VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

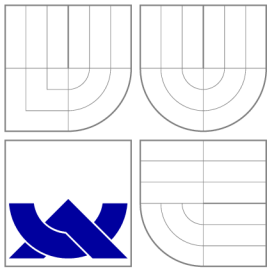
INFORMAČNÍ SYSTÉM NA CHYTRÉM TELEFONU PRO UČITELE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

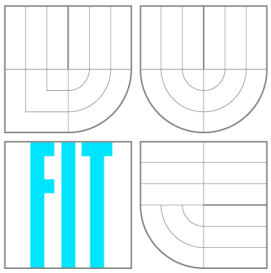
AUTOR PRÁCE
AUTHOR

JIŘÍ KRATOCHVÍL

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

INFORMAČNÍ SYSTÉM NA CHYTRÉM TELEFONU PRO UČITELE

INFORMATION SYSTEM ON SMARTPHONE FOR TEACHERS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ KRATOCHVÍL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROSLAV ROZMAN, Ph.D.

BRNO 2014

Abstrakt

Cílem práce je vytvoření informačního systému pro učitele na chytré telefony s operačním systémem *iOS*. Aplikace má zjednodušit práci učitelů s dokumenty třídní kniha, karty žáků a vlastní poznámky učitele. Aplikace byla vyvinuta v jazyce *Objective-C* v prostředí *Xcode*. Byla otestována prostředky k tomu určenými a následně předvedena potencionálním uživatelům. Na základě vyplněných dotázníků od uživatelů bylo stanoveno hodnocení aplikace. To je poměrně kladné a většina uživatelů si dokáže představit nasazení aplikace do reálného provozu.

Abstract

The purpose of this thesis was to create information system for teachers for smartphones with iOS operation system. The application should simplify work of teachers with documents such as classbook, student card and teachers own notes. The application was developed in Objective-C language in Xcode, then was tested by tools designed for this purpose and presented to potential users. The application was rated based on users feedback, taken by questionare, which was rather positive and most of asked users could imagine deploying the application into real traffic.

Klíčová slova

Informační systém, učitel, iOS, Xcode, Objective-C, iCloud.

Keywords

Information system, teacher, iOS, Xcode, Objective-C, iCloud.

Citace

Jiří Kratochvíl: Informační systém na chytrém telefonu pro učitele, bakalářská práce, Brno, FIT VUT v Brně, 2014

Informační systém na chytrém telefonu pro učitele

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana, Ph.D.

.....
Jiří Kratochvíl
20. května 2014

Poděkování

Chtěl bych poděkovat svému vedoucímu práce panu Ing. Jaroslavu Rozmanovi, Ph.D., který vedl moji práci a věnoval mi čas a cenné rady při konzultacích. Dále bych chtěl poděkovat mým rodičům za podporu a poskytnutí potřebných prostředků pro vytvoření této práce. Rád bych poděkoval také všem uživatelům fóra *stackoverflow*, kteří zodpovídali mé otázky a v neposlední řadě všem, kteří mě podporovali během tvorby práce.

Obsah

Kapitola 1

Úvod

Cílem práce je vytvoření aplikace, se kterou budou moci učitelé denně pracovat a bude jim přinášet ulehčení v jejich zaměstnání. Jedná se především o snížení časové náročnosti a zvýšení bezpečnosti práce s několika dokumenty, které je nutné uschovávat po určitou časovou periodu. Aplikace má zastupovat práci s třídní knihou, zápisníkem učitele a kartami žáků.

Důvod vzniku je také ten, že při ztrátě jakéhokoliv z výše zmíněných dokumentů přichází učitelé do nepříjemných situací, kdy musí pracně získávat ztracené informace nazpět. Toho se však při práci s aplikací, která má data uložena na sdíleném úložišti bát nemusí.

Vývoj informačního systému se skládá z několika částí, které na sebe navazují. Tato problematika vývoje je přehledně a velmi podrobně popsána ve skriptech [2], ze kterých bylo čerpáno i při vývoji tohoto informačního systému.

První kapitola je úvodem do problematiky programování *iOS* aplikací. Shrnuje potřebné prostředky pro vývoj aplikací a možnosti jak aplikace vyvíjet. Okrajově popisuje nástroje, které byly využívány při vývoji systému.

Další kapitola popisuje návrh aplikace. Ten je složen z několika kroků. Nejdříve byla provedena specifikace, jejímž základem bylo zadání práce. Mezi učiteli proběhl průzkum, jehož účelem bylo zmapovat jejich momentální práci s třídní knihou a dalšími dokumenty. Poté jim byla představena základní funkcionalita aplikace s prosbou o navržení dalších funkcí, které by jim ulehčily práci. Na základě jejich návrhů byla stanovena konečná specifikace a vytvořen diagram případů užití. Poté bylo navrženo grafické uživatelské rozhraní. V posledním kroku byl proveden návrh datového modelu pomocí *ERD*.

Čtvrtá kapitola, implementace, popisuje strukturu vytvořené aplikace. Převod návrhu datového modelu do modelu *Core Data*. Dále jsou zde popsány komponenty, které byly využity, a také se zde nachází několik popsaných ukázek metod včetně jejich kódu, který je uveden v rámci příloh. Popsána je zde i integrace vzdáleného úložiště *iCloud*.

Poslední kapitola se věnuje testování. Popisuje jednotlivé způsoby, kterými byla aplikace testována. Každá podkapitola odpovídá jednomu způsobu testování. V kapitolách je popsáno co a s jakým výsledkem bylo testováno.

Kapitola 2

Vývoj aplikací pro iOS

Vývoj aplikací pro platformu *iPhone* či *iPad* je zaštítěn několika věcmi, které si musí každý programátor opatřit a bez nichž není možné aplikace vyvíjet. Jelikož je vyvíjená aplikace určena primárně pro *iPhone* a bude na něm vyvíjena i testována, budu se dále zabývat pouze tímto zařízením.

2.1 iPhone

Chytrý telefon není nutně vyžadován pro vývoj aplikací. Na otestování aplikace poslouží vestavěný simulátor v prostředí *Xcode* (vývojové prostředí pro vývoj *iOS* a *OS X* aplikací), který zobrazuje výsledný produkt stejně, jak je tomu na displeji mobilního telefonu. Nemusí však vše fungovat v simulátoru naprosto identicky, jako na fyzickém zařízení. Například odezva aplikace v telefonu nemusí být tak svižná nebo ovládání pomocí dotykového displeje tak důmyslné jako v simulátoru. Je tedy doporučováno obstarat si potřebné zařízení pro kvalitní otestování aplikace.

2.1.1 Vybavení

Aplikace určené pro mobilní zařízení využívají různých funkcí telefonu. *iPhone* od verze *3GS* a vyšší nabízí mnoho technologických prvků, na kterých aplikace mohou stavět a nebo je vylepšovat. Informace o vybavení chytrých telefonů *iPhone* byly čerpány ze zdroje [4].

Mobilní telefony *iPhone* mají 3.5 nebo 4 palcový displej, který umožňuje kompletní ovládání pomocí dotyků. Ovládání aplikací lze rozšířit o gesta, která slouží například pro přiblížení, oddálení či přetočení zobrazované plochy. Ke komunikaci s ostatními zařízeními slouží technologie pro příjem a odesílání mobilních dat (*GSM, 3G, 4G*). Zařízení jsou schopna přijímat data i prostřednictvím *Wi-fi* a nebo *Bluethooth* ve verzi 2.0 nebo 4.0.

K dokumentaci okolí poslouží vestavěný fotoaparát a videokamera, jenž disponuje také bleskem (*iPhone* 4 a vyšší). U modelů *iPhone* 4 a vyšší lze využít i přední kameru/fotoaparát.

Zařízení obsahuje senzory pro snímaní intenzity světla, polohy či magnetického pole (*iPhone 4* a vyšší). Pro přesnou lokalizaci zařízení slouží vestavěná *GPS*. Vývoj herních aplikací podporuje multiplatformní rozhraní pro vytváření grafických aplikací *OpenGL* a gyroskop umožňující kvalitnější ovládání.

V poslední verzi *iPhone* se objevila i čtečka otisků prstů pro zabezpečený přístup. Operační systém verze 7.0 umožňuje plnohodnotný multitasking (*iPhone 4* a vyšší). Aplikace běží na pozadí, aniž by o tom uživatel musel vědět.

2.2 Objective-C 2.0

Nedílnou součástí vývoje jakýchkoliv aplikací je znalost potřebného programovacího jazyka. V případě vývoje pro *iOS* se jedná o objektový jazyk *Objective-C 2.0* [6].

2.2.1 Vývoj jazyka

Jazyk využívaný při vývoji *iOS* aplikací je nadstavbou jazyka C, který vznikl na začátku 70. let dvacátého století. Objektově orientovanou verzi jazyka C, vycházející z jazyka *SmallTalk-80*, navrhl v 80. letech Brad J. Box. Jazyk byl v roce 1988 licencován společností *NeXT Software*, kterou založil Steve Jobs potom, co byl donucen opustit společnost *Apple* v roce 1985. Po rozhodnutí společnosti *Apple* v roce 1996 o odkoupení *NeXT Software* a opětovném návratu S. Jobse do *Apple*, byl jazyk použit k vytvoření operačního systému *OS X*, což mělo zásadní vliv na další vývoj společnosti. V roce 2007 společnost uvolnila aktualizovanou verzi jazyka nazvanou *Objective-C 2.0*.

Po vydání prvního *iPhone* v roce 2007 a nátlaku vývojářů na společnost *Apple*, byla vytvořena první vývojářská sada *Software Development Kit* (SDK), která sloužila k rychlému vývoji a testování aplikací pro *iPhone*. Jak bylo zmíněno výše, základem jazyka *Objective-C 2.0* je samostatný jazyk C, který je plně využitelný i v aplikacích psaných v objektovém jazyce, včetně všech knihoven.

2.2.2 MVC – Model–View–Controller

Aplikace programované v jazyce *Objective-C* využívájí architekturu *MVC*, která zpřehledňuje a strukturuje aplikaci. Názvy jednotlivých částí architektury jsou zažité v anglické terminologii i v České republice, proto nebudu uvádět jejich český překlad. Architektura je složena ze tří vrstev *Model, View* a *Controller*. V následujícím zbytku kapitoly bylo čerpáno ze zdroje [1].

Model

Model definuje datovou strukturu aplikace. Zapouzdřuje data, která jsou perzistentně uložena a udávají stav systému. Data uložená v *Model* se využívají při zobrazení výstupu, s kterým pracuje uživatel. Jsou modifikována pomocí vrstvy *Controller* v závislosti na nějaké události, např. požadavky uživatele.

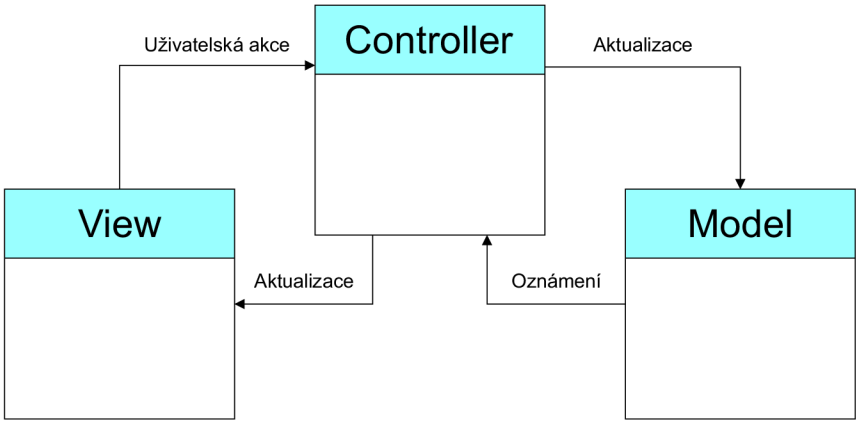
View

Vrstva *View* se stará o zobrazení dat a umožňuje uživateli data nepřímo editovat. Disponuje metodami pro optimalizaci výkonu zobrazených dat, jako je například ukládání dat do vyrovnávací paměti. Data, která zobrazuje, mohou tvořit jen určitou část vrstvy *Model* a nebo může být vrstva vizualizována celá. Vrstva *View* by měla zajistit korektní zobrazení dat. Zároveň musí být informována o aktualizaci dat, aby mohla data zobrazovat v požadovaném stavu.

