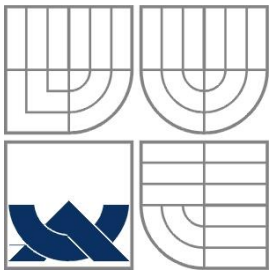
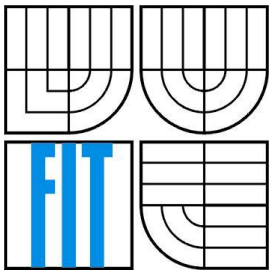


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

INFORMAČNÍ SYSTÉM PRO SPRÁVU NEMOVITOSTÍ

INFORMATION SYSTEM FOR MANAGEMENT OF REAL ESTATES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DAVID FOJTÍK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. VLADIMÍR BARTÍK, Ph.D.

BRNO 2016

Zadání bakalářské práce/18620/2015/xfojti11

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Fojtík David**
Obor: Informační technologie
Téma: **Informační systém pro správu nemovitostí**
Information System for Management of Real Estates
Kategorie: Informační systémy

Pokyny:

1. Seznamte se principy tvorby webových a mobilních aplikací a pořebnými technologiemi.
2. Analyzujte požadavky kladené na informační systém pro správu nemovitostí, který bude umožňovat evidenci vybavení nemovitosti, událostí (např. revize, údržba apod) a dalších investic souvisejících s nemovitostí. Požadavky konzultujte s Ing. Petrem Křenkem z firmy GetONE s.r.o.
3. Navrhněte koncepci informačního systému na základě analyzovaných požadavků, využijte k tomu vhodných modelovacích technik. Systém se bude skládat z webové i mobilní aplikace.
4. Navržený informační systém implementujte jako webovou aplikaci a proveďte transformaci do mobilních zařízení na platformu Android tak, aby byl dále přenositelný i na jiné platformy. Provedte přenos na druhou platformu.
5. Ověřte funkčnost systému na vhodném vzorku dat.
6. Zhodnoťte dosažené výsledky a další možné pokračování v tomto projektu.

Literatura:

- Zakas, N.Z.: JavaScript pro webové vývojáře - Programujeme profesionálně. Computer Press, 2009.
- Naramore, E. et al.: PHP5, MySQL, Apache: Vytváříme webové aplikace, Computer Press, 2009.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bartík Vladimír, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Hlavním cílem této bakalářské práce je navrhnout a implementovat informační systém pro správu nemovitostí. Tento informační systém bude umožňovat pro konkrétní byty či nemovitosti vedení evidence (ve smyslu servisní knížky). Systém eviduje přehledy investic a výdajů, údržby či servis spotřebičů, plánované a prováděné revize, evidence oprav nemovitosti a bytu nebo přidávání vlastních investic. Informační systém je navržen jako webová aplikace za použití HTML5, CSS a AngularJS. Webová aplikace je následně zabalena pro jednotlivé mobilní platformy pomocí Apache Cordova. Nativní vzhled aplikace je dosažen použitím Ionic frameworku. Apache Cordova zajišťuje integraci aplikace s mobilním systémem, díky tomu je možné v aplikaci využívat fotoaparát telefonu pro evidenci oprav či notifikace, které upozorňují na blížící se revize nebo plánované údržby.

Abstract

The aim of the bachelor thesis is to design and implement an information system for the management of real estates. The system will allow keeping records of specific apartments or real estates (in the sense of a service book). The system records the overviews of investments and expenditures, the maintenance and service of appliances, planned and executed inspections, repair records of real estates and apartments or adding your own investments. The information system is designed as a web application using HTML5, CSS and AngularJS. The web application is then packaged for individual mobile platforms using Apache Cordova. The native application view is achieved by the use of Ionic framework. Apache Cordova provides the integration of application with the mobile system, making it possible to use the camera in the phone for repairs or register notifications that warn of the impending revision or scheduled maintenance.

Klíčová slova

Informační systém, správa nemovitostí, mobilní webová aplikace, HTML5, CSS, AngularJS, Apache Cordova, Ionic framework

Keywords

Information systém, management of real estates, mobile web application, HTML5, CSS, AngularJS, Apache Cordova, Ionic framework

Citace

FOJTÍK, David. *Informační systém pro správu nemovitostí*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Bartík Vladimír

Informační systém pro správu nemovitostí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením. Ing. Vladimíra Bartíka, Ph.D. Další informace mi poskytl jednatel firmy GetONE s.r.o. pan Ing. Petr Křenek.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
David Fojtík
18. května 2016

Poděkování

Chtěl bych poděkovat jednateli firmy GetONE s.r.o. panu Ing. Petru Křenkovi, který mi poskytl odborné rady, konzultace požadavků na systém a v neposlední řadě výběr vhodných technologií pro úspěšné implementování aplikace. Dále bych chtěl poděkovat mému vedoucímu panu Ing. Vladimíru Bartíkovi, Ph.D. za jeho ochotu, vstřícné konzultace a poskytnutí užitečných rad při návrhu informačního systému.

Obsah

1 Úvod

V dnešní době se na trhu zatím nevyskytuje aplikace, která by obdobným způsobem jako servisní knížka, umožňovala evidovat a vytvářet přehledy investic, údržby, oprav či jiných výdajů v rámci nemovitosti či konkrétního bytu. Z tohoto důvodu vznikla myšlenka tuto skutečnost realizovat.

Dnes existuje celá řada chytrých telefonů s různými platformami, proto je potřeba zamyslet se nad tím, jaké máme možnosti vývoje aplikací pro různé mobilní platformy. První varianta je nativní aplikace, kde se specializujeme na vývoj pouze pro konkrétní platformu. Druhou variantou je webový vývoj aplikací, který poskytuje přenositelnost na více platforem. Nevýhodou však je absence přístupu k systémovému API. Poslední možností je hybridní mobilní vývoj aplikací, který kombinuje webový a nativní přístup.

Pro vývoj informačního systému byla zvolena poslední možnost. Informační systém je navržen jako webová aplikace využívající programovací jazyky HTML5, CSS a AngularJS. Aplikace je následně zabalena pomocí nástroje Apache Cordova pro konkrétní mobilní platformu. Vzhled nativní aplikace je dosažen díky použití Ionic frameworku.

V informačním systému může uživatel přidávat více nemovitostí. Ke každé nemovitosti pak přidává konkrétní byty. Uživatel si v systému zvolí aktivní nemovitost a byt, u kterých chce zobrazovat či přidávat přehledy investic, revizí, spotřebičů, oprav aj. Veškeré položky může uživatel přidávat, editovat, zobrazovat si u každé položky detailní náhled nebo položky odstraňovat.

Nástroj Apache Cordova zpřístupňuje systémová API telefonu (např. fotoaparát, geolokaci, kalendář atd.) přímo do JavaScriptu. Díky tomu je možné v aplikaci využít fotoaparát telefonu, kdy pořízené fotografie aplikace uchovává u oprav nemovitosti. Dále aplikace využívá notifikace telefonu, které budou uživatele upozorňovat na blížící se revize nemovitosti či údržbu spotřebičů. To nám vylepšuje použitelnost aplikace.

V následujících kapitolách je popsán celý proces tvorby informačního systému, nastudovaná problematika, popis použitých technologií, dále návrh a popis implementace, testování a nakonec v závěru je zmíněno možné pokračování v tomto projektu.

2 Informační systém

Tato kapitola obsahuje popis základních pojmů spojených s informačními systémy. Obsah této kapitoly vychází převážně z [3].

