturního vzoru MVC je využíván návrhový vzor delegát. Ten je ale limitován svojí násobností vztahů 1:1. Pokud bychom chtěli, aby na změnu stavu mohlo zareagovat více objektů najednou, musíme využít návrhový vzor observer. Pro jeho implementaci jsem využil notifikace a `NSNotificationCenter`. Dobrým příkladem využití tohoto vzoru v aplikaci Buttermilk je práce s vyhledávacím polem 4.4.

Při změně obsahu vyhledávání se spustí metoda `searchFieldTextChanged`, ve které zašlu notifikaci (ř. 4) s aktuálně vyhledávaným obsahem. Pomocí notifikačního centra se pro odběr této notifikace nezávisle na sobě přihlásí jak `BMMainWindowController`, tak `BMLocationViewController`. Každý z nich potom na změnu ve vyhledávání reaguje jinak. `BMMainWindowController` filtruje seznam úkolů, `BMLocationViewController` na základě zadaného textu vyhledává v mapě.

Listing 4.4: Návrhový vzor observer v aplikaci Buttermilk

```
1  /* soubor BMMainWindowController.m */
2  #pragma mark - Search
3  - (IBAction)searchFieldTextChanged:(NSSearchField *)sender {
4      NSNotificationCenter * ntfCenter = [NSNotificationCenter defaultCenter];
5      [ntfCenter postNotificationName:kSearchNotification
6                                object:nil
7                              userInfo:@{@"value" : sender.stringValue}];
8  }
9  /* soubor BMTaskSelectionViewController.m */
10 - (void)registerForNotifications { /* volano z viewWillAppear */
11     NSNotificationCenter *ntfCenter = [NSNotificationCenter defaultCenter];
12     [ntfCenter addObserver:self
13         selector:@selector(searchedTextChanged:) /* selektor, ktery se vola */
14                     name:kSearchNotification      /* pri obdrzeni notifikace */
15                   object:nil];
16 }
17 - (void)searchedTextChanged:(NSNotification *)notification {
18     /* prislusna reakce na zmenu ve vyhledavani */
19 }
20 /* soubor BMLocationViewController.m */
21 - (void)registerForNotifications {
22    /* identicka implementace jako v BMTaskSelectionViewController.m*/
23 }
```

4.4.3 Singleton

Návrhový vzor singleton funguje na principu existence pouze jedné instance třídy [15] v daný okamžik. Většinou se při implementaci využívá principu lazy inicializace, kdy se objekt vytvoří a nainicializuje až při prvotním zavolání getter metody. V Cocoa je tento návrhový vzor využíván na mnoha místech. Typickými příklady jsou `[NSFileManager defaultManager]`, `[NSApplication sharedApplication]` nebo `[NSNotificationCenter defaultCenter]`.

V práci jsem využil singleton pro třídu `BMDataSource`, která zajišťuje přístup k datům z RTM API. Největši nebezpečí vzoru singleton nastává ve vícevláknovém prostředí, kdy se může podařit objekt vytvořit více vláknům najednou. Tento nedostatek lze vyřešit konstruktem `dispatch_once` z technologie GCD, který zajistí atomicitu volání bloku, jenž se nachází uvnitř.

Listing 4.5: Šablona pro tvorbu singletonu v Cocoa

```
1 + (YourClass *)sharedInstance {
2     static dispatch_once_t pred = 0;
3     __strong static id _sharedInstance = nil;
4     dispatch_once(&pred, ^{
5         _sharedInstance = [[YourClass alloc] init];
6     });
7
8     return (YourClass *)_sharedInstance;
9 }
```

4.5 Úpravy chování objektů z frameworku AppKit

Ne vždy vyhovují objekty z knihoven Cocoa požadavkům, které na ně naše aplikace klade. Možnost jejich úpravy je sice nadstandartní, přesto někdy nastane situace, kdy je nutné, si určité funkce dopsat ručně. Programovací jazyk Objective-C nám poskytuje několik způsobů, kterými toho můžeme docílit:

- dědičnost
- kategorie
- method swizzling

Každý z těchto přístupů má svoje pro a proti a využívá se v jiných situacích.

4.5.1 Dědění

Dědění jsem použil například při oddědění třídy `NSTableRowView` třídou `BMTaskTableRowView`, kdy jsem přepsal její metodu `drawSelectionInRect:` tak, aby umožňovala uživatelské vykreslování efektu spojeného s označením buňky. Zde je dědění správnou cestou, jelikož nepotřebuji, a nechci toto chování u všech výskytů `NSTableRowView`.

4.5.2 Kategorie

Kategorie a jejich výhody jsem detailněji popsal v kapitole 2.2.5. Jak jsem již zmínil, využívají se především pro přidávání menších funkcí do kompletních tříd, kde dědění neni vyžadováno a vedlo by k zbytečné refaktorizaci již napsaného zdrojového kódu. Příkladem využití kategorií v aplikaci Buttermilk

je kategorie `NSColor+BMPriorityColors`. Jedná se o kategorii pro systémovou třídu `NSColor` z frameworku `Foundation`. Tato kategorie přidává třídě `NSColor` mimo jiné i metodu `bmPriority1Color`, která vrací objekt s barvou používanou na místech, kde se pracuje s úkoly s nejvyšší prioritou. Díky ní můžeme pracovat s vlastní barvou úplně stejným způsobem, jako se pracuje se základními barvami, a to voláním z následujícího kusu zdrojového kódu.