Controller

Zprostředkovatel mezi vrstvou *View* a daty (*Model*). Prostřednictvím rozhraní jsou jí odesílány jednotlivé požadavky od uživatele. Na základě typu požadavku získá potřebná data od vrstvy *Model* nebo data do vrstvy *Model* uloží. *Controller* může s přijatými daty dále

pracovat, modifikovat je, provést nějakou akci na základě jejich hodnot a nebo je pouze vrátit. Obrázek 2.1 zachycuje *MVC* model využívaný v architekturách iOS aplikací.



Obrázek 2.1: Struktura a vzájemná interakce mezi jednotlivými částmi aplikace založené na architektuře *MVC*.

2.3 Vývojová prostředí

Hlavním prostředím pro vývoj *iOS* aplikací je nástroj *Xcode*. Ten je podporován pouze pod operačními systémy *OS X*. Existují tři možnosti, jak aplikace vyvíjet.

Být vlastníkem jednoho z výrobků společnosti *Apple* (*Macbook Pro, Macbook Air, iMac, Mac mini, Mac pro*) a vyvíjet aplikace v prostředí *Xcode*.

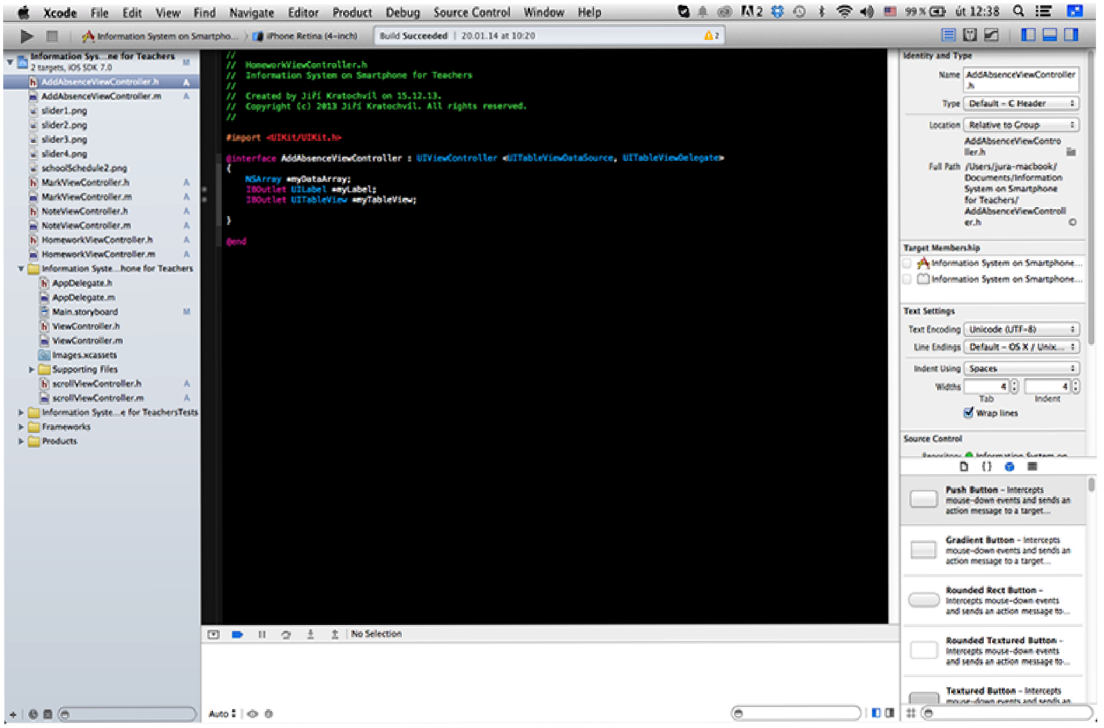
Další možností je koupě operačního systému *OS X* a instalace na dostupném zařízení. Systém by měl jít nainstalovat buďto přímo, například s využitím možnosti plnohodnotné instalace dvou operačních systému na jedné platformě (*dual boot*) a nebo za pomoci aplikace *VirtualBox* či *VMWare* jako virtuální mašina. Instalace není úplně jednoduchá a můžou se vyskytnout problémy. Například procesor v zařízení, na které je operační systém instalovaný, musí podporovat virtualizaci, jinak systém nebude fungovat.

Poslední možností je vyvíjet v nástrojích dostupných pro jiné platformy než je *Mac*. Například využití frameworků *PhoneGab, jQuery mobile* nebo vývojového prostředí *Oracle ADF mobile*, pomocí kterého lze vyvíjet aplikace v jazyce *Java* v kombinaci s *HTML5*. V tomto případě se jedná spíše o vývoj webových aplikacích primárně určených pro telefony, než nativních aplikací pro *iOS*.

2.4 Xcode

Xcode lze bezplatně stáhnout na stránkách určených pro vývojáře *Apple*. Jedná se o ucelený nástroj pro vývoj *iOS* a *OS X* aplikací. Přehledné a rychle pochopitelné prostředí sdružuje několik prostředků usnadňujících vývoj aplikace. *iSimulátor* pro testovaní *iOS* aplikací, prostředí pro návrh uživatelského rozhraní (*Storyboard*) nebo nástroj pro návrh datového modelu (*Core Data*).

Obrázek 2.2 zachycuje základní vývojové prostředí *Xcode*, které nabízí mnoho prvků a pro neznalého uživatele může působit robustně a těžkopádně, ale po seznámení s nástroji nečiní práce ve vývojovém prostředí problémy. Pomocné panely lze velmi jednoduše a rychle skrýt a zobrazit například jen samostatný editor kódu bez dalších rušivých elementů. Editor obsahuje asistenta, který poskytuje automatickou nabídku slov či opravu chyb.

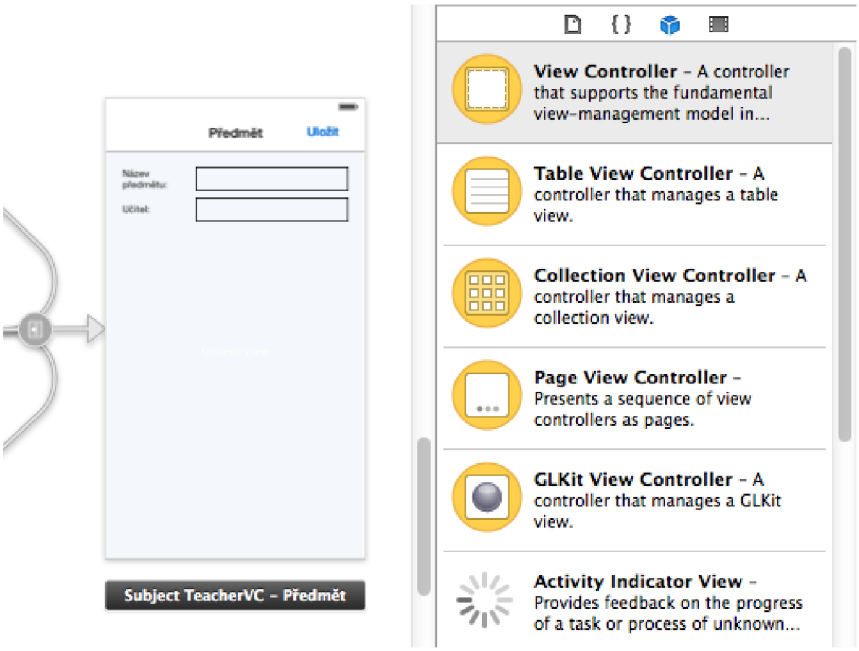


Obrázek 2.2: Vývojové prostředí *Xcode*.

2.4.1 Storyboard

Pro návrh uživatelského rozhraní je do prostředí začleněn nástroj *Storyboard*. Metodou táhni a pusť lze jednoduše navrhnout uživatelské rozhraní. Stačí vybrat z nabídky požadovaný prvek nebo obrazovku a přetáhnout na plochu pro zobrazení navrženého grafického uživatelské rozhraní.

Prvky lze mezi sebou spojovat šipkami, které znázorňují interakce mezi uživatelem a aplikací. Jinými slovy zobrazuje přechody mezi jednotlivými obrazovkami aplikace v závislosti na uživatelském vstupu. Na obrázku 2.3 je zobrazena práce s nástrojem. V pravé části obrázku se nachází výběr komponent. Na levé straně je plocha pro zobrazení a tvorbu návrhu.



Obrázek 2.3: Návrh grafického uživatelského rozhraní pomocí nástroje *Storyboard* (vpravo výběr z možných prvků, vlevo plocha pro tvorbu návrhu).

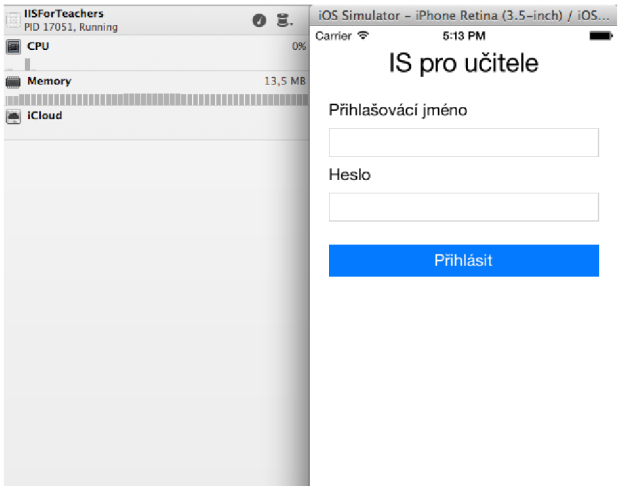
2.4.2 Core Data

Nástroj *Core Data* slouží k návrhu a implementaci datového modelu. Pomocí něj lze jednoduše vytvořit datové entity, přidávat jim atributy a vzájemně je propojovat mezi sebou podle jejich závislostí.

Výhodou návrhu datového modelu pomocí *Core Data* je fakt, že z návrhu lze ihned vygenerovat zdrojové soubory tříd, odpovídající jednotlivým entitám. Na obrázku 4.2 v kapitole 4.1 je vyobrazen jednoduchý datový model vytvořený pomocí *Core Data*.

2.4.3 iSimulator

K testování aplikace slouží nástroj *iSimulator*, který umožňuje rychlé testování aplikace včetně měření spotřebovávané paměti a procesoru. Aplikaci můžeme simulovat na několika druzích zařízení. K dispozici jsou *iPhone* s velikostí displeje 3.5 a 4 palce v provedení s *retina* displejem a *iPad* se zobrazením na normálním i retina displeji. Na obrázku 2.4 je zachycen *iSimulator* (pravá část) včetně využití paměti a procesoru (levá část).



Obrázek 2.4: *iSimulator* (společně s využitím procesoru a volatilní paměti).

