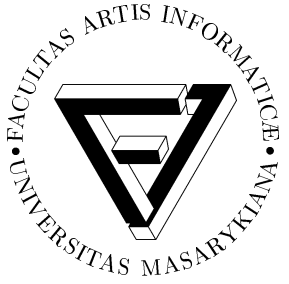
MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY



Rozšíření mobilního knihovního katalogu SmartLib o iOS klienta

BAKALÁŘSKÁ PRÁCE

David Jongepier

Brno, jaro 2013

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

David Jongepier

Vedoucí práce: Mgr. Bc. Jonáš Ševčík

Poděkování

Chtěl bych poděkovat vedoucímu práce Jonášovi Ševčíkovi za motivaci, podporu, konzultace a věcné připomínky. Dále bych rád poděkoval celému týmu SmartLib, jmenovitě Michalovi Halajovi, Hoa Quoc Tranovi a Tomášovi Hrabalovi, za nápady a vstřícné odpovědi na otázky ohledně problémů při implementaci.

Shrnutí

Získávání informací o knihách prostřednictvím webového rozhraní neposkytuje optimalizované uživatelské prostředí pro přístup z mobilních telefonů. Tato bakalářská práce má za cíl vytvoření mobilní aplikace SmartLib pro platformu iOS, která poskytne nativní prostředí pro přístup ke katalogu Aleph. Protože funkcionalita tohoto klienta má být ekvivalentní s existujícími aplikacemi SmartLib na platformách Android a Windows Phone, bude tento problém řešen metodou SASS. V této spojitosti budou v práci uvedeny základní informace o systému iOS a popsán vývoj na této platformě. Vytvořená aplikace splnila požadavky schvalovacího procesu firmy Apple, byla zveřejněna a je volně ke stažení z App Store pod názvem „SmartLib MU“.

Klíčová slova

SmartLib, iOS, Objective-C, Xcode, Aleph, knihy, mobilní aplikace

Obsah

Úvod

S příchodem prvních tzv. chytrých telefonů v roce 2007 byl změněn pohled na svět. Lidé vlastnící tyto telefony si zvykli mít všechny informace v kapse. Zmíněná zařízení totiž mimo jiné usnadnila organizaci informací a umožnila přístup na Internet. Časem si však uživatelé uvědomili, že ona hlavní funkce, prohlížení webových stránek, není na telefonech s malým displejem příliš pohodlná. Na přístrojích jako jsou stolní počítače, notebooky nebo tablety je tato činnost přívětivější, protože webové stránky jsou primárně vyvíjeny právě pro tato zařízení. Na telefonech se však stránka na malý displej nevejde celá, takže je potřeba ji často posouvat, přibližovat a oddalovat. To zhoršuje uživatelskou přívětivost a navíc trvá vyhledání informací na telefonu déle. Proto začaly vznikat aplikace optimalizované právě pro tato zařízení. Aplikace jsou jednoduché, přehledné a navíc mají dostatečně velké písmo a tlačítka. Práce s nimi je tedy mnohem příjemnější a rychlejší.

Jednou z těchto aplikací je i aplikace SmartLib, která slouží k získávání informací o knihách ve všech knihovnách Masarykovy univerzity. Tuto funkcionalitu již zajišťoval knihovní systém Aleph MU. Ten byl však vytvořen právě pro zařízení s velkými displeji, a práce s tímto katalogem je na prohlížeči v telefonu poněkud těžkopádná. Proto byl založen projekt SmartLib, který se zaměřuje právě na mobilní platformy. Kromě poskytování funkcionality ze systému Aleph přidává SmartLib další vlastnosti jako hodnocení knih, psaní recenzí nebo zobrazení náhledu knihy.

Mým úkolem je vytvoření verze této aplikace pro iOS. Jelikož již existují aplikace SmartLib na platformách Android a Windows Phone, je vhodné použít metodu strukturované analýzy a specifikace systému (SASS) (viz kapitola 4).

Kapitola 1

SmartLib

1.1 Historie

Historie projektu SmartLib sahá do roku 2011. V té době existoval prototyp aplikace pro platformu Android a serverová část napsaná v jazyce Java, kterou pro svou diplomovou práci [1] vyvinul Jonáš Ševčík. Jelikož bylo nutné aplikaci dále vyvíjet, byl postupně složen tým. Nejdříve se připojil Michal Halaj, který vytvářel verzi pro Windows Phone. Následně to byl Hoa Quoc Tran, který programoval verzi pro Android. Později se ukázalo, že je potřeba vyvíjet i serverovou část. Do týmu byl proto zapojen Tomáš Hrabal, který ji začal implementovat v jazyce Ruby. Dne 27. 12. 2012 byly zveřejněny aplikace pro Android a Windows Phone.

1.2 Serverové API

Komunikace se serverem probíhá pomocí protokolu HTTP, případně HTTPS pro registraci a autentizaci. Server používá princip REST (Representational State Transfer). Ten používá ke komunikaci pouze URL a metody `GET`, `POST`, `PUT` a `DELETE`.

Data se předávají pomocí formátu JSON (JavaScript Object Notation). Tento formát je datově nenáročný a může být jednoduše zpracován. JSON obsahuje dva základní datové objekty.

- Slovník – množina párů klíč-hodnota. Je ohraničen složenými závorkami. Pár klíč-hodnota je oddělen dvojtečkou. Jednotlivé páry jsou odděleny čárkami.
- Seřazené pole – je ohraničeno hranatými závorkami.

Kapitola 2

iOS

2.1 Historie

Dle [2] je iOS mobilní operační systém, který vyvinula společnost Apple. Byl uveřejněn v roce 2007. Zprvu se používal pouze v mobilním telefonu iPhone, později byl rozšířen i na další zařízení vyráběné touto firmou, jako jsou iPod Touch, iPad a Apple TV.

Prvotní verze neměla vlastní název. Jelikož je systém odvozen od OS X, byl prezentován pouze jako „OS X běžící na telefonu iPhone“. Tato verze také neumožňovala provozování aplikací třetích stran. Počítalo se totiž s tím, že aplikace budou vytvořeny pouze jako webové rozhraní. Později však bylo oznámeno, že se vyvíjí rozhraní pro tvorbu nativních aplikací, které bylo zveřejněno spolu s novým názvem systému „iPhone OS“ v roce 2008. Z tohoto názvu nakonec jeho zkrácením vznikl dnešní název „iOS“.

2.2 Funkce systému

2.2.1 Domovská obrazovka

Jako první je po spuštění systému zobrazena domovská obrazovka, což je plocha, která je zaplněna ikonami aplikací. Pokud je aplikace spuštěna, je možné se na plochu dostat stisknutím tlačítka Home, což je fyzické tlačítko nacházející se na jakémkoli zařízení s iOS. Mezi těmito plochami lze přepínat táhnutím prstu. Ve spodní části je tzv. dock, což je lišta s několika aplikacemi. Dock je možné vidět na jakékoli obrazovce. Aplikace lze dále uspořádat do složek. Složky však není možné zanořovat (složku ve složce).

2.2.2 Spotlight

Na ploše, která se nachází úplně vlevo, je umístěno hledání zvané Spotlight. Pomocí něj lze vyhledat téměř jakákoli data nacházející se na telefonu – od aplikací, přes média jako jsou fotky a videa až po kontakty a události v kalendáři.

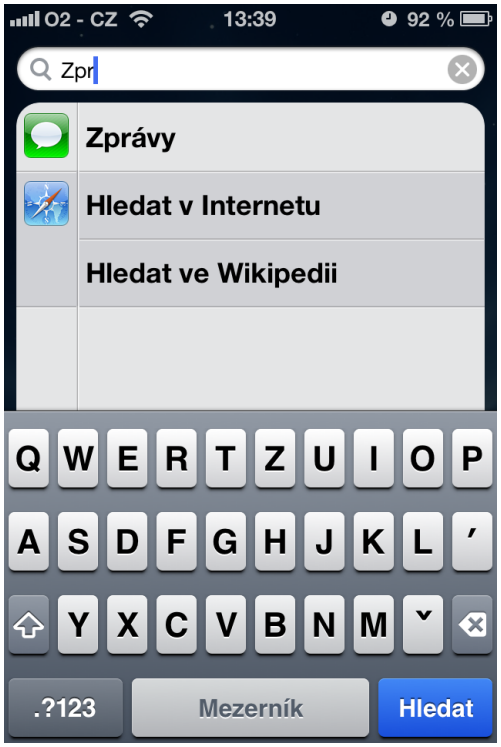
2.2.3 Aplikace

Spolu se samotným systémem je distribuována spousta aplikací. Patří mezi ně například přehrávač hudby, kalendář, mapy, e-mailový klient či webový prohlížeč Safari. Dodatečné aplikace lze nainstalovat přímo z internetového obchodu App Store.

