Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Interaktivní informační aplikace pro mobilní zařízení na platformě iOS

Plzeň, 2012 Jiří Zikmund

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 10. května 2012

Jiří Zikmund

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce Ing. Ladislavu Pešičkovi za skvělé vedení a čas strávený při konzultacích. Dále chci poděkovat rodičům za podporu a přítelkyni za přečtení práce a výbornou korekturu.

Abstract

Subject of this thesis is Interactive information application for iOS mobile devices. In the first part I describe possibilities in retrieving and displaying information by iOS applications in theory, taking into account its use in mobile devices. This part also focuses on classification of information and means of information retrieval from remote sources. In the second part of the thesis I proceed to the development process for iOS platform itself. Product of this task is a working application which provides data from selected information sources and runs on a real device.

Obsah

1 Úvod

Mobilní telefony jsou běžnou součástí našeho života a už dávno neslouží pouze k telefonování. Staly se z nich multifunkční zařízení s všestranným využitím a začaly částečně nahrazovat fotoaparáty, multimediální přehrávače nebo např. navigační přístroje. Díky mobilnímu internetu jsou také ideální pro okamžitý přístup k informacím.

Jedním ze způsobů prezentace těchto informací jsou mobilní internetové prohlížeče. Ty jsou již relativně vyspělé a mohou v mnoha případech sloužit jako plnohodnotná náhrada prohlížečů na počítači, nicméně jsou pro ně limitující zobrazovací schopnosti zařízení, na nichž jsou provozovány. Jde především o velikost displeje, protože úhlopříčky mobilních telefonů nepřesahují zpravidla 5 palců. Další možností jsou aplikace, které zpracovávají pouze konkrétní zdroj informací a přizpůsobují jejich zobrazení displejům mobilního zařízení. Příkladem mohou být zpravodajské servery, u nichž se stalo trendem poslední doby mít kromě webových stránek také odpovídající aplikaci dostupnou pro nejrozšířenější mobilní platformy.

V rámci bakalářské práce budu vyvíjet interaktivní informační aplikaci pro platformu iOS, konkrétně pro mobilní telefon iPhone. Aplikace bude získávat aktuální informace dostupné na internetu a s ohledem na velikost displeje je vhodně zobrazovat v telefonu. Získaná data bude možné použít např. pro plánování událostí v kalendáři nebo jiné interakce. Informační zdroje nejsou nijak blíže specifikovány, a proto by měla být aplikace navržena dostatečně obecně. Její funkčnost se ověří na konkrétní kategorii informací, např. kinech.

Protože mobilní připojení k internetu je v České republice stále dosti omezené, zejména co se týká množství přenesených dat, bude vhodné, aby aplikace umožňovala lokální ukládání získaných informací a jejich aktualizace probíhala pouze na vyžádání.

Výsledkem by měla být funkční aplikace běžící v reálném zařízení na platformě iOS.

2 Platforma iOS

2.1 Operační systém iOS

Operační systém iOS od společnosti Apple byl vydán společně s mobilním telefonem iPhone v roce 2007. Původně se jmenoval iPhone OS, ale protože firma začala tento systém nasazovat postupem času i do svých dalších mobilních zařízení, byl v souladu s jejich názvy přejmenován obecněji na iOS. Dnes se s ním setkáme v přístrojích iPhone, iPod Touch a iPad. Od svého vzniku si vybudoval opravdu širokou základnu uživatelů a do podzimu 2011 se prodalo přes 250 milionů zařízení s tímto systémem. [5]

Vydáním iOS vytvořila společnost Apple zcela nový pohled na mobilní telefony. Oproti konkurenci nabízely v té době iPhony velmi pohodlné dotykové ovládání jen pomocí prstů, systém byl uživatelsky přívětivý, stabilní a perfektně spolupracoval s hardwarem. To se nezměnilo ani po pěti letech. V systému samozřejmě přibyly nové funkce, ale základní principy ovládání jsou stále stejné.

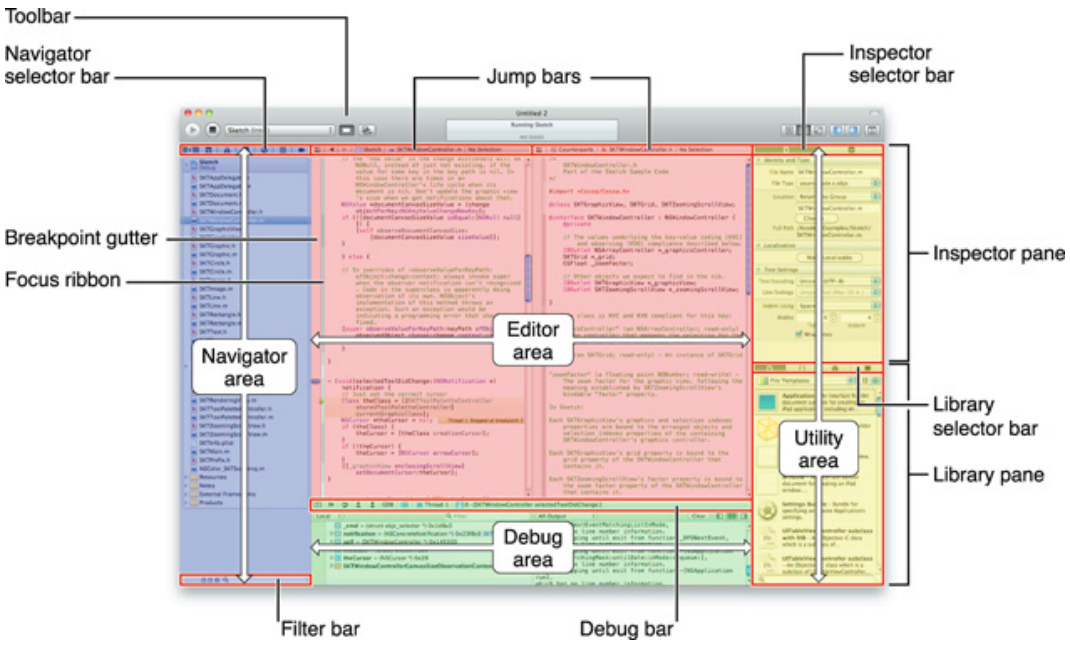
2.2 Vývoj aplikací pro iOS

Operační systém iOS je odlehčenou verzí Mac OS X, který je používán na počítačích společnosti Apple, a stejně jako on je tedy založený na Unixu. Původně byl vytvořen jako úplně uzavřený systém bez možnosti jakéhokoliv vývoje aplikací třetích stran, alespoň ne oficiální cestou. Přibližně po roce od uvedení, tedy v roce 2008, bylo ale uvolněno SDK pro vývojáře, čímž se otevřela cesta k vývoji pro tuto platformu.

SDK bylo vydáno pro vývojové prostředí Xcode, které je dostupné pouze pro operační systémem Mac OS X. Zdá se tedy, že jedinou možností vývoje aplikací pro iOS jsou počítače Apple. Protože ale v České republice nejsou zatím tolik rozšířené, snažil jsem najít alternativní způsob. Nabízela se možnost použití virtualizačního nástroje pro Windows, který by simuloval běh Mac OS X. Dle licenčních podmínek pro použití softwaru společnosti Apple to však není možné a pro svou práci jsem proto zvolil nativní platformu, počítač Apple Mac mini. [6]

2.3 Xcode a Objective C

Xcode je vývojové prostředí zdarma nabízené společností Apple. Lze ho získat např. stažením z jejich webových stránek. Integruje v sobě mnoho nástrojů, které usnadňují a zefektivňují vývoj aplikací pro platformu iOS. Podporuje velké množství programovacích jazyků, jako je např. C++, Java nebo Python, ale aplikace pro iOS lze vytvářet pouze v jazyce Objective C, případně v C. [7]

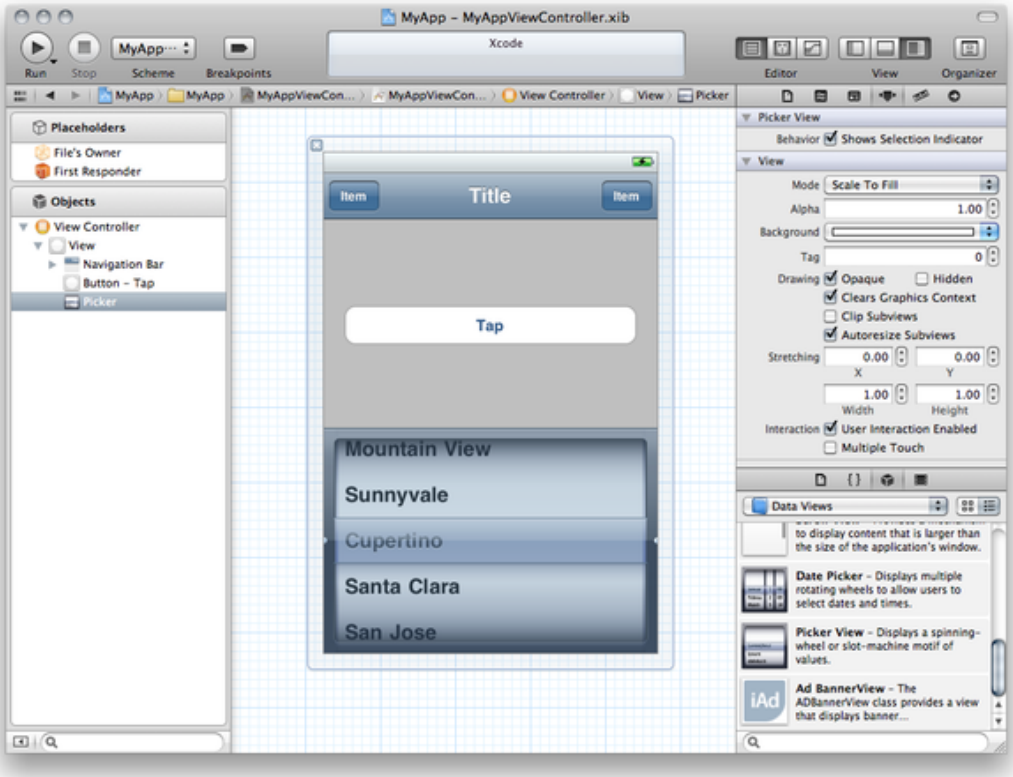


Obrázek 2.1: Základní obrazovka Xcode

Objective C vzniklo jako objektově orientované rozšíření jazyka C, s nímž byla zachována zpětná kompatibilita. Díky objektům a mnoha knihovnám ale nabízí efektivnější práci. Nad objekty je definována jediná operace, což je posílání zpráv, principiálně převzaté z jazyka Smalltalk.

Apple ve svých zařízeních se systémem iOS zavedl určitý standard, co se týká vzhledu a chování uživatelského prostředí, a dodržuje ho nejen ve svých aplikacích, ale snaží se k němu vést i ostatní vývojáře. Součástí Xcode je proto Interface Builder, velmi výkonný nástroj pro tvorbu vizuální části aplikace (viz obr. 2.2, str. 4). Vývojářům nabízí širokou paletu komponent, ze kterých mohou poskládat obrazovky aplikace, např. tlačítka, textová pole, přepínače, posuvníky a další. [8]

Všechny tyto prvky mají totožný vzhled s nativním prostředím iOS, čímž se zaručuje sjednocený vzhled celého prostředí telefonu. Vydal také příručku *iOS Human Interface Guidelines*, ve které popisuje doporučené zásady návrhu aplikace. [9]



Obrázek 2.2: Interface builder

2.3.1 UI Elements

Při návrhu aplikace je v Xcode k dispozici pro iPhone poměrně široké množství komponent. Následuje popis nejdůležitějších z nich. [8]

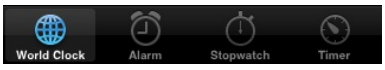
Status Bar zobrazuje důležité informace týkající se přístroje. Vývojář může ovlivnit pouze jeho zobrazení nebo skrytí v aplikaci, nikoliv jeho jednotlivé prvky.

Obrázek 2.3: Status Bar

Navigation Bar ve většině případů ukazuje název aktuální obrazovky aplikace a umožňuje navigaci zpět, případně další akce. Umístění této komponenty je vždy v horní části obrazovky.

Obrázek 2.4: Navigation Bar

Tab Bar je hlavní komponenta pro navigaci. Její tlačítka znázorňují jednotlivé obrazovky aplikace a umožňují přechod mezi nimi.



Obrázek 2.5: Tab Bar

Toolbar obsahuje kontextová tlačítka pro práci s aktuálním obsahem obrazovky a je zobrazen vždy ve spodní části aplikace.