Jestliže mluvíme o informačních systémech, vždy jde v zásadě o zpracování dat. Informacemi se stávají až po interpretaci uživatelem.

2.1 Informace

Informací se chápe údaj o reálném prostředí, jeho stavu a procesech v něm probíhajících. Informace je nehmotná, má význam a popisuje skutečnost vždy v rámci jistého modelu na určitém stupni abstrakce. V oblasti výpočetní techniky se za informaci považuje kvantitativní vyjádření obsahu zprávy. Za jednotku informace se považuje rozhodnutí mezi dvěma alternativami (0, 1) a vyjadřuje se jednotkou zvanou bit. Informace jsou data, která mají sémantiku (význam). Definice sémantiky je však obtížná, většinou je popsána neformálně a nepřesně. Důležitou roli proto hraje správná reprezentace dat takovou formou, aby nedocházelo k různým interpretacím.

2.2 Data

Obecně jsou data jakékoli vyjádření (reprezentace) skutečnosti, schopné přenosu, uchování, interpretace či zpracování. Z hlediska počítačového jsou to pouze hodnoty různých datových typů. Data reprezentují stav informačního systému. Data se ukládají jako posloupnost bajtů, nemají sémantiku, zpracovávají se na základě syntaxe.

2.3 Znalost

Znalost je informace zařazená do souvislostí. Znalosti chápeme jako sekundární (odvozené) informace. Na základě použití informací lze řešit problémy.

2.4 Systém

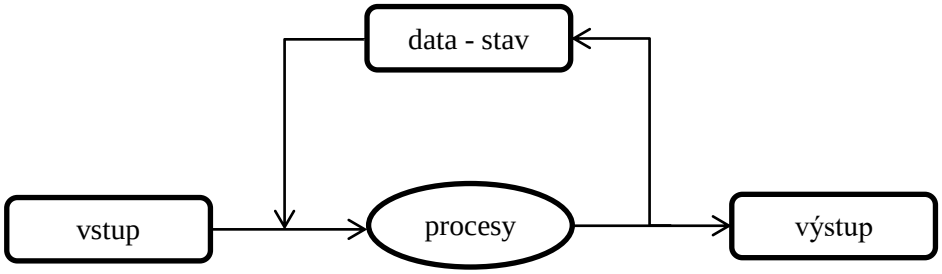
Systém lze chápat jako množinu prvků a vazeb mezi nimi, které jsou definovány na nosiči. Nosičem je množina prvků systému ve vzájemných informačních a procesních vztazích. Prvky nosiče se nazývají zdroje. Podle připojení na okolí dělíme systémy na otevřené (je připojen k okolí tokem zdrojů, má vstupní a výstupní část) a uzavřené (není v interakci se svým okolím).

2.5 Informační systém

V této práci je popsán otevřený informační systém. Informační systém zobrazený na obrázku 2.1 je obecným systémem pracujícím s konceptuálními zdroji. Informační systém se skládá ze vstupní a výstupní části. Ve vstupní části jsou zadána data určená ke zpracování a ve výstupní části jsou zpřístupněna uživatelům pro interpretaci a získání informací.

Mezi vstupem a výstupem probíhá transformace těchto dat. Transformaci dat typicky provádí různé algoritmy. Probíhají zde procesy, proto se musíme zabývat způsobem definice procesů, jejich vzájemnou interakcí, paralelním prováděním apod. Důležitou roli hraje zpětná vazba. Některé procesy mohou být závislé na ostatních procesech, nikoliv na okamžitých vstupech.

Data uchovávají stav systému a procesy realizují transformace často ve formě transakcí.



Obrázek 2.1: Schéma informačního systému

Informační systémy jsou tvořeny pro řízení určitých fyzických systémů a modelují jejich chování. Protože jsou systémy v určité míře abstraktní, nemohou nikdy obsahovat veškeré procesy fyzického systému. Úroveň abstrakce se tedy volí podle zaměření systému.

Nejčastějším prostředím pro trvalé, neboli perzistentní uchování stavů informačních systémů je relační databáze. Relační databáze může být centralizovaná nebo distribuovaná. Centralizovaná databáze uchovává stav systému na jednom místě, naopak u distribuované probíhá získávání z různých zdrojů.

3 Tvorba aplikací pro mobilní zařízení

Pokud chceme vyvíjet aplikace na mobilní zařízení, musíme se nejen zamyslet nad tím, pro kterou platformu chceme aplikaci vytvořit, ale také nad tím, jaké máme možnosti pro tvorbu této aplikace.

Pro mobilní zařízení je možné vytvářet aplikace nativní, webové nebo hybridní. Pro každý typ existuje celá řada vývojových prostředí, programovacích jazyků, a je jen na nás, jak se rozhodneme. V posledních letech se však začínají objevovat nástroje pro multiplatformní vývoj mobilních aplikací, které umožňují přenos kódu mezi platformami a tím pokrýt co největší část trhu.

Popsané typy mobilních aplikací vycházejí z textů [4] a [5].

3.1 Nativní mobilní aplikace

Nativní aplikace je specifikována pouze pro konkrétní platformu. Tato platforma může být například Android, iOS nebo Windows Phone. Aplikace pro konkrétní platformu se vyvíjí v primárním programovacím jazyce daného systému. iOS aplikace jsou programovány v Objective-C, aplikace pro Android v jazyce Java a pro Windows Phone v jazyce C#. Rozdíly jsou nejen v programovacích jazycích, ale i celková struktura projektu, přístup k systémovému API[1] nebo principy designu a uživatelského rozhraní dané platformy.

Další typickou charakteristikou nativních aplikací je jejich distribuce. Aplikace jsou distribuovány nejčastěji pomocí tržišť aplikací zaměřených na jednotlivé platformy. Každá platforma má svůj vlastní obchod, ze kterého může uživatel stahovat a instalovat aplikace. Aplikace pro telefony se systémem iOS jsou k dispozici přes AppStore, na telefony se systémem Android v Google Play a pro Windows Phone přes Windows Phone Store. Přítomnost nativních aplikací v těchto obchodech je zcela běžné, uživatelé jsou na to navyklí.

Výhodou nativních aplikací je především výkon oproti webovému nebo hybridnímu vývoji. Je to z důvodu absence JavaScriptového běhového prostředí, které běží skrz jádro webového prohlížeče. Nativní aplikace totiž běží přímo v systému telefonu.

Pokud bychom chtěli uskutečnit přenos již hotové aplikace na další platformu, znamenalo by to začít úplně nový samostatný vývoj. Z tohoto důvodu je tvorba nativních aplikací nejnákladnějším řešením (ať už časovým či finančním). Výhodou však zůstává fakt, že aplikace budou vždy pro konkrétní platformu lepší, ať už z hlediska rychlosti či vzhledu.

Existují však sady nástrojů, které umožňují vývoj nativních aplikací v jednom programovacím jazyce, které mohou být spuštěny na více platformách. Zástupcem tohoto přístupu je například

[1] Application Programming Interface (rozhraní pro programování aplikací) - sbírka procedur, funkcí, tříd či protokolů nějaké knihovny, které může programátor využívat

Xamarin, ve kterém probíhá vývoj aplikací v jazyce C# a .NET. Obsahuje základní knihovny pro operační systém Android a iOS. Jelikož je vývoj v jazyce C# a .NET, je tento kód kompatibilní i s platformami Windows Phone.