2.2.4 Notifikační centrum

Některé aplikace v iOS potřebují určitým způsobem notifikovat uživatele. Nejzásadnější jsou upozornění o zmeškaném hovoru nebo příchozí zprávě. Všechny nepřečtené notifikace je možné zobrazit ve zvláštním okně. Stačí táhnout prstem směrem dolů z horní

Obrázek 2.1: Domovská obrazovka.



Obrázek 2.2: Hledání Spotlight.

části obrazovky. Poté se zobrazí notifikační centrum. Kromě notifikací je zde možné zobrazovat počasí, akcie nebo tlačítka pro zveřejňování příspěvků na sociální sítě Twitter a Facebook.

Obrázek 2.3: Notifikační centrum.

2.2.5 Multitasking

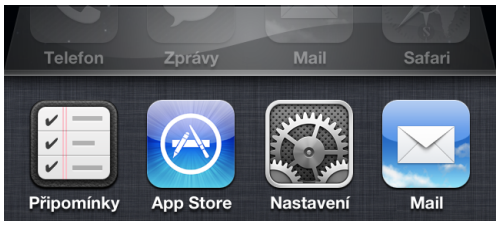
Multitasking je funkcionalita zajišťující provádění několika úkonů naráz. Na systému iOS je však tento způsob pro externí aplikace omezen.

Po pozastavení běhu (typicky stisknutí tlačítka Home) má aplikace přibližně deset minut, během kterých může provádět úkony. Pokud je však spuštěna jiná aplikace, která potřebuje více paměti, může být běh předchozí aplikace úplně zastaven za libovolně kratší dobu.

Kromě provádění běžných úkonů může aplikace využít některé ze specializovaných funkcí v API, které umožňují provádění na pozadí neustále. Jsou to:

- audio na pozadí,
- získávání polohy,
- notifikace,
- VOIP (voice over IP),
- newsstand (stahování nových položek, například elektronických časopisů nebo novin),
- komunikace pomocí bluetooth.

Mezi běžícími aplikacemi lze přepínat pomocí dvojitého stisknutí tlačítka Home.



Obrázek 2.4: Multitasking.

2.3 Grafické rozhraní (UI) a uživatelský prožitek (UX)

Každý systém má své určité zásady, které by se měly dodržovat napříč systémem. U iOS tomu není jinak. Tyto zásady se týkají zejména grafického rozhraní (user interface) a uživatelského prožitku (user experience).

2.3.1 Grafické rozhraní

Aplikace jsou často navrženy tak, aby většina grafických prvků pocházela z frameworku UIKit (viz kapitola 3.4). To má dvě výhody:

- uživatelé jsou na tyto prvky zvyklí, protože se používají napříč systémem;
- není potřeba psát další kód pro nové prvky, což urychluje vývoj aplikace.

Elementy z frameworku UIKit je možné od verze iOS 5 graficky upravit při zachování funkcionality; není potřeba vytvářet vlastní prvek pouze kvůli změně vzhledu. To dále redukuje množství kódu a urychluje vývoj.

V oficiální dokumentaci [3] je každý prvek z frameworku UIKit popsán, včetně jeho vzhledu, chování a doporučení k použití.

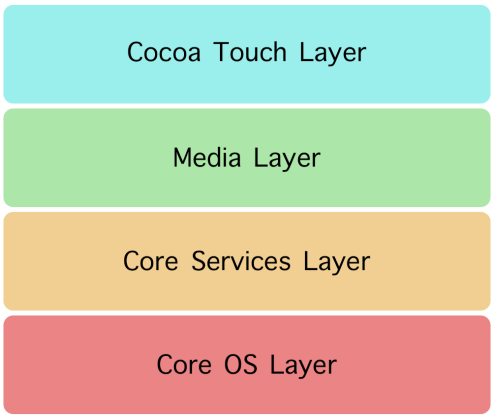
2.3.2 Uživatelský prožitek

Uživatelský prožitek se nesoustředí na jednotlivé grafické prvky, ale na návrh a použití aplikace z širšího hlediska. Zabývá se korektním rozmístěním prvků a rozdělením aplikace na menší logicky související části. Kromě toho zdůrazňuje zejména správné použití grafických prvků, použití animací a intuitivní užívání aplikace.

Veškeré informace týkající se uživatelského prožitku jsou opět popsány v oficiální dokumentaci [4].

2.4 Architektura systému

Systém iOS je složen ze čtyř vrstev. [5]



Obrázek 2.5: Vrstvy systému iOS. [5]

2.4.1 Cocoa Touch Layer

Tato vrstva je nejvyšší vrstvou celého systému. Zahrnuje klíčovou funkcionalitu pro vytváření a provoz aplikací. Její největší částí je UIKit, což je framework umožňující interakci s aplikacemi. Patří sem mimo jiné jejich správa, rozpoznávání dotyků, notifikace, manipulace s grafickými prvky a fotoaparátem nebo získávání dat ze senzorů.

2.4.2 Media Layer

Media Layer obsahuje manipulaci s grafikou, audiem a videem. Vrstva pokrývá vykreslování 2D a 3D grafických objektů spolu s jejich manipulací, animací, konverzí a ukládáním. Také zahrnuje podporu přehrávání a nahrávání audia i videa.

2.4.3 Core Services Layer

Obsahuje fundamentální systémové služby. Z vyšší vrstvy to jsou například iCloud, podpora sociálních sítí či přístup ke kontaktům. Mezi nízkoúrovňové patří Automatic Reference Counting, ochrana dat, sdílení souborů, SQLite, podpora XML a mnohé další.

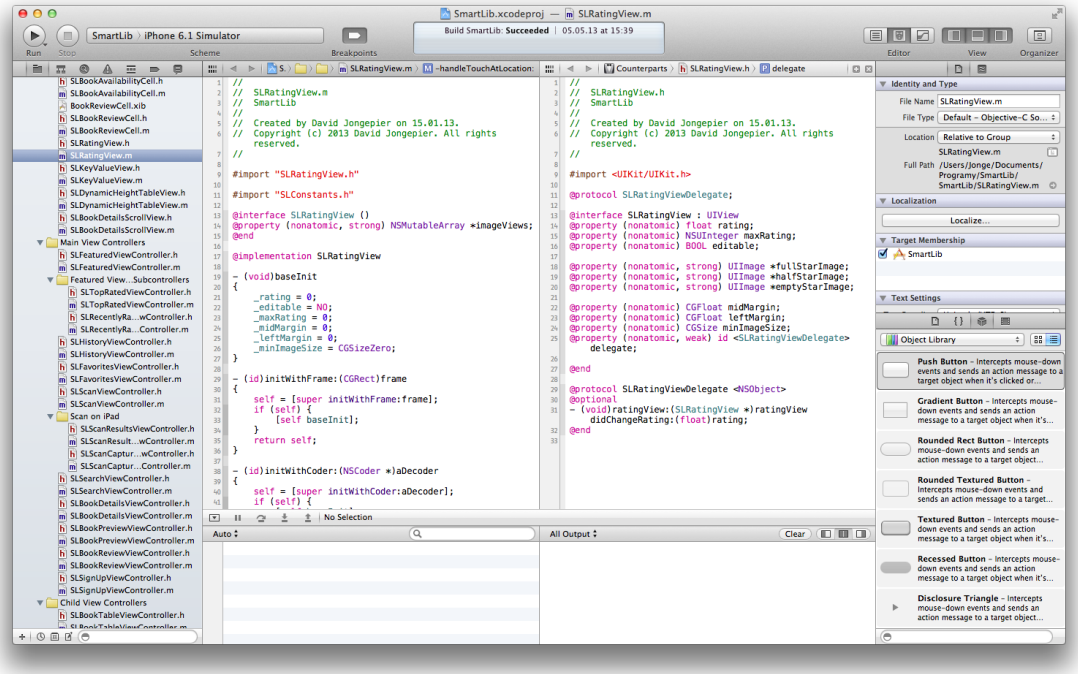
2.4.4 Core OS Layer

Funkce této vrstvy jsou využívány velmi řídce, poskytují však funkcionalitu nadřazeným vrstvám. Přímo jsou používány pouze v situacích, kdy je potřeba zajistit bezpečnost či komunikaci s jiným zařízením. Zahrnuje frameworky jako Core Bluetooth (pro komunikaci přes bluetooth) nebo Security (pro zajištění bezpečnosti).