Listing 4.6: Využití kategorií pro rozšířeni systémové třídy `NSColor`

```
1 NSColor *standartColor = [NSColor redColor]; // systemova verze
2 NSColor *ourColor= [NSColor bmPriority1Color]; // verze z~pridane kategorie
```

Další výhodou je, že v případě nutnosti změny barvy pro prioritu jedna stačí udělat změna pouze na jednom místě v kódu.

4.5.3 Method swizzling

Method swizzling [16] je přístup, při kterém se, zjednodušeně řečeno, vykonání jedné metody nahradí metodou druhou. Swizzling využívá faktu, že v ObjC se posílají objektům zprávy. Runtime si drží tabulku se jmény selektorů a ukazateli do paměti na implementaci příslušných metod. Kódem z ukázky 4.7 můžeme tyto ukazatele na metody mezi sebou prohodit. Tím lze zcela změnit chování již existujících tříd. Nutno podotknout, že tento postup by se neměl používat ve standartních situacích a slouží především pro opravování chyb ve zdrojových kódech Cocoa náhradou za vlastní verzí implementace nefunkční metody. Při implementaci aplikace Buttermilk nebylo zapotřebí tuto metodu použít.

Listing 4.7: Ukázka method swizzling

```
1  #import <objc/runtime.h>
2  + (void)load {
3      static dispatch_once_t onceToken;
4      /* dispatch_once zajisti, ze kod se zavola jen jednou v metode load */
5      dispatch_once(&onceToken, ^{
6          Class class = [self class];
7          SEL originalSel = @selector(viewDidLoad:);
8          SEL swizzledSel = @selector(my_viewDidLoad:);
9         /* zde ziskame implementace metod podle nazvu selektoru */
10         Method originalMethod = class_getInstanceMethod(class, originalSel);
11         Method swizzledMethod = class_getInstanceMethod(class, swizzledSel);
12        /* zamenime implementace funkci method_exchangeImplementations */
13         method_exchangeImplementations(originalMethod, swizzledMethod);
14     });
15 }
```

Kód se do tříd zavádí pomocí kategorií a měla by se dodržovat dvě hlavní pravidla. Spouštět kód pro nahrazení pouze v metodě `load`, která je volaná, když je třída poprvě nahraná do runtimu. A použít pro toto volání konstrukt `dispatch_once` používaný také u singletonu.

4.6 Úprava knihovny pro práci s RTM API

Celá knihovna rtm2cocoa [14] je napsaná synchronně a tudíž nepodporuje současný způsob práce se sítí. Všechna volání byla blokující, a docházelo kvůli nim k zástavě grafického rozhraní. Problém jsem vyřešil s použitím technologie Grand Central Dispatch [17], kde jsem volání obalil do `dyspatch_async` bloků, které se volají na pozadí. Po jejich vykonání se zavolá completion blok (uzávěr/lambda funkce), který přidám vlastní funkci jako parametr. V tomto completion bloku si poté zažádám o přístup k hlavnímu vláknu, kde se provede překreslení grafického prostředí.

Pro ukázku přikládám ukázku metody pro práci s úkoly a jejího volání.

Listing 4.8: Ukázka asynchronní práce s funkcemi rtm2cocoa

```
/* soubor BMDataSource.m */
typedef void (^BMCompletion)(NSRTMList *list);
- (void)deleteTaskWithLocator:(NSRTMLocator *)locator
            andCompletion:(BMCompletion)completion {
   /* napsat informace o probihajici ukolu do progress view aplikace */
    [[BMProgressManager sharedInstance]
      startTaskWithDescription:[NSString stringWithFormat:@"deleting task %@",
                                                 locator.getTaskId]];

    dispatch_queue_t queue =
       dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0ul);
    dispatch_async(queue, ^{
       /* volani rtm2cocoa, ktere smaze ukol */
        NSRTMList *list = [self.rtmController deleteTask:locator];

        [[BMProgressManager sharedInstance] taskEndedWithStatus:list != nil];
       /* po splneni se zavola completion block */
        completion(list);
    });
}
/* ukazka volani teto funkce, vsimnete si parametru completion (blok) */
[[BMDataSource sharedInstance] deleteTaskWithLocator:locator
                                       andCompletion:^(NSRTMList *list) {
    if (list) {
       /* dispatch_get_main_queue() zazada o pristup na hlavni vlakno */
        dispatch_async(dispatch_get_main_queue(), ^{
            self.m_SelectedList = list;
            [self updateTableView]; /* prekresli GUI */
         });
    } else {
        /* reakce na neuspech */
    }
 }];
```

4.7 Autocomplete

Jednou ze stěžejních a velice oblíbených funkcí RTM, kterou ocení zkušení uživatelé a mnohonásobně jim zjednodušuje práci, je `Smart Add` [11]. Pomocí ní je možné využít pole pro přidávání úkolů k vykonání několika procesů najednou. Například je možné vytvořit nový úkol a současně mu přiřadit prioritu, tagy a lokaci.