Za výhodu nástroje Xamarin je možné považovat možnost implementace uživatelského rozhraní pro každou platformu zvlášť. Výsledný vzhled a ovládací prvky jsou stejné, jako kdyby probíhal vývoj pro konkrétní platformu zvlášť. [6]

3.2 Webové mobilní aplikace

Webový vývoj mobilních aplikací umožňuje přenositelnost mezi platformami. Obdobně jako webová stránka je aplikace naprogramována pouze jednou a přizpůsobena tak, aby fungovala na požadovaných platformách.

Programovacími jazyky pro tento přístup vývoje jsou především jazyky HTML5, CSS a JavaScrtipt. Uživatel k těmto aplikacím přistupuje a využívá je pomocí svého webového prohlížeče. Webové aplikace jsou aplikace poskytované uživatelům z webového serveru, kde je aplikace uložena, prostřednictvím počítačové sítě. Počítačová síť může být vnitropodniková (intranet) či globální síť Internet. [7]

Výhodou webových aplikací je nenáročnost na prostor, protože aplikace běží na vzdáleném serveru, na kterém mohou probíhat i složité výpočty. Umožňuje to také snadné řešení aktualizací aplikace, kdy stačí nahrát novou aktuální verzi na server.

Server je většinou připojen k databázi, se kterou komunikuje. Uživatel komunikuje s webovým serverem prostřednictvím webového prohlížeče protokolem HTTP[2]. Prohlížeč přijatá data od serveru zformátuje a prezentuje je uživateli. Webovému prohlížeči se také říká tzv. tenký klient. Tenký klient je počítač nebo počítačový program, který je při plnění své funkce závislý na jiném počítači (serveru).

V dnešní době se od webové aplikace požaduje, aby měla svou aplikační logiku a přístup k databázi, kam ukládá data. Proto byla vytvořena třívrstvá architektura, která se při vývoji webových aplikací používá. Jedná se o MVC architekturu.

Webové aplikace mají však nevýhodu. Nemohou přistupovat k systémovému API telefonů, takže není možné využívat informace o poloze, fotoaparátu apod. Další nevýhodou může být možná nekonzistence uživatelského rozhraní v porovnání s nativní aplikací.

[2] Hypertext Transfer Protocol – protokol určený pro výměnu hypertextových dokumentů ve formátu HTML

3.3 Hybridní mobilní aplikace

Posledním možným způsobem je vytvořit hybridní aplikaci. Tento přístup kombinuje oba výše zmíněné přístupy, tedy nativní a webový. Aplikace se skládá z webového rámce, ve kterém se nachází nativní komponenta umožňující zobrazit webovou stránku. Této komponentě se říká WebView. Tato komponenta je buď součástí aplikace, nebo se může načítat vzdáleně.

Vývoj hybridních aplikací probíhá obdobně jako u webových, tedy použití jazyků HTML5, CSS a JavaScript. Liší se však v tom, že tato aplikace je potom ve výsledku zabalena pro konkrétní platformu pomocí nějakého nástroje jako nativní.

Jako každý přístup má i hybridní vývoj své výhody a nevýhody. Mezi první výhodu patři distribuce aplikace v oficiálních obchodech u konkrétních platforem, kdy je uživatel instaluje obdobným způsobem jako běžné nativní aplikace. Oproti webovému vývoji mají hybridní aplikace zásadní výhodu v tom, že zajišťují propojení webové aplikace se systémem telefonu. Díky tomu je již možné využívat systémové API a přistupovat k funkcím přímo z JavaScriptu. Toto API je navíc totožné pro všechny platformy. Jsme jen omezeni množstvím funkcí, které nám na dané platformě umožňuje provádět.

Nevýhodou hybridních aplikací je menší výkon z důvodu běhu aplikace na základě webového jádra. Další nevýhodou je vzhled uživatelského rozhraní oproti nativním aplikacím. Nativního vzhledu však můžeme docílit pomocí nějakého frameworku, který imituje nativní vzhled prvků.

Popis zvolených technologií je popsán v kapitole 4.

3.4 MVC architektura

MVC architektura je jedna z nejoblíbenějších architektur, která se využívá u webových aplikací. Je například součástí i některých webových frameworků jako Nette, Zend, Ruby On Rail. Jedná se o trojúhelníkovou typologii, která rozděluje aplikaci na tři logické části tak, aby je šlo upravovat samostatně a dopad změn byl na ostatní části co nejmenší.

Základní myšlenkou MVC architektury je oddělení logiky od výstupu, kód se díky tomu lépe udržuje, je přehlednější a rozšiřitelný. Části MVC architektury jsou Model, View a Controller (z jejich počátečních písmen je tvořen název této architektury). Obsah podkapitoly popisující MVC architekturu vychází z informací dostupných z [9] a [10].

3.4.1 Model

Model obsahuje logiku a vše, co do ní spadá. Mohou to být výpočty, databázové dotazy, validace a podobně. Model pouze přijímá parametry, podle kterých má vydat data. Neví tedy nic o tom, odkud

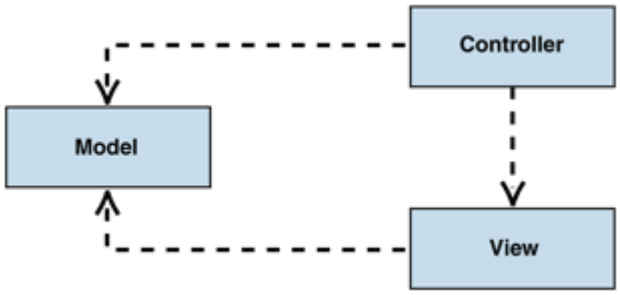
se data v parametrech berou a ani jakým způsobem budou výstupní data reprezentována či formátována.

3.4.2 View (pohled)

View nebo pohled se stará o zobrazení výstupu z Modelu uživateli. Nejčastěji se jedná o HTML stránku a tagy[3] značkovacího jazyka, který umožňuje do této šablony vkládání proměnných, případně využívá cykly a podmínky. View stejně jako Model neví, odkud mu data přišla, stará se pouze o jejich zobrazení uživateli.

3.4.3 Controller

Controller je onen chybějící prvek, který si lze představit jako prostředníka, se kterým komunikuje uživatel, Model i View. Drží celý systém pohromadě, propojuje všechny části mezi sebou, stará se o aktualizaci Modelu, aby došlo k případným výpočtům nových hodnot a aby se překreslily všechny pohledy.



Obrázek 3.1: MVC architektura [8]

Z obrázku 3.1 je patrné, že View a Controller přímo závisí na Modelu. Model nezávisí ani na Controlleru ani na View. To je klíčová výhoda takového rozdělení aplikace, protože dovoluje, aby byl Model sestaven a testován samostatně a nezávisle na vizuální prezentaci.

[3] Značka, podle které se prohlížeč řídí

4 Popis použitých technologií

Prvním krokem pro vytvoření informačního systému je vytvoření webové aplikace v HTML, CSS a použití AngularJS. Aplikace je založena na principu SPA[4], která minimalizuje server a klade důraz na klientskou část. Celou aplikaci představuje jediná stránka, na které jsou přítomny všechny podstránky, které jsou skrývány a zobrazovány podle potřeby.

Pokud chceme dosáhnout toho, aby se aplikace podobala co nejvíce nativní, většinou se využívá framework[5], který je k tomu určený. V implementaci informačního systému byl zvolen Ionic Framework. Druhým krokem pro vytvoření mobilní verze je zabalení webové aplikace pomocí nástroje Apache Cordova. Nástroj Apache Cordova si lze představit jako předpřipravené nativní aplikace pro jednotlivé platformy, které mají přes celou plochu WebView komponentu, kam se načte webová aplikace. Hlavní výhodou tohoto přístupu (oproti samostatné webové aplikaci) je možnost odeslání aplikace do obchodů Google Play, App Store nebo Windows Phone Store.