Kapitola 3

Vývoj aplikací na iOS

Před započetím vývoje je potřeba nainstalovat program Xcode, který obsahuje i iOS Software Development Kit. Tento program zahrnuje vše, co je pro vývoj potřeba. Je v něm možné editovat kód, vytvářet grafické rozhraní, používat debugger, spravovat zařízení i zobrazovat dokumentaci. Xcode zahrnuje také program na analýzu aplikací zvaný Instruments, kompilátor a veškeré frameworky.

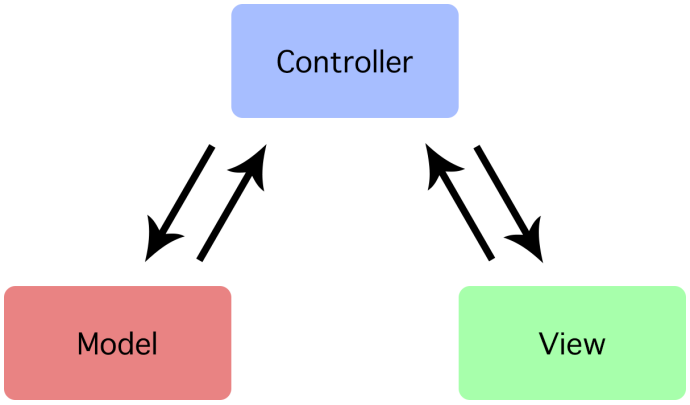


Obrázek 3.1: Xcode.

V této kapitole budou zmíněny pouze základy. Detailní přehled o vývoji aplikací pro iOS lze získat například z knihy iOS Programming: The Big Nerd Ranch Guide [12].

3.1 Návrhový vzor Model-View-Controller

Návrhový vzor Model-View-Controller (MVC) [6] je pravděpodobně nejrozšířenějším návrhovým vzorem používaným při vývoji aplikací pro iOS. Předpokládá, že se v aplikaci vyskytují tři typy objektů: model, view (pohled) a controller (řadič). Ty musí být na sobě nezávislé. Při vývoji je důležitým krokem zvolení správného typu objektu pro každou třídu.



Obrázek 3.2: Návrhový vzor MVC.

3.1.1 Model

Objekty typu model obsahují aplikační data a definují manipulaci s nimi. Pokud je například nutné ukládat auta, je vhodné vytvořit si objekt typu model s názvem Car, ve kterém se ukládají vlastnosti auta (značka, barva, rok výroby). Také by zde měly být implementovány metody pro změnu těchto parametrů, ukládání do databáze apod.

3.1.2 View (pohled)

Jakékoli objekty, které zahrnují zobrazování informací uživateli, by měly být typu view. Jsou to tedy jakákoli tlačítka, posuvníky, tabulky a jiné grafické prvky.

3.1.3 Controller (řadič)

Část controller by měla obsahovat objekty, které svazují předchozí dva typy objektů. Měly by tedy předávat data z view do modelu (a naopak).

3.2 Objective-C

Pro vytváření iOS aplikací se používá vysokoúrovňový objektově orientovaný programovací jazyk zvaný Objective-C. Je to vlastně jazyk C, který je rozšířen o tzv. zasílání zpráv, podobné jazyku Smalltalk.

3.2.1 Historie

Podle zdroje [7] sahá počátek jazyka Objective-C do roku 1980, kdy Brad Cox a Tom Love začali vyvíjet vlastní preprocesor jazyka C, který byl rozšířen o některé vlastnosti jazyka Smalltalk. V roce 1988 byla licence Objective-C zakoupena společností NeXT, kterou v té době vedl Steve Jobs, tehdy bývalý ředitel společnosti Apple. NeXT tento jazyk začal rozšiřovat o knihovny AppKit a Foundation Kit, které použil ve vlastním operačním systému NeXTSTEP. Tento systém dal později základ dnešním systémům OS X a iOS.

V roce 1996 získal Apple společnost NeXT a začal vyvíjet nový operační systém Mac OS X, odvozený od systému NeXTSTEP. Ten zahrnoval Objective-C a aplikaci Project Builder, ze které je dnes odvozen program Xcode.

3.2.2 Rozhraní, implementace

Každá třída musí mít jak rozhraní (interface), tak implementaci (implementation). Jejich syntaxe je následující.

```
@interface MyClass : NSObject
@property (nonatomic, copy) NSString *name; // atribut
- (void)printName; // metoda
@end

@implementation MyClass
// Implementace uvedených metod
- (void)printName
{
    NSLog(@"%@", self.name);
}
@end
```

Zde je vytvořena třída `MyClass`, která dědí z `NSObject`. `MyClass` obsahuje atribut `name`, což je řetězec. Také obsahuje metodu pro vypsání jména. V sekci označené jako `@implementation` je tato metoda implementována. Funkce `NSLog()` funguje podobně jako funkce `printf()` v jazyce C.

3.2.3 Zasílání zpráv

Zatímco v jiných jazycích se „volají metody“, v Objective-C se „zasílají zprávy“.

```
[objekt metoda:argument];
```

Interně je tato část také implementována odlišně. V běžných objektově orientovaných jazycích je potřeba ošetřit to, že objekt není nulový (null/nil). V jazyce Objective-C je však možné volat metodu na nulovém objektu. To usnadňuje ošetřování chyb a zvyšuje čitelnost kódu, protože není potřeba mít podmínky které rozhodují, zda je objekt nenulový. Na druhou stranu je však obtížnější identifikovat chyby.

3.2.4 Prefix tříd

V jazyce Objective-C neexistují jmenné prostory (namespace). Název každé třídy proto musí být unikátní. Je tedy vhodné vlastním třídám zavést prefix, aby byla tato unikátnost zaručena. V již existujících souborech tříd (framework) předchází jejich názvům prefix, který je často odvozený z jejich názvu, například `UI` (User Interface) nebo `CG` (Core Graphics).

3.2.5 Protokoly, delegáti

Při vývoji aplikací pro iOS se musí dodržovat návrhový vzor MVC. Je proto nutné nějakým způsobem zajistit zasílání zpráv mezi jednotlivými typy objektů. Pro dodržení abstrakce byl zaveden princip protokolů a delegátů. Protokol je návrh metod a atributů které nemají implementaci. Implementace je přenechána třídám, které tento protokol dodržují.

```
@protocol MyProtocol
- (void)printSomething;
@end
```

Rozhraní třídy, která dodržuje protokol `MyProtocol`, by vypadalo následovně.

```
@interface MyClass <MyProtocol>
// metody a atributy
@end
```

V implementační části je pak implementována metoda `printSomething` z protokolu `MyProtocol`.

```
@implementation MyClass
- (void)printSomething
{
    NSLog(@"Something");
}
@end
```

Díky tomuto návrhu je možné mít objekt, kterým může být jakákoli třída dodržující určitý protokol.

```
id <MyProtocol> protocolObject;
```

Tato vlastnost nabývá síly až při kombinaci s delegáty. Delegát je atribut třídy, který dodržuje protokol. Lze například definovat metodu, která se zavolá po stisknutí tlačítka. Třída `MyButton` obsahuje atribut `delegate`. Tento atribut je třída, která dodržuje protokol `ButtonDelegate`. Smí se mu tedy přiřadit jakákoli třída dodržující tento protokol. Tou je v tomto případě třída `ButtonHandler`.

```
@protocol ButtonDelegate
- (void)buttonPressed;
@end

@interface MyButton
@property id <ButtonDelegate> delegate;
@end

@implementation ButtonHandler <ButtonDelegate>
- (void)buttonPressed
{
    // kód vykonaný po stisknutí tlačítka
}
@end
```

Při stisknutí tlačítka zašle třída `MyButton` zprávu `buttonPressed` delegátovi.

```
[delegate buttonPressed]
```

Jakákoli reakce na stisk tlačítka je díky tomu předána další třídě. To umožňuje snadnou abstrakci a díky tomu je jednodušší dodržet návrhový vzor MVC.

3.2.6 Kategorie

Kategorie jsou alternativou k dědičnosti. Umožňují rozšiřovat třídy o další metody, ale ne o atributy.

```
@interface NSString (Reverse)
+ (NSString *)stringByReversingString:(NSString *)string;
@end

@implementation NSString (Reverse)
+ (NSString *)stringByReversingString:(NSString *)string
{
    // vlastní implementace
}
@end
```

Zde je již existující třída `NSString` rozšířena o vrácení řetězce pozpátku. Díky tomu, že není použita dědičnost, je možné tuto metodu volat na třídách `NSString`, které jsou již použity jinde. Stačí pouze pomocí direktivy `#import` soubor s kategorií vložit. Není nutné jakkoli přepisovat kód či vytvářet nové třídy.