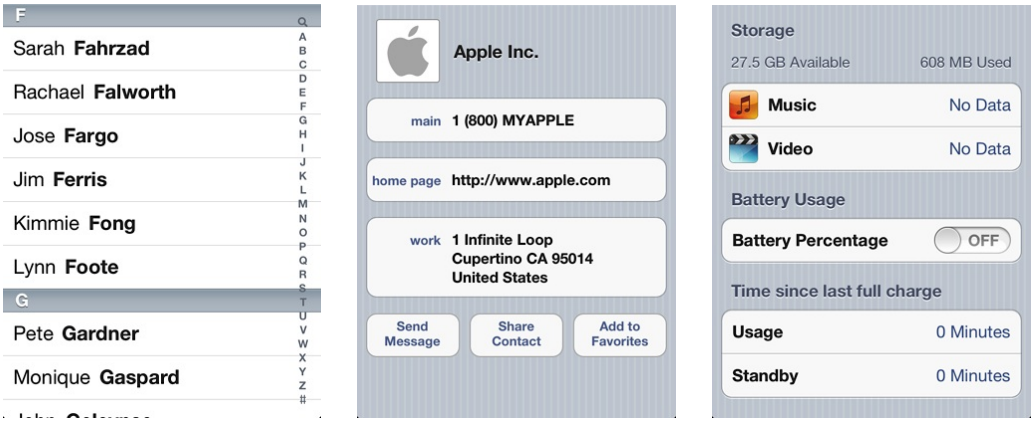
Obrázek 2.6: Toolbar

Alert má několik možností zobrazení. Jeho počet tlačítek je variabilní, takže kromě jednoduchého informačního hlášení může sloužit i pro interakci s uživatelem.

(a) Základní podoba (b) Interakce s uživatelem

Obrázek 2.7: Alert

Table View představuje typicky seznam položek. Umožňuje seskupování a vyhledávání podle názvů. Jednotlivé buňky nemusí obsahovat pouze text, ale i tlačítka, obrázky, nebo jakékoliv další prvky. Tato komponenta je díky svému všestrannému využití jednou z nejpoužívanějších vůbec.



(a) Základní podoba (b) Upravená podoba 1 (c) Upravená podoba 2

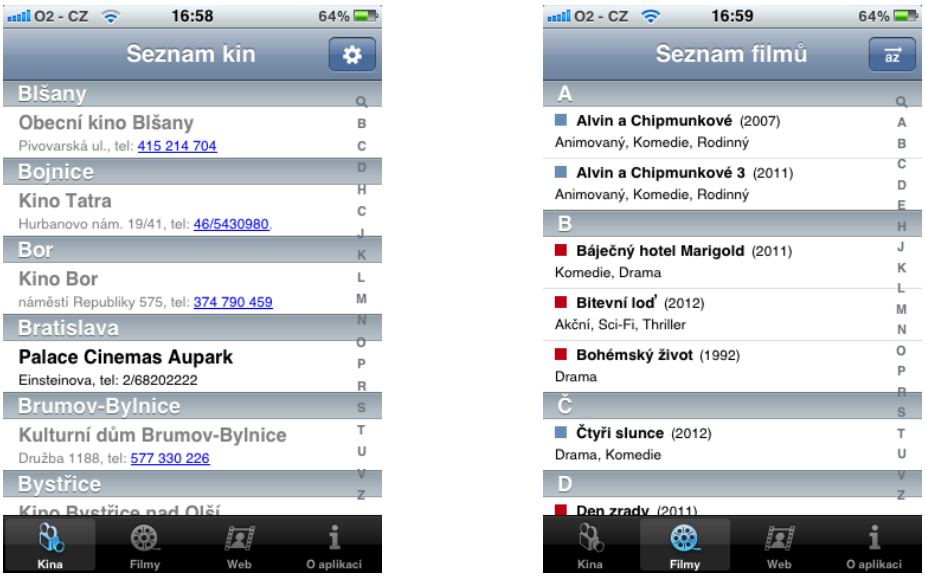
Obrázek 2.8: Table View

2.4 Ukázka vybraných informačních aplikací

Pro platformu iOS existuje velké množství aplikací a mezi nimi jsou samozřejmě i aplikace s podobným zaměřením, jako moje práce.

Aplikace Kina CSFD.cz

Aplikace Kina CSFD.cz se zaměřuje na kinematografii v ČR. Nabízí seznam kin a aktuálně vysílané filmy. Kromě časů představení v jednotlivých kinech se u detailu filmu zobrazuje uživatelské hodnocení. Jak je vidět na obrázcích, používá k zobrazování dat převážně modifikovanou komponentu Table View.

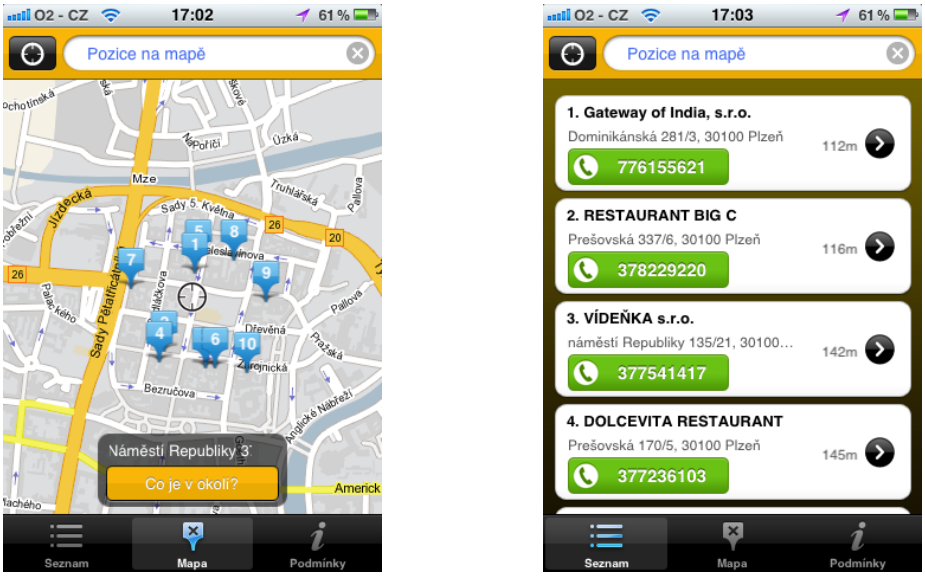


(a) Seznam kin (b) Seznam filmů

Obrázek 2.9: Aplikace Kina CSFD.cz

Aplikace Kde jíst a pít

Kde jíst a pít je aplikace Zlatých stránek a slouží jako alternativa k jejich webovým stránkám. Využívá GPS telefonu pro zjišťování aktuální polohy a zobrazuje pomocí ní nejbližší místa v okolí. Kromě klasického zobrazení v seznamu nabízí i náhled míst na mapě. Díky rozsáhlému katalogu je aplikace v praxi dobře použitelná.



(a) Mapa (b) Seznam míst

Obrázek 2.10: Aplikace Kde jíst a pít

3 Analýza problému

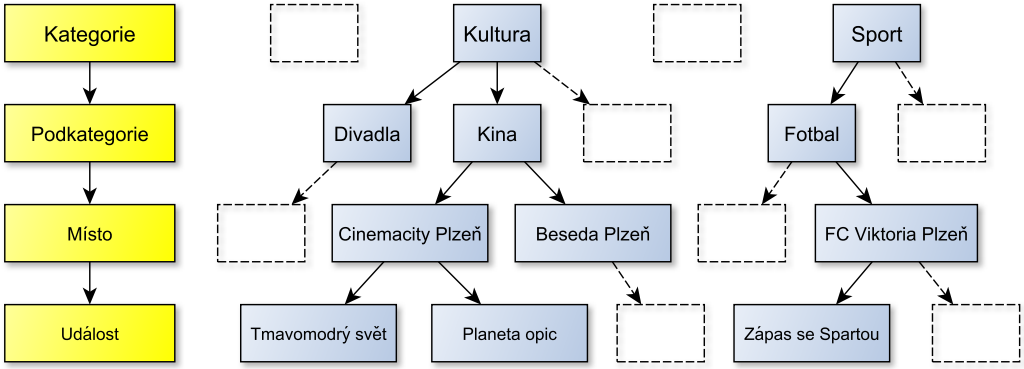
3.1 Třídění informací

Jednou z prvních věcí, kterou jsem při analýze problému musel vyřešit, byl způsob, jakým budu získaná data třídit. Protože se nejedná o žádný konkrétní zdroj informací, muselo být vše zvoleno dostatečně obecně.

Určil jsem, že základní informační jednotkou celého systému bude událost, pro kterou bude definováno místo a čas. Může tedy představovat např. sportovní utkání, divadelní představení nebo hudební koncert. Navzdory názvu se ale nemusí jednat pouze o událost jako takovou, lze si představit i např. nabídku jídel v restauraci. Vše záleží na vhodném způsobu zpracování informačního zdroje. Dalšími vlastnostmi události jsou její název, popis a délka trvání, tedy dostatečně obecné informace pro mnoho různých zdrojů.

S přibývajícím množství událostí by se ale jejich rozumné a přehledné zobrazování, navíc s přihlédnutím k malému displeji mobilního telefonu, stalo značně obtížné. Bylo tedy nutné události nějakým způsobem třídit. Rozumné se to jeví podle jejich místa. A protože k jednomu místu se může vztahovat více událostí, určil jsem místo jako nadřazené událostem. Místo potom patří do podkategorie, která dále spadá do kategorie, což je nejobecnější prvek kategorizace (viz obr. 3.1, str. 10).

- **Kategorie** - název
- **Podkategorie** - název a kategorie, do které patří
- **Místo** - název, poloha a podkategorie, do které patří
- **Událost** - název, popis, čas a místo, kam patří



Obrázek 3.1: Příklad třídění dat (prázdná pole značí další prvky)

3.2 Agregace informačních zdrojů

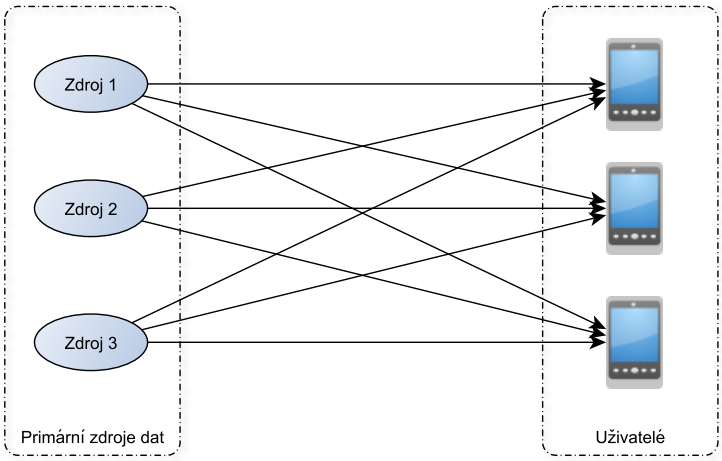
3.2.1 Přímý přístup

Základním pilířem aplikace jsou data a ta je potřeba nějak získávat. První možností je přímý přístup, což znamená, že aplikace získává informace z primárního zdroje. Dobrým příkladem jsou programy kin. Uživatel spustí aplikaci a chce např. zjistit, jaké filmy jsou aktuálně promítány v jeho oblíbeném kině. V tu chvíli se aplikace musí připojit k nějakému zdroji této informace, jímž jsou zřejmě webové stránky daného kina, načíst jejich obsah, zpracovat ho a získat z něj potřebná data. Takový proces může být velmi zdlouhavý a musel by být prováděn při každé aktualizaci (viz obr. 3.2, str. 11).

Další nevýhodou přímého přístupu je to, že zdroj musí existovat a být nějakým způsobem stále dostupný. Navíc pokud bychom chtěli zobrazit nový zdroj informací, musel by být pevnou součástí aplikace, což by kvůli aktuálnosti dat ztrácelo svůj smysl.

3.2.2 Nepřímý přístup pomocí serveru

Kvůli zmíněným nevýhodám přímého přístupu k informacím bylo potřeba najít alternativní způsob získávání dat. Jako elegantní řešení se ukázalo vložení serveru mezi uživatelskou aplikaci a primární zdroje dat. Tento server má na jedné straně na starosti získávání aktuálních dat ze všech určených zdrojů a na druhé straně jejich poskytování aplikaci uživatele (viz obr. 3.3, str. 12).



Obrázek 3.2: Přímý přístup k informacím

Pojem informační zdroje je nutné v tomto případě brát velmi obecně. Mohou totiž představovat mnoho různých věcí a hlavně mohou mít libovolnou formu. Nejčastěji však zřejmě půjde o internetové stránky, jejichž obsah bude nutné zpracovat a získat z nich potřebná data. Tento proces bude pro každý zdroj velmi individuální a přidávání nových zdrojů tak nebude triviální záležitostí.

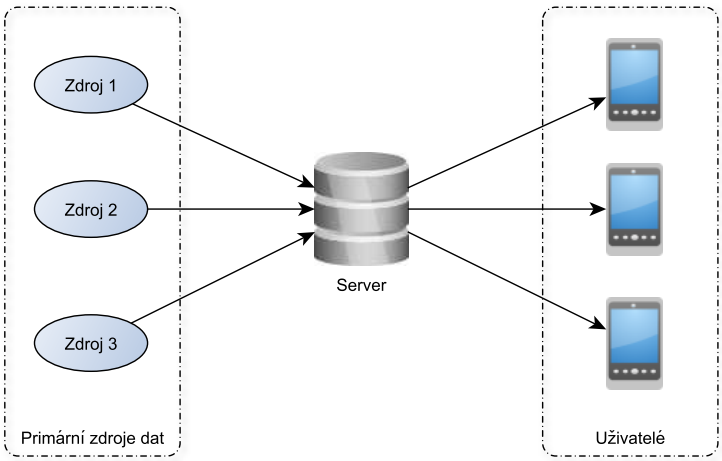
Získaná data budou uložena na serveru a připravena pro zaslání uživatelské aplikaci až ve chvíli, kdy si to vyžádá. Tím se celý proces přesouvání informací z primárního zdroje k uživateli rozdělí na dvě části:

- **zpracování dat** z informačního zdroje a uložení na server
- **poskytnutí dat** uživatelské aplikaci

To je velmi výhodné, protože zpracování dat je časově relativně náročné a může být prováděno nezávisle na jejich poskytování uživateli. Otázkou zůstává, jak často data z informačních zdrojů zpracovávat. To je individuální a záleží to na konkrétním zdroji. V případě programů kin stačí aktualizace jednou denně, ale pokud by se jednalo o zpravodajství, bylo by vhodné proces opakovat např. každých deset minut tak, aby uživatel získal vždy aktuální informace.