V ukázkách kódů na příkladech u konkrétních použitých technologií je vysvětlena problematika a princip fungování, které hrají důležitou roli a byly rovněž využity při vývoji aplikace.

4.1 HTML a CSS

HTML neboli HyperText Markup Language je značkovací jazyk definující strukturu stránky a používá se nejčastěji pro tvorbu webových stránek. Je také označován jako Document Object Model (DOM) se stromovou strukturou. Jazyk HTML je charakterizován množinou značek (tzv. tagů) a jejich vlastností (atributů) definovaných pro danou verzi. [11] Některé značky mohou být párové a některé nepárové.

1) Ukázka párové značky pro odstavec: `<p> Text odstavce </p>`
2) Ukázka nepárové značky obrázku: `<img src="/images/obrazek.jpg" />`

Na ukázce výše jsou dva příklady. První příklad pro párovou značku `<p>` pro odstavec platí, že koncová ukončující značka je shodná se značkou počáteční. Koncová značka však musí obsahovat před svým názvem lomítko. Mezi těmito značkami se nachází text odstavce. Nepárová značka v druhém příkladu nepoužívá koncovou značku, lomítko pak může nepovinně obsahovat uvnitř sebe. Zde je také vidět vlastnost `src` u elementu `img`, která definuje cestu, kde se obrázek nachází.

Jazyk CSS neboli Cascading Style Sheets je určený pro definování vzhledu a pozic elementů. Styl HTML elementu se může definovat přímo nastavením jeho atributu `style`. Další možnost, která

[4] Single Page Application
[5] Softwarová struktura, která slouží jako podpora při programování a vývoji. Může obsahovat podpůrné programy, knihovny API, podporu pro návrhové vzory nebo doporučené postupy při vývoji.

se často používá a umožňuje tak vícenásobné zadávání stejného stylu pro více elementů, je definování tříd.

Třídu lze nastavit u elementů, u kterých chceme tuto třídu aplikovat, nastavením jejich atributu `class="nazevtridy"`. Třída a její vlastnosti, které definují vzhled, mohou být definovány přímo v hlavičce dokumentu mezi značkami `<style></style>`. Častější způsob je definování stylů ve vlastních souborech s příponou `.css`. Element může zároveň obsahovat i více definujících tříd například `class="nazevtridy1 nazevtridy2"`. Místo tříd lze použít u elementu atribut identifikátoru `id="nazevid"`, který je jedinečný pro konkrétní element a nesmí se dále v dokumentu vyskytovat. Toto označení se využívá především při používání a manipulaci JavaScriptem.

4.2 AngularJS

Nástupem technologií HTML5 se JavaScript nebývale rozrostl. Aplikace v tomto jazyce se stávají čím dál víc komplexní a samotná knihovna jQuery[6] na to již přestává stačit. Má totiž jednu nevýhodu a tou je, že míchá dohromady aplikační logiku, zpracování událostí a manipulaci s DOM. U menších aplikací tento problém není tolik znát, ale u větších již ano. Kód začíná být nepřehledný a lehce se v něm programátor ztratí. V takovém případě nastupuje na řadu architektura MVC či nějaký její derivát.

Je zde však problém, jak architekturu MVC aplikovat v JavaScriptu a jak udělat template[7] systémem. HTML je výhodné pro použití ve statických dokumentech. Ne však v případech, pokud potřebujeme vytvořit dynamické aplikace, kde se informace ve View (tedy v HTML) neustále mění. A právě toto řeší framework od společnosti Google – AngularJS.

AngularJS je postaven na MVC architektuře, umožňuje vytvářet vlastní HTML elementy a atributy, navíc obsahuje *two – way data binding* a další užitečné vlastnosti, díky nimž se stává přehledný, snadno pochopitelný a kód není tak rozsáhlý. [12]

```
<ul>
  <li>
    <span> Nexus S </span>
    <p> Fast just got faster with Nexus S.</p>
  </li>
  <li>
    <span> Xperia Z5 Tablet </span>
    <p> The Next, Next Generation tablet.</p>
  </li>
</ul>
```

Kód 4.1: Ukázka části statické HTML stránky

[6] https://jquery.com/
[7] HTML šablona

Ukázky kódů 4.1, 4.2, 4.3 a 4.4 vychází z demonstračního příkladu z dokumentace AngularJS [15]. Na těchto ukázkách je popsáno využití AngularJS v praxi.

Na ukázce kódu 4.1 je vidět odrážkový seznam, který je napsán ve statické HTML stránce. Odrážkový seznam obsahuje pouze dva produkty telefonů v elementu `<span>` a jejich konkrétní popis v odstavci `<p>`. Dokážeme si však představit, co bychom museli udělat, pokud bychom chtěli mít třeba 100 takovýchto produktů. Museli bychom ručně přidat dalších 100 atributů `<li>` a u každého změnit text řádkového elementu `<span>` a text odstavce `<p>`.

V následující ukázce kódu 4.2 je na stejném příkladu ukázáno použití AngularJS. V AngularJS je pohled (View) projekcí Modelu přes šablonu HTML. To znamená, že kdykoliv se změní model, AngularJS tuto skutečnost zaznamená a aktualizuje pohled na patřičných místech.

```
<html ng-app="phonecatApp">
<head>
  ...
  <script src="bower_components/angular/angular.js"></script>
  <script src="js/controllers.js"></script>
</head>
<body ng-controller="PhoneListCtrl">

  <ul>
    <li ng-repeat="phone in phones">
      <span>{{phone.name}}</span>
      <p>{{phone.snippet}}</p>
    </li>
  </ul>

</body>
</html>
```

Kód 4.2: Ukázka kódu za použití AngularJS

V ukázce je vidět klasický HTML dokument, který však obsahuje speciální atributy – direktivy, které do jazyka HTML nepatří. Tyto direktivy právě zpracovává AngularJS. AngularJS využívá návrhový vzor Dependency Injection, který řeší závislost mezi jednotlivými komponentami programu. AngularJS obsahuje zabudovaný systém, který řeší tuto závislost napříč celou vyvíjenou aplikací a umožňuje vždy přesně specifikovat, které další komponenty jsou používány uvnitř konkrétní komponenty.

4.2.1 Direktivy

Jakmile narazí AngularJS na počáteční direktivu `ng-app`, začne od tohoto místa vyhodnocovat a zpracovávat všechny direktivy a vazby, na které narazí. AngularJS obsahuje spoustu dalších[8] direktiv. Dále budou popsány důležité direktivy, které byly využity při implementaci informačního systému.

[8] https://docs.angularjs.org/api/ng/directive

4.2.1.1 ng-app

Tato direktiva automaticky spouští aplikaci. Jako parametr očekává název hlavního modulu aplikace, v našem příkladu (ukázka kódu 4.2) to je `phonecatApp`. Dále v dokumentu musí být v hlavičce definována knihovna AngularJS a případně další používané skripty, kde jsou definovány controllery.

4.2.1.2 ng-controller

Pro celý element `<body>` je přidána direktiva `ng-controller` s názvem `PhoneListCtrl`. Controller plní důležitou roli. Poskytuje prostor, kam můžeme uložit informace tedy `$scope.` Každá aplikace má jediný kořenový `$scope` zvaný `$rootScope`. Všechny další jsou pak jeho potomky. `$scope` hraje důležitou roli, umožňuje oddělení mezi modelem, pohledem a cotrollerem a také obsahuje mechanismus, který sleduje změny v modelu a udržuje tak všechny části synchronizované. Změní-li se `$scope` v data modelu, promítne změny v šabloně a naopak, změní-li se `$scope` v šabloně, změní se i data v modelu.