3.3 Foundation

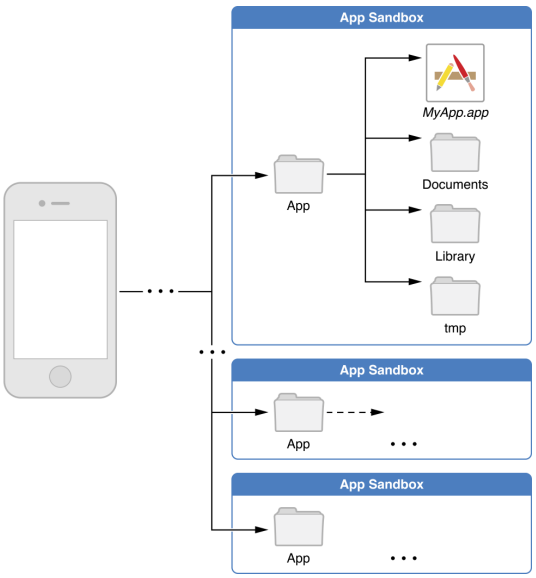
Framework Foundation [8] poskytuje fundamentální vrstvu tříd pro jazyk Objective-C. Patří sem třídy nezbytné pro práci s textovými řetězci, poli a množinami, časy a daty, čísly a spousta dalších.

3.4 UIKit

UIKit [9] je pravděpodobně nejdůležitější framework týkající se grafického rozhraní při vývoji pro iOS. Poskytuje veškeré třídy, které je zapotřebí pro sestrojení a správu UI, včetně tlačítek, tabulek, navigačních lišt, přepínacích tlačítek nebo zobrazení obrázků.

3.5 Sandbox

Z bezpečnostních důvodů je každá aplikace na iOS spouštěna v režimu sandbox [10]. Znamená to, že aplikace má omezený přístup k jednotlivým částem systému. Když případný útočník z nějakého důvodu ovládne aplikaci, může měnit data pouze této aplikace. Žádným způsobem se ale nedostane ke klíčovým segmentům systému ani k jiným aplikacím a nemůže tak ohrozit jeho bezpečnost.



Obrázek 3.3: Sandbox. [10]

Kapitola 4

Vývoj aplikace SmartLib

Pro vývoj aplikace byla použita metoda strukturované analýzy a specifikace systému (SASS), kterou popsal Tom DeMarco v roce 1978 [11]. Prvním krokem je zde studie stávajícího fyzického modelu, což jsou v tomto případě již existující aplikace SmartLib. Při této studii je odvozen logický ekvivalent stávajícího modelu, z nějž vyplyne funkcionalita stávajících aplikací. Dále je nutné tento model přetransformovat na nový logický model, který je možné částečně odvodit z modelu stávajícího. Posledním krokem je odvození nového fyzického modelu, což bude v tomto případě výsledná aplikace pro platformu iOS.

4.1 Studie stávajícího fyzického modelu

Prvotním krokem metody SASS byla studie stávajícího fyzického modelu, kterým byly aplikace pro Android a Windows Phone.

4.2 Odvození logického ekvivalentu stávajícího modelu

Díky předchozímu kroku byl odvozen stávající logický model. Ten zahrnoval následující funkcionalitu:

- seznam nejlépe hodnocených knih,
- seznam nedávno hodnocených knih,
- historie zobrazených knih,
- seznam oblíbených knih,
- vyhledávání knih, a to podle názvu titulu, autorů, ISBN a všech údajů,
- skenování čárových kódů a vyhledání knihy podle nich,
- zobrazení detailů knihy, včetně ISBN, názvu vydavatelství, data vydání, jazyku, počtu stran, dostupnosti a recenzí,
- psaní recenzí a hodnocení knih,
- zobrazení náhledu knih.

Kromě zmíněné funkcionality dodržuje stávající model vzhled a zákonitosti platforem. Tuto část není možné zakomponovat do nového logického modelu. Systém iOS se totiž v mnohém liší. Pokud by se ze stávajícího modelu použilo i grafické rozhraní, výrazně by to zhoršilo uživatelskou přívětivost. Je tedy vhodné agregovat pouze zmíněnou část s vlastnostmi.

4.3 Odvození nového logického modelu. Návrh UI

Ze stávajícího modelu se podařilo zjistit, které klíčové vlastnosti má budoucí aplikace zahrnovat. Návrh grafického rozhraní však bylo nutné kompletně přepracovat. Některými vlastnostmi stávající aplikace se lze při návrhu inspirovat, avšak ne na úkor uživatelské přívětivosti a zvykům systému.

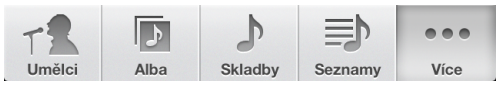
Aplikaci je vhodné navrhnout od nejobecnějších prvků po detaily.

4.3.1 Rozdělení sekcí

Vzhledem k velkému množství různých sekcí aplikace bylo vhodné použít třídu, která toto rozdělení podporuje – `UITabBarController`.

Obrázek 4.1: Třída `UITabBarController`.

`UITabBarController` však zobrazuje pouze pět panelů, pokud jich je na telefonu iPhone potřeba použít více, poslední panel se nahradí za panel „Více“, ve kterém jsou zobrazeny odkazy na další panely.



Obrázek 4.2: Třída `UITabBarController` se zobrazením více než pěti panelů.

V této aplikaci bylo potřeba mít sekcí šest (seznam nejlépe hodnocených knih, seznam nedávno hodnocených knih, historie zobrazených knih, seznam oblíbených knih, vyhledávání knih a skenování kódů). Avšak to znamenalo, že kvůli jednomu panelu navíc by místo posledního panelu musel být panel „Více“. V takovém případě je možné v jednom panelu použít přepínací tlačítko, které po přepnutí zobrazí alternativní prvky. Tady bylo vhodné sloučit nejlépe a nedávno hodnocené knihy.

Obrázek 4.3: Výsledný `UITabBarController` v aplikaci SmartLib.

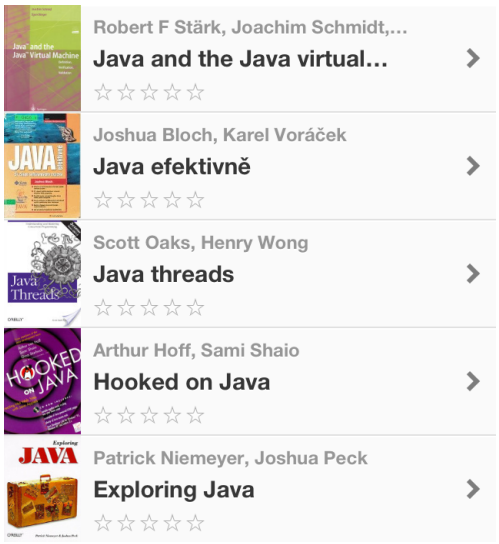
4.3.2 Seznam knih

Po rozdělení aplikace na sekce bylo dalším krokem rozhodnutí, jak bude vzhledově vypadat seznam knih. Existují v zásadě dvě možnosti. Je možné použít buď `UITableView` nebo `UICollectionView` (podobně jako v aplikaci iBooks).

Ke knize bylo potřeba zobrazit více informací (název knihy, autory a hodnocení). `UICollectionView` se používá spíše pouze pro zobrazování obrázků, případně jejich

krátkých popisů. Proto tady bylo lepší možností použití `UITableView`. Tento způsob zobrazení se také používá v již existujících aplikacích SmartLib.

Po kliknutí na položku se zobrazení pomocí tzv. segue přesune na detail knihy.



Obrázek 4.4: Finální podoba seznamu knih za použití třídy `UITableView`.

4.3.3 Detail knihy

V detailu knihy se zobrazují dodatečné informace o knize. Uspořádání položek bylo vhodně zpracované na jiných platformách. Úplně nahoře by se totiž měly vyskytovat položky, které mají pro uživatele nejvyšší informační hodnotu. Čím níže se vyskytují, tím je pro uživatele informační hodnota nižší. Nahoře jsou tedy nejdůležitější prvky jako název knihy, její autoři a celkové hodnocení. Pod tím jsou dodatečné informace o knize (ISBN, název vydavatelství, datum vydání, jazyk a počet stran) a vedle nich obrázek obálky dané knihy. Protože jednou z nejžádanějších funkcí bylo hodnocení knih, byla tato možnost přidána hned za tyto informace. Ještě níže je dostupnost kopií knihy spolu s informací o tom, ve které knihovně se vyskytuje, a zdali je výpůjčka absenční nebo prezenční. Úplně naspodu jsou pak uživatelské recenze.