Stačí začít psát a při zadávání úkolu použít jeden z klíčových znaků pro přidání vlastnosti. Těmi jsou:

– přidávání tagů

^ – datum pro splnění

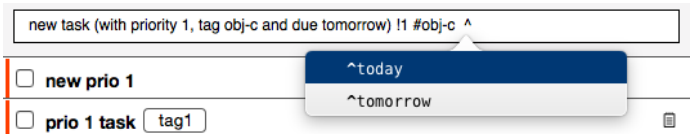
@ – místo

! – priorita

= – očekávaná doba pro splnění

***** – opakování

Příkazem ze snímku 4.4 lze jednoduše, přímo v aplikaci Buttermilk, vytvořit jedním příkazem úkol s prioritou 1, tagem obj-c a datem pro splnění na aktuální den.



Obrázek 4.4: Ukázka funkce autocomplete

Pro implementaci této funkčnosti bylo zapotřebí obohatit systémové `NSTextView` o možnost doplňování textu. K tomu jsem použil knihovnu `NCRAutocompleteTextView`, která od něj dědí a zobrazování napovídaného textu řeší pomocí systémových tabulek `NSTableView`. Dále bylo zapotřebí implementovat metody protokolu `NCRAutocompleteTableViewDelegate` a stát se delegátem nového textview. Z těchto metod byla nejpodstatnější metoda z ukázky 4.9.

4.8 Rozšíření a integrace do OS X

Jelikož jedním z cílů práce bylo zaměření na efektivitu používání a ovládání, rozhodl jsem se do aplikace přidat dvě rozšíření, které zvětší produktivitu práce a docílí se jimi větší integrace do samotného operačního systému.

Listing 4.9: Metoda z `NCRAutocompleteTableViewDelegate`

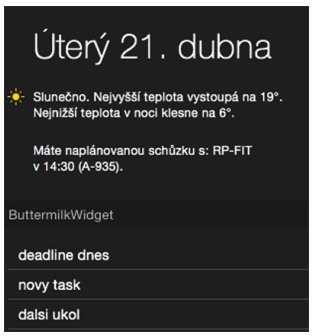
```
- (NSArray *)textView:(NSTextView *)textView
         completions:(NSArray *)words
  forPartialWordRange:(NSRange)charRange
  indexOfSelectedItem:(NSInteger *)index {
    NSMutableArray *matches = @[].mutableCopy;
    NSString *tvStr = [textView string];

   /* vyzkouset jestli napsany retezec neni podretezcem nektereho retezce
   z pole doplnovanych slov (vytvarena na zacatku z uzivatelskych dat) */
    for (NSString *string in self.m_Wordlist) {
        if ([string rangeOfString:[tvStr substringWithRange:charRange]
             options:NSAnchoredSearch | NSCaseInsensitiveSearch
             range:NSMakeRange(0, [string length])].location != NSNotFound) {
            [matches addObject:string];
        }
    }
    [matches sortUsingSelector:@selector(compare:)]; /* seradit */
    return matches; /* tyto budou zobrazeny v tabulce nabizenych slov */
}
```

4.8.1 Widget v notifikačním centru

Notifikační centrum přibylo do OS X teprve ve verzi 10.8. Slouží pro rychlý náhled aktuálního stavu aplikací (aplikace Akcie, Počasí) nebo nadcházejících událostí (Kalendář). Uživatelská rozšíření (widget) je do centra možné vytvářet až od verze 10.10 Yosemite, která je k dnešnímu dni nejnovější. Jedná se tedy o technologickou novinku, alespoň co se OS X týče. Naprostá většina aplikací tímto rozšířením zatím nedisponuje a Buttermilk je tím pádem mezi nimi výjimkou.

Pro jednoduchost a přehlednost jsem se rozšíření rozhodl implementovat pouze jako rychlý přehled dnešních úkolů. Větší množství informací by bylo zbytečně rušivé a porušovalo by principy notifikačního centra. Náhled, jak rozšíření pro Buttermilk vypadá, je na obrázku 4.5.



Obrázek 4.5: Ukázka widgetu v notifikačním centru

V projektu jsem musel zařídit nový cíl pro kompilaci, kde nám Xcode umožní vybrat si mezi aktuálně dostupnými druhy rozšíření. Widget funguje jako samostatný program a je na základní aplikaci nezávislý. Z toho plyne hlavní problém – komunikace a výměna dat mezi těmito dvěma aplikacemi. K jeho vyřešení jsem zvolil přístup, kdy obě aplikace budou v jedné společné app group `group.ungurdav.ButtermilkSharingDefaults` a budou komunikovat přes sdílené `NSUserDefaults`, které je dostupné právě díky jednoté skupině.

Listing 4.10: Uložení dat do společných `NSUserDefaults`

```
- (void)updateWidgetContent {
   /* misto [NSUserDefaults standartDefaults] vyuzije spolecne defaults*/
    NSUserDefaults *sharedDefaults = [[NSUserDefaults alloc]
          initWithSuiteName:@"group.ungurdav.ButtermilkSharingDefaults"];

    NSArray *todaysTasks = [self getTodaysTasks];
    [sharedDefaults setObject:todaysTasks forKey:@"tasks"];

    [sharedDefaults synchronize];  /* vynuti zapis na disk */
}
```

Získání dat ve widgetu potom provedeme analogicky přes stejné user defaults.