Další výhodou použití serveru je možnost ručního přidávání zdrojů, resp. již konkrétních dat. Můžeme tak poskytovat informace, které nejsou nikde k dispozici a nedají se ukládat automaticky.

Takto navržený systém se při správně zvoleném způsobu komunikaci mezi serverem a aplikací stává velmi univerzálním. K přenosu dat proto budu používat značkovací jazyk XML, který je nezávislý na použité platformě. Díky tomu by se aplikace dala portovat i na jiné systémy než je iOS.



Obrázek 3.3: Nepřímý přístup k informacím pomocí serveru

4 Cíl práce

Práce je rozdělena na serverovou část a klientskou aplikaci pro platformu iOS. Server bude schopen zpracovávat informační zdroje na internetu a ukládat získaná data, která bude následně při zažádání poskytovat klientovi. Pro ověření funkcionality bude server schopný získávat programy několika kin v ČR.

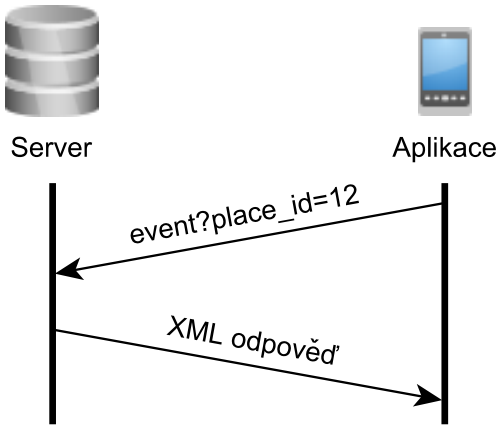
Aplikace se bude připojovat k serveru a stahovat data. Ta se budou lokálně kešovat v zařízení a aktualizace proběhne pouze při žádosti uživatele. Výpis informací bude prováděn s ohledem na dostupnost pro mobilní zařízení. Kromě textové reprezentace nabídne aplikace i jejich zobrazení na mapě s využitím geografické polohy přístroje. Události s definovaným časem bude možné využít pro plánování kalendáře, odeslání SMS a emailu. Oproti aplikacím v kapitole 2.4 nebude tato práce zaměřena na žádný konkrétní zdroj informací a bude možné je doplňovat.

Součástí práce je i návrh vhodné síťové komunikace mezi serverem a aplikací.

5 Komunikace mezi serverem a aplikací

Server a aplikace jsou dva nezávislé prvky a musí spolu komunikovat. Ve směru z aplikace na server se bude posílat jen informace o tom, jaká data jsou požadována. To se dá velmi jednoduše zařídit pomocí parametrů v http protokolu. Odpovědí ale bude relativně velké množství dat a je proto nutné určit jejich formát. Protože server i aplikace fungují na odlišných platformách, ukázalo se jako dobrá volba XML (viz obrázek 5.1).

Množství dat uložených na serveru může být při rozsáhlejším počtu informačních zdrojů příliš velké, a proto by nebylo vhodné přenášet je do aplikace všechna najednou. Z tohoto důvodu jsem rozdělil přenos podle vrstev třídění informací (viz obrázek 3.1, str. 10). Posílají se tak zvlášť kategorie, podkategorie, místa i události. Navíc pouze v rámci jedné své nadřazené třídy. Najednou se tak přenáší např. všechny události patřící k jednomu místu nebo všechna místa z jedné podkategorie.



Obrázek 5.1: Komunikace serveru s aplikací

5.1 Formát XML

Aby komunikace probíhala v pořádku, musí být XML ze serveru odesláno ve stejném formátu, jak ho očekává aplikace. Pro formát jsem zvolil následující hlavní značky:

- `root` - kořenový element
- `status_code` - číselný kód stavu odpovědi (viz tabulka 5.1)
- `status_message` - slovní reprezentace status_code
- `data` - element zaobalující všechna přenášená data
- `node` - obecný element, obsahuje vždy jednu informaci (např. jedno místo nebo jednu událost)

status_code	**status_message**
200	`OK`
201	`Service OK`
403	`Parameter category_id is missing`
404	`Parameter category_id is empty`
405	`Parameter subcategory_id is missing`
406	`Parameter subcategory_id is empty`
407	`Parameter place_id is missing`
408	`Parameter place_id is empty`
501	`No results`

Tabulka 5.1: Číselné kódy stavu odpovědi a jejich slovní reprezentace

Další značky jsou použity pro popis informací a liší se podle své třídy (viz tabulka 5.2, str. 16).

Kategorie	
`id`	Identifikační číslo
`name`	Název
`timestamp_unix`	Čas poslední aktualizace v unixovém formátu
Podkategorie	
`id`	Identifikační číslo
`category_id`	Číslo kategorie, do které patří
`name`	Název
`timestamp_unix`	Čas poslední aktualizace v unixovém formátu
Místo	
`id`	Identifikační číslo
`subcategory_id`	Číslo podkategorie, do které patří
`name`	Název
`timestamp_unix`	Čas poslední aktualizace v unixovém formátu
`description`	Popis
`coordinates`	Geolokační souřadnice
Událost	
`id`	Identifikační číslo
`place_id`	Číslo místa, do kterého patří
`name`	Název
`timestamp_unix`	Poslední aktualizace v unixovém formátu
`date_unix`	Den události v unixovém formátu
`times_unix`	Časy události v daném dnu v unixovém formátu
`description`	Popis
`length`	Délka v minutách

Tabulka 5.2: Značky pro popis informací jednotlivých tříd

Kód 1 Příklad XML odpovědi při dotazu `subcategory?category_id=1`

```
<?xml version="1.0" encoding="utf-8"?>
<root>
    <status_code>200</status_code>
    <status_message>OK</status_message>
    <data>
        <node>
            <id>1</id>
            <category_id>1</category_id>
            <name>Kina</name>
            <timestamp_unix>1335599267</timestamp_unix>
        </node>
        <node>
            <id>2</id>
            <category_id>1</category_id>
            <name>Divadla</name>
            <timestamp_unix>1335599062</timestamp_unix>
        </node>
    </data>
</root>
```

6 Návrh serveru

Pro vytvoření serveru jsem zvolil jazyk PHP. Rozhodl jsem se především na základě mých předchozích zkušeností a bral jsem ohled i na to, že je tento jazyk velmi rozšířen a podporován mnoha webhostingy v ČR.

Aby byla zaručena bezpečnost serveru a dal se snadno rozšiřovat, rozhodl jsem se použit PHP framework. K dispozici je jich několik a většina je volně dostupná. Vybral jsem Nette Framework v aktuální verzi 2.0, který má v ČR velmi širokou komunitu vývojářů a je zdarma dostupný jako svobodný software pod licencí New BSD. [10]

Serverová aplikace nemá žádné uživatelské rozhraní, její funkcionalita spočívá ve shromažďování a poskytování dat.

6.1 Požadavky serveru a vybraný webhosting

Požadavky serveru závisí především na použitém frameworku (viz tabulka 6.1, str. 6.1). Kromě toho je nutné předpokládat, že zpracování informačních zdrojů bude probíhat převážně z webových stránek a musí být proto povolena funkce `cURL` pro načítání jejich obsahu.

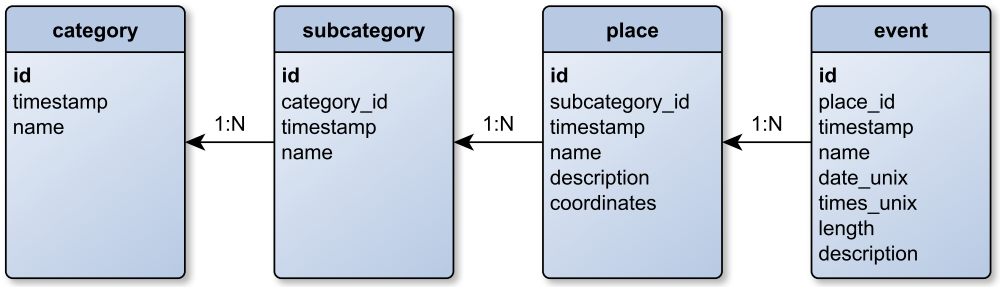
Pro účely bakalářské práce jsem se snažil nalézt webhosting, který by odpovídal zmíněným požadavkům a byl zároveň nabízen zdarma. Jako vhodný se nakonec ukázal webhosting na doméně php5.cz, jehož poskytovatelem je ZONER software, a.s. [11]

Požadavek	**Popis**
PHP verze 5.2.0 nebo vyšší	Vyžadováno frameworkem
.htaccess file protection	Omezení přístupu do složek
.htaccess mod_rewrite	Potřebné pro routování
Function ini_set()	Bezpečnostní důvody
Function error_reporting()	Bezpečnostní důvody
Function flock()	Vyžaduje cache
Register_globals	Musí být vypnuto
Zend.ze1_compatibility_mode	Musí být vypnuto
Session auto-start	Musí být vypnuto
Reflection extension	Vyžadováno frameworkem
SPL extension	Vyžadováno frameworkem
PCRE extension	Vyžadováno frameworkem
ICONV extension	Vyžadováno frameworkem
PHP tokenizer	Vyžadováno frameworkem
PDO extension	Vyžadováno Nette\Database
Multibyte String extension	Vyžadováno
Multibyte String function overloading	Nebezpečné, musí být vypnuto
Memcache extension	Podporováno úložištěm cache
GD extension	Vyžadováno Nette\Image
Bundled GD extension	Vyžadováno Nette\Image
Fileinfo extension or mime_content_type()	Detekce uploadovaných souborů

Tabulka 6.1: Požadavky Nette Frameworku na webový server [12]

6.2 Databáze

Databáze je realizována v jazyce MySQL. Jak je vidět na ERA diagramu (viz obr. 6.1), její model se drží návrhu třídění informací (viz obr. 3.1, str. 10) a každá tabulka tak představuje jednu vrstvu informací.



Obrázek 6.1: ERA diagram databáze

6.3 Struktura serveru

Struktura aplikace serveru vychází z Nette Framework a je doplněna o složky `objects` a `parsers` (viz kód 2, str. 21). Následuje popis hlavních složek.

- **config** - složka s konfiguračními soubory. Kromě základního souboru `config.neon`, který obsahuje např. údaje o připojení k databázi, zde nalezneme ještě `parser.neon` a `xml.neon`. V nich je uložené nastavení parserů a formát XML odpovědi.
- **models** - modely, tedy třídy sloužící jako datová vrstva
  - `ParserModel.php` ukládá informace do databáze během parsování informačních zdrojů
  - `ServiceModel.php` získává informace z databáze při dotazu klienta
- **objects** - složka s datovými strukturami
- **parsers** - složka s parsery, pro každý informační zdroj je zde jeden soubor

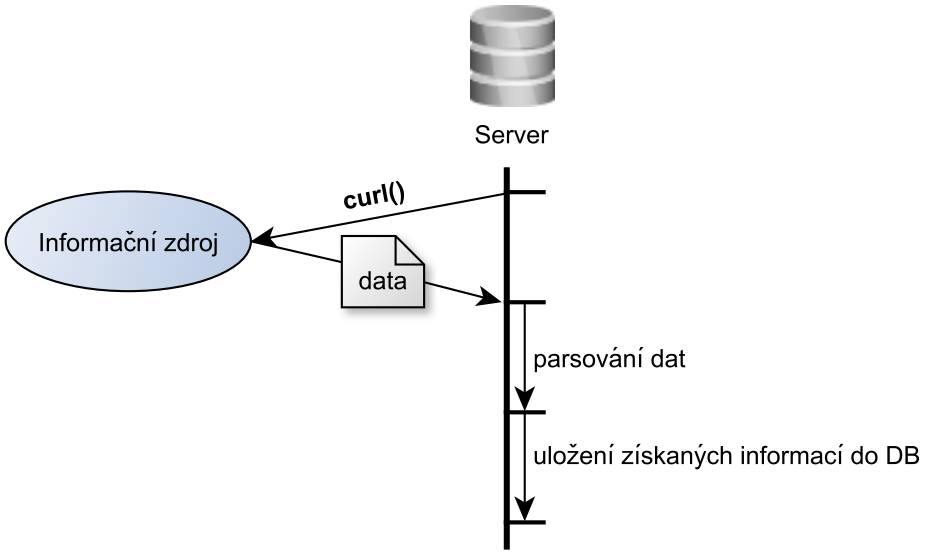
- **presenters** - zde nalezneme presentery, které mají za úkol odpovídat na akci volanou uživatelem
  - `ParserPresenter.php` je volán pro zahájení zpracování informačního zdroje
  - `ServisPresenter.php` je volán pro získání XML se zpracovanými daty
- **templates** - obsahuje šablony pro jednotlivé stránky
- **libs** - knihovny třetích stran včetně samotného Nette Framework
- **log** - složka pro ukládání logovacích souborů při ladění aplikace
- **temp** - dočasné soubory, např. zkompilované šablony
- **www** - výchozí složka Nette Framework obsahující `index.php`

Kód 2 Struktura aplikace serveru a významné soubory

```
- app
    + config
    - models
          ParserModel.php
          ServiceModel.php
    - objects
          Events.php
    + parsers
    - presenters
          ParserPresenter.php
          ServicePresenter.php
    + templates
+ libs
+ log
+ temp
+ www
     index.php
```

6.4 Zpracování informačních zdrojů

Informační zdroj je definován obecně, ale ve většině případů se jedná o webové stránky. Server z nich stahuje data a pomocí parseru z nich získává potřebné informace, které následně ukládá do své databáze (viz obr. 6.2). Problém ale je, že tyto webové stránky mohou mít jakoukoliv formu a pro každý zdroj je tak potřeba vytvořit jiný parser. Přidávání nových zdrojů proto není triviální záležitostí, která by byla proveditelná na uživatelské úrovni.