Uspořádání tedy nebylo potřeba řešit, bylo ale nutné vytvořit vlastní vzhled. Jeho výsledná podoba je na obrázku 4.5.

4.3.4 Skenování

Implementaci skenování čárových kódů po kliknutí na panel je možné provést dvěma možnostmi. Buď je možné zobrazit obrazovku s kamerou ihned, nebo až po kliknutí na tlačítko. Nevýhoda první možnosti je ta, že pokud uživatel tento panel zvolí omylem musí několik sekund čekat, než se kamera spustí. Druhá možnost je rychlejší při přepínání panelů, ale pomalejší, pokud chce uživatel skenovat ihned. V tomto případě je lepší použít první možnost, protože k překlikům dochází méně často.

Dále bylo nutné přidat text, aby bylo jasné, co se po uživateli v této obrazovce požaduje. Skenování navíc funguje jen pro jednu dimenzi. Bylo proto potřeba přidat další grafické prvky. Díky nim je již jasné, jak přesně skenování funguje.

Obrázek 4.5: Grafický vzhled detailu knihy.

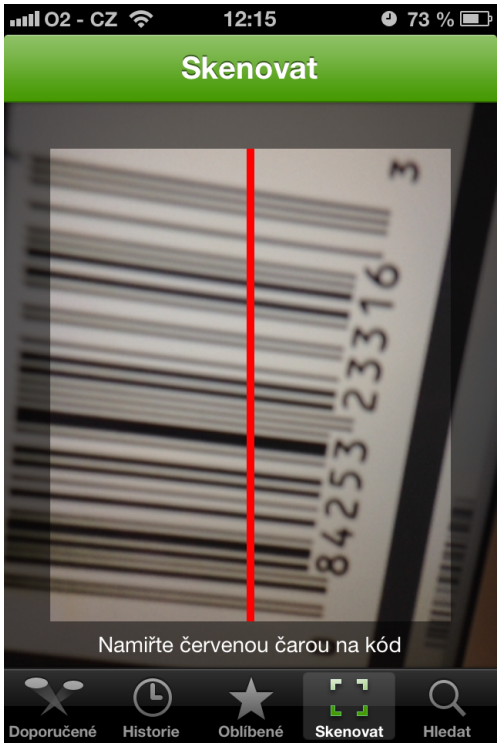
4.3.5 Hledání

Podobně jako ve spoustě jiných aplikací vypadá hledání tak, že se úplně nahoře vyskytuje standardní prvek pro hledání, používaný napříč systémem. Po klepnutí na něj se vysune tzv. scope bar, kde si uživatel může zvolit, zda chce hledat podle názvu titulu, autorů, ISBN nebo všech údajů. Po stisknutí tlačítka „Hledat“ se po získání údajů ze serveru zobrazí tabulka s knihami, popsaná v kapitole 4.3.2.

Hledání bylo také možné naimplementovat tak, že by se po pouhém kliknutí na spodní panel vysunula klávesnice, a tím pádem by mělo být započetí hledání rychlejší. Při zkoumání této možnosti však paradoxně vyplynulo to, že výsledný efekt byl opačný. Uživatel byl spíše zmaten, protože vše probíhalo bez jeho interakce. Navíc s vysunutou klávesnicí nebylo možné okamžitě se přepnout do dalšího panelu. Bylo nutné ji nejprve skrýt. Je tedy lepší zvolit původní možnost.

4.3.6 Náhled knihy

Některé knihy mají náhled na Google Books. Jedním z požadavků bylo jejich zobrazení. Tuto část je možné naimplementovat dvěma způsoby. Buď lze přepnout uživatele do výchozího prohlížeče (na iOS do Safari) a náhled zobrazit tam, nebo jej zobrazit v apli-



Obrázek 4.6: Skenování čárových kódů.

Obrázek 4.7: Hledání.

kaci pomocí Web View. V aplikaci SmartLib na platformě Android je implementována první možnost. Avšak na iOS se až na výjimky vyskytuje pouze druhá možnost. Navíc je pro uživatele přívětivější zobrazit náhled přímo v aplikaci. Přepínání z aplikace do aplikace jen kvůli náhledu knihy není úplně nejvhodnější. Je však velmi doporučené přidat možnost zobrazení stránky v prohlížeči Safari. Uživatel pak může stránku přidat do oblíbených. Pokud používá iCloud, může si ji dokonce zobrazit na jiných zařízeních.

4.3.7 Registrace, přihlášení a psaní recenzí

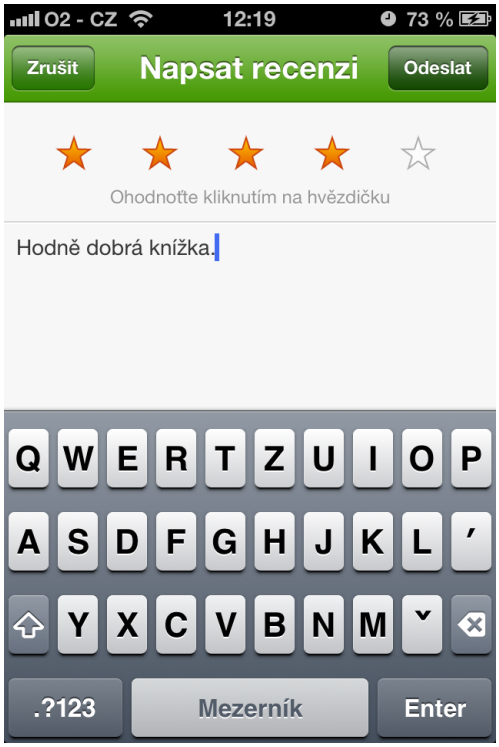
Aplikace měla umožňovat psaní uživatelských recenzí. Jelikož je v tomto případě vyžadován účet, bylo nutné vytvořit i registrační formulář a přihlašovací okno.

Po kliknutí na možnost „Napsat recenzi“ v detailu knihy se zobrazí okno s dotazem pro přihlášení. Zde má uživatel možnost přihlásit se pod stávajícím účtem, případně jej vytvořit. Po úspěšné autentizaci je zobrazeno hodnocení a textový rámeček.

4.4 Odvození nového fyzického modelu. Struktura aplikace

V aplikacích na iOS se používá návrhový vzor MVC (model-view-controller), popsaný v kapitole 3.1. Struktura proto byla rozdělena následovně.

- Model – obsahuje jakoukoli práci s knihami (jejich základní informace, detaily, recenze, dostupnost, zda patří do nějakého seznamu knih apod.). Vývoj této části zahrnoval návrh databáze a naprogramování metod pro operaci s knihami (mimo jiné přidávání do oblíbených a historie). Dále model umožňoval získávání dat ze serveru, jejich parsování a následné ukládání do databáze.



Obrázek 4.8: Psaní recenze.

- View – tato část byla z větší části hotová již po návrhu UI, protože obsahuje většinou grafické prvky. Kromě toho však bylo potřeba ručně naprogramovat i vlastní komponenty, například pro zobrazení hodnocení pomocí hvězdiček, kde bylo potřeba reagovat na dotyk prstu.