4.8.2 Plug-in pro Spotlight

Spotlight je vyhledávací systém zabudovaný do operačního systému OS X. Kromě vyhledávání souborů na disku umožňuje i vyhledávat na wikipedii, spouštět aplikace, otevřít záložku z webového prohlížeče nebo spočítat matematické výrazy. Díky těmto vlastnostem se Spotlight stalo velice populárním a denně používaným nástrojem většiny uživatelů.

Jak jsem již zmínil v předchozí sekci, rozšíření jsou novinkou OS X 10.10. Na Spotlight se bohužel zatím nedostalo, a tak v současné době neexistuje oficiálně podporovaný způsob, jak změnit, co bude Spotlight zobrazovat.

4.8.2.1 Flashlight

Flashlight [18] je opensource aplikace rozšiřující Spotlight o systém zásuvných modulů. Poskytuje neoficiální API pro psaní těchto plug-inů v jazyku Python. Pro vytvoření vlastního modulu stačilo vytvořit strukturu `bm.bundle`

```
bm.bundle
├── examples.txt
├── info.json
└── plugin.py
```

, kde soubor `examples.txt` definuje, jaké příkazy bude Spotlight přiřazovat k našemu plug-inu. Vybral jsem příkaz `bm` a `buttermilk`. Soubor

`info.json` obsahuje meta data a `plugin.py` samotný kód modulu, který je možný vidět na následujícím úryvku zdrojového kódu.

Listing 4.11: Obsah souboru `plugin.py`

```
1  def results(fields, original_query):
2    msg = fields['~message']
3    return {
4      "title": "Create new task: '{0}'".format(msg),
5      "run_args": [msg]
6    }
7
8  def run(msg):
9    import os, pipes
10   os.system("open bm://'{0}'".format(msg))
```

Nejpodstatnější část skriptu je řádka 10, která má na starosti spuštění příkazu `open` pomocí terminálu. Ten spustí URL `bm://hodnota_url` pomocí aplikace Buttermilk. Aby tento mechanismus fungoval, musíme aplikaci registrovat k otevírání URL tohoto typu. Toho lze docílit rozšířením souboru s metadaty `info.plist` o následující tag.

Listing 4.12: Úprava `info.plist` pro otevírání vlastních URL schémat

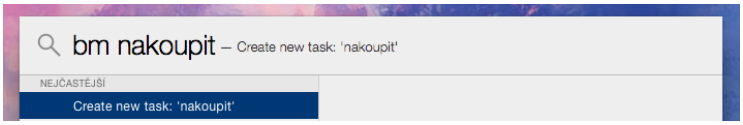
```
1  <key>CFBundleURLTypes</key>
2  <array>
3      <dict>
4          <key>CFBundleURLName</key>
5          <string>Buttermilk</string>
6          <key>CFBundleURLSchemes</key>
7          <array>
8              <string>bm</string>
9          </array>
10     </dict>
11 </array>
```

Jelikož mazání nebo úprava úkolů by byla příliš komplikovaná a vyžadovala by překlad názvu úkolu na jeho id, rozhodl jsem se implementovat pouze přidávání. Navíc spotlight, dle mého názoru, chce uživatel používat pouze pro rychlé přidávání úkolů, ke kterým se může později vrátit. Pro komplikovanější operace je zde hlavní aplikace.

Zpracování požadavku na straně Buttermilk vypadá následovně.

1. V metodě `handleURLEvent: withReplyEvent:` naparsuji obsah URL.
2. Assynchronně zavolám funkci pro přidání nového úkolu se získaným názvem
3. Po skončení funkce přidám přes notifikační centrum notifikaci o potřebě překreslení GUI
4. Notifikaci si odchytí příslušný controller a zareaguje na ni (pokud je zobrazený)

Pro zadání nového úkolu stačí do Spotlight napsat příkaz *bm nakrmit rybu* nebo *buttermilk nakoupit*. Automaticky se vytvoří nové úkoly *nakrmit rybu* respektive *nakoupit* a nahrají se do účtu RTM. Ukázka zadání je vidět na obrázku 4.6.



Obrázek 4.6: Ukázka rozšíření pro Spotlight

Kapitola 5

Testování

Testování je jednou ze základních fází cyklu vývoje software. V této kapitole proberu všechny testy, které jsem při vývoji Buttermilk použil.

S ohledem na fakt, že cílem vývoje byla mimo jiné uživatelská přívětivost a jednoduché ovládání, vytvořil jsem vedle unit testů i usability testy uživatelského rozhraní reálnými uživateli.

5.1 Unit testy

Pro testování aplikace jsem použil testovací framework XCTest [19], který je doporučován společností Apple pro testování OS X a iOS aplikací. Dokazuje to jeho integrace do vývojového prostředí XCode a automatické vytvoření nového balíčku s testy při prvotním vytvoření projektu.