Obrázek 6.2: Zpracování informačního zdroje

6.5 Tvorba parseru

Vytvořil jsem třídu `BaseParser`, od které se všechny parsery dědí. Obsahuje následují proměnné a metody:

Kód 3 Proměnné a hlavičky funkcí třídy BaseParser

```
private $events;
private $originalData;
private $placeId;

public function __construct($placeId)
public function getPlaceId()
public function getEvents()
public function getOriginalData()
protected function addEvent($name, $date, $dateUnix, $times,
                            $timesUnix, $length, $description)
protected function remote_data_by_post($url, $params=null,
                                       $timeout = 5)
protected function remote_data_by_get($url, $params=null,
                                      $timeout = 5)
```

V odděděné třídě je nutné dopsat funkci pro zpracování dat, typicky pojmenovanou `parseData()`. Pro stažení dat z webů jsou připraveny metody `remote_data_by_post()` a `remote_data_by_get()`, které používají metodu cURL. Ze získaných dat je potřeba dostat následující informace vztahující se ke každé události:

- název
- datum v textové podobě
- datum v unixovém formátu
- časy události v daném dnu v textové podobě oddělené čárkami
- časy události v daném dnu v unixovém formátu oddělené čárkami
- délku události v minutách
- popis

Každá takto získaná událost se musí přidat pomocí funkce `addEvent()`. Ta ji totiž správným způsobem uloží do objektu `Events`, který se nakonec z parseru vrací jako výsledek.

6.6 Použití parseru, uložení dat

Parsery se budou pouštět při určitém volání serveru. K tomu v Nette Frameworku slouží presentery. Pro parsery jsem vytvořil `ParserPresenter` a jeho volání se provádí následujícím způsobem:

```
url_serveru/www/parser/render/id
```

Položky `render` a `id` slouží jako "parametry" stránky. To je dané frameworkem. Presentery se v něm vždy dále dělí na rendery, id není povinné. Při zadání adresy bez renderu se volá renderDefault(). Určil jsem, že rendery budou představovat název parseru, volání tedy bude:

```
url_serveru/www/parser/nazev_parseru
```

Id lze využít například u náročného parseru, u něhož se parsování musí provádět v několika krocích.

V praxi se parser používá velice jednoduše. Nejprve se vytvoří jako objekt, přičemž parametrem v konstruktoru je číslo místa, ke kterému se vztahuje. Poté se nad ním zavolá metoda pro získání dat, která se typicky jmenuje `parseData()`. Nakonec se metodou `getEvents()` získá objekt s událostmi, který bude pomocí modelu `ParserModel` uložen do databáze.

Společně s renderem nového parseru ve třídě ParsePresenter je nutné vytvořit i odpovídající šablonu ve složce `templates/Parser/` (např. `novyParser.latte`). I přesto, že stránka nemusí mít žádný uživatelský výstup, Nette Framework to vyžaduje.

Kód 4 Příklady použití parserů ve třídě ParserPresenter

```
class ParserPresenter extends BasePresenter
{
    public function renderDefault()
    {
    // není zadané jméno parseru, nic se neprovádí
    }
    public function renderParserJedna()    // voláno bez id
    {
        $parser = new Parser1(5);          // vytvoření parseru
                                           // pro místo 5
        $parser->parseData();              // zpracování dat
        $events = $parser->getEvents();    // získání dat
        $this->parserModel->updateEvents($events); //uložení
    }
    public function renderParserDva($id)   // voláno s id
    {
        if($id!=1) return;
        $parser = new Parser2(7);          // vytvoření parseru
                                           // pro místo 7
        $parser->parseData();              // zpracování dat
        $events = $parser->getEvents();    // získání dat
        $this->parserModel->updateEvents($events); //uložení
    }
}
```

6.7 Vytvořené parsery

V rámci bakalářské práce jsem vytvořil parsery dvou informačních zdrojů. Jedná se o multikina Cinestar a Cinemacity.

6.7.1 Parser Cinemacity

Společnost Cinemacity nabízí centralizované webové stránky s programem všech svých kin. Parser tedy nezpracovává pouze jedno místo, ale je vytvořen obecnějším způsobem tak, aby získával informace o všech kinech. Základem pro fungování parseru je adresa informačního zdroje, ze které se získávají potřebná data. V tomto případě se jedná o adresu `http://cinemacity.cz/scheduleInfoRows`, která je společná pro všechna kina. Konrétní požadavek se specifikuje připojením následujících parametrů:

- **locationId** - Slouží k určení pobočky kina. Pro jednotlivá kina bylo nutné zjistit odpovídající kódy (viz tabulka 6.2).
- **date** - Datum požadovaného programu ve formátu `DD/MM/YYYY`.
- **venueTypeId** - Účel tohoto parametru jsem nezjistil, ale vždy má hodnotu 0.

Kino	**locationId**
Praha Flora	1010105
Praha Galaxie	1010113
Praha Letňany	1010110
Praha Nový Smíchov	1010101
Praha Slovanský dům	1010111
Praha Zličín	1010104
Brno Olympia	1010103
Brno Velký Špalíček	1010107
Plzeň	1010106
Liberec	1010102
Pardubice	1010108
Ústí nad Labem	1010109
Ostrava	1010114

Tabulka 6.2: Parametr locationId pro adresu infromačního zdroje Cinemacity

Chceme-li získat např. data o programu kin v Plzni pro den 4.5.2012, bude adresa vypadat následovně:

`http://cinemacity.cz/scheduleInfoRows?locationId=1010106 &date=04/05/2012&venueTypeId=0`

Musí být zavolána metodou GET a výsledkem, který se vrátí, je HTML tabulka s následujícím formátem:

Kód 5 Ukázková data pro získání informací ze zdroje Cinemacity

```
...
<tbody>
  ...
  <tr>
    <td class="featureName">
      <a href="#" class="featureLink" data-feature_code="9765">
        Bitevní lod'
      </a>
    </td>
    <td>MP-12</td>
    <td>ČT</td>
    <td>129</td>
    ...
    <td class="prsnt">
      <a class="presentationLink expired" href="#">
        16:50
      </a>
    </td>
    ...
  </tr>
  ...
</tbody>
...
```

Pro zpracování této tabulky a získání konrétních informací o programech kin jsem použil volně dostupný `PHP Simple HTML DOM Parser`. [13]

Informační zdroj Cinemacity obsahuje celkem 13 kin a programy se stahují až týden dopředu (podle aktuální dostupnosti). Celý proces parsování je tak časově relativně náročný. Pro každé kino se zpracuje a uloží do databáze

přibližně 100 filmových představení, což trvá na použitém serveru asi 7 vteřin. Jelikož má server omezení 30 vteřin na vykonání jednoho dotazu, bylo nutné rozdělit parsování na několik částí. Spouštění parseru tedy probíhá opakovaným voláním adresy `url_serveru/www/parser/cinemacity/id` s hodnotami id 1 až 4 (viz kapitola 6.6).

6.7.2 Parser Cinestar

Cinestar stejně jako Cinemacity nabízí centralizované stránky s programem pro všechna svá kina. I parser proto bude fungovat na podobném principu. Rozdíl je v adresách pro získávání dat, nyní je pro každé kino jiná (viz tabulka 6.3). V parametrech se určuje pouze datum v unixovém formátu.

Kino	**Adresa**
Praha Černý Most	`http://praha9.cinestar.cz/program_multikino.php`
Praha Anděl	`http://praha5.cinestar.cz/program_multikino.php`
Mladá Boleslav	`http://boleslav.cinestar.cz/program_multikino.php`
Liberec	`http://liberec.cinestar.cz/program_multikino.php`
Hradec Králové	`http://hradec.cinestar.cz/program_multikino.php`
Ostrava	`http://ostrava.cinestar.cz/program_multikino.php`
Olomouc	`http://olomouc.cinestar.cz/program_multikino.php`
Pardubice	`http://pardubice.cinestar.cz/program_multikino.php`
Plzeň	`http://plzen.cinestar.cz/program_multikino.php`
Jihlava	`http://jihlava.cinestar.cz/program_multikino.php`
České Budějovice	`http://budejovice.cinestar.cz/program_multikino.php`

Tabulka 6.3: Adresy informačních zdrojů pro kina Cinestar

Adresu je v tomto případě nutné volat metodou `POST`. Příklad pro získání dat o programu kin v Plzni pro den 4.5.2012 je:

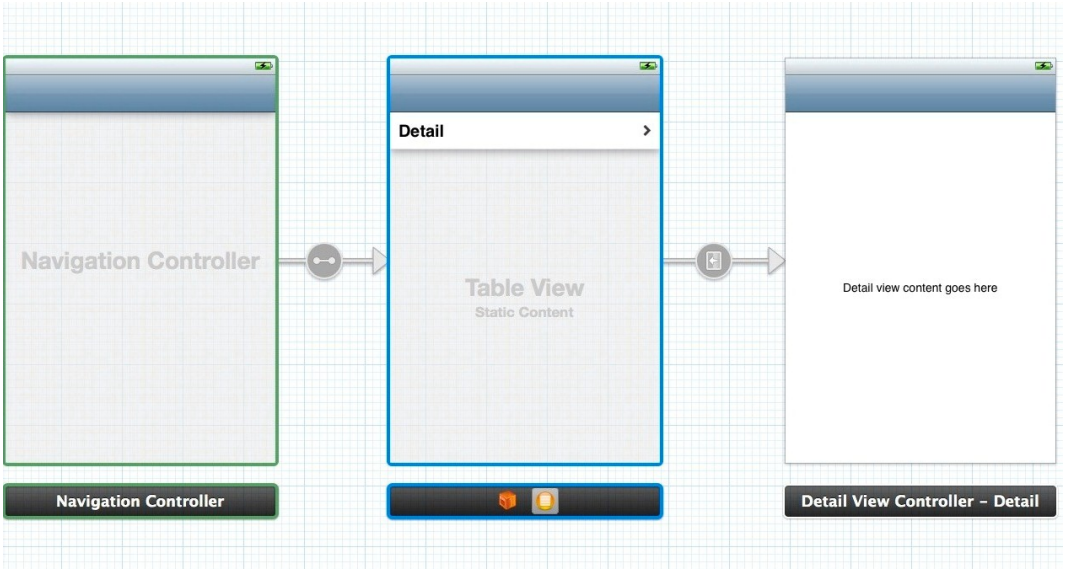
```
http://plzen.cinestar.cz/program_multikino.php?datum=1336089600
```

Výsledkem dotazu je opět HTML tabulka, kterou zpracovávám pomocí `PHP Simple HTML DOM Parser` [13]. Zpracovává se 11 kin až týden dopředu (podle aktuální dostupnosti), což opět překračuje 30 vteřinový limit serveru. Parser se tedy volá v několika krocích voláním adresy `url_serveru/www/parser/cinestar/id`, nyní s hodnotami id 1 až 3 (viz kapitola 6.6).

7 Návrh aplikace

Při vytváření aplikace jsem používal vývojové prostředí Xcode ve verzi 4.2. Ta přináší dvě důležité změny, které jsem využil při vývoji. První z nich je Automatic Reference Counting, což je funkce umožňující automatickou správu paměti pro objekty v Objective C. Programátor se díky tomu nemusí starat o uvolňování objektů z paměti a při překladu programu je kód pro uvolnění automaticky doplněn.

Druhou novinkou je funkce Storyboarding, která slouží k celkovému návrhu aplikace. Vizuální prostředí se díky tomu nenavrhuje pro každou obrazovku zvlášť, ale k dispozici je náhled kompletní aplikace se všemi obrazovkami a se znázorněnými přechody mezi nimi (viz obrázek 7.1).