2.5 PhoneGab – jQueryMobile

Aplikační platforma *PhoneGab* je založena na jazycích *HTML5* a *JavaScript*. Umožňuje vytvářet aplikace pro více mobilních platforem současně (*iOS, BlackBerry, Windows Phone 7, Bada* atd.) s minimálními změnami ve zdrojovém kódu. *PhoneGab* umožňuje přistupovat k funkčním prvkům mobilních zařízení, které jsou důležité pro tvorbu aplikací(fotoaparát, kompas, senzory světla, senzory polohy atd.).

Uživatelské rozhraní je vytvářeno pomocí jazyků *HTML5* a *CSS3*. Vytvářet aplikace lze v libovolném textovém editoru. V této kapitole bylo čerpáno ze zdroje [3].

2.6 Oracle ADF mobile

Oracle ADF mobile slibuje nejjednodušší současný vývoj aplikací pro platformy *iOS i Android* se stejným zdrojovým kódem aplikace. Je založen na jazycích *Java, Javascript, HTML 5 a CSS*.

Kapitola 3

Návrh aplikace

Návrh aplikace se skládá z následujících bodů:

1. Specifikace aplikace - určení funkcionality.
2. Vytvoření diagramu užití a jeho textový popis.
3. Návrh uživatelského rozhraní. Vytvoření všech obrazovek a vzájemné interakce mezi nimi.
4. Navržení datového modelu pomocí *ERD*.

3.1 Specifikace

3.1.1 Předmět zájmu

Tento projekt vznikl pro úsporu času a zjednodušení práce učitelů zejména na středních a základních školách. Výsledná aplikace přináší pro učitele sjednocení důležitých informací do jednoho místa, mobilního telefonu. Díky uložení dat prostřednictvím služby *iCloud* je možné k datům přistupovat odkudkoliv prostřednictvím různých zařízení. Data jsou zároveň zálohována a tedy i jednoduše obnovitelná při ztrátě mobilního řazení.

3.1.2 Průzkum mezi učiteli

Průzkum mezi učiteli byl proveden na střední škole. Učitelům byla popsána aplikace, její smysl a základní funkcionalita. Zároveň byli požádáni o doplnění dalších funkcí aplikace a popsání současného stavu na škole.

Současný stav Práce učitele v oddělení administrace se skládá z několika částí. Jedná se o práci s kartami žáků, třídnicemi, rozvrhy a vlastním deníkem pro zápis známek, aktivit, laboratorních cvičení atd. Případně s elektronickým systémem pro zápis známek.

Karta žáka Karta žáka je dokument, který uchovává všeobecné informace o žákovi. Obsahuje jméno, příjmení, datum narození, pojišťovnu, kontakt, číslo občanského průkazu, zdravotní údaje a údaje o rodičích. Vzor karty můžeme vidět na obrázku 3.1, který byl zapůjčen od Masarykovy základní školy Klánovice v Praze.

Kartu udržuje v aktuálním stavu třídní učitel. Pravidelná aktualizace je prováděna po jednom roce a nebo při nahlášené změně u žáka. Originál karty je uchováván u třídního

učitele, kopie pak u zástupce ředitele. Pokud chce jiný vyučující nahlédnout do karty, například chce kontaktovat jednoho z rodičů, musí si kartu vypůjčit od výše zmíněných osob. Jiný přístup ke kartě neexistuje. Karta je po ukončení studia uložena do školního archivu.

MASARYKOVA ZÁKLADNÍ ŠKOLA, SLAVĚTÍNSKÁ 200, PRAHA 9 - KLÁNOVICE

Žádost o přijetí žáka do ZŠ Klánovice

K rukám ředitele školy PaedDr. Michala Černého

Žádám o přijetí dítěte:

Příjmení a jméno: ..... Rodné číslo:.....

Místo nar.: ..... Okres nar.(příp. stát):.....

Stát.příslušnost: ..... **do ..... ročníku ZŠ Klánovice.**

Počet ukončených let školní docházky: ..... Nástup od (datum): .....

Trvalé bydliště:

PSČ ..... Adresa .....

Bydliště (liší-li se od trvalého)

PSČ ..... Adresa .....

Telefon domů: ..... Zdravotní pojišťovna: ..... Kód ZP: .....

Zdravotní problémy: .....

Otec

Příjmení a jméno: .....

Bydliště (liší-li se) .....

Zaměstnání:..... Telefon do zaměstnání:.....

Mobilní telefon: ..... e-mail: .....

Matka

Příjmení a jméno: .....

Bydliště (liší-li se): .....

Zaměstnání:..... Telefon do zaměstnání:.....

Mobilní telefon: ..... e-mail: .....

Pokud nechcete sdělit své zaměstnání, vyplňte prosím pouze telefony.

Cizí jazyk, který žák dosud studoval jako hlavní (ve škole): .....

Adresa předchozí školy: .....

.....

Další přání a sdělení škole:

V ..... dne .....

.....

podpis zákonného zástupce žáka

Obrázek 3.1: Vzorový příklad karty žáka (Masarykova základní škola Klánovice v Praze).

Třídní kniha Do třídní knihy se zapisuje každá hodina, která byla odučena. U hodiny je uvedena zkratka předmětu, číslo hodiny, téma hodiny, absence žáků a podpis učitele. Učitel může napsat poznámku o tom, kdo byl neukázněný, dorazil pozdě nebo jiným způsobem narušil výuku. První strana třídní knihy je určena pro absenci a píší se do ní součty zameškaných hodin v rámci celého týdne. Na konci pololetí se dělá celkový součet hodin podle týdnu.

Kopie třídní knihy ani její elektronický ekvivalent neexistuje. Při ztrátě nebo odcizení je nutné knihu napsat znovu s následujícím postupem. Podle rozvrhu a změn v rámci jednotlivých dní (změnami je myšleno suplování předmětů) vyhledat učitele a nechat jej dopsat hodiny, které odučil. Téma hodiny je odvozeno od tematického plánu školního roku. Záznamy o absenci jsou obnoveny podle omluvenek v indexech studentů. Jediná možná obrana proti této situaci je průběžné zálohování třídní knihy (např. vytváření kopií třídní knihy v pravidelných časových intervalech).

Zápisník známek Každý učitel používá svůj vlastní zápisník nebo sešit, do kterého si značí známky, odevzdané domácí úkoly, laboratorní cvičení atd. Některé školy mají svůj informační systém, kam mohou známky zapisovat. Škola, na které byl proveden průzkum, používá informační systém SaS (systém agendy škol). Do systému je možno zapisovat známky. K nim mají přístup i rodiče žáků prostřednictvím webové aplikace.

Systém využívá každý učitel rozličně. Někteří do něj zapisují ihned po vyhodnocení testu, někteří jednou za měsíc a někdo vůbec. Do systému se dají zapisovat pouze známky, nikoliv laboratorní cvičení, aktivita nebo známkám přidělovat nějakou váhu, jak to dělají někteří učitelé. Například pololetní písemná práce má dvojnásobnou váhu, nežli ostatní písemky. Proto všichni kantoři používají své zápisníky. Při jejich ztrátě nastávají problémy a musejí známky zjišťovat z indexu studentů. Tam si známky zapisují studenti sami a ne všichni jsou tak svědomití a poctiví a mají zapsány všechny známky.

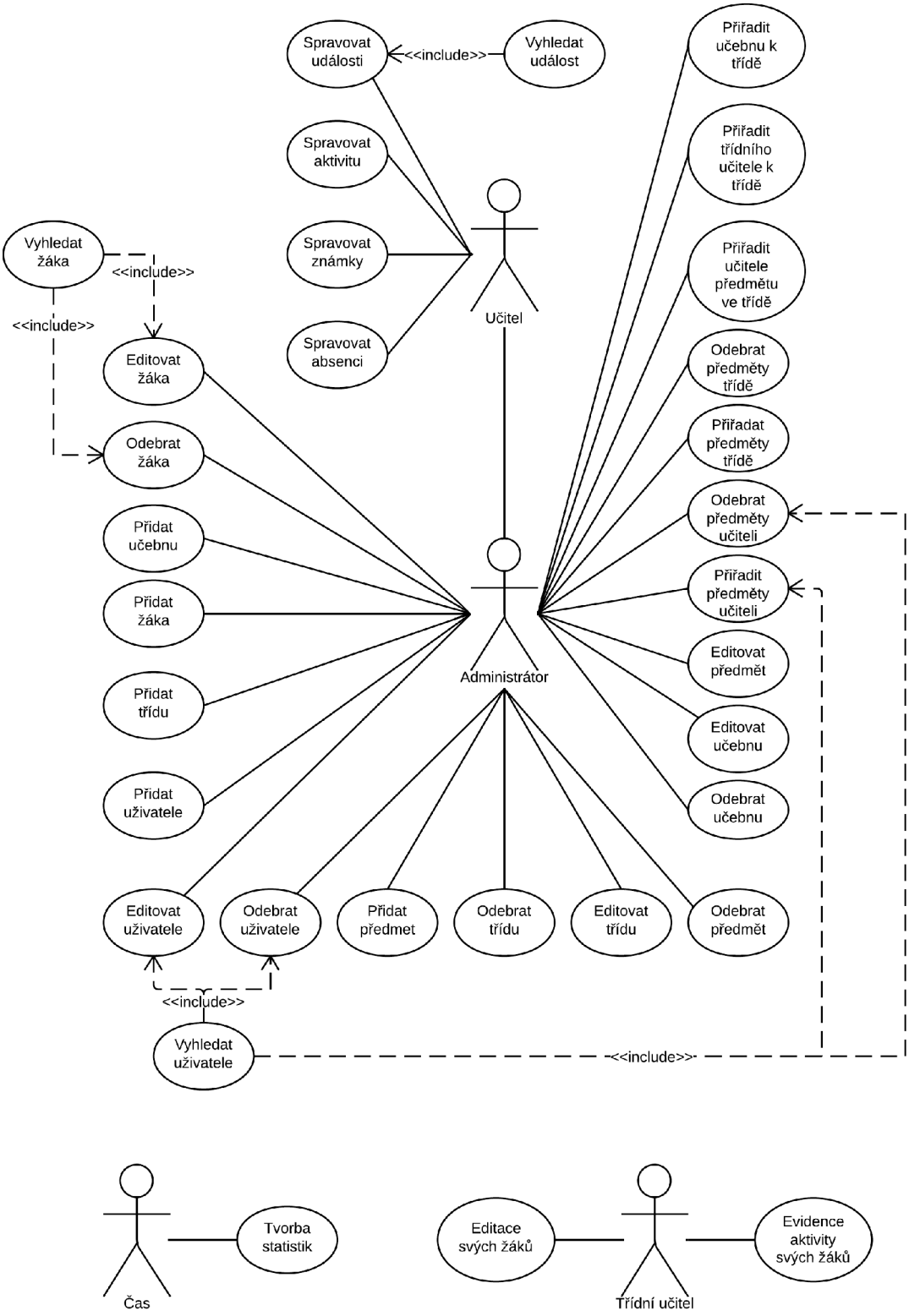
Rozvrhy Rozvrh vytváří zástupce ředitele ručně. Každý učitel může vznést návrh na úpravu svého osobního rozvrhu před jeho vytvořením. Například učitel zažádá o volné pátky kvůli navštěvování vysoké školy. Na základě rozvrhu jednotlivých učitelů a jejich zaměření je zúžen výběr učitele při suplování. Učitel je také povinen mít dozor o přestávkách v určitých částech školy. Rozvrh dozoru je stejně jako klasický rozvrh vygenerován a platí celý školní rok.

Funkcionalita Základní funkcionalita byla určena v rámci zadání bakalářské práce. Po provedení průzkumu a zjištění představ učitelů byly doplněny následující funkce.