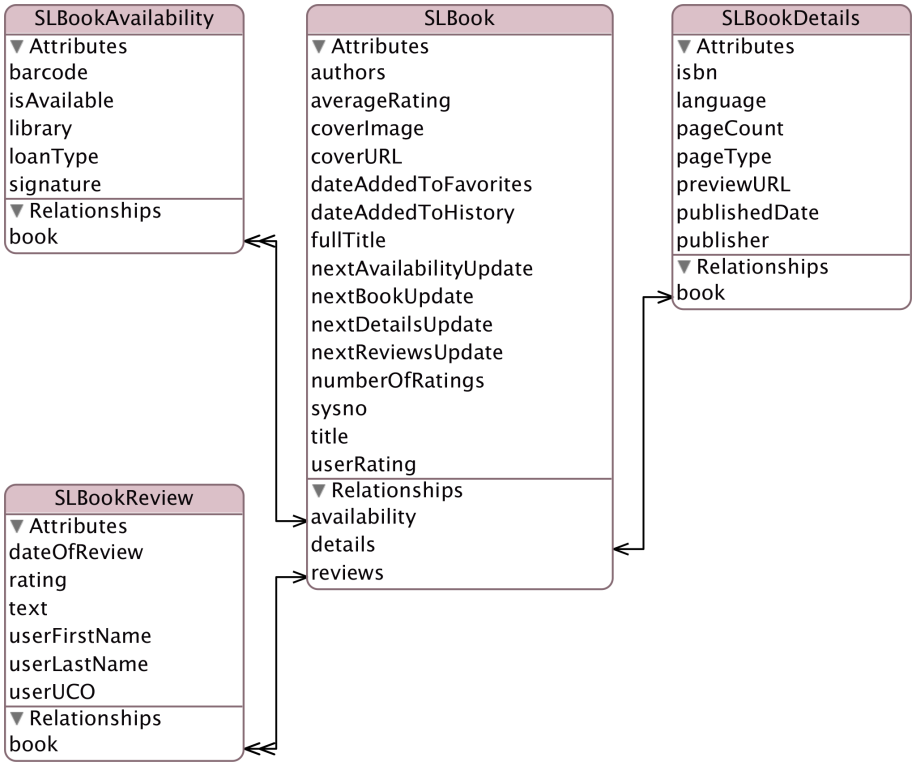
- Controller – jelikož controller pracuje s modelem a jeho výsledky předává do view, je v této části naimplementováno předávání dat z databáze do tabulek a do detailu knihy. Také je touto částí zajištěna komunikace opačným směrem (z view do modelu), což jsou například akce, které se mají vykonat po stisknutí tlačítka.

4.4.1 Model – databáze

Knihy je potřeba ukládat do seznamů. V takovém případě je vhodné použít framework Core Data, který tuto funkcionalitu zajišťuje. Jeho použití usnadňuje to, že po vytvoření projektu ze šablony je automaticky naimplementováno veškeré vytváření a ukládání souborů. Poté je nezbytným krokem návrh databázového modelu pro knihy. Jeho konečná verze je na obrázku 4.9.

Detaily knihy bylo možné ukládat buď přímo do části s knihou, nebo je ukládat zvlášť. V tomto případě je vhodnější druhá možnost. Je díky tomu jednodušší zjistit, zda už byly detaily knihy získány ze serveru nebo ne. Kniha totiž obsahuje vztah k detailům, který neexistuje, pokud detaily ještě načteny nebyly.

Po vytvoření modelu je potřeba vytvořit třídy pro jednotlivé entity. Tyto třídy jsou generovány automaticky v programu Xcode. Díky Key-Value Coding lze pak s těmito objekty komunikovat stejně jako s běžnými objekty. Problém však nastává v situaci, kdy je nutné třídy rozšířit o další funkce. V případě této aplikace je to ukládání knih do ob-



Obrázek 4.9: Návrh databázové struktury.

líbených a do historie a jejich případné mazání. Z důvodu automatického generování není vhodné implementovat tuto část přímo ve třídě, protože při změně modelu a novém vytváření je jakákoli rozšířená funkcionalita zahozena. Proto se používají elementy jazyka Objective-C zvané kategorie, popsané v kapitole 3.2.6, které tyto funkce obsahují v dodatečném souboru.

Dalším logickým krokem by bylo do modelu zahrnout vytvoření dotazů na databázi pro získání různých seznamů knih a jejich detailů. Avšak protože je pak potřeba pak s těmito daty pracovat a zobrazovat je uživateli, je vhodné tuto část přenechat objektu typu controller.

4.4.2 Model – získávání dat

Kromě ukládání informací zodpovídá model i za jejich získávání a překládání. Další velkou částí implementace proto byla komunikace se serverem. Je vhodné vytvořit si třídu, která bude zodpovědná za asynchronní stahování dat ze serveru a případné řešení chyb při přenosu, výpadcích sítě apod. Tuto třídu lze pak pomocí agregace použít v dalších třídách.

Po získání dat ze serveru je potřeba je přeložit a přiřadit objektům. Ze serveru jsou data vrácena ve struktuře JSON. Jelikož třída `NSJSONSerialization`, obsažená ve frameworku Foundation, zajišťuje překlad z JSONu do tříd poskytující rozhraní pro pole a slovník (třídy `NSArray` a `NSDictionary`), je parsování dat snadnou záležitostí. Avšak vzhledem k tomu, že překlad může trvat delší dobu, je potřeba zajistit jeho vykonávání asynchronně.

V tomto případě je vhodné použít třídy `NSOperation` a `NSOperationQueue`. Nej-

dříve se musí vytvořit vlastní třída, která dědí z `NSOperation`. Poté je v ní potřeba překrýt metodu `main()`. V té probíhá vlastní parsování, které zahrnuje i vytváření databázových objektů. Data se této třídě předávají typicky pomocí `@property`. Většinou je také potřeba zajistit, aby se vykonal kód po skončení této operace. Z důvodu abstrakce je vhodné předat této třídě například blok, který se v takovém případě vykoná. Do fronty `NSOperationQueue` se přidávají třídy dědící z `NSOperation`. Tato fronta si již sama zajistí vykonávání operací asynchronně.

Poté, co je implementováno získávání, zpracovávání a ukládání dat, je potřeba tato data zobrazit uživateli.

4.4.3 View – grafické prvky

Protože se část view týká pouze grafického rozhraní a prezentování informací uživateli, byla tato část z větší části hotova již po návrhu UI. Z grafických prvků bylo potřeba vytvořit třídy. Velkou část bylo také potřeba věnovat buňkám v tabulkách a manipulací s nimi. V neposlední řadě bylo nutné vytvořit vlastní třídu pro zobrazení a zpracování hodnocení pomocí hvězdiček.

Jelikož spousta grafických prvků nelze upravit přímo v grafickém editoru, musí se změnit v kódu.

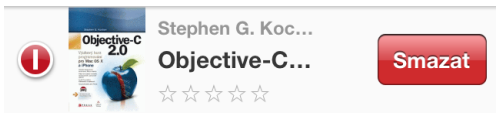
4.4.4 View – úpravy grafiky v kódu

V tabulce nelze přímo v návrháři UI změnit pozadí buňky při jejím výběru. Je možné jen vybrat mezi modrým nebo šedým pozadím. To se však pro aplikaci SmartLib nehodí, protože celá aplikace je laděná do zelené barvy. Proto byla barva pozadí potřeba změnit. Tuto změnu lze však uskutečnit jen v kódu.

Další nepříjemností je, že text v buňce je při výběru automaticky zobrazen s bílým stínem. Text je však také bílý, což zhoršuje jeho čitelnost. Bylo tedy nutné tento stín odstranit.

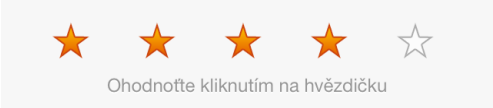
Obrázek 4.10: Změna pozadí buňky při výběru.

Poslední úpravou buňky byl její vzhled při stavu editace, protože pro příjemnější práci s aplikací je zapotřebí přímo z tabulky mazat knihy z oblíbených a z historie. Při zpřístupnění režimu editace se na levé straně zobrazí červené tlačítko, které poté, co jej uživatel stiskne, zajistí smazání položky. Ve výchozím stavu však na pravé straně tohoto tlačítka není dostatečně velký okraj a je potřeba jej zvětšit.



Obrázek 4.11: Režim editace a následné smazání knihy ze seznamu.

Další částí, kterou bylo potřeba zajistit v kódu, bylo zobrazení hodnocení pomocí hvězdiček. Jelikož se v aplikaci vyskytovala i možnost hodnocení po stisknutí hvězdičky, muselo se do této třídy přidat i rozpoznávání dotyků.



Obrázek 4.12: Hodnocení pomocí hvězdiček.

4.4.5 Controller

Objekty typu controller zastávají největší část aplikace SmartLib. Zajišťují totiž jakoukoli komunikaci mezi modelem a pohledem. Také poskytují pokročilou interakci aplikace s uživatelem.

Prvním krokem při jejich vytváření je vytvořit jeden pro každý view, který vyplňuje celou obrazovku. V této aplikaci to znamená mít samostatné controllery pro:

- nejlépe hodnocené,
- nedávno hodnocené,
- historii,
- oblíbené,
- skenování,
- hledání,
- detail knihy,
- náhled knihy,
- psaní recenzí,
- registraci.