Schéma průběhu testů je následující. Před spuštěním se zavolá metoda `setUp`, která připravý testovací prostředí. Po ní jsou spuštěné konkrétní testy. Všechny testy by měly mít formát názvu ve stylu `testSubject`, kde název vždy začína slovem *test* a po něm pokračuje předmět testování (*subject*). Pro otestování podmínek se používají makra `XCTAssert`, `XCTAssertNotNil`, `XCTAssertEqual` a další, které při nesplnění zvýrazní v editoru, na kterém místě nastala chyba. Po skončení všech testů proběhne metoda `tearDown`, ve které by programátor měl uvolnit naalokovanou pamět a uzavřít otevřené soubory.

Při testování jsem se primárně zaměřil na správnou inicializaci aplikace a přítomnost všech důležitých prvků, jako jsou okna a viewControllery. Dalším důležitým předpokladem, pro správnou funkčnost bylo korektní chování RTM API a mého wrapperu, který tvoří mezivrstvu mezi rtm2cocoa frameworkem [14] a aplikací. Toto otestování jsem provedl v metodě `testDataSource`.

Jelikož XCTest poskytuje rozhraní pro testování asynchronních volání, mohl jsem otestovat i komplikovanější metody. Na následujícím úryvku zdrojového kódu je vidět, jakým způsobem tento mechanismus funguje. Vytvoří se `XCTestExpectation`, která očekává, že bude notifikována, a dokud tomu tak nenastane, nebo neuběhne nastavený timeout, bude blokovat ukončení testu.

Notifikace je vidět na řádku 8 a blokování na řádku 11. Pokud uběhne timeout a objekt expectation nebyl notifikován, test bude vyhodnocen jako neúspěšný.

```
1  /* uryvek z testovani asynchronnich metod */
2    BMDataSource *ds  [BMDataSource sharedInstance]
3    XCTestExpectation *exp = [self expectationWithDescription:@"listExp"];
4
5    [ds getListWithId:list.getListId.integerValue
6        andCompletion:^(NSRTMList *list) {
7           XCTAssertNotNil(list, @"Returned list cant be nil");  /* vlakno */
8           [listDownloadedExpectation fulfill]; /* dobehlo -> notifikovat exp */
9       }];
10   /* zde se ceka na splneni expectation po dobu 5 sekund */
11   [self waitForExpectationsWithTimeout:5 handler:nil];
```

5.2 Usability testy

5.2.1 Testovací subjekty

Aby výsledky testů odpovídaly co nejvíce realitě, pokusil jsem se sehnat rozmanitou skupinu testovacích subjektů. Počínaje vývojáři pro mobilní systém iOS s dlouholetými zkušenostmi s aplikacemi pro OS X, konče uživateli OS Windows. Celkově dotazník vyplnilo 10 subjektů, jejich složení bylo následující:

- vývojáři iOS (2)
- uživatelé iPhone a iPad (2)
- uživatelé OS X
  - pokročilí (2)
  - začátečníci (1)
- uživatelé Windows
  - pokročilí (3)

5.2.2 Úkoly

Úkoly jsem postavil tak, aby svým rozsahem pokryly funkční požadavky a případy užití.

1. Přihlásit se do aplikace a povolit ji přístup k RTM účtu
2. Vytvořit nový úkol
3. Vytvořit nový úkol s dvěma tagy a nastavit mu datum na zítřek

4. Upravit existující úkol změnou data
5. Splnit více úkolů najednou
6. Odložit úkol
7. Smazat úkol
8. Přidat nový kontakt
9. Přidat poznámku k úkolu
10. Vyhledat úkol podle názvu

5.2.3 Ohodnocení

Po splnění každého úkolu, uživatelé museli rozhodnout, jak náročné bylo jeho vykonání. Na výběr měli z pěti možností.

- velmi jednoduché (VJ)
- jednoduché (J)
- neutrální (N)
- obtížné (O)
- velmi obtížné (VO)

5.2.4 Výsledek

Tabulka 5.1: Výsledek dotázníku z testu usability

Uživatel / Úkol	Ú1	Ú2	Ú3	Ú4	Ú5	Ú6	Ú7	Ú8	Ú9	Ú10
vývojář iOS 1	VJ	VJ	J	VJ	VJ	J	VJ	J	VJ	VJ
vývojář iOS 1	VJ	VJ	N	J	VJ	J	VJ	N	VJ	VJ
OS X pokročilý 1	VJ	VJ	J	VJ	VJ	N	VJ	J	VJ	VJ
OS X pokročilý 2	J	VJ	N	VJ	J	N	VJ	J	VJ	VJ
OS X začátečník 2	VJ	VJ	N	J	J	N	VJ	N	VJ	VJ
Windows 1	J	VJ	N	VJ	VJ	J	VJ	J	J	VJ
Windows 2	VJ	VJ	J	J	VJ	J	VJ	N	VJ	VJ
Windows 3	VJ	VJ	N	VJ	VJ	J	VJ	J	VJ	J
Průměr	**VJ**	**VJ**	**N**	**VJ**	**VJ**	**J**	**VJ**	**J**	**VJ**	**VJ**

5.2.5 Závěr

Z výsledků je vidět, že i přes relativní komplexnost aplikace nečinilo běžné užívání většině uživatelů žádný problém. Nejméně nesnází s úkoly měli podle

očekávání pokročilí uživatelé OS X. Méně zkušení uživatelé potom měli problém s komplikovanějšími úkoly, jako jsou například ty s čísly 3 a 8.