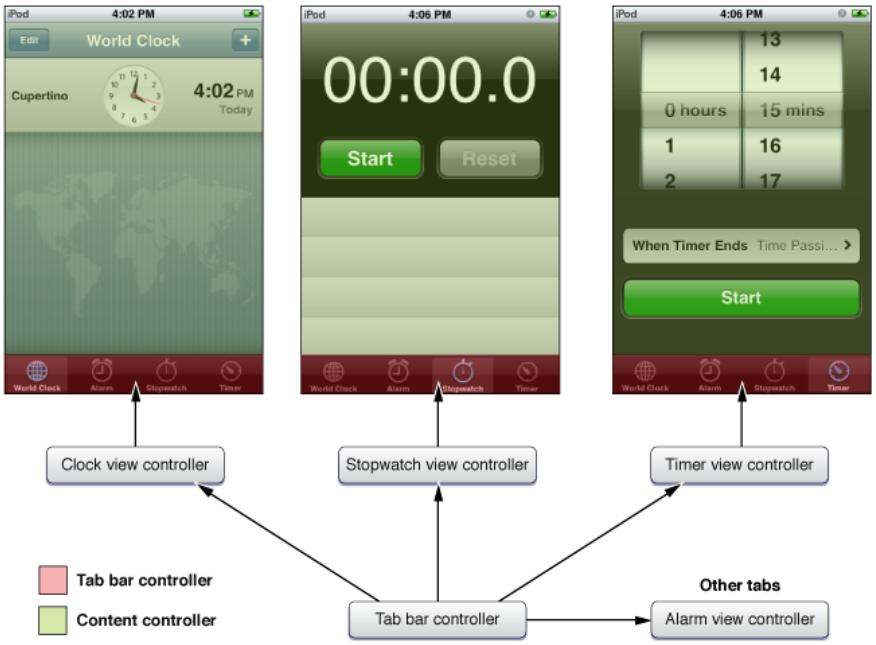
Obrázek 7.1: Ukázka storyboardu

7.1 Struktura aplikace

Aplikace pro systém iOS mají kvůli malým úhlopříčkám displejů velmi omezený prostor pro zobrazování svého obsahu. Můžou využívat pouze jedno okno a to je pro složitější aplikace nedostačující. Jsou proto k dispozici ob-

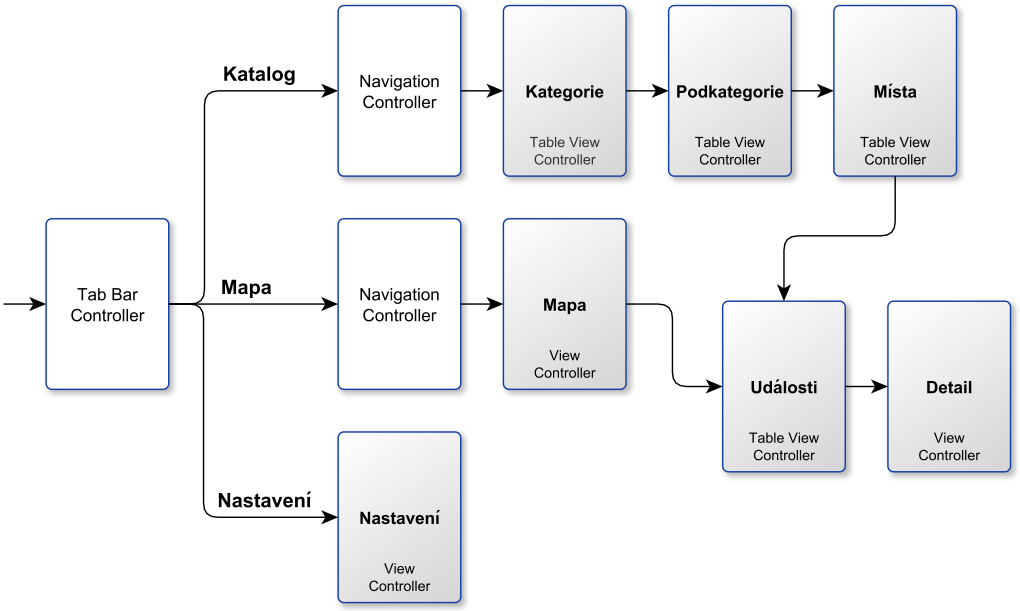
jekty **View Controller**, které poskytují infrastrukturu pro správu obsahu aplikace. Jejich použití umožní členit uživatelské rozhraní na více částí a vytvořit různé pohledy aplikace. View controllery lze rozdělit na následující dvě skupiny. [14]

- **Content View Controller** pomocí jedné nebo více komponent **View** zobrazuje obsah aplikace na obrazovce. Běžně se používá pro zobrazení dat uživateli, pro jejich získání a nebo k vykonání určitě činnosti. K dispozici je několik odvozených tříd a asi nejpoužívanější z nich je **Table View Controller**, která je přizpůsobena pro zobrazování dat ve stylu tabulky. Vzhled buněk může být libovolně upraven a může obsahovat jakékoliv další komponenty.
- **Container View Controller** slouží pro organizaci dalších view controllerů a zobrazování jejich obsahu. Funguje jako navigační prvek v aplikaci a zajišťuje přechody mezi jednotlivými pohledy. Nejdůležitější a nejpoužívanější z nich jsou **Navigation Controller** a **Tab Bar Controller** (viz obr. 7.2).



Obrázek 7.2: Tab Bar Controller v aplikaci Hodiny

Při návrhu aplikace jsem bral ohled na velikost displeje a snažil se všechna data, která potřebuji zobrazit, vhodně rozdělit do jednotlivých pohledů. Výsledek je znázorněn na obrázku 7.3, kompletní storyboard potom je k dispozici v příloze.



Obrázek 7.3: Schéma struktury aplikace

Aplikace je pomocí tab bar controlleru rozdělena na tři základní sekce:

- **Katalog** zobrazuje všechna dostupná data v tabulkových výpisech. Katalog se drží návrhu třídění informací (viz kapitola 3.1) a pro každou třídu informací je vytvořen samostatný pohled s tabulkou pomocí table view controlleru. Pro přechod mezi nimi je použit navigation controller.
- **Mapa** nabízí alternativní zobrazení všech míst z katalogu, jejich polohy jsou znázorněny jako špendlíky na mapě. Jedná se o základní view controller. Z jednotlivých míst je potom pomocí navigation controlleru umožněn přechod na daný seznam událostí.
- **Nastavení** je pouze základní view controler.

Každý pohled v aplikaci v sobě uchovává data, která zobrazuje. Bylo nutné definovat strukturu, ve které budou tato data uložena. Určil jsem, že se

bude jednat o pole objektů, přičemž objekt reprezentuje konkrétní informaci a musí tak být pro každý pohled jiný. V tabulce 7.1 jsou uvedeny objekty pro jednotlivé pohledy a informace, které obsahují.

Category - pro pohled Kategorie	
`ID`	identifikační číslo
`name`	název
`timestamp`	čas v unixovém formátu
`stringTime`	čas jako řetězec
Subategory - pro pohled Podkategorie	
`ID`	identifikační číslo
`name`	název
`timestamp`	čas v unixovém formátu
`stringTime`	čas jako řetězec
`categoryId`	číslo kategorie, do které patří
Place - pro pohledy Místa a Mapa	
`ID`	identifikační číslo
`subcategoryId`	číslo podkategorie, do které patří
`name`	název
`timestamp`	čas v unixovém formátu
`stringTime`	čas jako řetězec
`description`	popis
`coordinates`	souřadnice
Event - pro pohled Události	
`ID`	identifikační číslo
`placeId`	číslo místa, do kterého patří
`name`	název
`timestamp`	čas v unixovém formátu
`stringTime`	čas jako řetězec
`date`	datum události v unixovém formátu
`stringDate`	datum události v textové podobě
`times`	pole časů v unixovém formátu
`stringTimes`	pole časů v textové podobě
`description`	popis
`length`	délka v minutách

Tabulka 7.1: Použité objekty a informace, které obsahují

7.2 Komunikace se serverem

Protože komunikace se serverem probíhá na mnoha místech aplikace, vytvořil jsem třídu **ServerController**, která vše zajišťuje. Její úlohou je připojit se k serveru, načíst z něj požadovaná data a vrátit je jako výsledek. Zobrazuje také informační hlášení při načítání dat nebo při nedostupnosti internetového připojení.

Protože se během spojení může přenášet i větší množství dat, což nějakou dobu trvá, je realizováno asynchronně. Tím se předejde pozastavení celé aplikace při čekání na dokončení přenosu. Z tohoto důvodu třída obsahuje protokol ServerControllerDelegate, který je při použití nutno přijmout a naimplementovat jeho jedinou metodu `serverController:didReceiveData:`. Ta se volá ve chvíli, kdy se dokončí přenos a vrací získaná data.

Základní metodou třídy je `sendToServerOnUrl:WithParams:`. Jejími parametry jsou url adresa a parametry, se kterými se pošle dotaz na vzdálený server. Pro připojení používá **NSURLConnection**, které se inicializuje následujícím způsobem:

Kód 6 Inicializace NSURLConnection

```
NSURL *nsURL = [[NSURL alloc] initWithString:initUrl];
NSMutableURLRequest *nsMutableURLRequest =
                [[NSMutableURLRequest alloc] initWithURL:nsURL];
[nsMutableURLRequest setHTTPMethod:@"POST"];
NSData *paramData = [paramDataString
                     dataUsingEncoding:NSUTF8StringEncoding];
[nsMutableURLRequest setHTTPBody: paramData];
nsUrlConnection=[[NSURLConnection alloc]
                  initWithRequest:nsMutableURLRequest
                         delegate:self];
```

Proměnné `initUrl` a `paramDataString` jsou parametry funkce. Jak je v kódu vidět, požadavky se na server posílají metodou **POST**. Metodu `sendToServerOnUrl:WithParams:` využívají všechny následující metody:

- `loadXmlCategoriesAll` - načtení všech dostupných kategorií
- `loadXmlSubcategoriesByCategoryId:` - načtení podkategorií, které patří do kategorie určené parametrem

- `loadXmlSubcategoriesAll` - načtení všech dostupných podkategorií
- `loadXmlPlacesBySubcategoryId:` - načtení míst, které patří do podkategorie určené parametrem
- `loadXmlPlacesAll` - načtení všech dostupných míst
- `loadXmlEventsByPlaceId:` - načtení událostí, které patří do místa určeného parametrem
- `loadXmlEventsAll` - načtení všech dostupných událostí

Dále bylo nutné ve třídě přijmout a naimplementovat metody protokolu NSURLConnection:

- `connection:didReceiveResponse` - odezva na připojení
- `connection:didReceiveData` - přijmula se data
- `connection:didFinishLoading` - přijímání dat bylo dokončeno, v tuto chvíli se volá metoda delegáta `serverController:didReceiveData`
- `connection:didFailWithError` - nastala chyba
- `connection:willSendRequest:redirectResponse` - došlo k přesměrování

7.3 Parsování XML

Přijatá XML data ze serveru je nutné zpracovat a získat z nich potřebné informace. K tomu slouží třída **XmlParser**. Obsahuje pouze tzv. metody třídy, což je obdoba statických metod v jiných programovacích jazycích. S třídou se tedy nepracuje jako s objektem, pouze se volají její metody:

- `parseCategories:`
- `parseSubcategories:`
- `parsePlaces:`
- `parseEvents:`

Parametrem uvedených metod jsou vždy data přijatá ze serveru a vracejí se již hotové objekty naplněné zpracovanými informacemi, se kterými se pracuje v aplikaci. K parsování jsem použil knihovnu **Google Data Objective-C Client** [15].

7.4 Kešování dat

Kešování dat je realizováno pomocí ukládání do souborů a probíhá při každé aktualizaci ze serveru. Protože množství dat potřebných k uložení může být při rostoucím počtu informačních zdrojů velké, nepoužívá se pouze jeden soubor. Čas načítání a ukládání by totiž mohl být příliš dlouhý a aplikace by se stala špatně použitelnou. Zvlášť jsou proto ukládány informace o kategoriích, podkategoriích a místech. V případě událostí je členění ještě podrobnější a události vztahující se k určitému místu jsou vždy uloženy ve vlastním souboru (viz tabulka 7.2).

Soubor	**Použití**
`SavedCategories.plist`	uložení kategorií
`SavedSubcategories.plist`	uložení podkategorií
`SavedPlaces.plist`	uložení míst
`SavedEventsForPlace_1.plist`	uložení událostí pro místo 1
`SavedEventsForPlace_2.plist`	uložení událostí pro místo 2
`SavedEventsForPlace_3.plist`	uložení událostí pro místo 2
...	
`SavedEventsForPlace_n.plist`	uložení událostí pro místo n

Tabulka 7.2: Použité soubory pro ukládání dat

Pro ukládání a načítání souborů slouží třída **StorageController**. Jejími hlavními metodami jsou: `writeObject:ToFile:` a `readObjectFromFile:`, které zapisují a čtou libovolný objekt do nebo ze souboru. Každá aplikace v iOS má pro práci se soubory k dispozici pouze jednu svojí složku, tzv. sandbox. Cestu k této složce lze zjistit pomocí standardní služby `NSSearchPathForDirectoriesInDomains`, kterou jsem pro zjednodušení použil v metodě `storagePatch` (viz kód 7). Implementace čtení a zapisování objektů je ukázána v kódu 8.