Spousta těchto controllerů dělá podobnou činnost, kterou je zobrazování seznamu knih v tabulce. Je proto vhodné vytvořit si podřízený controller, který se bude vkládat do výše zmíněných. Ten bude zajišťovat celé zobrazování tabulky a reakci na výběr knihy. Jelikož jedinou věcí, která se mezi různými obrazovkami liší, jsou data, musí se tomuto podřízenému controlleru přidat rozhraní pro práci s daty.

Protože všechny controllery zajišťují pouze komunikaci mezi modelem a pohledem, zahrnuje většinová část jejich implementace pouze volání akcí modelu při interakci v pohledu a aktualizaci pohledu při změně dat, například při stažení. Menší část implementace zahrnují funkce pro zlepšení uživatelského prožitku. Jsou to například notifikace při nedostupnosti připojení k internetu či serveru, načítání dodatečných dat při dosažení konce tabulky nebo zákaz stisku tlačítka při nevyplněném formuláři.

4.5 Skenování čárových kódů

V aplikaci SmartLib pro platformy Android a Windows Phone byla pro skenování čárových kódů použita open-source knihovna ZXing. Tato knihovna je primárně implementována v jazyce Java, avšak je přeložena do spousty dalších jazyků. I když existuje verze pro jazyk Objective-C, není přeložena úplně. Umožňuje totiž pouze skenování QR kódů, a pro tuto aplikaci by tedy nebyla vhodná, jelikož je potřeba skenovat spoustu dalších typů kódů.

Existuje však určitá odnož této knihovny. Jmenuje se ZXingObjC, což je neoficiální verze kompletně přeložená právě do Objective-C. Tuto knihovnu tedy již bylo možné použít.

Instalace sice není úplně triviální, ale následné jednoduché použití tento zápor vyvažuje. Je potřeba vytvořit instanci třídy `ZXCapture` a nakonfigurovat dodatečné atributy – která kamera se má používat, který pohled má zobrazit informace z fotoaparátu a delegát této instance, který bude zpracovávat požadavky. Po této konfiguraci začne třída automaticky snímat data z fotoaparátu. Pokud narazí na čárový kód, zavolá se metoda v delegátovi, ve které je potřeba skenování zastavit. Následně se už jen odešle požadavek na server, který podle získaného čárového kódu vrátí seznam knih s tímto kódem.

4.6 Konverze aplikace na iPad

4.6.1 UI

Tablet iPad má oproti telefonu mnohem větší displej. Je velmi nevhodné aplikaci se stejnou úpravou jako na iPhonu publikovat i na větším zařízení. Velká část místa by totiž zůstala nevyužita. Proto je nutné částečně přetvořit UI.

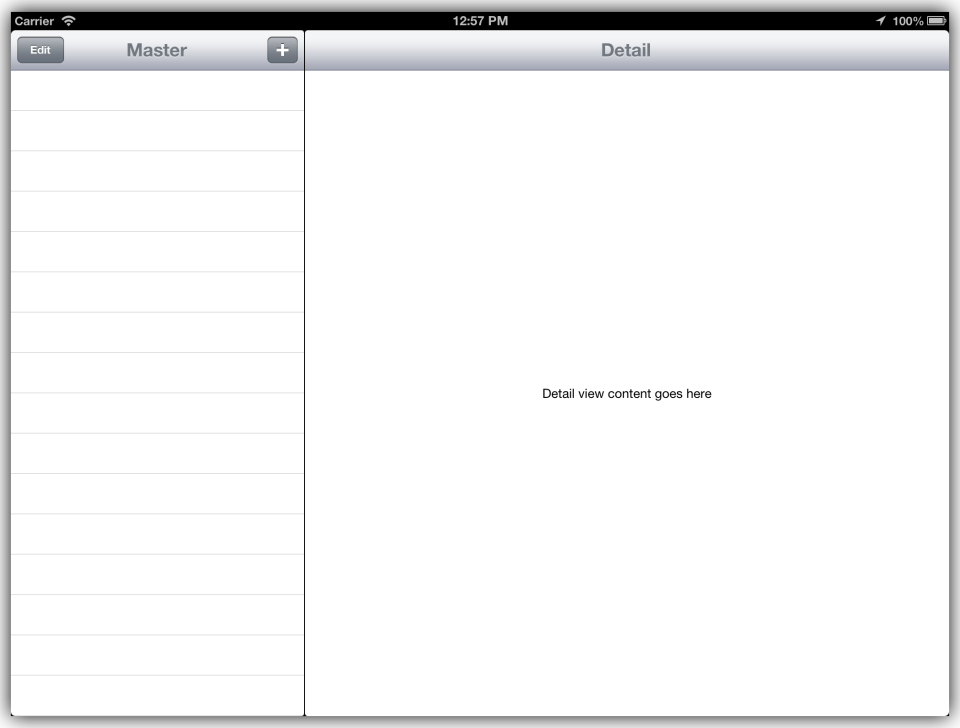
`UISplitViewController` je hojně používaným prvkem, který je možné použít pouze na iPadu. Je to controller, který je sestavený ze dvou dílčích controllerů.

Levé části se říká „master“, pravé „detail“. Při vybrání položky v levé části se zobrazí detail této položky v pravé části. Tento způsob zobrazování informací je pro aplikaci SmartLib téměř ideální. Vždy se totiž nejdříve zobrazuje seznam knih a po vybrání knihy je uživatel přesunut na detail této knihy. Díky tomu je možné v části master použít controller, který zajišťuje zobrazení seznamu knih, a v části detail controller, který zobrazuje její detail.

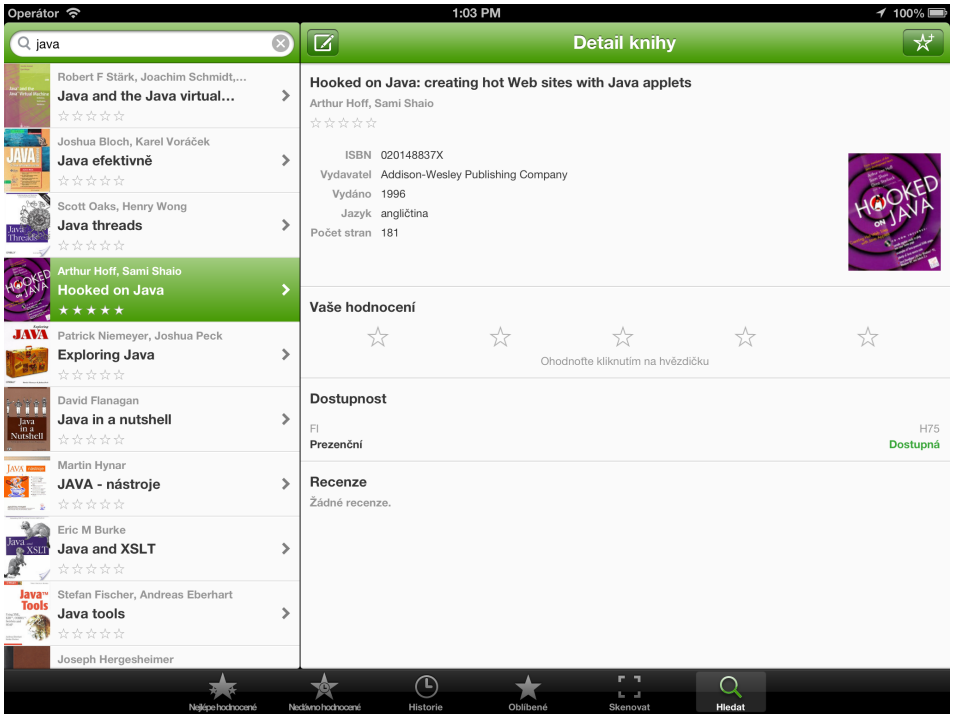
V detailu knihy je potřeba přidat prvky pro zajištění dodatečné funkcionality, jako je přidání do oblíbených, zobrazení náhledu knihy či napsání recenze. Na iPhonu je v navigační části místo pouze pro jedno tlačítko na pravé straně, na levé je tlačítko pro přechod zpět na seznam knih. Do tohoto tlačítka je tedy nutné zakomponovat více funkcí. Na iPadu je však tento postup zbytečný. iPad má mnohem širší navigační lištu, na kterou se vejde více tlačítek, a navíc v levé části nemá tlačítko pro přechod zpět, protože seznam knih je zobrazen vždy. Proto je vhodné na navigační lištu umístit tlačítek několik. Díky tomu je práce s těmito prvky na iPadu rychlejší a přehlednější.