Dobrou zprávou je, že nejfrekventovanější úkony, mezi které patří tvorba a plnění úkolů, prošly testem s nejlepším možným výsledkem. Nízkou úspěšnost úlohy 6 (odložení úkolu) připisuji neintuitivní ikoně. Uživatel si na první pohled není jistý, co daná ikona reprezentuje.

Dalšími pozitivy jsou kladné výsledky úloh 1 a 10. Úvodní spuštění, i přes moje obavy, nedělalo uživatelům vesměs žádný problém a vytvoření úkolů přes Spotlight si všichni vychvalovali.

Aplikace byla celkově hodnocena velmi pozitivně. Nejkladnější ohlasy jsem obdržel od pokročilejších uživatelů, kteří si ihned oblíbili funkce jako je autocomplete nebo tvorbu úkolů přes Spotlight. Ostatní uživatelé aplikaci hodnotili rovněž kladně. Občas napoprvé nevěděli, jak některý úkol splnit, ale vždy se jim to po několika pokusech podařilo.

Na základě napozorovaných výsledků jsem se rozhodl do aplikace zapracovat následující změny

- změnit ikonu tlačítka pro odložení úkolu
- přidat dočasný text s ukázkou do pole pro tvorbu úkolu

Závěr

Prvním cílem této bakalářské práce bylo provedení rešerše nejlepší praxe návrhu a implementace architektury a grafického rozhraní aplikací pro operační systém Mac OS X 10.9 a vyšší. Cílem druhým bylo praktické využití znalostí nabytých z rešerše při tvorbě desktopové aplikace pro systém Remember the Milk. Oba tyto cíle se mi podařilo splnit.

V první části rešerše jsem se zaměřil na programovací jazyk a knihovny používané pro tvorbu aplikací na operační systém OS X. Následující třetina byla zaměřena na stukturu a model architektury. V poslední části jsem se věnoval návrhu grafického rozhraní, kde jsem při srovnávání konkurenčních aplikací poukazoval na často používané prvky.

Praktickou část práce jsem začal analýzou a návrhem aplikace Buttermilk. Na základě webové verze RTM jsem sestavil seznam funkčních a nefunkčních požadavků. Pokračoval jsem tvorbou diagramů v UML, návrhem architektury a grafického uživatelského rozhraní.

Při implementaci jsem kladl co největší důraz na rozšiřitelnost a dodržování návrhových vzorů, které jsou při vývoji těchto aplikací kritické. Dalším z vedlejších cílů práce bylo zaměření se na efektivitu používání a ovládání. Tuto část práce bych chtěl vyzdvihnout, jelikož se mi podařilo naimplementovat systém přidávání úkolů *SmartAdd*, a to i s využitím napovídání textu. Využil jsem i několika zbrusu nových technologií. Například při tvorbě zásuvného modulu do notifikačního centra, který mnoho současných aplikací zatím nemá. Zkušení uživatelé dále ocení podporu zadávání nových úkolů pomocí systémového rozhraní Spotlight. Aplikaci je s pomocí těchto technologií možné používat velice efektivně a při většině základních úkonů ji není potřeba ani přímo spouštět, stačí využít zásuvných modulů.

Funkčnost výsledné aplikace jsem podrobil sérii jednotkových testů. Na otestování uživatelského rozhraní jsem vytvořil usability testy formou dotazníku. Všechny testy dopadly úspěšně. Aplikace je v současném stavu schopna nahradit webového klienta ve vykonávání všech základních činností spojených se správou úkolů.

Plány do budoucna

V blízké budoucnosti bych chtěl do aplikace přidat méně používané funkce uživatelů premium, ke kterým jsem při implementaci neměl přístup. Také by bylo na místě nahradit současný vzhled grafického rozhraní návrhem od grafika. Dalším místem pro rozšíření je zásuvný modul v notifikačním centru, který v současné době slouží pouze jako přehled nadcházejících úkolů.

Literatura

[1] APPLE Inc.: Cocoa Fundamentals Guide. [online], 18.09.2013 [cit. 2015-03-16]. Dostupné z: <https://developer.apple.com/legacy/library/documentation/Cocoa/Conceptual/CocoaFundamentals/>

[2] APPLE Inc.: Mac Technology Overview. [online], 10.06.2014 [cit. 2015-04-06]. Dostupné z: <https://developer.apple.com/library/mac/documentation/MacOSX/Conceptual/OSX_Technology_Overview/>

[3] APPLE Inc.: Survey the Major Frameworks. [online], 23.04.2013 [cit. 2015-03-05]. Dostupné z: <https://developer.apple.com/library/mac/referencelibrary/GettingStarted/RoadMapOSX/chapters/07_Frameworks.html>

[4] HILLEGASS, Aaron a Adam PREBLE: *Cocoa programming for Mac OS X*. Addison-Wesley, čtvrté vydání, 2012, ISBN 978-0-321-77408-8.