4.2.1.3 ng-repeat

Z názvu této direktivy může být patrné, že se jedná o cyklus, který bude opakovat ten element, u kterého je definován – v našem případě element `<li>`. Vytváří uvnitř elementu speciální proměnné, jednak proměnnou `phone`, která obsahuje aktuální položku a pak speciální proměnnou `$index`, která obsahuje aktuální index položky z pole `phones`.

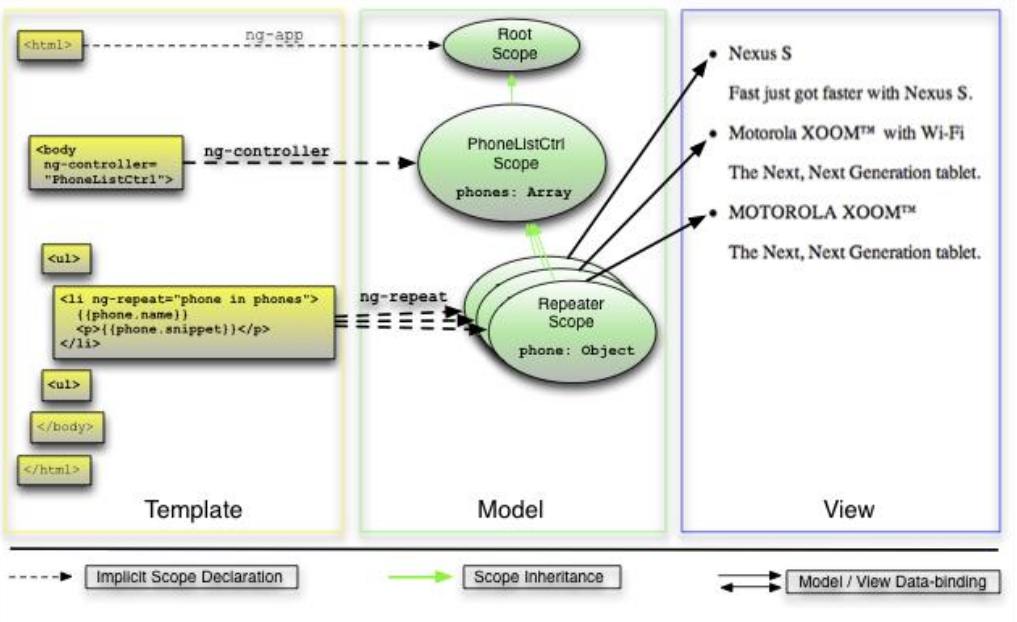
Další speciální proměnné jsou `$first`, `$middle` a `$last`, tedy proměnné obsahující logickou hodnotu pravda/nepravda, pokud je aktuální položka první, uprostřed mezi prvním a posledním prvkem nebo je položka posledním prvkem. Tyto speciální proměnné obsahují znak `$` a značí proměnné AngularJS. Je to z toho důvodu, aby nevznikl konflikt mezi proměnnými vytvořenými námi a proměnnými AngularJS. Výsledky se zde ještě navíc dají filtrovat a řadit např. přidáním za název pole následující výraz `| filter: $index`.

Výrazy ve složených závorkách značí AngularJS výrazy. Syntaxe je stejná jako v JavaScriptiu. V tomto výrazu se může vypisovat prvek pole, objekt, proměnné nebo volat funkce či používat filtry. V našem případě výrazy `{{phone.name}}` a `{{phone.snippet}}` značí vazbu odkazující na konkrétní aplikační model definován v controlleru.

```
angular.module('phonecatApp', []){
   .controller('PhoneListCtrl', function ($scope) {
     $scope.phones = [
       {'name': 'Nexus S',
        'snippet': 'Fast just got faster with Nexus S.'},
       {'name': Xperia Z5 Tablet',
        'snippet': 'The Next, Next Generation tablet.'},
       {'name': 'MOTOROLA XOOM™',
        'snippet': 'The Next, Next Generation tablet.'}
     ];
   };
});
```

Kód 4.3: Ukázka modelu a controlleru

V AngularJS je modelem jakýkoliv objekt přidaný do `$scope`. V ukázce kódu 4.3 reprezentuje datový model pouze jednoduché pole objektů `phones` s danými informacemi (`name` a `snippet`). Pole je vytvořeno v controlleru `PhoneListCtrl` na objektu `$scope`, který controlleru předává AngularJS parametrem. Na následujícím obrázku 4.1 je pro lepší představu znázorněn náš příklad.



Obrázek 4.1: Znázorňující ukázkový příklad [14]

4.2.1.4 ng-model

Direktiva `ng-model` hraje důležitou roli při *two – way data bindingu*. Znamená to, že veškeré změny ve `$scope` se přímo promítnou na tomto elementu a naopak. Využití je na formulářových elementech[9] například `input`, `select`, `textarea` aj. V implementaci informačního systému má tato direktiva klíčovou vlastnost pro editace nebo předvyplnění konkrétních formulářových elementů právě načtenými daty z databáze.

[9] https://docs.angularjs.org/api/ng/directive/ngModel

4.2.1.5 ng-submit

Tato direktiva zpracovává JavaScriptovou událost `onSubmit`, tedy vyhodnocení formuláře. V parametru této direktivy je potom možné volat funkci z controlleru a předat jí parametrem data z formuláře. Jinými slovy „posbírá“ všechny direktivy `ng-model` a předá k vyhodnocení.

4.2.1.6 ng-show

Direktiva `ng-show` umožňuje zobrazovat určitý HTML element, ke kterému je přidružena. Vyhodnocení je na základě výrazu jejího atributu. Obdobou této direktivy je `ng-hide`, která element skrývá. Direktiva je v implementaci využívána pro zobrazení varovné hlášky, která upozorňuje, že uživatel nemá zvolenou aktivní nemovitost nebo byt.

4.2.1.7 ng-click

Direktiva `ng-click` umožňuje zadat vlastní chování elementu při kliknutí na tento prvek. Většinou se využívá u tlačítek `<button>`.

4.2.1.8 ng-checked

Tato direktiva nastavuje hodnotu checked (zatrženo) u elementu `<input>`, podle podmínky uvnitř atributu `ng-checked`. V implementaci informačního systému je tato direktiva využívána při zvolení aktivní nemovitosti nebo bytu.

4.2.1.9 ng-change

Tato direktiva najde uplatnění u prvků, které nějakým způsobem změnil uživatel. Může to být například textové pole, selecty nebo checkboxy. Při této změně je často volána funkce, která provede reakci na tuto změnu.

4.2.1.10 ng-if

`ng-if` umožňuje nastavit elementu testovací podmínku podle zadaného výrazu. Pokud je podmínka vyhodnocena jako nepravda, je tento element odstraněn z DOM. V opačném případě je do DOM přidán. Rozdíl oproti `ng-show` nebo `ng-hide` je právě v tomto odstranění z DOM.

4.2.1.11 ng-value

Direktiva `ng-value` je provázána s `ng-model`, kde nastavuje výchozí hodnotu vybraného prvku. Uplatňuje se např. u selectu, radio buttonu nebo checkboxů.

4.2.1.12 ng-src

Pokud bychom chtěli u cesty k obrázku v elementu použít následující AngularJS výraz `<img src=“{{path}}“`, došlo by k chybě a obrázek by nebyl načten. Cesta by byla vyhodnocena přesně

tak, jak je zapsaná v uvozovkách, nikoliv načtena z AngularJS proměnné `path`. Tento problém nám však odstraňuje direktiva `ng-src`, která se použije namísto atributu `src`.