4.6.2 Úprava struktury aplikace

Protože má iPad své zákonitosti a některé vlastnosti se oproti iPhonu liší, je potřeba do aplikace přidat části kódu, které s tímto počítají. Většinou se jedná pouze o přidání podmínky, díky níž je možné zjistit, na jakém zařízení je kód vykonáván.



Obrázek 4.13: `UISplitViewController`.



Obrázek 4.14: Výsledná podoba seznamu a detailu knihy na iPadu.

V aplikaci SmartLib bylo například potřeba zajistit to, aby se detail knihy nezobrazoval do master controlleru ale do detailu.

Dále bylo nutné připustit zobrazení tabulky ve všech orientacích zařízení. Ve výchozím stavu je totiž část master skryta v orientaci na výšku, a zobrazí se po stisku tlačítka. V této aplikaci však v této orientaci více místa pro detail nic nepřináší.

Posledním krokem byla úprava skenování. Ve verzi pro iPhone je skenování řešeno na jedné obrazovce, v jednom controlleru. U iPadu je však potřeba mít tuto část rozdělenou na dvě. V části master by měla být tabulka s výsledky skenování, v části detail buď obrazovka se skenováním, nebo detail knihy. Jediným možným řešením je v tomto případě zkopírování existujícího controlleru a vytvoření dvou nových. Ty obsahují již existující kód, avšak rozdělený na dvě části a upravený pro iPad.

4.7 Lokalizace

Po zkompletování aplikace bylo nutné vytvořit lokalizaci pro češtinu, slovenštinu a angličtinu. Jako první se musí vytvořit soubor s názvem „Localizable.strings“. Ten na každém řádku obsahuje klíč a hodnotu v následujícím formátu.

```
"No results" = "Žádné výsledky";
```

Pokud je potřeba lokalizovat řetězec, nahradí se funkcí `NSLocalizedString`. Ta se postará o to, že je text zobrazen ve správném jazyce.

```
NSLocalizedString(@"No results", @"Message when search returns
       no results");
```

Prvním argumentem je zde jméno klíče v souboru „Localizable.strings“, druhým je pouze komentář pro případného překladatele.

Lokalizace grafického rozhraní se musí dělat zvlášť, mimo tento soubor. Do verze iOS 5 včetně bylo nutné vytvořit soubor storyboard pro každý jazyk zvlášť. Od iOS 6 však stačí nechat vygenerovat soubor s řetězci nutnými pro překlad, a tyto řetězce poté přeložit.

Závěr

Cílem této práce bylo vytvoření funkční aplikace SmartLib pro platformu iOS. Tato aplikace měla zahrnovat zobrazení seznamů knih, detail knihy, náhled knihy, možnost psaní recenzí s čímž souvisela registrace a přihlášení a skenování čárových kódů.

Pro vývoj byla použita metoda SASS. Jejím prvním krokem byla analýza stávajícího fyzického modelu, neboli prostudování již existujících klientů na platformách Android a Windows Phone. Díky tomuto kroku byl odvozen stávající logický model, z nějž vyplynula požadovaná funkcionalita. Nový logický model pak obsahoval funkcionalitu ekvivalentní existujícímu logickému modelu. Jeho vzhled však musel být odlišný.

Před provedením konečné fáze, vytvoření nového fyzického modelu, bylo potřeba seznámit se s jazykem Objective-C a programem Xcode, které se pro vývoj na iOS používají. Dále bylo nutné prostudovat iOS API a s tím související hlavní frameworky Foundation a UIKit. V neposlední řadě bylo potřeba seznámit se se zásadami vývoje aplikací pro iOS. Znalost API serverové části SmartLibu také patřila mezi nutné předpoklady.

Po prostudování bylo možné přejít k vývoji samotné aplikace. Jako první bylo navrženo UI. Po něm bylo zpracováno získávání dat ze serveru. To se napojilo na zobrazování těchto dat pomocí objektů typu controller. Posledním krokem bylo přidání možnosti skenování čárových kódů a lokalizace.

Metoda SASS se ukázala být jako vhodně zvolená a úspěšná. Jejím výsledným fyzickým modelem je funkční aplikace SmartLib fungující na zařízeních iPhone, iPod Touch a iPad. Funkcionalita je ekvivalentní klientům na platformách Android a Windows Phone. Výsledná aplikace byla odeslána do App Store a úspěšně prošla schvalovacím procesem. Spustitelnou verzi lze stáhnout pod názvem „SmartLib MU“.

Do budoucna je naplánován další vývoj a rozšíření této aplikace. Tyto změny jsou cíleny zejména na uživatele. Mezi hlavní budoucí úpravy patří následující položky.

- Odstranění registrací. Tento krok totiž uživatele zdržuje a kvůli tomu odrazuje od psaní recenzí. Registrovat se mohou jen studenti Masarykovy univerzity, což znamená, že nová registrace pouze pro SmartLib je redundantním krokem. Registrace jsou zavedeny pouze z technických důvodů.

- Gamifikace. Je to použití prvků z her v neherním prostředí. Bylo by například vhodné přidat žebříčky s nejaktivnějšími uživateli nebo virtuální odměny za psaní recenzí, což by zvýšilo aktivitu uživatelů.

- Překlad signatur do lidsky čitelné podoby. Díky tomu by lokace knihy byla na první pohled zřejmá a její následné nalezení v dané knihovně by bylo mnohem jednodušší.

- Propojení se sociálními sítěmi. Týkalo by se zejména sdílení knih. Toto propojení by pomohlo rozšíření povědomí o projektu SmartLib a vedlo by k většímu rozšíření mezi veřejnost.

Literatura

[1] ŠEVČÍK, Jonáš. *Rozšířená realita na mobilní platformě Android a její aplikace v knihovnictví* [online]. 2011 [cit. 2013–05–08]. Diplomová práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Jaroslav Škrabálek. Dostupné z: <http://is.muni.cz/th/255493/fi_m/>.

[2] iOS. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001–2013 [cit. 2013–05–08]. Dostupné z: <http://en.wikipedia.org/wiki/iOS>

[3] iOS UI Element Usage Guidelines. APPLE INC. *iOS Developer Library* [online]. 2013 [cit. 2013-05-11]. Dostupné z: <http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/UIElementGuidelines/UIElementGuidelines.html>

[4] User Experience Guidelines. APPLE INC. *iOS Developer Library* [online]. 2013 [cit. 2013-05-11]. Dostupné z: <http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/UEBestPractices/UEBestPractices.html>

[5] iOS Technology Overview. APPLE INC. *iOS Developer Library* [online]. 2012 [cit. 2013-05-08]. Dostupné z: <http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html>

[6] Model-View-Controller. APPLE INC. *iOS Developer Library* [online]. 2012 [cit. 2013-05-08]. Dostupné z: <http://developer.apple.com/library/ios/#documentation/general/conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html>

[7] Objective-C. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001–2013 [cit. 2013–05–08]. Dostupné z: <http://en.wikipedia.org/wiki/Objective-C>

[8] Foundation Framework Reference. APPLE INC. *iOS Developer Library* [online]. 2012 [cit. 2013-05-08]. Dostupné z: <http://developer.apple.com/library/ios/#documentation/Cocoa/Reference/Foundation/ObjC_classic/_index.html>

[9] UIKit Framework Reference. APPLE INC. *iOS Developer Library* [online]. 2012 [cit. 2013-05-08]. Dostupné z: <http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIKit_Framework/_index.html>

[10] The iOS Environment. APPLE INC. *iOS Developer Library* [online]. 2013 [cit. 2013-05-08]. Dostupné z: <http://developer.apple.com/library/ios/

#documentation/iphone/conceptual/iphoneosprogrammingguide/
TheiOSEnvironment/TheiOSEnvironment.html>

[11] DEMARCO, Tom. *Structured analysis and system specification*. New York: Yourdon, 1978, xiv, 352 p. ISBN 09-170-7207-3.

[12] CONWAY, Joe a HILLEGASS, Aaron. *iOS Programming: The Big Nerd Ranch Guide*. 3. vyd. Atlanta: Big Nerd Ranch, 2012. ISBN 978-0-321-82152-2.

Přílohy

Součástí práce je příloha obsahující implementační část se zdrojovými kódy projektu. Příloha neobsahuje použitou knihovnu ZXingObjC ani výsledný spustitelný soubor. Aktuální verzi ZXingObjC lze stáhnout na adrese `http://github.com/TheLevelUp/ZXingObjC`. Aplikaci spustitelnou na iOS zařízeních lze stáhnout z App Store z adresy `http://itunes.apple.com/cz/app/smartlib-mu/id644335694`.