Funkce přidané na základě průzkumu:

- Vytvoření domácích úkolů a testů pro celou třídu
- Evidence účasti na písemkách a domácích úkolech
- Plnohodnotné informace o žákovi (karta žáka)
- Možnost evidence absence (včetně omluvenek)

3.2 Diagram případu užití



Obrázek 3.2: Diagram případu užití navrhované aplikace.

V diagramu užití (obrázek 3.2) vystupují čtyři role. Učitel, třídní učitel, administrátor a čas. Administrátor je nadřazený učiteli. Využívá stejné funkce aplikace jako on. Navíc pracuje s databází tříd, předmětů a žáků, která je sdílená pro všechny učitele. Také upravuje databázi uživatelů (přidání/odebíraní). Třídní učitel disponuje dvěma funkcemi navíc, narozdíl od učitele. Detailní popis všech případů užití následuje níže.

Role: učitel, třídní učitel, administrátor

Případy užití:

- Vyhledat žáka
  - Vyhledání žáka podle jeho jména. Zahrnující (*include*) případ užití využívaný při správě známek, absence a editace karty žáka.
- Spravovat známky
  - Přidávání nových známek k žákovi a možnost jejich další editace. Známky jsou přidávány v rámci předmětu.
  - Známce lze přiřazovat váhu podle důležitosti písemky. Váha známky se projeví ve výpočtu průměru.
- Spravovat absenci
  - Evidence absence žáka. Přidání absence žákovi podle data, předmětu a třídy. Rozlišení omluvených a neomluvených hodin, jejich editace (změna stavu omluvené/neomluvené).
- Spravovat události
  - Přidání nové události pro třídu (test, domácí úkol). Evidence všech přidaných událostí.
- Vyhledat událost
  - Zahrnující případ užití využívaný při správě událostí (test, domácí úkol).
- Spravovat aktivitu
  - Rychlé přidávání a odebírání bodů určující aktivitu žáka. Možnost evidence aktivity žáka v závislosti na ostatních žácích. Evidence aktivity ve všech proběhlých hodinách.
- Přidat/odebrat předměty učiteli
  - Přidávání a odebírání předmětů, na které má učitel akreditaci.
- Přidat/odebrat předměty třídě
  - Přidávání a odebírání předmětů, které jsou vyučovány ve třídě.
- Přiřadit učitele předmětu ve třídě

  - Přiřazení učitele ke konkretnímu předmětu ve třídě. Jinými slovy určuje např. kdo vyučuje v 7.A předmět biologie.

- Přiřadit třídního učitele k třídě
  - Přiřazení třídního učitele k třídě. Učitel má poté pravomoce sledovat veškeré aktivity, známky, absence atd. svých žáku a editovat jejich karty.

- Přiřadit učebnu k třídě
  - Přiřazení mateřské učebny k třídě.

Role: administrátor

Případy užití:

- Přidat/odebrat žáka
  - Přidání/odebrání žáka, včetně jeho vyplněné karty z/do sdílené databáze.

- Přidat/odebrat učebnu
  - Přidání/odebrání učebny jednotlivých tříd.

- Přidat/odebrat třídu
  - Přidání/odebrání třídy z/do sdílené databáze.

- Přidat/odebrat předmět
  - Přidání/odebrání předmětu z/do sdílené databáze.

- Odebrání/přidání uživatele
  - Vytvoření nového uživatelského účtu s rolí učitele nebo administrátora.

- Editace třídy
  - Editování detailních informací o třídě.

- Editace předmětu
  - Editovaní informací o předmětu.

- Editace učebny
  - Editovaní informací o učebně.

- Editace uživatele
  - Úprava uživatelského profilu (heslo, jméno, informace o učiteli nebo administrátorovi).

- Vyhledání uživatele
  - Zahrnující případ užití pro správu uživatelů. Při editaci uživatele existuje možnost uživatele nejdříve vyhledat.

- Editovat žáka
  - Úprava detailních informací (karta žáka) žáka.

Role: třídní učitel

Případy užití:

- Editace svých žáků
  - Třídní učitel může editovat karty žáků ve své třídě.
- Evidence aktivity svých žáků
  - Evidence veškeré aktivity (známky, absence, aktivita v hodinách atd.) všech žáků třídy, kde je učitel třídním učitelem.

Role: čas

Případy užití:

- Tvorba statistik
  - S přidávaním domácích úkolů, testů, absence atd. se vytváři statistiky o třídách a žácích.

3.3 Grafické uživatelské rozhraní

3.3.1 Ručně kreslený návrh

Návrh grafického uživatelského rozhraní pomocí tužky a papíru je způsob poměrně přehledný a rychlý. Lze nakreslit jednotlivé obrazovky na zvolené úrovni abstrakce (nemusí obsahovat všechny prvky) a poté je vzájemně propojit a znázornit tak interakci aplikace. Při kreslení návrhu ve větším měřítku je nutné zvolit správné rozmístění prvků (např. obrazovku s hlavním menu umístit doprostřed náčrtu) aby se nestalo, že čáry představující přechody mezi jednotlivými obrazovkami povedou skrz celý návrh.

Pokud je navrhována větší aplikace, je výhodné a v některých případech i nutné učinit dekompozici návrhu na několik podsekcí. To značně zpřehledňuje návrh.

3.3.2 Storyboard

Pro převod návrhu do digitální podoby slouží nástroj *Storyboard* umístěný přímo ve vývojovém prostředí *Xcode*. Vytváření je podobné kreslení na papír, jen není potřeba kreslit každou obrazovku a element v ní zvlášť, ale pouze metodou táhni a pusť vytvořit jednotlivé obrazovky a stejným způsobem je naplnit požadovanými prvky. Mezi prvky lze vyznačit přechody pomocí šipek, které se však umí nepříjemně kroutit a přímo z grafického návrhu není mnohokrát jasné odkud a kam přesně vedou. Pokud aplikace obsahuje větší počet obrazovek, s velkou pravděpodobností vznikne větší skrumáž obrazovek, v které nemusí být jednoduché se vyznat. Proto je důležité již od začátku vývoje členit jednotlivé obrazovky aplikace do menších skupin dle logické struktury.

Výhodou *Storyboard* je fakt, že návrh probíhá přímo ve vývojovém prostředí a navržené rozhraní lze ihned použít v aplikaci. Návrh lze spustit pomocí *iSimulator* a otestovat správné propojení mezi jednotlivými prvky.

V následujicím přehledu jsou uvedeny základní prvky pro návrh uživatelského rozhraní. Názvy prvků jsou uvedeny v anglickém jazyce, jelikož se jedná o zavedené pojmy a neexistují pro ně jednotné překlady. Některé prvky jako *Buttons* či *Switch* se dají přeložit, ale pro udržení celistvosti jsou uvedeny také v anglickém jazyce.

Storyboard obsahuje mnoho elementů, které lze vkládat a poté využívat v aplikaci. Každý prvek má vlastní popisek přímo v nápovědě *Xcode*.

Jednotlivé prvky se vkládají do kontroléru (obrazovka), který je obaluje a poté zobrazuje. Existuje několik druhů kontrolérů. Kontroléry jsou uváděny v anglickém názvosloví, protože český ekvivalent neexistuje.

Controllers: Každá obrazovka, v které se nachází obsah, se nazýva *Controller*. Jeho název je z pravidla vytvořen na základě jeho využití.

- ViewController
  - Základní obrazovka pro všechny aplikace. Může obsahovat většinu elementů jako *Toolbar, NavigationBar, Buttons, Labels* atd. Podporuje změnu velikostí a orientace prvků při změně orientace zařízení.
- TableViewController
  - Modifikovaná obrazovka pro zobrazování většího počtu záznamů stejného typu. Hodí se například na výpis seznamů.
- PageViewController
  - Má v sobě několik obsahů (*ControllerView*), mezi obrazovkami lze přecházet pomocí gest nebo stiskem tlačítka. Jinými slovy se jedná o posuvník (*Slider*), umožňující rychlý přechod mezi obrazovkami.
- TabBarController
  - Funguje na podobné bázi jako *PageViewController*. Je opět složen z několika obrazovek, s tím rozdílem, že přepínaní mezi obrazovkami je ovládáno pomocí spodního panelu (*Toolbaru*) složeného z několika záložek (*Tab*).

Základní prvky: Pod *Controllers* se nacházejí ve výběru klasické prvky, kam patří obvyklé elementy každé aplikace. Například *Label*, *Button*, *Text field* atd. Mezi více specifické prvky pro mobilní aplikace patří:

- Switch
  - Používá se např. v nastavení aplikace pro zapnutí/vypnutí vlastnosti.
- Activity Indicator
  - Je využívá při načítání dat do aplikace (stav čekání).

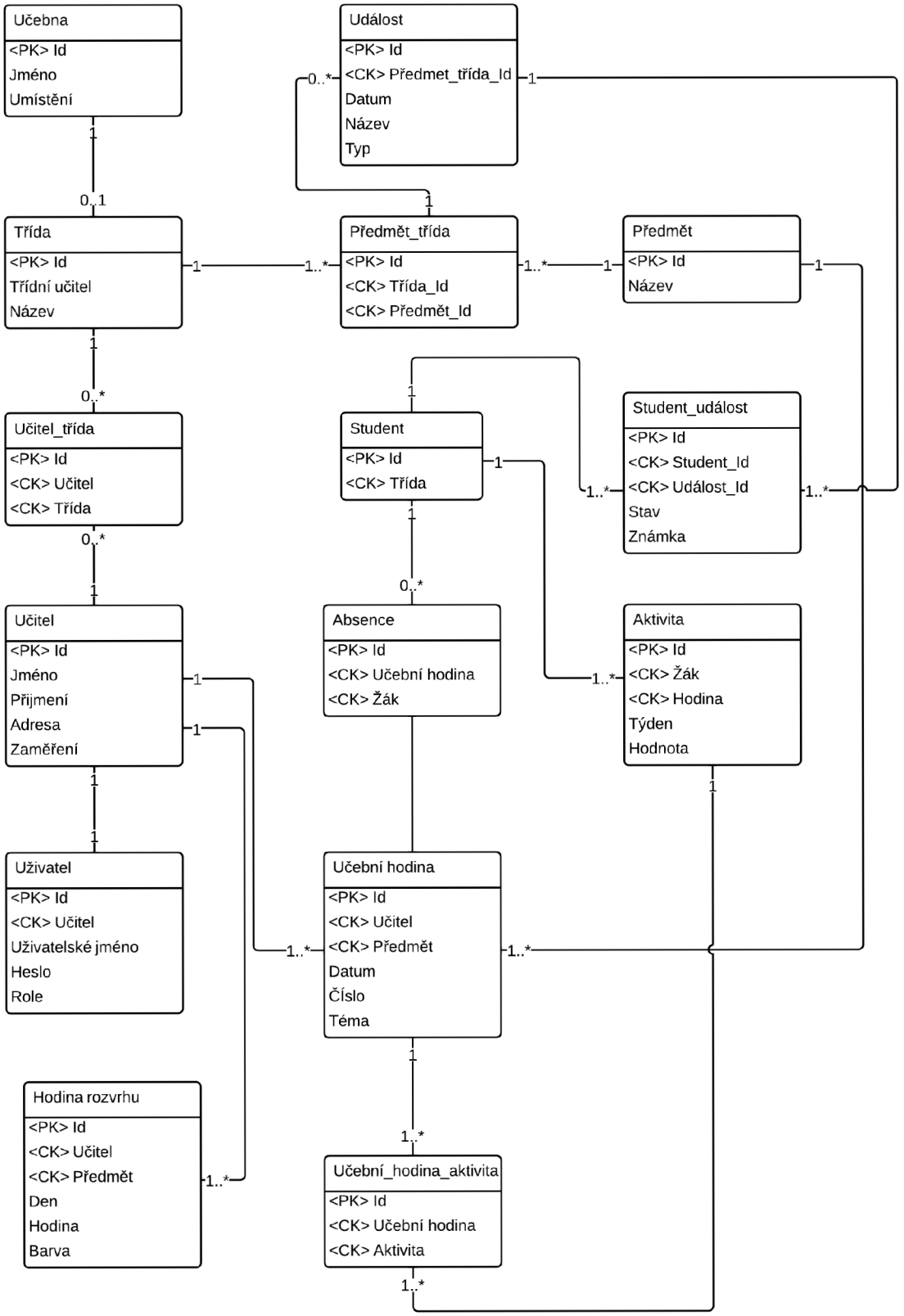
- Progress Bar
  - Informuje o stavu události (v jaké fázi se nachází).
- Navigation Bar
  - Zobrazuje název (titulek) aktuální obrazovky. Obsahuje tlačítko zpět pro návrat na minulou obrazovku a lze do něj přidávat další tlačítka.
- Status Bar
  - Obsahuje základní systémové informace jako stav baterie, úroveň signálu, čas atd.
- Search Bar
  - Vestavěný panel pro vyhledávání.