Kód 7 Metoda **storagePatch** pro zjištění složky aplikace

```
+(NSString *)storagePath {
  return [NSSearchPathForDirectoriesInDomains(
    NSDocumentDirectory,NSUserDomainMask,YES) objectAtIndex: 0];
}
```

Kód 8 Čtení objektu ze souboru a zápis objektu do souboru

```
// čtení:
NSString *storagePath = [self storagePath];
NSString *filePath = [storagePath
                      stringByAppendingPathComponent:fileName];
return [NSKeyedUnarchiver unarchiveObjectWithFile:filePath];

// zápis:
NSString *storagePath = [self storagePath];
NSString *filePath = [storagePath
                      stringByAppendingPathComponent:fileName];
[NSKeyedArchiver archiveRootObject:object toFile:filePath];
```

Dále jsou ve třídě k dispozici metody zajišťující již konkrétní načítání a ukládání dat kategorií, podkategorií, míst a událostí:

- `loadAllCategories` - nahrává všechny kategorie
- `saveAllCategories:` - ukládá všechny kategorie
- `saveSubcategories:forCategoryId:` - ukládá podkategorie pro danou kategorie
- `loadSubcategoriesForCategoryId:` - nahrává podkategorie z dané kategorie
- `saveAllSubcategories:` - ukládá všechny podkategorie
- `loadAllSubcategories` - nahrává všechny podkategorie
- `savePlaces:forSubcategoryId:` - ukládá místa k dané podkategorie
- `loadPlacesForSubcategoryId:` - nahrává místa z dané podkategorie

- `loadAllPlaces` - nahrává všechna místa
- `saveAllPlaces:` - ukládá všechna místa
- `saveEvents:forPlaceId:` - ukládá události k danému místu
- `loadEventsForPlaceId:` - nahrává události z daného místa
- `saveAllEvents:` - ukládá všechny události

7.5 Práce s mapami

Pohled aplikace, kde je zobrazena mapa, je reprezentován třídou **Map ViewController**, odvozené od základního view controlleru. Stejně jako ostatní pohledy si musí uchovávat informace o zobrazovaných datech. V tomto případě se jedná o místa, která jsou ve výsledku reprezentována jako špendlíky na mapě. Načítají se vždy při inicializaci třídy nebo při aktualizaci ze serveru. Pro zobrazování samotné mapy je použita třída **Map View** dostupná v **MapKit framework**. Sama o sobě při správné inicializaci nabízí náhled mapy, který lze libovolně posouvat, přibližovat a oddalovat. Navíc zobrazuje i aktuální polohu uživatele. Zbývá tedy zobrazit všechna potřebná místa na mapě. Ty jsou realizována pomocí tzv. anotací, což jsou objekty třídy **MKAnnotatin**. Při jejich vytváření je potřeba kromě jejich názvu určit také jejich pozici na mapě, což se provádí pomocí objektu třídy **MKCoordinateRegion**. Anotace se na mapu přidává pomocí funkce `addAnnotation:` (viz kód 9).

Kód 9 Přidání anotace do mapy

```
MKCoordinateRegion region = { {0.0, 0.0}, {0.0, 0.0} };
region.center.latitude = 49.72488;
region.center.longitude = 13.350891;
region.span.longitudeDelta = 0.02f;
region.span.latitudeDelta = 0.02f;
MapAnnotation *ann = [[MapAnnotation alloc] init];
ann.title = @"Jméno místa";
ann.subtitle = @"Popis místa";
ann.coordinate = region.center;
[mapView addAnnotation:ann];
```

Samotné anotace však nejsou žádným způsobem na mapě zobrazeny. U každé je nejdříve nutné nastavit objekt třídy **MKAnnotationView**, který představuje špendlík na mapě. K tomu je potřeba, aby třída MapViewController přijmula protokol **MKMapViewDelegate** a byla naimplementována metoda `mapView:viewForAnnotation:`. Jejím parametrem je anotace a návratovou hodnotou objekt třídy **MKAnnotationView**, tedy špendlík. V těle metody je třeba tento špendlík vytvořit a vrátit ho, čímž bude přiřazen k dané anotaci. Při tom lze např. specifikovat jeho barvu nebo přidat tlačítko do detailu. To jsem využil, aby byl umožněn přechod na seznam událostí daného místa.

7.6 Interakce s událostmi

Aplikace nabízí u detailu události odeslání SMS, emailu nebo její přidání do kalendáře. Tato akce se vztahuje vždy k vybranému času.

Odeslání SMS

Pro odeslání SMS slouží třída **MFMessageComposeViewConroller**. Z názvu je patrné, že je odvozena od základního view cotrolleru, jedná se tedy o samostatný pohled aplikace. Nabízí předpřipravené rozhraní totožné s aplikací Zprávy ze systému iOS. Lze předvyplnit tělo zprávy a příjemce. Inicializace a zobrazení pohledu je ukázáno v kódu 10.

Kód 10 Zobrazení obrazovky pro odeslání SMS

```
MFMessageComposeViewController *controller =
                  [[MFMessageComposeViewController alloc] init];
  if([MFMessageComposeViewController canSendText])
  {
     controller.body = [NSString  stringWithFormat:
                          @"Ahoj, co takhle jít na %@, %@ v %@?",
                               eventTitle, eventDate, eventTime];
     controller.messageComposeDelegate = self;
     controller.recipients = nil;
     [self presentModalViewController:controllere animated:YES];
  }
```

Odeslání emailu

Email se odesílá stejným způsobem jako zprávy SMS. Rozdíl je pouze v použité třídě, kterou je tentokrát **MFMailComposeViewController**. Kromě těla zprávy a příjemců se u ní nastavuje ještě předmět emailu (viz kód 11).

Kód 11 Zobrazení obrazovky pro odeslání emailu

```
MFMailComposeViewController *mailer =
                    [[MFMailComposeViewController alloc] init];
mailer.mailComposeDelegate = self;
[mailer setSubject:@"Pozvánka na akci"];
NSString *emailBody = [NSString  stringWithFormat:
                         @"Ahoj, co takhle jít na %@, %@ v %@?",
                              eventTitle, eventDate, eventTime];
[mailer setMessageBody:emailBody isHTML:NO];
[self presentModalViewController:mailer animated:YES];
```

Přidání události do kalendáře

Přístup ke kalendáři v telefonu probíhá pomocí třídy **EKEventStore**. Pomocí ní se do kalendáře přidá událost reprezentovaná objektem třídy **EKEvent**, u které se musí určit čas začátku, konce (oba údaje v unixovém formátu), její název a kalendář, který pro přidání chceme použít. V ukázkovém kódu 12 je použit výchozí kalendář systému.

Kód 12 Přidání události do kalendáře

```
EKEventStore *eventStore = [[EKEventStore alloc] init];
EKEvent *newEvent = [EKEvent eventWithEventStore:eventStore];
newEvent.title = eventTitle;
newEvent.startDate =
           [NSDate dateWithTimeIntervalSince1970:eventTimeUnix];
newEvent.endDate =
      [newEvent.startDate dateByAddingTimeInterval:eventLength];
[newEvent setCalendar:[eventStore defaultCalendarForNewEvents]];
[eventStore saveEvent:newEvent span:EKSpanThisEvent  error:nil];
```

8 Testování

8.1 Funkčnost parserů

Server byl v rámci testování bakalářské práce v provozu na doméně `http://george.php5.cz`. Nemá žádné uživatelské prostředí a tak test spočíval v kontrole uložených dat pro vybrané informační zdroje. Postupně jsem spustil parsery pro Cinestar a Cinemacity (viz kapitola 6.6) zadáním následujících adres ve webovém prohlížeči:

```
http://george.php5.cz/www/parser/cinemacity/1
http://george.php5.cz/www/parser/cinemacity/2
http://george.php5.cz/www/parser/cinemacity/3
http://george.php5.cz/www/parser/cinemacity/4

http://george.php5.cz/www/parser/cinestar/1
http://george.php5.cz/www/parser/cinestar/2
http://george.php5.cz/www/parser/cinestar/3
```

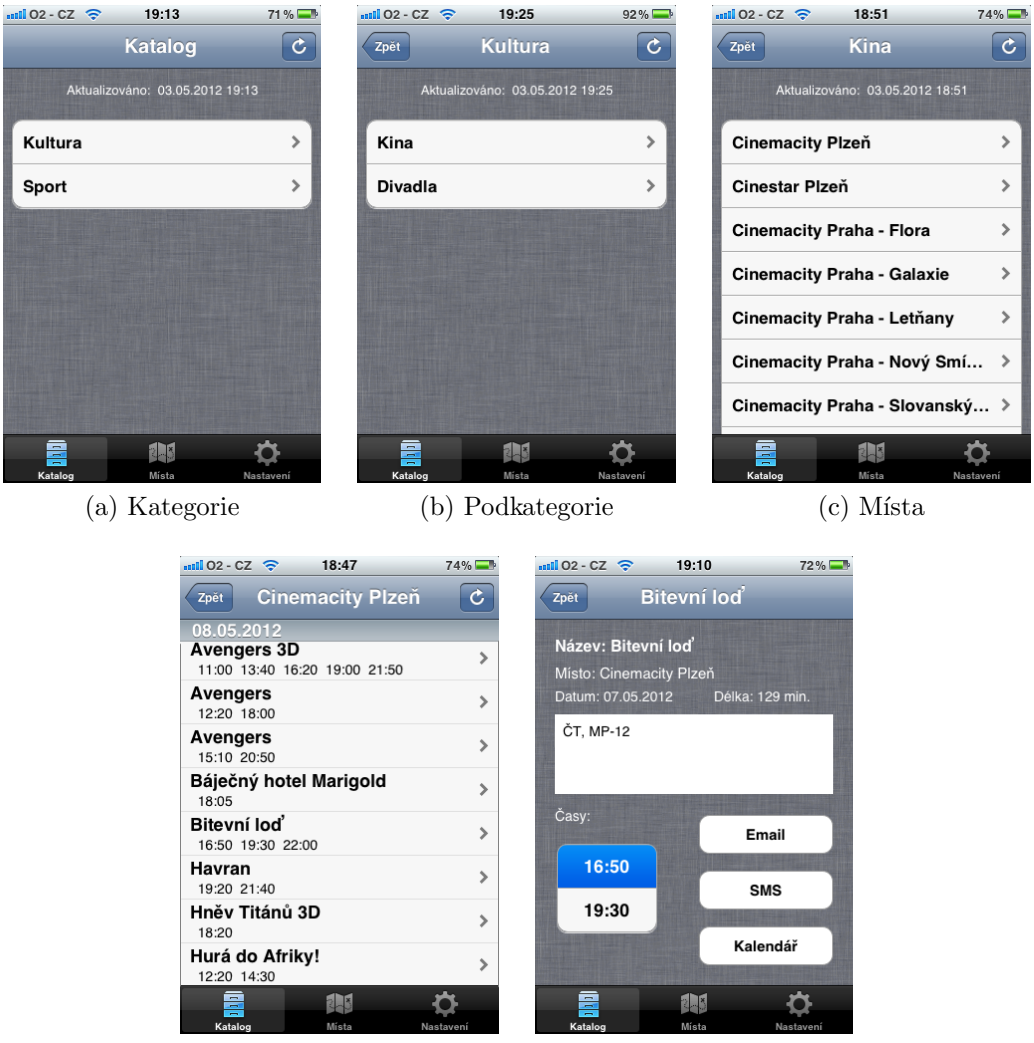
Následně jsem se pomocí nástroje phpMyAdmin připojil k databázi. Výsledkem je 2528 událostí, což odpovídá očekávanému průměrnému počtu 100 představení pro každé z 24 kin (13 kin Cinemacity, 11 kin Cinestar). Toto číslo je závislé na aktuálním programu, který je k dispozici. Kontrola jednotlivých událostí probíhala z důvodu jejich velkého množství náhodným výběrem. Všechna zkoumaná data byla uložena v pořádku.

8.2 Běh aplikace

Na testování aplikace jsem během vývoje používal iPhone Simulator, který je součástí prostředí Xcode. Testy probíhaly na počítači Apple Mac mini a MacBook Pro. Funkčnost finální verze jsem ověřoval i v reálném zařízení, mobilním telefonu iPhone 3GS, což umožnilo v praxi vyzkoušet například geolokační funkce. Z něj jsou také pořízeny všechny snímky aplikace v této kapitole.