4.2.2 Routování

Aby AngularJS věděl, jakou šablonu má pro konkrétní URL načíst, využívá k tomu stavy `$stateProvider` a službu `$urlRouterProvider`. Pro každý stav specifikujeme URL, pro které pravidlo platí, šablonu, která se má načíst a případně controller, který se zavolá a načte. Musí se také nastavit výchozí stav, pokud nevyhovuje ani jedna námi definovaná podmínka. [13][15] Příklad takového routování (z angl. routing), které je využito i při implementaci informačního systému, je vidět na ukázce kódu 4.4.

```
.config(function($stateProvider, $urlRouterProvider) {
 $stateProvider
   .state('app.myEstate',{
     url: '/myEstate',
     views: {
     'menuContent': {
         templateURL:'templates/my-estate.html',
         controller:'MyEstateCtrl'
     }
   }
 });
 $urlRouterProvider.otherwise('/app/homepage');
});
```

Kód 4.4: Routování

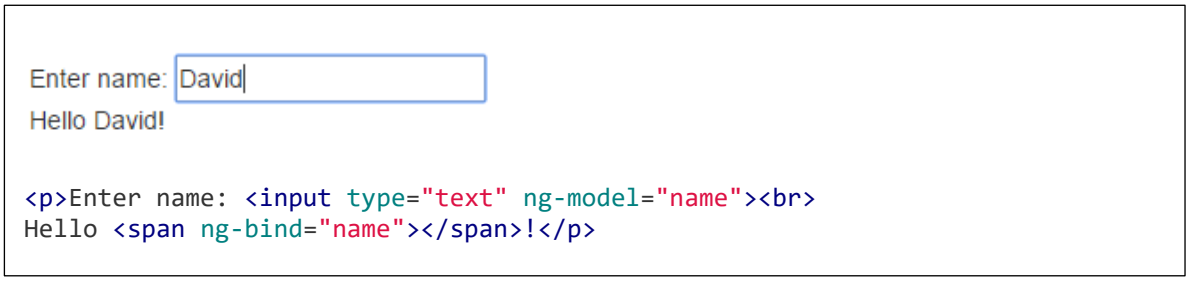
4.2.3 Services

V AngularJS je možné vytvořit vlastní služby. Jedná se o funkce, které jsou zodpovědné za provádění určité operace. Díky tomu je kód dobře udržovatelný a testovatelný. [15]

V implementaci informačního systému jsou využity jako služby práce s databází, služba pro uchovávání aktivní nemovitosti a bytu a práce s obrázky.

4.2.4 Two – way data binding

Jednou z předností AngularuJS je právě two – way data binding. Jedná se o synchronizaci stavů mezi modelem a view. Můžeme tento pojem přeložit do češtiny jako „dvoucestná synchronizace dat“. [13] Na následujícím příkladu 4.5, ve kterém se nachází jedno textové pole, do kterého uživatel zadává data, je vysvětleno, o co se jedná.



Kód 4.5: Ukázka Two-way data bindingu

Ve chvíli, kdy uživatel napíše nějaký řetězec do textového pole, se tento text přenese do modelu (to je první cesta). Následně se z modelu automaticky přepíše do ostatních částí view (druhá cesta). Textovému inputu přidáváme direktivu `ng-model`, která přesune do modelu `name` to, co napíše uživatel. Druhý řádek obsahuje element `<span>` s direktivou `ng-bind`, která říká, že do tohoto elementu se mají vypsat data, která jsou v modelu `name`. A to je vše.

Jedná se o deklarativní způsob implementace, tedy programátor nemusí imperativně popisovat, jak data synchronizovat, místo toho deklaruje vztah a AngularJS už zajistí synchronizaci.

4.3 Apache Cordova (Adobe PhoneGap)

Apache Cordova je open – source[10] framework pro mobilní vývoj. V roce 2009 společnost Nitobi vyvinula PhoneGap, který umožňoval vývoj pomocí webových technologií HTML, CSS a JavaScriptu a ty zabalit jako nativní. Posléze byla společnost Nitobi koupena společností Adobe, která darovala PhoneGap komunitě pod Apache Foundation. Jelikož si chtěli značku PhoneGap ponechat, tak došlo k přejmenování na Apache Cordova. Pod značkou PhoneGap teď společnost Adobe nabízí online build aplikací pro různé platformy, který je právě založen na Apache Cordova. [16] Tato podkapitola vychází z [17], [18] a [19].

Předností nástroje Apache Cordova je především podpora multiplatformního vývoje aplikací. Na obrázku 4.2 jsou znázorněny všechny podporované platformy a jejich systémové API.

[10] Počítačový software s otevřeným zdrojovým kódem, který může být šířen, prohlížen nebo upravován.

	android	blackberry10	ios	Ubuntu	wp8 (Windows Phone 8)	windows (8.1, 10, Phone 8.1)
cordova CLI	✓ Mac, Windows, Linux	✓ Mac, Windows, Linux	✓ Mac	✓ Ubuntu	✓ Windows	✓
Embedded WebView	✓ (see details)	✗	✓ (see details)	✓	✗	✗
Plugin Interface	✓ (see details)	✓ (see details)	✓ (see details)	✓	✓ (see details)	✓
	Core Plugin APIs					
Accelerometer	✓	✓	✓	✓	✓	✓
BatteryStatus	✓	✓	✓	✗	✓	✓ * Windows Phone 8.1 only
Camera	✓	✓	✓	✓	✓	✓
Capture	✓	✓	✓	✓	✓	✓
Compass	✓	✓	✓ (3GS+)	✓	✓	✓
Connection	✓	✓	✓	✓	✓	✓
Contacts	✓	✓	✓	✓	✓	partially
Device	✓	✓	✓	✓	✓	✓
Events	✓	✓	✓	✓	✓	✓
File	✓	✓	✓	✓	✓	✓
File Transfer	✓	✓ * Do not support onprogress nor abort	✓	✗	✓ * Do not support onprogress nor abort	✓ * Do not support onprogress nor abort
Geolocation	✓	✓	✓	✓	✓	✓
Globalization	✓	✓	✓	✓	✓	✓
InAppBrowser	✓	✓	✓	✓	✓	uses iframe
Media	✓	✓	✓	✓	✓	✓
Notification	✓	✓	✓	✓	✓	✓
Splashscreen	✓	✓	✓	✓	✓	✓
Status Bar	✓	✗	✓	✗	✓	✓ Windows Phone 8.1 only
Storage	✓	✓	✓	✓	✓ localStorage & indexedDB	✓ localStorage & indexedDB
Vibration	✓	✓	✓	✗	✓	✓ * Windows Phone 8.1 only