3.4 Návrh datového modelu

Datový model aplikace se vytváří pomocí nástroje *Core Data*. Jednotlivé entity v modelu odpovídají skutečným fyzickým entitám v modelovaném systému.

Core Data nově nabízí integraci komunikace datové struktury se sdíleným úložištěm dat *iCloud*. Díky tomu mohou uživatelé aktualizovat sdílená data pouze jedenkrát a nemusí je dál rozšiřovat jednotlivě mezi ostatní uživatele. Navíc jsou data zálohována a lze k nim přistupovat z více zařízení.

Na obrázku 3.3 je zobrazen model datové struktury aplikace pomocí *ERD*. Model byl vytvořen pomocí nástroje *Lucidchart*.



Obrázek 3.3: *ERD* zobrazující datový model navržené aplikace.

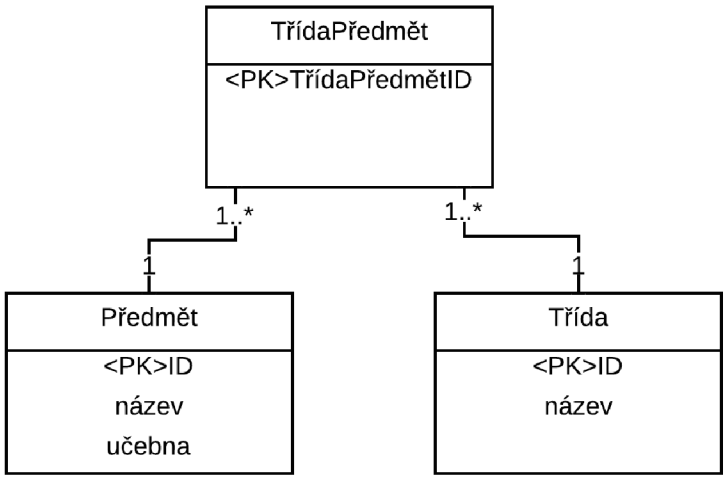
Kapitola 4

Implementace

4.1 Převod ERD do datového modelu Core Data

Při tvorbě *iOS* aplikací je jako datové úložiště často využíván datový model *Core Data*. Jeho základem je grafický a nebo také textový model. Na základě tohoto modelu je vytvořena datová struktura, do níž jsou ukládána a následně čerpána data.

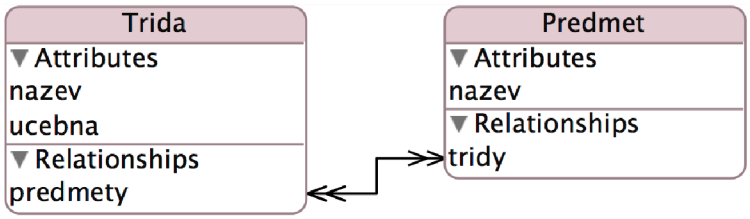
Na obrázku 4.1 je zobrazen *ERD* představující dvě entity (*Předmět* a *Třída*). Jeden předmět je vyučován ve více třídách, např. biologie je vyučována v třídě 1.A a zároveň v třídě 3.A. Zároveň v jedné třídě může být vyučováno více předmětů. Ze specifikace vztahu dvou entit vyplývá, že tyto entity jsou ve vztahu M:N. Proto je nutné zavést třetí tabulku *TřídaPředmět*, která obsahuje jedtnotlivé dvojice typu předmět-třída a jednoznačně určuje, které předměty jsou učeny v jednotlivých třídách. Struktura využívající pomocnou tabulku pro zaznamenání vztahů se běžně využívá např. v *MySQL* databazích.



Obrázek 4.1: Příklad řešení vztahu M:N mezi dvěma entitami v návrhu datového modelu pomocí *ERD*.

Při tvorbě datového modelu v *Core Data* není tabulka vztahů (*TřídaPředmět*) využívána. Z navrženého diagramu entit je tedy nutné takovéto tabulky odstranit a nahradit vztahem. Na obrázku 4.2 lze vidět dvě entity navzájem propojeny vztahem. Vztahy existují pouze dva, „k jednomu"(*to one*) a „k více"(*to many*). Vztah „k jednomu"je značen

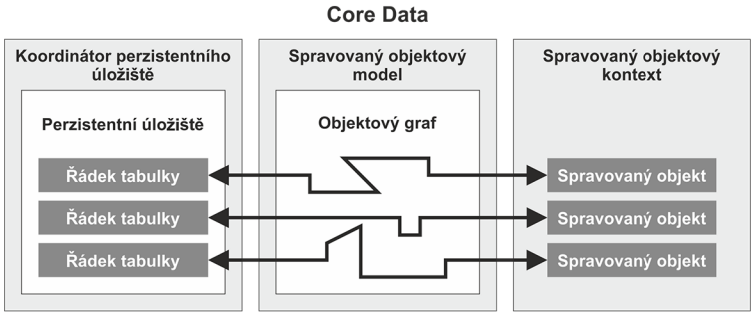
jednoduchou šipkou „k více"poté dvojitou. Dalším rozdílem je absence atributu primárního klíče. Jednoznačný identifikátor je automaticky vytvářen při tvorbě nového objektu a odpadá nutnost přidávání primárního klíče jako samostatného atributu.



Obrázek 4.2: Příklad řešení vztahu M:N mezi dvěma entitami v návrhu datového modelu pomocí Core Data

4.2 Správa dat

Na obrázku 4.3 jsou vyobrazeny části, ze kterých se *Core Data* skládají. Všechny části jsou vzájemně mapovány. Vývojář přijde do styku s každou z nich a je nutné jim náležitě rozumnět.



Obrázek 4.3: Zobrazení mapování dat z volatilní paměti do perzistentní přes datový model.

Koordinátor perzistentního úložiště Obsahuje perzistentní úložiště, kam jsou ukládána všechna data. Jedná se o tabulky většinou ve formě *SQLite* databáze. Lze využívat i např. *XML* úložiště, to je ale díky své atomičnosti v mnoha případech nevhodné. Atomičností úložiště je myšlen způsob zápisu dat. I v případech, kdy je zapisované menší množství dat, je ukládán a přepisován celý *XML* soubor. To je velmi nepraktické, zejména pokud je aplikace integrována s *iCloud* a nebo jiným vzdáleným úložištěm.

Spravovaný objektový model Leží mezí perzistentním úložištěm a nestálým objektovým kontextem. Jedná se o model struktury dat, který lze zobrazovat i v grafické podobě. V rámci schématu jsou vytvářeny entity. Jedna entita odpovídá právě jedné tabulce v perzistentním úložišti. Podrobnější popis objektového modelu se nachází v podkapitole 4.1.

Spravovaný objektový kontext Je vytvářen ve vysokorychlostní volatilní paměti a obsahuje všechny momentálně spravované objekty. Pevné disky jsou mnohem pomalejší než volatilní pamět (*RAM*), proto je přímá práce s daty prováděna zde. Do kontextu jsou

přidávány nově vytvořené objekty a objekty získané na základě dotazů do databáze. Spravovaný objekt odpovídá jednomu řádku tabulky v perzistentním úložišti. Práce s objektem je zmíněna v podkapitole 4.4.

4.3 Rozhraní pro přístup k datům

Část rozhraní pro správu dat byla čerpána z knihy [7]. Části kódů byly převzaty. Práce s daty není zaštítěna pouhým vytvořením datového modelu, ale také několika funkcemi. Tyto funkce jsou obsaženy ve třídě `CoreDataHelper`. Jedná se o funkce pro vytvoření souboru s datovým úložištěm, získání cesty k souboru, vytvoření a inicializace výše zmíněných komponent (spravovaný objektový kontext, spravovaný objektový model, koordinátor perzistentního úložiště a perzistentní úložiště), získání a ukládání spravovaného objektového kontextu. Jako ukázka je v příloze A.1 uvedena funkce pro získání adresáře, kde je umístěno perzistentní úložiště.

Všechny funkce jsou přehledně popsány v knize [7]. Po vytvoření třídy pro přístup k perzistentním datům a vytvoření kontextu je samotná práce s objekty jednoduchá. Tato problematika je popsána v následujicí kapitole.

4.4 Základní práce s objekty

Na základě vytvořeného modelového schématu lze vygenerovat třídy, které jsou dále využitelné v aplikaci. Např. z entity *Predmet* Xcode vygeneruje dva soubory `predmet.m` a `predmet.h`. Obsah souboru `predmet.h` je uveden v příloze A.2:

Třída obsahuje ekvivalentní atributy, jaké byly uvedeny v datovém modelu. V tomto případě se jedná o název. Dalším atributem je sada třídy (`NSSet *tridy`). Sada je velmi podobna klasickému poli, až na několik malých rozdílů. M:N vztah mezi entitami je tedy vytvořen pomocí dvou sad. Do sady `tridy` jsou přidávány instance třídy `trida`, kde je předmět vyučován. Naopak v instancích třídy `trida` je sada s názvem `predmety`, do které jsou přidávány předměty (instance třídy `predmet`) vyučované ve třídě. Pod definicí tříd se nacházejí prototypy metod, které slouží k přidávání a odebírání objektů třída. Základní práce s objekty je velmi jednoduchá a intuitivní. Ukázka práce s objekty je uvedena v příloze A.3

4.5 Implementace základních komponent

Pro implementaci iOS aplikací existuje několik základních komponent z pohledu oken aplikace. V aplikaci byly využívány zejména komponenty *ViewController*, *TabBarController* a *TableViewController*. Možné využití jednotlivých prvků je popsáno v kapitole 3.3.2.

Pokud je v návrhu uživatelského rozhraní přidána nějaká z výše zmíněných komponent, pravděpodobně bude nutné komponentě vytvořit vlastní kontrolér, který bude řídit její chování. Kontroléry dědí své chování z téměř stejnojmenných tříd. Například komponenta *ViewController* dědí z třídy `UIViewController`. Vytvořený kontrolér dědí metody, které jsou využívány pro realizaci navrženého chování. Některé metody jsou povinné a musí být správně použity pro chod komponenty. Jiné jsou pouze doplňkové. Nově vytvořenou třídu lze využít u více komponent stejného druhu. Tak je tomu např. u většiny komponent typu *TableViewController*, které dědí z třídy `CoreDataTVC`. Práce s jednotlivými komponentami a možnosti využití jsou přehledně popsány v knize [5].