Byly otestovány následující situace:

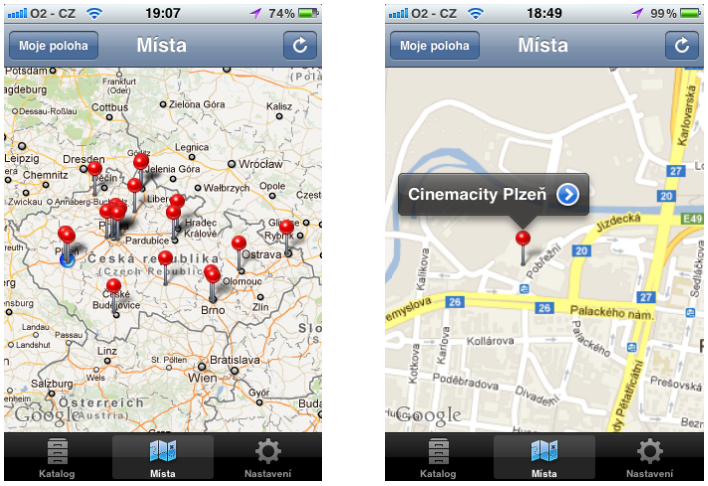
- zobrazení informací v katalogu (viz obr. 8.1)
- zobrazení míst na mapě (viz obr. 8.2)
- interakce s událostmi(viz obr. 8.3)
- chování aplikace při nedostupném internetovém připojení (viz obr. 8.5)
- chování aplikace bez načtených dat (viz obr. 8.4)



(a) Kategorie (b) Podkategorie (c) Místa

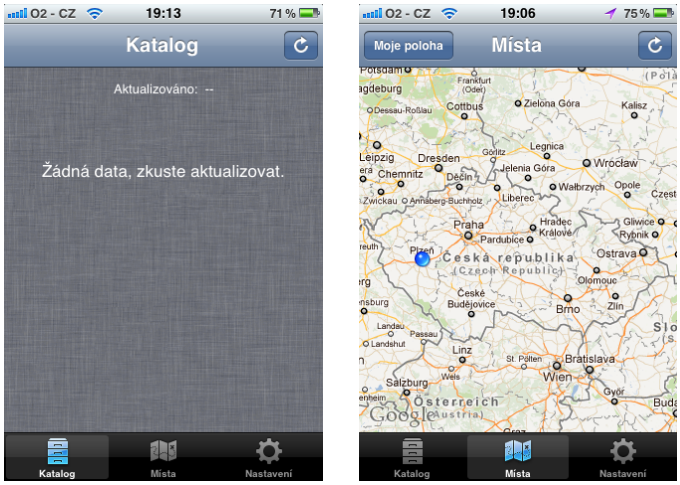
(d) Události (e) Detail události

Obrázek 8.1: Zobrazení informací v katalogu



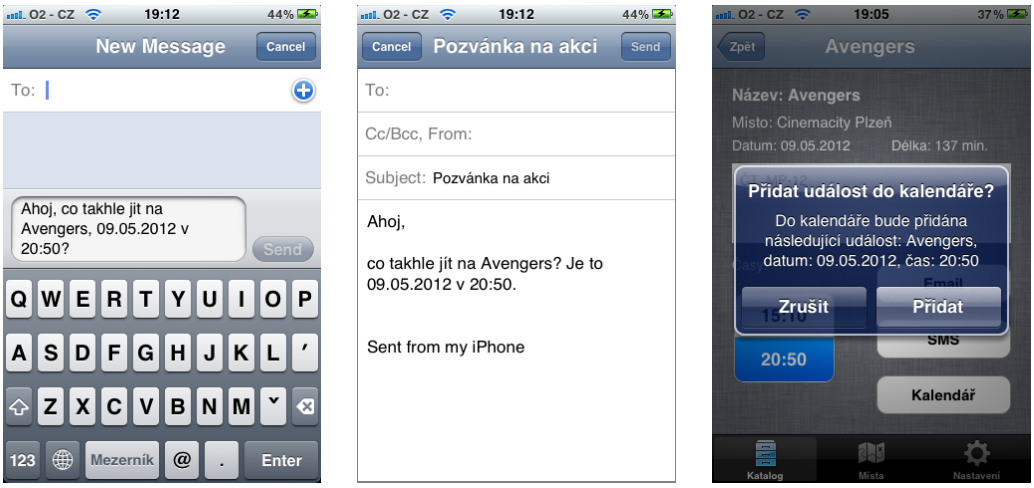
(a) Celá ČR (b) Detail na místo

Obrázek 8.2: Zobrazení míst na mapě



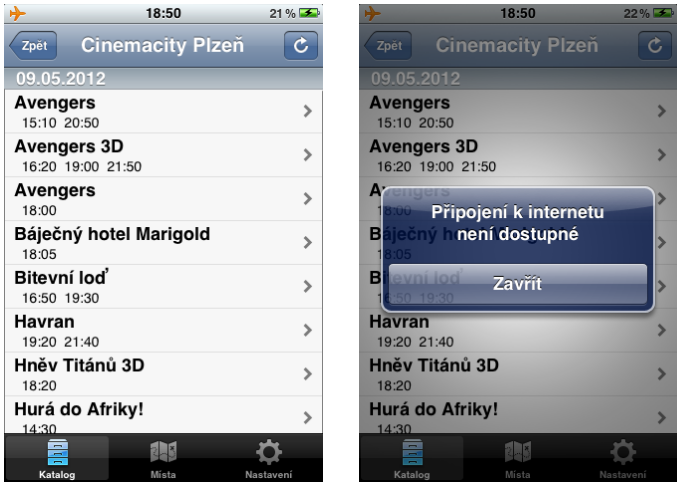
(a) Katalog (b) Mapa

Obrázek 8.3: Chování aplikace bez načtených dat



(a) Odeslání SMS (b) Odeslání emailu (c) Přidání události do kalendáře

Obrázek 8.4: Interakce s událostmi

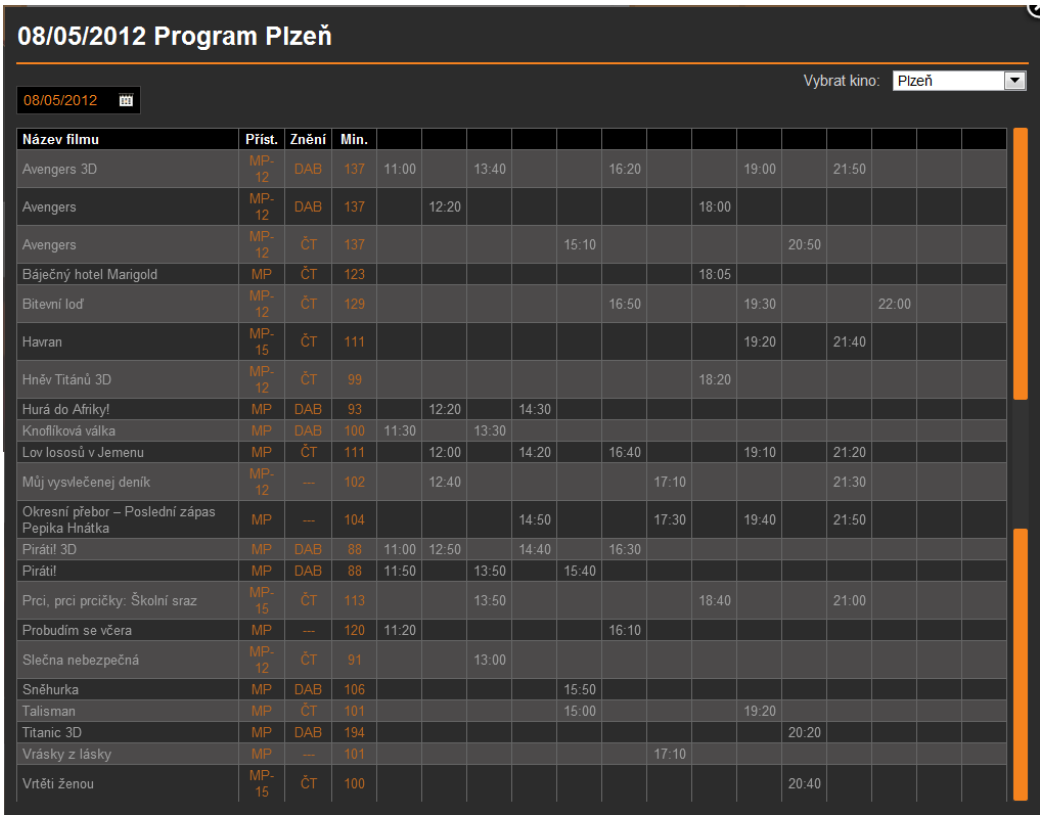


(a) Zobrazení uložených dat (b) Pokus o aktualizaci

Obrázek 8.5: Chování aplikace při nedostupném internetovém připojení

8.3 Validita zobrazovaných informací

Pro kontrolu validity zobrazovaných informací jsem porovnal výstup aplikace s primárním zdrojem informací, jímž jsou v případě programů filmových představení webové stránky daného kina (viz obr. 8.6 a 8.7, str. 44).

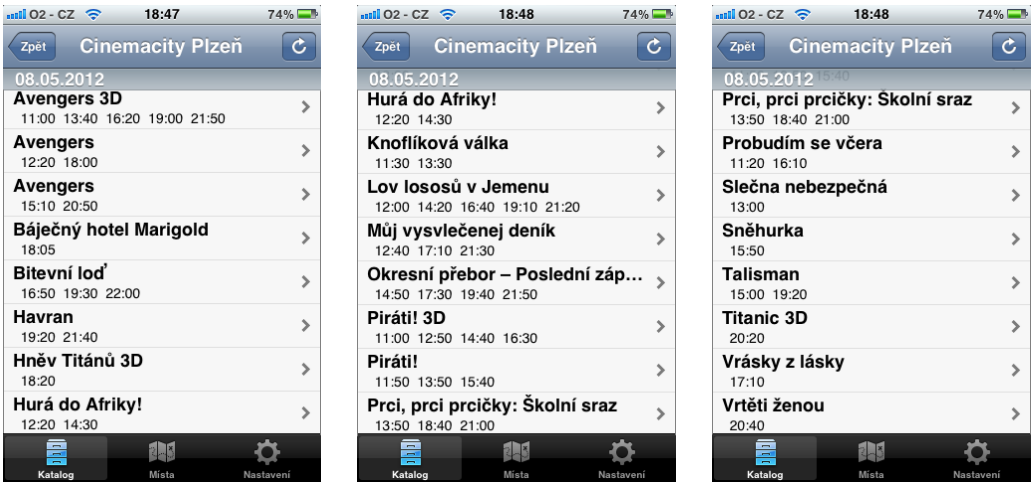


08/05/2012 Program Plzeň

08/05/2012 Vybrat kino: Plzeň

Název filmu	Přís.	Znění	Min.													
Avengers 3D	MP-12	DAB	137	11:00		13:40			16:20			19:00		21:50		
Avengers	MP-12	DAB	137		12:20						18:00					
Avengers	MP-12	ČT	137					15:10					20:50			
Báječný hotel Marigold	MP	ČT	123								18:05					
Bitevní loď	MP-12	ČT	129						16:50			19:30			22:00	
Havran	MP-15	ČT	111									19:20		21:40		
Hněv Titánů 3D	MP-12	ČT	99								18:20					
Hurá do Afriky!	MP	DAB	93		12:20		14:30									
Knoflíková válka	MP	DAB	100	11:30		13:30										
Lov lososů v Jemenu	MP	ČT	111		12:00		14:20		16:40			19:10		21:20		
Můj vysvlečenej deník	MP-12	---	102		12:40					17:10				21:30		
Okresní přebor – Poslední zápas Pepika Hnátka	MP	---	104				14:50			17:30		19:40		21:50		
Piráti! 3D	MP	DAB	88	11:00	12:50		14:40		16:30							
Piráti!	MP	DAB	88	11:50		13:50		15:40								
Prci, prci prcičky: Školní sraz	MP-15	ČT	113			13:50					18:40			21:00		
Probudím se včera	MP	---	120	11:20					16:10							
Slečna nebezpečná	MP-12	ČT	91			13:00										
Sněhurka	MP	DAB	106					15:50								
Talisman	MP	ČT	101					15:00				19:20				
Titanic 3D	MP	DAB	194										20:20			
Vrásky z lásky	MP	---	101							17:10						
Vrtěti ženou	MP-15	ČT	100										20:40			

Obrázek 8.6: Program kin dostupný na www.cinemacity.cz



Obrázek 8.7: Program kin zobrazený v aplikaci

9 Uživatelská příručka

9.1 Spuštění

Aplikace se spouští stisknutím příslušné ikonky na obrazovce telefonu. Po načtení se objeví úvodní obrazovka s katalogem.

9.2 Ovládání aplikace

Aplikace je rozdělena do třech hlavních částí zobrazených ve spodní navigační liště (viz obr. 9.1: A).

- **Katalog** - zde je k dispozici tabulkový výpis dostupných informací zobrazený v několika úrovních. Výběrem položky se vždy přesuneme do příslušné podkategorie. Tlačítkem zpět (viz obr. 9.1: B) se vracíme o úroveň výš. Nejnižší úrovní katalogu je detail události, u kterého lze vybraný čas konání přidat do kalendáře, případně ho odeslat pomocí SMS nebo emailu.
- **Mapa** - zobrazuje místa na mapě znázorněná pomocí špendlíku. Zmáčknutím špendlíku se zobrazí název místa s tlačítkem pro přechod na příslušné události (viz obr. 9.1: E). Mapu lze posouvat a pomocí gest dvou prstů i přibližovat a oddalovat. Stisknutím tlačítka *Moje poloha* (viz obr. 9.1: D) se náhled přesune na aktuální pozici uživatele. Lze tak prozkoumávat místa v okolí.
- **Nastavení** - umožňuje načítání a mazání všech dostupných dat (viz níže).