[5] APPLE Inc.: Streamline Your App with Design Patterns. [online], 23.04.2013 [cit. 2015-04-21]. Dostupné z: <https://developer.apple.com/library/mac/referencelibrary/GettingStarted/RoadMapOSX/chapters/08_DesignPatterns.html>

[6] ENGEL, Ruben: Skeuomorphism gone in Notes app, new PDF export feature. [online], 24.06.2013 [cit. 2015-04-13]. Dostupné z: <http://www.tips-and-tricks-in-mavericks.com/skeuomorphism-gone-in-notes-app-new-pdf-export-feature/>

[7] CUTEDGE SYSTEMS: LDAP Enabler for Mountain Lion. [online], 9.11.2012 [cit. 2015-04-13]. Dostupné z: <https://cutedgesystems.com/software/LDAPEnablerForMountainLion/>

[8] APPLE Inc.: OS X Human Interface Guidelines. [online], 08.04.2015 [cit. 2015-03-15]. Dostupné z: <https://developer.apple.com/

library/mac/documentation/UserExperience/Conceptual/OSXHIGuidelines/>

[9] PIXSHARK Inc.: Safari on OS X Mavericks. [online], 22.06.2013 [cit. 2015-04-12]. Dostupné z: <http://pixshark.com/os-x-mavericks-safari.htm>

[10] TRAXSOURCE NEWS: OS X Mavericks Review. [online], 04.11.2012 [cit. 2015-04-13]. Dostupné z: <http://news.traxsource.com/articles/763/os-x-mavericks-review>

[11] REMEMBER THE MILK Pty Ltd.: Remember The Milk - What is Smart Add? [online], 12.04.2015 [cit. 2015-03-10]. Dostupné z: <https://www.rememberthemilk.com/help/?ctx=basics.smartadd.whatis>

[12] APPLE Inc.: Mac App Programming Guide. [online], 09.03.2015 [cit. 2015-04-05]. Dostupné z: <https://developer.apple.com/library/mac/documentation/General/Conceptual/MOSXAppProgrammingGuide/>

[13] APPLE Inc.: Auto Layout Guide. [online], 18.09.2013 [cit. 2015-03-14]. Dostupné z: <https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/AutolayoutPG>

[14] KKILLIAN: rtm2cocoa. [online], [cit. 2015-04-12]. Dostupné z: <https://code.google.com/p/rtm2cocoa/>

[15] GANEM, Eli: iOS Design Patterns. [online], 04.09.2013 [cit. 2015-04-17]. Dostupné z: <http://www.raywenderlich.com/46988/ios-design-patterns>

[16] APPLE Inc.: Objective-C Runtime Reference. [online], 22.10.2014 [cit. 2015-04-17]. Dostupné z: <https://developer.apple.com/library/mac/documentation/Cocoa/Reference/ObjCRuntimeRef/>

[17] APPLE Inc.: Grand Central Dispatch (GCD) Reference. [online], 09.17.2014 [cit. 2015-04-01]. Dostupné z: <https://developer.apple.com/library/prerelease/mac/documentation/Performance/Reference/GCD_libdispatch_Ref/>

[18] PARROTT, Nate: Flashlight — do anything with a keystroke. [online], [cit. 2015-04-10]. Dostupné z: <http://flashlight.nateparrott.com>

[19] APPLE Inc.: Testing with Xcode. [online], 16.10.2014 [cit. 2015-04-12]. Dostupné z: <https://developer.apple.com/library/mac/documentation/DeveloperTools/Conceptual/testing_with_xcode/>

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

ARC Automatic reference counting

BSD Berkeley Software Distribution

GCD Grand Central Dispatch

GUI Graphical user interface

IB Interface Builder

MRC Manual reference counting

MVC Model-view-controller

POSIX Portable Operating System Interface

RTM Remember the Milk

SDK Software development kit

SQL Structured query language

UML Unified Modeling Language

URL Uniform Resource Locator

WYSIWYG What you see is what you get

XML Extensible markup language

Příloha B

Struktura tříd projektu

Obrázek B.1: Stuktura tříd projektu Buttermilk v Xcode

Příloha C

Instalační příručka

Tato příloha obsahuje návod na instalaci aplikace a přídavného rozšíření pro vyhledávací systém Spotlight.

C.1 Instalace hlavní Aplikace

Pro nainstalování stačí zkopírovat soubor `Buttermilk.app` z adresáře `/exe` na přiloženém disku do adresáře s aplikacemi na počítači Mac. Pokud chceme aplikaci nainstalovat pro všechny uživatele daného systému, použijeme adresář `/Applications`. Pokud bychom naopak vyžadovali nainstalování pouze pro konkrétního uživatele, nakopírujeme ji do adresáře s aplikacemi daného uživatele. Obvykle se nachází na cestě `/Users/<username>/Applications`.

C.2 Instalace rozšíření pro Spotlight

Pro správnou funkčnost zásuvného modulu je hlavní podmínkou mít nainstalovaný software Flashlight [1].

Dalším krokem je nakopírování souboru `/exe/bm.bundle` na cílový počítač do adresáře `/Library/FlashlightPlugins`. Funkčnost by se měla projevit obratem. Pokud tomu tak nenastalo, restartujte Flashlight kliknutím na tlačítko *Enable Spotlight Plugins*.