4.5.1 ViewController

V kontroléru základní obrazovky jsou v první řadě využívány metody `-(void)viewDidLoad` a `-(void)viewWillAppear:(BOOL)animated`. Zde jsou často volány další metody, které například inicializují kontrolér a nebo komponenty obsažené v něm. Na opačném konci se náchází metoda `-(void)viewDidAppear:(BOOL)animated`, v které lze naopak odstraňovat již nepotřebné objekty a nebo jiné komponenty. Metody jsou volány automaticky při zobrazení či zmizení obrazovky. To zaručuje pravidelné provádění kódu uvnitř zmíněných metod.

Jelikož *ViewController* často obsahuje komponentu textové pole (*UITextField*), je právě zde využíváno metod spojených s textovými poli. Jedná se například o metodu `-(void)textFieldDidBeginEditing:(UITextField *)textField` nebo `-(void)textFieldDidEndEditing:(UITextField *)textField`. Jakmile se libovolné textové pole stane či přestane být aktivní, lze jej v těchto metodách detekovat a libovolně s ním pracovat. Využití metod je podmíněno použitím protokolu `<UITextFieldDelegate>`, který je nutné zahrnout v hlavičkovém souboru daného *ViewController*.

4.5.2 TableViewController

Skládá se z jednotlivých buněk, které mají stejnou šablonu. Pro vytvoření šablon buněk bylo využíváno prostředí *Storyboard* a kontroléru dědícího z třídy `UITableViewCell`. Třídy všech použitých buněk se nácházejí ve složce `Cell Controllers` . V rámci kontroléru je nutné volat metody, které definují obsah. Jedná se o následujicí metody:

- `-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView`
  - Definuje počet sekcí v kontroléru. Počet není omezen, pouze musí být větší než jedna. V aplikaci nebylo sekcí využíváno.
- `- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section`
  - Definuje počet řádků v jedné sekci.
- `- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath`
  - Pro každý řádek je metoda jedenkrát volána. V každém volání je vytvořena jedna buňka, která je poté vykreslena na základě její definice. Pokud buňka obsahuje více komponent, jsou incializovány v této metodě.

Stejně jako u ostatních kontrolérů lze využívat metody volané po načtení a zmizení obrazovky. Další v práci využité metody jsou:

- `(void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath`
  - Metoda je volána při stisku libovolné buňky. Proměnná `indexPath` udává číslo řádku, takže lze jednoduše zjistit, o kterou buňku se jedná. V aplikaci bylo využíváno této metody pro hromadné označování buněk a následné přidavání nebo mazání. Dalším využitím byl přechod na jinou obrazovku na základě indexu buňky.

- `-(void)tableView:(UITableView *)tableView commitEditingStyle:(UITableViewCellEditingStyle) editingStyleforRowAtIndexPath:(NSIndexPath *)indexPath`
  - Umožňuje rozbalení editační nabídky, pomocí které se mažou jednotlivé řádky z tabulky. Tento způsob jednotlivého mazání byl využit i v aplikaci.

4.5.3 TabBarController

Skládá se z libovolného počtu komponent typu *View Controller*. *TabBarController* byl využíván například pro zobrazení karty žáka nebo celkového přehledu žáka v rámci předmětu.

NavigationBar, který je umístěn v horní části obrazovky a informuje o aktualně zobrazovaném okně je v rámci *TabBarController* společný pro všechny komponenty typu *ViewController*. Pokud mají být titulky jednotlivých komponent *ViewController* měněny. Je nutné získat index aktivního *ViewController* a podle indexu nastavit titulek. Nastavení titulku provádí metoda uvedena v příloze A.4.

Další metoda nastavuje aktivní obrazovku na základě atributu *TabBarController*. Při přístupu z jiné obrazovky není vždy požadováno zobrazení úvodního *ViewController*. Proto je před přechodem nastaven atribut `activeView` a na základě jeho hodnoty je aktivována potřebná obrazovka. Nastavení obstarává metoda uvedena v příloze A.5.

4.6 Ošetření vstupu

Validace vstupních parametrů byla prováděna ve dvou fázích. První fáze byla zvolení klávesnice, která se uživateli zobrazí při započetí editace pole. Pokud se jedná pouze o číselný vstup, zobrazí se uživateli klávesnice, která obsahuje pouze čísla. Není nutné dále kontrolovat, jaké hodnoty uživatel zadal.

Druhá fáze byla kontrola zadání povinných údajů. Povinné údaje jsou takové, ze kterých je následovně vytvářen jedinečný identifikátor. Identifikátor by nebyl potřebný, jelikož každému nově vytvořenému objektu je přiřazeno jedinečné *ID*. Byl vytvořen z důvodů duplikace stejných objektů. Například pokud budou existovat dva administrátoři školy a nebo si škola zvolí, že administrátorem má být každý učitel, je možné, že dva administrátoři vloží třídu se stejným jménem. Pokud k této situaci dojde, kontrola identifikátorů to zjistí a smaže objekt, který byl vytvořen dříve. Jinými slovy, posledně vložený objekt ponechá. K těmto situacím by nemělo ve škole docházet, protože administrátor by měl být jen jeden a pokud je jich více, měli by se domluvit na změnách prováděných v administraci školy. I proto je použita pouze jednoduchá logika ponechání aktuálnějšího objektu.

Kontrola povinných hodnot probíhá pomocí podmínek. Pokud je nějaké povinné pole prázdné, vytvoří se instance třídy `Alerts` a následně je zavolána metoda `-(void)showAlert:(int)error`. Metoda přijímá jediný argument a tím je konstanta. Všechny konstanty jsou pomocí výčtového typu uvedeny v souboru `Errors.h`. Ná základě tvaru konstanty je poté zobrazena chybová hláška. Ukázka v příloze A.6 zobrazuje jednu ze dvou chybových hlášek.

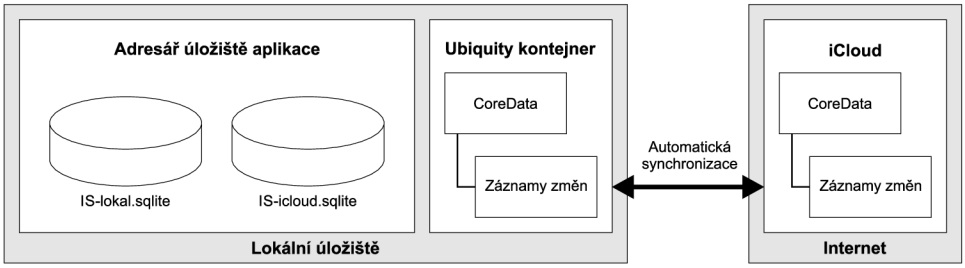
4.7 Integrace iCloud

Integrace vzdáleného úložiště v iOS aplikaci lze provádět několika způsoby. Lze ji provést pomocí uložišť *Dropbox* a *iCloud* nebo hostingových serverů, které jsou pro mobilní aplikace určeny. Nevýhodou integrace pomocí *iCloudu* je fakt, že pokud chceme používat apli-

kaci s internetovým připojením, je nutné být přihlášen k *iCloudu* účtu, kde je perzistentní úložiště umístěno. To může být menší problém pro uživatele, kteří využívají svůj vlastní *iCloud* pro sdílení poznámek, fotek atd. Na druhou stranu jim nic nebrání se k *iCloudu* připojit pouze když chtějí odeslat nebo přijmout nová data.

Pro synchronizaci dat mezi lokálním úložištěm a úložištěm na serveru je nutno doplnit správu dat o několik metod. Tyto metody byly částečně přebrány z knihy [7]. Jedna z metod je kontrola připojení uživatele k *iCloud* účtu. Pokud je uživatel připojen, další metoda sleduje změny na obou úložištích. Jakmile dojde ke změně na jednom z nich, dochází k automatické synchronizaci.

Na obrázku 4.4 je zobrazen způsob uložení dat. Na koncovém zařízení v lokálním adresáři se nacházejí dvě úložiště. Pokud je v nastavení aplikace vypnuta funkce *iCloud*, probíhá ukládání dat do *IS-local.sqlite*. Při zapnutí služby *iCloud* je uživatel dotázán zda-li chce svá data z lokálního úložiště sjednotit s daty z *iCloud*. V lokálním prostoru se nachází také speciální souborový systém, tzv. *Ubiquity* kontejner, který zaznamenává změny a zprostředkovává synchronizaci mezi lokálním a sdíleným *iCloud* úložištěm.



Obrázek 4.4: Struktura uložení a synchronizace dat mezi fyzickým zařízením a vzdáleným úložištěm.

Kapitola 5

Testování

5.1 iSimulator a iPhone

Aplikace byla v průběhu celého vývoje testována na vestavěném simulátoru v prostředí *Xcode*. Testování probíhalo způsobem testování jednotlivých částí aplikace. Vždy byla implementována část aplikace, u které byla následně otestována správná funkčnost podle návrhu. Správná funkčnost implementované části byla zpravidla ověřována přímo v grafickém uživatelském rozhraní aplikace. Například byl implementován oddíl přidávání učitelů do aplikace. Testování proběhlo pomocí spuštění aplikace a následného vyzkoušení vložení nového učitele atd.

Po otestování jednotlivých částí se přecházelo na testování aplikace jako celku, např. její provázanosti. Pokud byl přidán nový učitel, měl být přidán také do nabídky potenciálních třídních učitelů, kteří se zobrazovali u jednotlivých tříd při volbě třídního učitele atd.

Pokud testování neprobíhalo úspešně, bylo nutné využít ladících prostředků pro nalezení chyby. Pro tento účel lze v prostředí *Xcode* využít konzoli, která zobrazuje všechny hodnoty, které uživatel vytisknul. V aplikaci bylo přidáno makro pro zobrazování názvu všech volaných metod. Názvy jsou vypisovány do konzole a tento výpis lze jednoduchým způsobem vypnout pomocí nastavení makra. Tento způsob hledání chyby je využitelný v mnoha příkladech. Překladač totiž ne ve všech případech dokáže kvalitně detekovat chybu. V mnoha případech pádu aplikace pouze skončí v hlavní funkci `main` s mnohdy nejasným chybovým hlášením. Při využití makra lze alespoň detekovat metodu, ve které došlo k chybě. Pro detekci konkrétního řádku lze využít ladicí nástroj (*debugger*), pomocí něhož lze krokovat kód řádek po řádku a detekovat konkrétní instrukci, která pád programu způsobuje. Zároveň lze při ladění sledovat hodnoty proměnných a objektů.

Jednotlivé části aplikace, které byly otestovány na *iSimulátoru* byly poté odzkoušeny na fyzickém zařízení. Konkrétně se jednalo o model *iPhone 4*. Na tomto zařízení byly prováděny obdobné testy funkčnosti jako na simulátoru s tím rozdílem, že aplikace již byla otestována a proto nebylo nutné podrobně testovat každou funkci. Jednalo se spíše o plynulost chodu aplikace. Aplikace byla na mobilním zařízení podrobně otestována po kompletním dokončení vývoje. Nevýhodou testování aplikace na fyzickém zařízení je absence možnosti využít testovací nástroje, které jsou zmíněny v kapitole 5.4.

5.2 iCloud

V nastavení *iSimulator* stejně jako v nastavení fyzického zařízení se lze připojit k *iCloud* účtu. Testování probíhalo mezi zařízením *iPhone 4* a *iSimulator* v prostředí *Macbook Pro*.

Testy spočívaly v úpravách dat na jednotlivých zařízení a sledování následné synchronizace na druhém zařízeních. V *iSimulator* byla například přidávána, mazána či editována data. Poté se sledovalo, kdy se změny projeví na mobilním zařízení. Toto testování probíhalo také v opačném pořadí tzn. data byla měněna pomocí *iPhonu* a sledování změn probíhalo na *iSimulatoru*.