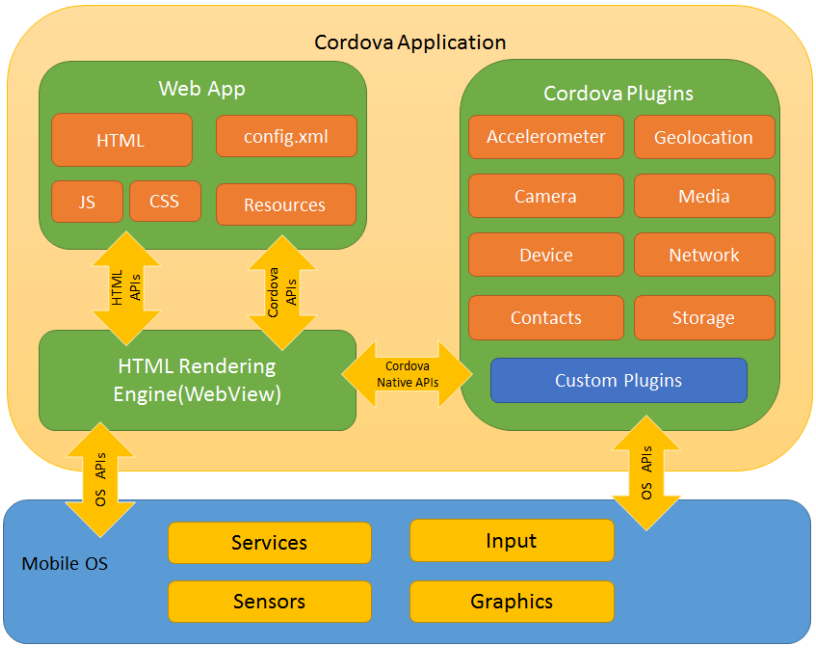
Obrázek 4.2: Podporované platformy a systémové API Apache Cordovou [18]

Aplikace má přes celý display jednu WebView komponentu, webovou stránku, která je ve výchozím nastavení v souboru `index.html`, která dále odkazuje na další soubory (CSS, JS, obrázky, soubory a další, které jsou nezbytné pro spuštění). Web App je ta část, která obsahuje vlastní naprogramovaný kód aplikace. Důležitou roli hraje soubor `config.xml`, který poskytuje informace o aplikaci a definuje její parametry. Tyto parametry mohou být například, jak bude aplikace reagovat na otočení telefonu, zda se bude zobrazovat aplikace přes celý display telefonu atd.

Pluginy[11] jsou nedílnou součástí Apache Cordova, protože tyto pluginy zpřístupňují nativní API zařízení (fotoaparát, geolokaci, souborový systém, orientaci zařízení atd.) přímo do JavasCriptu uvnitř WebView.

Na následujícím obrázku 4.3 je zobrazena architektura Apache Cordova.

[11] Doplňkový modul jiné aplikace, rozšiřuje tak její funkčnost



Obrázek 4.3: Architektura Apache Cordova [17]

Spousta pluginů již existuje v Cordova Plugin Repository[12]. Můžeme si však vytvořit vlastní pluginy. Instalace pluginů i samotné Cordovy je pomocí nástroje `npm`[13], který funguje na serveru Node.js[14]. Tento balíčkovací systém zajišťuje nejen správu pluginů, ale i aktualizace. Instalace Cordovy se spouští příkazem:

```
$ npm install -g cordova
```

Konzolový nástroj vytvoří strukturu aplikace, vytvoří adresář `www`, ve které je umístěn soubor `index.html`, adresář `/templates` obsahující podstránky (html soubory) a adresář `/js`, ve které je uložena logika aplikace – tedy skripty. Dále se zde nachází adresář `/css` pro vlastní styly nebo se zde mohou nacházet adresáře s obrázky či soubory, které jsou potřebné v aplikaci. Dalším adresářem je adresář `/lib`, do které se potom přidávají pluginy. Při prvním spuštěním aplikace se zde žádné pluginy nenainstalují, programátor si tedy sám přidává pluginy, které potřebuje. Instalace těchto pluginů probíhá také pomocí konzolového nástroje `npm`. Například přidání pluginu podporující fotoaparát telefonu:

```
$ cordova plugin add cordova-plugin-camera
```

Následně se příkazem určí, pro kterou platformu má být aplikace přeložena:

[12] http://cordova.apache.org/plugins/
[13] Balíčkovací systém pro JavaScript https://www.npmjs.com/
[14] Javascriptová platforma používaná jako server založený na Chrome JavaScript runtime.

```
$ cordova platform add android ios
```

4.4 Ionic framework

Tato podkapitola popisující Ionic framework vychází z [20] a z oficiální dokumentace Ionic frameworku [21].

Ionic framework je SDK, tedy soubor nástrojů, pomocí něhož lze vyvíjet software. Ionic framework byl vytvořen na bázi AngularJS pro sestavení aplikací na cílovou platformu za využití Apache Cordova. Ionic framework se rychle dostal mezi nejvíce používané frameworky pro vývoj hybridních mobilních aplikací a to i přes to, že je na trhu poměrně krátkou dobu. Vývoj začal na konci roku 2013 a od té doby bylo vytvořeno přes 600 000 mobilních aplikací.

Ionic framework obsahuje velice dobře zpracovanou dokumentaci, fórum[15] s řešením nejrůznějších problémů nebo funkční dema[16].

Ionic framework se nainstaluje pomocí příkazu:

```
$npm install -g ionic
```

Následně se vytvoří nový projekt. Můžeme si vybrat, zda vytvořit prázdný, nebo již předpřipravený projekt od Ionicu využívající postranní menu nebo záložky. Pro vytvoření projektu se záložkami se například použije příkaz:

```
$ionic start myApp tabs
```

Ionic framework je primárně zaměřen na vzhled a dojem uživatelských prvků aplikace. V současné verzi 1.2.4 podporuje uživatelské prvky WebView pro platformy verze iOS7+ a Android 4.1+. Windows phone zatím podporován není. Vývojáři však plánují v nejbližší době i podporu Windows phone. Ionic však podporuje od verze 1.2 pouze Windows 10, tedy i telefony s Windows 10 mobile a prohlížeč MS Edge.

Příkaz, který přidá do projektu platformy Android a iOS, ve kterých chceme framework využívat:

```
$ionic platform add android ios
```

Ionic framework definuje nové značky podobné jazyku HTML. Jelikož je založen na AngulaJS, je možné si definovat vlastní značky, které zpracovává. Většina značek je párových a začínají slovem `<ion-`. Každý soubor z adresáře `tepmplates` (soubory s konkrétními podstránkami

[15] https://forum.ionicframework.com/
[16] http://codepen.io/ionic/pens/public/?grid_type=list

aplikace) začíná párovou značkou `<ion-view>`. Vše, co je mezi tímto tagem (např. tagy `<ion-content>` určující viditelnou hlavní část ve View), se dosadí do tagu `<ion-nav-view>` v hlavním souboru `index.html`. Dále zde můžeme definovat např. záhlaví `<ion-header-bar>` nebo zápatí `<ion-footer-bar>`. Tagy Ionicu je však možné definovat i jinak. Lze použít běžné tagy z jazyka HTML. Stačí nastavit CSS třídu konkrétnímu elementu. Na ukázce kódu 4.6 jsou ukázány dva ekvivalentní zápisy.

```
<ion-list>                                  <div class="list">
  <ion-item> Item 1 </ion-item>               <div class="item"> Item 1</div>
  <ion-item> Item 2 </ion-item>               <div class="item"> Item 2</div>
</ion-list>                                 </div>
```

Kód 4.6: Dva ekvivalentní zápisy kódu pro seznam položek

Všechny prvky, které je možné použít, jsou popsané v oficiální dokumentaci[17]. Navíc Ionic obsahuje i vlastní ikonky[18] nebo efekty[19].

Z využitých efektů, které nabízí Ionic jsou v aplikaci využity například:

Slide-box

`<ion-slide-box>`, který umožňuje pomocí tahu zobrazovat další komponenty. Tento slide-box je využit u zobrazování obrázků u konkrétní investice.