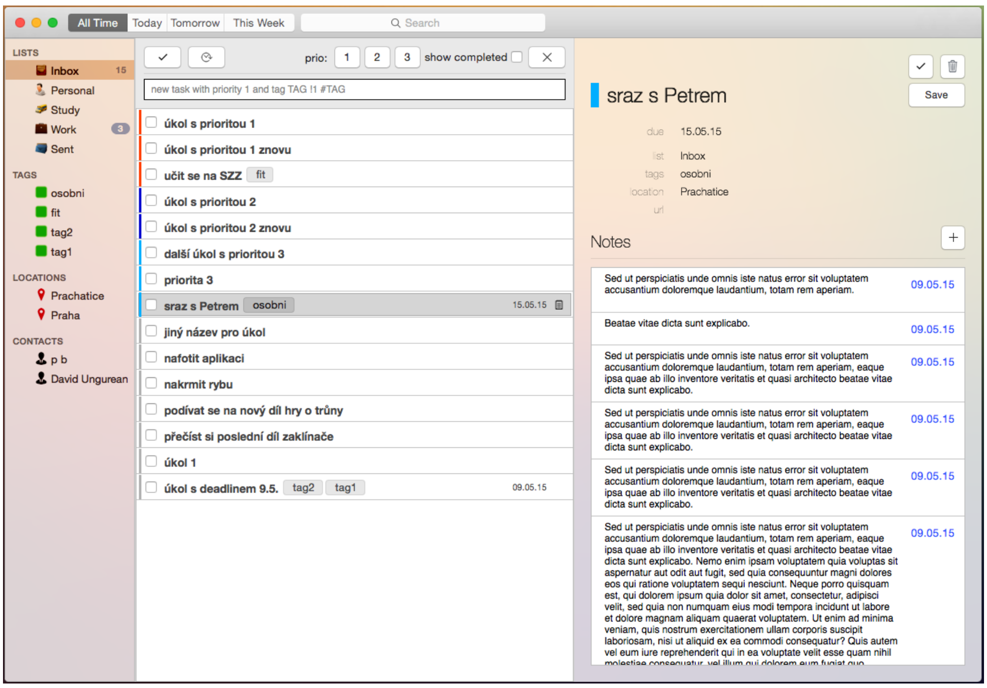
C.3 Prvotní přihlášení

Při prvotním přihlášení se objeví dialogové okno s možností přidělení práv aplikaci. Po kliknutí na souhlas se otevře internetový prohlížeč, kde budete přesměrováni na domovskou stránku Remember the Milk. Zde je nutné přidat aplikaci vyžadovaná práva (číst, psát a mazat). Potom se stačí vrátit zpět do aplikace a kliknout na tlačítko *Continue*. Autorizace je tímto u konce.

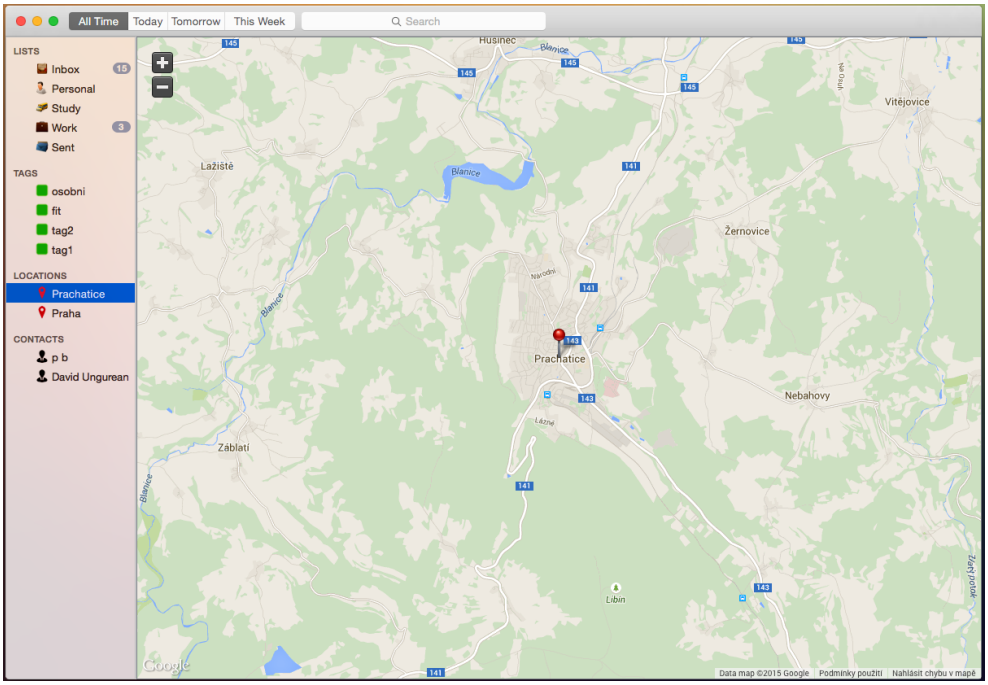
[1] dostupný na adrese `<http://flashlight.nateparrott.com>`

Příloha D

Finální vzhled aplikace



Obrázek D.1: Ukázka hlavního okna aplikace s detailem úkolu



Obrázek D.2: Ukázka hlavního okna aplikace s detailem lokace

Příloha **E**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `docs`
  - `html` ..... vygenerovaná dokumentace zdrojových kódů
    - `index.html` ..... hlavní stránka HTML dokumentace
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `buttermilk_src` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Ungurean_David.pdf` ..... text práce ve formátu PDF