Rychlost aktualizace změněných dat na druhém zařízení byla poměrně proměnlivá. Ale pohybovala se v rozmezí několika desítek sekund.

5.3 SQLite databáze

Pro testování korektní práce s daty lze využít grafické rozhraní telefonu a nebo simulátoru. Všechna data objektů ovšem nejsou uživateli viditelná. Proto je nutné zvolit jiný postup. Otestovaná data lze vypsat jednoduchým způsobem pomocí volání metody `NSLog` se správnými parametry těsně před editací objektu. Další možností je využití ladícího nástroje, který umožňuje získání hodnoty libovolné proměnné či objektu v konkretním místě kódu. Nevýhodou prvního způsobu je fakt, že výpis hodnoty objektu musí být vkládán na konkrétní pozice v kódu a následně opět odstraňován. Druhý způsob je na tom podobně v tom smyslu, že pokud je ladicí nástroj využíván, tak jeho využití je v rámci určité části aplikace, např. v jedné metodě.

Jednodušší možností je využití ladícího nástroje pro *SQL* databáze. V nastavení schématu aplikace lze přidat několik argumentů, které se projeví při spuštění. Pokud je vložen argument `-com.apple.CoreData.SQLDebug 1` v terminálu se začnou vypisovát hlášení informující o všech obdržených datech. Z jaké tabulky byly data obdržena, počet dat a jak dlouho trvalo přijetí dat. Pomocí těchto výpisů byla testována rychlost aplikace při získávání dat.

Argument `-com.apple.CoreData.SQLDebug` lze kombinovat s různými parametry. V aplikaci bylo využíváno také argumentu `-com.apple.CoreData.SQLDebug 3`. Pomocí argumentu lze kontrolovat vložení všech objektů do kontextu. Jakmile je vložen nový objekt, do konzole je vypsáno hlášení informujicí o vložení a také o typu objektu.

Kontrolu dat lze provést také procházením databáze. Stačí vyhledat *SQL* soubor s daty a otevřít jej v programu k tomu určeném. Při ladění aplikace byl využíván program *SQL database browser*, ve kterém je možné přehledným způsobem zkontrolovat data. Program nabízí i možnost editace.

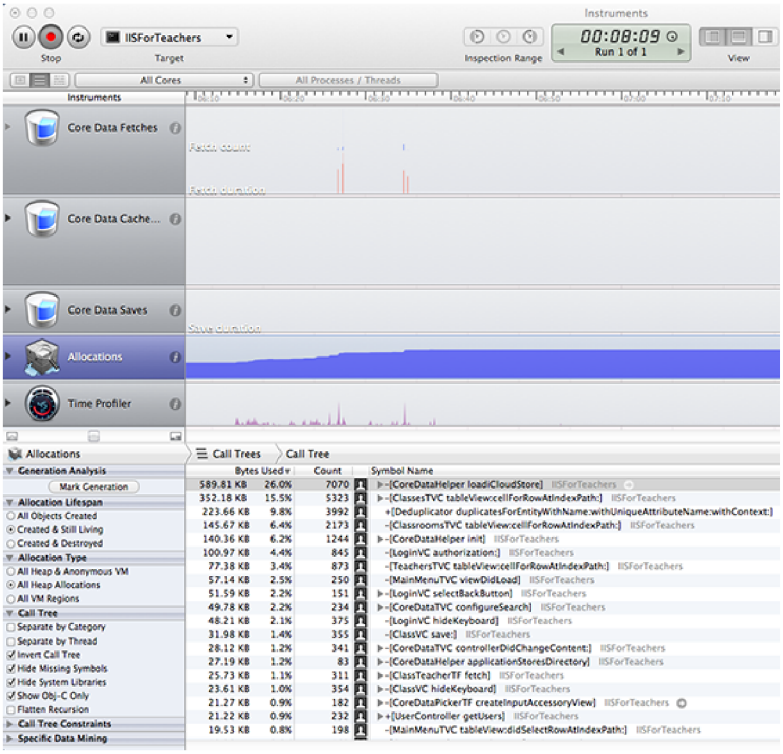
5.4 Nástroje v prostředí Xcode

V *Xcode* exituje několik nástrojů pro otestování chodu aplikace. Všechny jsou umístěny pod položkou *Profile*, kterou lze vybrat při překladu programu. Pro testování aplikace bylo využíváno třech nástrojů *Automation*, *Allocation* a *Core Data*. Názvy jsou uváděny v anglickém jazyce, jelikož český překlad aplikace neexistuje. Práce s jednotlivými nástroji je automaticky nahrávána. Pokud je tedy objevena nějaká chyba, lze záznam opakovaně přehrávat a zjiš'tovat, proč k chybě dochází.

Allocation Nástroj zobrazuje všechny alokované objekty, proměnné atd. Na časové ose je graficky znázorněna celková alokovaná paměť. Nástroj nabízí přehledy jako celková alokovaná paměť, momentálně využívaná paměť atd. Na obrázku 5.1 je zobrazen detail všech metod zodpovědných za alokaci paměti a množství paměti, která byla pro tuto metodu alokována. Nástrojem byla testována paměťová náročnost jednotlivých metod a kontrolováno, zda-li alokovaná paměť nepřesahuje rozumné množství.

Core data Dalším nástrojem jsou *Core data*, které se skladají ze tří částí. Každá část opět na časové ose zobrazuje detekované akce a jejich velikost. *Core Data Saves* zobrazuje všechny data, která byla uložena do perzistentního úložiště. *Core Data Cache Missings* zobrazuje pokusy o přístup k datům, které měly být uloženy ve volatilní paměti, ale už se zde nenachází. Tímto způsobem bylo testováno, zda-li nedochází k předčasnému mazání objektů z kontextu. *Core Data Fetches* zobrazuje jakákoliv obdržená data. Nástroji pro obdržená a uložená data bylo testováno, zda jsou data ukladáná a příjmána v očekávaných momentech.

Automation Nástroj *Automation* slouží k automatickému testování aplikace. K testování slouží skripty napsané v jazyce *Javascript*. Skripty lze napsat buďto ručně nebo využít nahrávání. Pokud je využito nahrávání, je nutné spustit aplikaci a provést potřebné operace. Na základě provedených operací je automaticky vygenerován skript. Tento skript lze uložit a opakovaně používat. Po spuštění skriptu jsou automaticky provedeny všechny operace, které jsou ve skriptu uvedeny. Takto byla testována stabilita aplikace. Opakovaným prováděním jednotlivých skriptů.



Obrázek 5.1: Zobrazení testovacích nástrojů *Core Data*, *Allocation* a *Time Profiler* pro testovaní práce s pamětí a objekty.

5.5 Testování mezi učiteli

V rámci testování grafického uživatelského rozhraní byla aplikace předvedena patnácti učitelům. Někteří z nich aplikaci přímo vyzkoušeli, ostatním byla předvedena na informačním videu. Ve videu byla zobrazena a okomentována celá aplikace. Po shlédnutí, respektive vyzkoušení aplikace, byl učitelům předložen dotazník s prosbou o vyplnění. Dotazník je k náhladnutí v příloze B.

Dotazník vyplnilo celkem 15 učitelů. Celkově 47% učitelů vyučovalo na gymnáziu, 27% na 1.stupni základní školy, 13% na 2. stupni základní školy a 13% na jiné škole. Jejich zaměření bylo spíše přirodovědné (90%). Vyučovali předměty zejména matematika, fyzika a dále biologii, chemie, zeměpis případně všechny předměty vyučované na 1. stupni základní školy mimo jazyků.

Pro zápis svých poznámek, známek atd. využívalo 87% učitelů nějaký sešit, zápisník, který nikdo z nich za několik let praxe ještě neztratil. Chytré zařízení pro zápis známek používali pouze dva učitelé. Využití aplikace místo současných zápisníků, třídních knih atd. si dokázala představit valná většina učitelů (pouze dva ne). Polovina dotázaných učitelů na nižším stupni základní školy považovala aplikaci za více využitelnou ve vyšších ročnících. Tento fakt komentovali často nutnou osobní spoluprácí s rodiči a nepovažovali zápis známek a absence za tak důležité. Navíc v některých nižších ročnicích se známky neudělují, ale udělují např. jen razítka. Dokonce mnoho z nich souhlasilo s tím, že aplikace zjednodušší jejich práci a nabídne vyšší mobilitu a bezpečnost dat. Někteří učitelé odmítali práci s aplikací kvůli vysoké náročnosti na zaškolení. Jednalo se o učitele vyššího věku (oba v kategorii 50-59 let).

Všichni učitelé by ocenili také webové rozhraní určené pro rodiče pro zobrazování známek, docházky atd. Požadavky na velikost zařízení byly individuální, někomu postačoval telefon, jiný by zase uvítal tablet, další by rád obě zařízení, které by využíval dle situace. Pouze polovina dotázaných byla majiteli chytrého zařízení. Všichni vlastníci na zařízení využívali internet, což je kvůli aplikaci velmi důležité.

Logická stavba aplikace byla ohodnocena v průměru 7.1 body a intutivnost a přehlednost 7.6 body. Oba tyto výsledky považuji za úspěch. Nepřehledné se zdály pouze popisky některých položek menu a to jen malému procentu učitelů. Na aplikaci se líbila zejména jednoduchost, přehlednost a provázanost aplikace. Ze samostatných funkcí to bylo hodnocení aktivity žáka v hodině a také celkové zvýšení přehlednosti informací o žákovi.

Naopak nelíbilo se nebo chyběla možnost zadávat body z písemek, nikoliv pouze známky, možnost pořízení digitálních záznamů jako vyfocení testu žáka nebo vytvoření krátké zvukové nahrávky. Učitelé také hodnotili svoji technickou zdatnost, jelikož i tento fakt se může projevit v hodnocení. Stejně jako jejich věk. Tyto kritéria spolu mohou být úzce spjata. Hodnocení technické zdatnosti bylo průměrně 7.3 bodů a věk 41 let. V závislosti na technické zdatnosti jsem nepozoroval zasádní rozdíly v hodnocení aplikace. Naopak mladší kantoři hodnotili aplikaci více pozitivně než starší.

Kapitola 6

Závěr

V rámci bakalářké práce byla vytvořena aplikace pro učitele primárně určená pro střední a základní školy. Aplikace byla vyvinuta v prostředí *Xcode* v jazyce *Objective-C* a slouží ke zjednodušení práce učitelů.

Hlavním přínosem je sjednocení několika dokumentů do jediné aplikace. Učitel má díky aplikaci detailní přehled o žákovi. Aplikaci může využívat místo svého zápisníku, jelikož zde může zapisovat prospěch žáků, evidovat domácí úkoly, aktivitu v hodinách či vytvářet libovolné poznámky k žákům. V systému se nachází karty žáků, které obsahují obecné informace o žákovi a jeho rodičích. Odpadá tedy nutnost udržování karet žáků v papírové podobě. Poslední dokument, který aplikace zastupuje nebo alespoň z části, je třídní kniha. Učitel může vytvářet nové vyučovací hodiny v rámci předmětu. Vyučovací hodině přiřazovat téma, číslo hodiny a chybějící žaky. Aplikace také disponuje jednoduchou správou omluvenek.