9.3 Částečná aktualizace dat

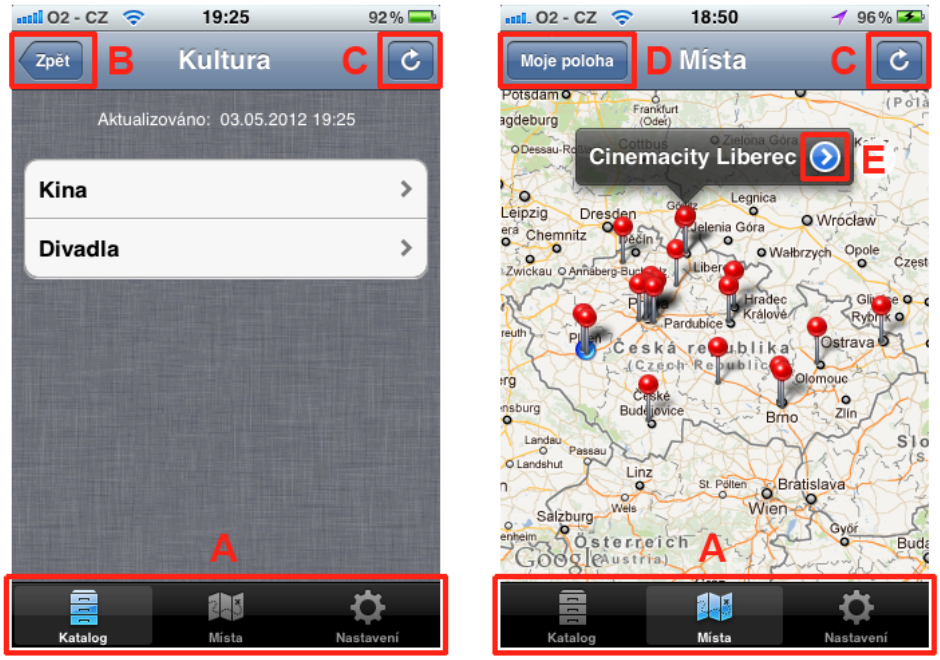
Částečná aktualizace dat se inicializuje tlačítkem pro obnovu (viz obr. 9.1: C). Načítají se vždy pouze data potřebná k obnově informací na aktuální obrazovce. To je výhodné např. při omezeném připojení k internetu. Data zůstávají uložena v telefonu pro další zobrazení.

9.4 Kompletní aktualizace dat

Kompletní aktualizace dat se provede v nastavení aplikace stisknutím tlačítka *Načíst všechna data*. Tím se načtou ze serveru všechny aktuálně dostupné informace. Při jejich velkém množství může ale načítání trvat delší dobu a stáhne se při tom větší množství dat. Je tedy vhodné tento proces provádět např. při připojení přes síť wi-fi. Výhodou je, že načtená data zůstanou uložena v telefonu a všechny kategorie lze potom procházet bez nutnosti internetového připojení.

9.5 Mazání dat

Mazání dat se provádí v nastavení, tlačítkem *Smazat všechna data*. Aplikace se tak uvede do původního stavu. Tento proces je nevratný a pro opětovné zobrazení informací je nutné je znovu načíst ze serveru.



Obrázek 9.1: Navigační prvky aplikace

10 Závěr

Aplikace pro mobilní telefony se staly díky své okamžité dostupnosti velice oblíbeným nástrojem pro získávání informací. Přispěl k tomu jistě i rychlý vývoj těchto zařízení, která dnes již dokážou poskytnout dostatečně pohodlné a přehledné uživatelské rozhraní.

V rámci bakalářské práce jsem se zabýval vývojem aplikace pro platformu iOS, která čerpá informace ze vzdálených zdrojů a vhodným způsobem je zprostředkovává uživateli. Během analýzy problému jsem určil třídění informací do kategorií takovým způsobem, aby členění bylo přehledné a zároveň byl systém použitelný pro široké množství informačních zdrojů.

Přístup k primárnímu zdroji informací neprobíhá přímo z aplikace. Namísto toho je použit mezičlánek ve formě serveru, který přistupuje ke zdrojům, zpracovává dostupná data a získané informace ukládá do své databáze. Aplikace se potom připojuje pouze k němu. Důležitý byl návrh komunikace mezi serverem a aplikací. Díky zvolenému formátu XML je systém nezávislý na platformě a aplikace by se tak dala portovat i na jiné systémy.

Výsledkem je funkční aplikace otestovaná v reálném zařízení. Umožňuje uživateli zobrazení informací z vybraných informačních zdrojů, jimiž jsou aktuální programy filmových představení ve všech multikinech Cinestar a Cinemacity v ČR.

Díky vhodnému návrhu by se práce dala v budoucnu snadno rozšiřovat. Na straně serveru by to bylo určitě přidání dalších informačních zdrojů, což znamená především vytvoření nových parserů. V případě aplikace se otevírá možnost již zmiňované portace na jiné platformy než iOS, např. Android nebo Windows Phone. V úvahu by připadal i webový portál, čerpající ze stejného zdroje informací.

Použité výrazy

- **framework** - softwarová struktura a knihovna potřebných utilit sloužící k podpoře programování a vývoje softwarových projektů
- **HTTP**, *Hypertext Transfer Protocol* - internetový protokol určený pro výměnu hypertextových dokumentů ve formátu HTML
- **PHP** - skriptovací programovací jazyk určený především pro programování dynamických internetových stránek
- **phpMyAdmin** - nástroj napsaný v jazyce PHP umožňující jednoduchou správu obsahu databáze MySQL
- **SDK**, *Software Developement Kit* - software umožňující vývoj pro určitou platformu
- **SMS**, *Short Message Service* - služba pro zasílání krátkých textových zpráv mezi mobilními telefony
- **unixový čas** - čas vyjádřený počtem vteřin uplynulých od 1.1.1970
- **webhosting** - pronájem prostoru pro webové stránky na cizím serveru
- **XML**, *Extensible Markup Language* - obecný značkovací jazyk vyvinutý a standardizovaný konsorciem W3C (`http://www.w3c.org`)

Literatura

[1] LAMARCHE, Jeff - MARK, Dave, *iPhone SDK*, Computer Press, 2010. ISBN 9788025128206.

[2] ČADA Ondřej, *Programování pro iOS*, 2011 [cit. 12.2.2012]. Dostupné z `http://www.mujmac.cz/rubriky/software/nastal-cas-na-kakao--59269cz`.

[3] HOLLEMANS, Matthijs, *Beginning Storyboards in iOS 5*, 14.11.2011 [cit. 7.2.2012]. Dostupné z `http://www.raywenderlich.com/5138/beginning-storyboards-in-ios-5-part-1`.

[4] WENDERLICH, Ray, *Introduction to MapKit on iOS*, 22.3.2011 [cit. 16.3.2012]. Dostupné z `http://www.raywenderlich.com/2847/introduction-to-mapkit-on-ios-tutorial`.

[5] AOL Inc., *Apple: 250 million iOS devices sold*, 4.10.2011 [cit. 2.2.2012]. Dostupné z `http://www.engadget.com/2011/10/04/apple-250-million-ios-devices-sold/`.

[6] Apple inc., *SOFTWARE LICENSE AGREEMENT FOR MAC OS X*, 2011 [cit. 6.5.2012]. Dostupné z `http://images.apple.com/legal/sla/docs/macosx107.pdf`.

[7] Apple inc., *Xcode*, 2012 [cit. 6.2.2012]. Dostupné z `http://developer.apple.com/xcode/`.

[8] Apple inc., *iOS UI Element Usage Guidelines*, 2012 [cit. 16.2.2012]. Dostupné z `http://developer.apple.com/library/ios/\#documentation/UserExperience/Conceptual/MobileHIG/UIElementGuidelines/UIElementGuidelines.html\#//apple\_ref/doc/uid/TP40006556-CH13-SW1`.

[9] Apple inc., *iOS Human Interface Guidelines*, 2012 [cit. 29.4.2012]. Dostupné z http://developer.apple.com/library/ios/\#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html\#//apple_ref/doc/uid/TP40006556-CH1-SW1.

[10] Nette Foundation, *Licenční politika Nette Framework*, 2012 [cit. 1.5.2012]. Dostupné z http://nette.org/cs/license.

[11] ZONER software, a.s., *Podmínky hostingu PHP5.cz*, 2012 [cit. 3.5.2012]. Dostupné z http://www.php5.cz/index/conditions/.

[12] Nette Foundation, *Požadavky Nette Framework*, 2012 [cit. 3.5.2012]. Dostupné z http://doc.nette.org/cs/requirements.

[13] CHEN, S.C., *PHP Simple HTML DOM Parser*, 2012 [cit. 9.4.2012]. Dostupné z http://simplehtmldom.sourceforge.net/.

[14] Apple inc., *View Controller Basics*, 2012 [cit. 2.3.2012]. Dostupné z http://developer.apple.com/library/ios/\#featuredarticles/ViewControllerPGforiPhoneOS/AboutViewControllers/AboutViewControllers.html.

[15] Google Inc., *Google Data APIs Objective-C Client Library*, 2012 [cit. 4.3.2012]. Dostupné z http://code.google.com/p/gdata-objectivec-client/.

A Přílohy

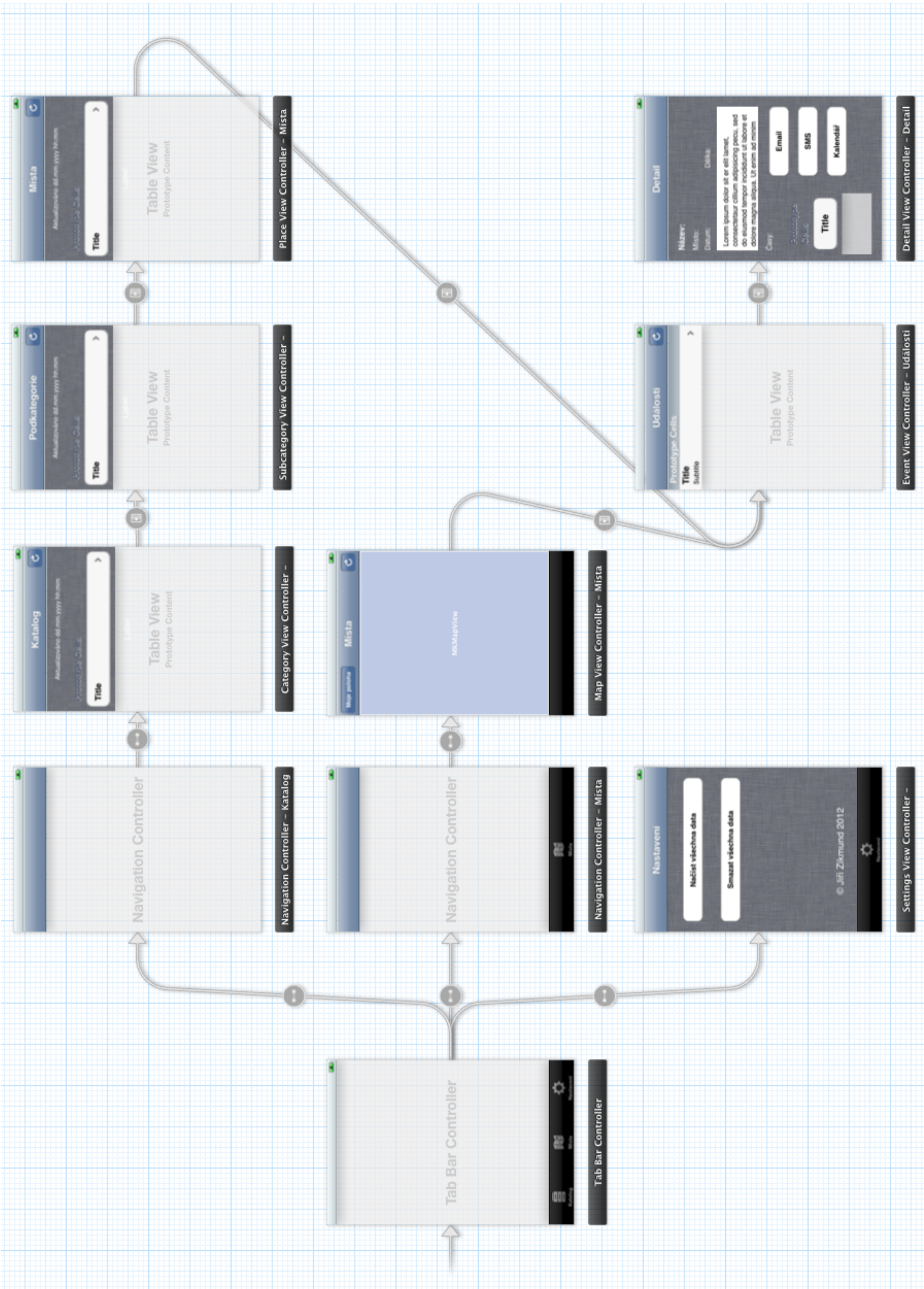
A.1 CD

K práci je přiloženo CD s následující strukturou:

- `/dokument/` - tento dokument v elektronické podobě
- `/aplikace/source/` - zdrojové kódy klientské aplikace
- `/aplikace/readme.txt` - instrukce pro spuštění aplikace v Xcode
- `/server/source/` - zdrojové kódy serverové aplikace
- `/server/readme.txt` - instrukce pro instalaci serveru

A.2 Instalace serverové aplikace

Použitý server musí splňovat požadavky uvedené v kapitole 6.1. Je potřeba zkopírovat celý osbsah složky `/server/source/` z přiloženého CD do kořenového adresáře. Poté je nutné nastavit databázi. Parametry pro připojení se vyplňují v konfiguračním souboru `config.latte` umístěném ve složce `/app/config/`. Podrobnější popis je k dispozici v souboru `readme.txt`.



Příloha A.3: Storyboard aplikace v prostředí Xcode