Druhou výhodou mimo pouhého sjednocení dokumentů je fakt zvýšení mobility a bezpečnosti dat. Aplikace je integrována pomocí vzdáleného úložiště *iCloudu*. Při ztrátě zařízení, pomocí kterého byly data do systému zadána, nedochází ke ztrátě dat. Tento fakt zvyšuje bezpečnost uložených dat, jelikož při ztrátě třídní knihy či vlastních poznámek se učitelé dostávali do velmi nepříjemných situací. Důležitá je také mobilita, kdy učitel má data stále při sobě, aniž by momentálně u sebe musel mít třídní knihu rozměrů A4, či vlastní notýsek.

V rámci dalšího vývoje aplikace bylo na základě dotazníků vyplněného učiteli a zhodnocení dalších potřeb učitelů navrženo několik možných rozšíření aplikace. Učitelé určitou dobu musí archivovat všechny písemné práce a testy, proto by bylo dobré tuto archivaci také převést do digitalní podoby a mít možnost u jednotlivých testů pořídit fotografii. V rámci zrychlení práce s aplikací by mohly být poznámky doplněny o možnost vytvoření poznámky pomocí zvukového záznamu. Rodiče by také rádi evidovali výkony svých dětí. Pro tento účel by se dalo využít webové rozhraní aplikace, kde by se výsledky zobrazovaly. Posledním navrženým vylepšením je možnost zadávat výsledky z testů nejen pomocí známek, ale i bodů, či procent.

Myslím si, že práce splnila svůj cíl a dokáže učiteli zjednodušit jeho práci a zároveň zvýšit bezpečnost informací. I přes kladná hodnocení učitelů bude podle mého názoru ještě nějaký čas trvat, než se začne podobný systém nasazovat do reálného provozu ve školách. Tento fakt je způsoben i tím, že tato aplikace nebo jí podobná má v dnešní době stále jisté procento odpůrců, kteří nechtějí upouštět od zažitých rutin.

Literatura

[1] APPLE, I.: Concepts in Objective-C Programming. https://developer.apple.com/library/ios/documentation/General/Conceptual/CocoaEncyclopedia/CocoaEncyclopedia.pdf [online], 2012 [cit. 2013-12-05].

[2] Hruška, T.; Máčel, M.: *Pokročilé informační systémy : Analýza, návrh, implementace informačního systému, Studijní opora*. Fakulta informačních technologií v Brně VUT, Únor 2012.

[3] Jiří, V.: *IPhone: vývoj aplikací*. Praha: Grada, první vydání, 2012, ISBN 978-80-247-4457-5, 192 s.

[4] Kochan, S. G.: *Objective-C 2.0 : Výukový kurz programování pro Mac OS X a iPhone*. Brno: COMPUTER PRESS, 2010, ISBN 978-80-251-2654-7, 552 s.

[5] Nahavandipoor, V.: *iOS 7 Programming Cookbook*. Sebastopol: O'Reilly Media, 2013, ISBN 978-1449372422, 1032 s.

[6] RITCHIE, R.: History of the iPhone: From revolution to what comes next [online]. http://www.imore.com/history-iphone, 11 2013 [cit. 2013-11-11].

[7] Roadley, T.: *Learning Core Data for iOS: A Hands-On Guide to Building Core Data Applications*. Indiana: Addison-Wesley Professional, Listopad 2013, ISBN 978-0-321-90576-5, 480 s.

Příloha A

Ukázky kódu

A.1 Získání adresáře s perzistentním úložištěm

```
-(NSURL *)applicationStoresDirectory{
  /*vytvoření cesty do adresaře obsahující
    persistentní uložiště získáním cesty
    k aplikaci a pridanim adresare Store */
    NSURL *storesDirectory = [[NSURL fileURLWithPath:[self
            applicationDocumentsDirectory]]URLByAppendingPathComponent:@"
            Stores"];

    NSFileManager *fileManager = [NSFileManager defaultManager];
   // Pokud adresar neexistuje, je vytvoren
    if(![fileManager fileExistsAtPath:[storesDirectory path]]){
        NSError *error = nil;
        if([fileManager createDirectoryAtURL:storesDirectory
                withIntermediateDirectories:YES attributes:nil error:&error]){
            if(debug == 1){
                NSLog(@"Successfully created Stores directory");}
        }
        else
            NSLog(@"FAILED to create Stores directory: %@", error);
    }

    return storesDirectory;
}
```

A.2 Obsah souboru predmet.h

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@class Trida;

@interface Predmet : NSManagedObject

@property (nonatomic, retain) NSString nazev;
@property (nonatomic, retain) NSSet *tridy;
@end

@interface Predmet (CoreDataGeneratedAccessors)
```

```
- (void)addTridyObject:(Trida *)value;
- (void)removeTridyObject:(Trida *)value;
- (void)addTridy:(NSSet *)values;
- (void)removeTridy:(NSSet *)values;

@end
```

A.3 Ukázka základní práce s objekty

```
#import "Predmet.h"
#import "Trida.h"
#import  "CoreDataHelper.h"
#import "Appdelegate.h"

// ziskani kontextu spravovaneho modelu
CoreDataHelper *cdh = [(AppDelegate *)[[UIApplication sharedApplication]
        delegate]cdh];

// vlozeni novych objektu do spravovaneho kontextu
Predmet *biologie = [NSEntityDescription insertNewObjectForEntityForName:@"
        Predmet" inManagedObjectContext:cdh.context];
Trida *2.A = [NSEntityDescription insertNewObjectForEntityForName:@"Trida"
        inManagedObjectContext:cdh.context];

// nastaveni nazvu tridy a predmetu
biologie.nazev = @"Biologie";
trida.nazev = @"2.A";

// prirazeni predmetu ke tride
[2.A addPredmetyObject:biologie];

// odebrani predmetu ze tridy
[2.A removePredmetyObject:biologie];

// tisk ID objektu do konzole
NSLog{@"%@", biologie.objectID};

// ulozeni dat do perzistentniho uloziste
[cdh saveContext];
```

A.4 Nastavení titulku u TabBarController

```
-(void)setTitleForView{
    if ([self selectedIndex] == 0) {
        self.title = @"Poznámky";
    }
    else if ([self selectedIndex] == 1) {
        self.title = @"Absence";
    }
    else if ([self selectedIndex] == 2) {
        self.title = @"Domácí úkoly";
    }
    else if ([self selectedIndex] == 3) {
        self.title = @"Testy";
    }
    else if ([self selectedIndex] == 4) {
        self.title = @"Aktivita";
    }
```

```
}
```

A.5 Nastavení aktivního pohledu v TabBarController

```
-(void)initializationOfTBC{
    if (self.activeView == 0)
        self.selectedViewController = [self.viewControllers objectAtIndex:0];
    else if (self.activeView == 1)
        self.selectedViewController = [self.viewControllers objectAtIndex:1];
    else if (self.activeView == 2)
        self.selectedViewController = [self.viewControllers objectAtIndex:2];
    else if (self.activeView == 3)
        self.selectedViewController = [self.viewControllers objectAtIndex:3];
    else if (self.activeView == 4)
        self.selectedViewController = [self.viewControllers objectAtIndex:4];
}
```

A.6 Zpracování chyb

```
#import "Alerts.h"
#import "Errors.h"

@implementation Alerts

-(void)showAlert:(int)error{

    NSString *title = @"";
    NSString *messsage = @"";
    NSString *cancel = @"";
    NSString *otherTitles = @"";

    // vyber chybove hlasky
    switch (error) {
        case LOGIN_ERROR:
            title = @"Chyba přihlášení";
            messsage = @"Špatné uživatelské jméno nebo heslo.";
            cancel = @"OK";
            otherTitles = nil;
            break;

        case CLASS_NAME_MISS:
            title = @"Název třídy nezadán";
            messsage = @"Prosím zadejte název třídy.";
            cancel = @"OK";
            otherTitles = nil;
            break;

        default:
            break;
    }
    // inicializace chybove hlasky
    UIAlertView *alert = [[UIAlertView alloc]initWithTitle:title message:
        messsage delegate:nil cancelButtonTitle:cancel otherButtonTitles:
        otherTitles, nil];
    // zobrazeni chybove hlasky
    [alert show];
```

Příloha B

Dotazník

Vážená paní, vážený pane,
jsem studentem školy Vysoké učení technické v Brně, Faukulta informačních technologií. Dovoluji si Vás požádat o vyplnění tohot dotazníku, který bude anonymně zpracován a získaná data budou chráněna před zneužitím. Zjištěné udajé budou použity pro účely bakalářské práce.

Pokyny k vyplnění dotazníku: zakroužkujte vždy jednu odpověď, pokud nebude uvedeno jinak.

Děkuji za Váš čas a ochotu. Jiří Kratochvíl

1. Na jaké škole vyučujete?
   a) gymnázium b) střední odborná škola c) základní škola 1. stupeň
   d) základní škola 2. stupeň d) jiné

2. Jaké předměty vyučujete?
   a) technické (informatika, deskriptivní geometrie aj.)
   b) společenské (psychologie, ekonomie aj.)
   c) přírodovědné (matematika, biologie, zeměpis, chemie, fyzika aj.)
   d) humanitní (český jazyk, dějepis, základy společenských věd aj.)
   e) jiné

3. Používáté při své práci nějaké mobilní zařízení (tablet, chytrý telefon) pro správu známek, poznámek, aktivity atd. u žáků?
   a) ano b) ne

4. Pokud jste odpověděl/a ano, jaké zařízení?

5. Jaké aplikace využíváte a k čemu?

6. Pokud jste odpověděl/a ne, kam si známky, poznámky apod. zapisujete?

7. Co děláte, pokud se Vám pomůcka (např. zápisník) ztratí?

8. Dokázali byste si představit nahrazení používaných pomůcek (zápisník, třídní kniha, karty žáka) touto nebo podobnou aplikací? Svou odpověď rozviňte.

9. Myslíte, že by používání této aplikace zjednodušilo učitelům práci? (zvažte, že jste plně zaškolení v rámci aplikace)
   a) ano b) spíše ano c) spíše ne d) ne

10. Uvítali byste kromě mobiní aplikace i webové rozhraní? (možnost pro rodiče, aby se příhlásili pomocí webové aplikace do systému a mohli sledovat známky svého dítěte)
    a) ano b) spíše ano c) spíše ne d) ne

11. Dali byste přednost mobilnímu zařízení s větší obrazovkou (např. tablet) nebo by Vám mobilní telefon postačoval?
    a) ano b) spíše ano c) spíše ne d) ne

12. Pokud jste odpověděl/a kladně. Proč?

13. Jste vlastníkem chytrého mobilního zařízení?
    a) ano b) ne

14. Pokud jste odpověděl/a ano, jakého?

15. Využíváté mobilní internet ve Vašem zařízení?
    a) ano b) ne

16. Ohodnoťte logickou stavbu aplikace v rozmezí 1 až 10 bodů (10 nejvyšší).

17. Ohodnoťte přehlednost a intiutivnost aplikace v rozmezí 1 až 10 bodů.

18. Co se Vám zdá nepřehledné/nejasné/neintutivní?

19. Co se Vám v aplikaci líbí?

20. Co se Vám v aplikaci nelíbí?

21. Co Vám v aplikaci chybí?

22. Ohodnoťte Vaši technickou zdatnost v rozmezí 1 až 10 bodů. (jak rychle se dokážete sžít s novým zařízením/aplikací a osvojit si práci s ním)

23. Jaký je Váš věk?
    a) 24 – 30 let b) 30 – 39 let c) 40 – 49 let d) 50 – 59 let e) 60 a více let

Příloha C

Metriky kódu

Celková velikost souborů:	1.5MB
Celkový počet řádků kódu:	23 909
Celkový počet řádků komentáře:	4 744
Celkový počet souborů:	279

Příloha D

Použité zkratky

ERD ..... Entity Relationship Diagrams
MVC ..... Model–View–Controller