Modální okna

Další efekty Ionicu jsou jeho služby pro otevírání modálních oken. Modální okna vyžadují nějakou akci od uživatele, kterou musí provést, aby se mohlo pokračovat dále. Modální okna jsou využívána při přidávání položek, editací nebo mazání. Službu modálních oken zajišťuje `$ionicModal`. Tato služba potřebuje identifikátor `id` konkrétního okna, které se má zobrazit. Dále se musí nastavit parametry, ze které šablony se má okno otevřít a nakonec zvolení animace okna. Animace okna byla použita `slide-in-up`, tedy odspodu nahoru.

`$ionicActionSheet` je služba, která otevře modální okno s výběrem možností. Okno není přes celý display, ale pouze přes čtvrt obrazovky. Uživatel nemusí provádět žádnou akci a okno lze kdykoliv zrušit kliknutím mimo něj. `$ionicActionSheet` je využit pro výběr přehledů událostí, které se zobrazí v kalendáři.

Postranní menu

`$ionicSideMenuDelegate` je další služba, která umožňuje práci s navigačním menu aplikace. Menu se zobrazuje buď kliknutím na ikonku menu, nebo přejetím po obrazovce zleva doprava.

[17] http://ionicframework.com/docs/components/
[18] http://ionicons.com/
[19] http://ionicframework.com/docs/api/

Záložky

Záložky umožňují konzistentní navigaci mezi více okny nebo šablonami. Většinou jsou kombinovány s textem a ikonkou. Mohou být umístěny v hlavičce nebo patičce stránky. V systému jsou využity u oddělení nemovitostí a bytů, dále u správy revizí a nakonec u přehledů investic.

4.5 SQLite

SQLite je typ lokální databáze, která se často uplatňuje při vývoji aplikací pro mobilní telefony, a také jako datové úložiště desktopových aplikací, zejména internetových prohlížečů. Nejedná se o plnohodnotnou databázi, jako například MySQL[20], nýbrž o knihovnu, která je přilinkovaná k některému skriptovacímu jazyku a umožňuje otevřít datový soubor a pracovat s ním prostřednictvím jazyka SQL. Díky této vlastnosti je zajištěna rychlost a přenositelnost. Uplatnění najde u paměťově omezených zařízení. [23]

Transakce SQLite jsou ACID. [22] Zkratka ACID je odvozena od počátečních písmen názvů vlastností, které má. Jsou to:

- **Atomicity** (atomičnost) – transakce představuje z hlediska provedení jeden celek, tzn. Je provedena buď celá transakce, nebo nic
- **Consistency** (konzistence) – izolovaná transakce[21] neporušuje konzistenci databáze. Platí tedy, že pokud byla databáze v konzistentním stavu před zahájením provádění transakce, musí být v konzistentním stavu i po skončení transakce.
- **Isolation** (izolace) – pokud bude probíhat několik transakcí souběžně, bude zajištěno, že transakce budou z hlediska provádění operací s daty v databázi izolované. Efekt jejich operací bude takový, jako by probíhaly jedna za druhou.
- **Durability** (trvalost) – po úspěšném dokončení transakce budou mít všechny změny v databázi trvalý charakter, a to u po výpadku systému.

Další vlastností je CRUD, tedy schopnost databáze vytvářet (create), číst (read), modifikovat (update) a mazat (delete) data podle požadavků aplikace. Dále potřebujeme znát jazyk SQL. Na následující ukázce kódu 4.7 je ukázán jednoduhý SQL dotaz pro výběr všech hodnot z tabulky `estates` podle konkrétního identifikátoru.

```
"SELECT * FROM estates WHERE estate_id = " + tid
```

Kód 4.7: Ukázka SQL dotazu

Databázové soubory se lokálně otevírají v režii jádra operačního systému a nejsou problémy s vyrovnávacími pamětmi. Zámky fungují, i když se souborem pracují stovky skriptů současně. Do

[20] https://www.mysql.com/
[21] Transakce, která není ovlivněna žádnými jinými souběžně probíhajícími transakcemi

databáze můžeme ukládat hodnoty jakéhokoli typu a můžeme zde používat integritní omezení, cizí klíče nebo používat triggery.

4.6 Pluginy

Všechny pluginy Apache Cordova nalezneme v jejich dokumentaci na oficiálních webových stránkách[22]. V této oficiální dokumentaci nalezneme popis konkrétních pluginů, jejich použití, instalaci a odkaz na kompletní zdrojový kód. Při instalaci pluginu lze místo názvu konkrétního pluginu dosadit URL adresu z repozitáře GitHub[23].

Při implementaci informačního systému byly použity tyto pluginy:

- `Cordova-plugin-camera`: pro využívání fotoaparátu v aplikaci
- `Cordova-plugin-file`: pro transport fotografií do aplikace
- `Cordova-plugin-local-notification`: pro využívání notifikací a upozorňování
- `Ionic-calendar`: pro využívání vlastního kalendáře v aplikaci

[22] http://ngcordova.com/docs/plugins/
[23] Webová služba podporující vývoj softwaru při používání verzovacího nástroje Git.

5 Návrh a implementace

Hlavní myšlenkou pro vytvoření tohoto informačního systému je udělat aplikaci ve smyslu servisní knížky, která bude uchovávat, zobrazovat a spravovat investice či jiné výdaje do nemovitosti a bytu. Aplikace byla pojmenována jako *„Moje nemovitost“* a skládá se z šesti základních částí:

1) Moje nemovitost
2) Investice do nemovitosti
3) Investice do bytu
4) Přehled spotřebičů
5) Revize a údržba
6) Kalendář

Jednotlivé části aplikace budou popsány dále. Pro lepší představu je v příloze č. 1 zobrazena adresářová struktura adresáře `www` obsahující konkrétní zdrojové soubory aplikace.

Aplikace uchovává data v lokální databázi SQLite popsané v kapitole 4.5. Pro práci s databází je vytvořena služba `SQLService`, která umožňuje přístup k tabulkám databáze, obsahuje důležité funkce pro vytvoření databáze a tabulek, dále funkce pro vkládání, mazání a výběr dat z databáze podle konkrétních parametrů z konkrétních tabulek.

Další službou je služba `Active`, která zajišťuje uložení a získání zvolené aktivní nemovitosti nebo bytu. Údaj o aktivní nemovitosti nebo bytu ukládá do `localStorage`. Oproti `sessionStorage` nemá čas expirace, tedy uchová údaje trvale a to i po vypnutí aplikace.

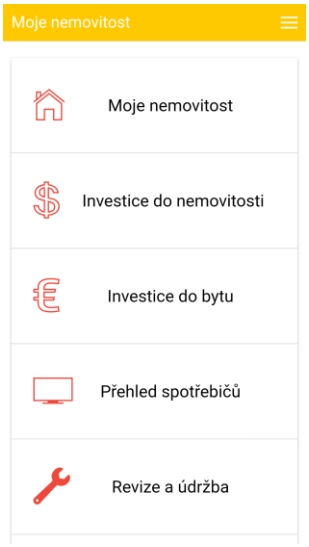
Poslední služby jsou `FileService` a `ImageService`, které zajištují uložení a odstranění obrázků (pořízených fotografií) u detailu investice. Služba `ImageService` uloží pořízenou fotografii pouze do aplikace, neukládá fotografii do galerie telefonu. Nastaví parametry fotografie jako je velikost a typ. Cesta k obrázku a primární klíč investice tvoří pole ve formátu `json`, kde výběr položek (přidružených fotografií ke konkrétní investici) probíhá na základě primárního klíče investice. Všechny výše zmíněné služby jsou implementovány v souboru `service.js`.

5.1 Homepage a menu

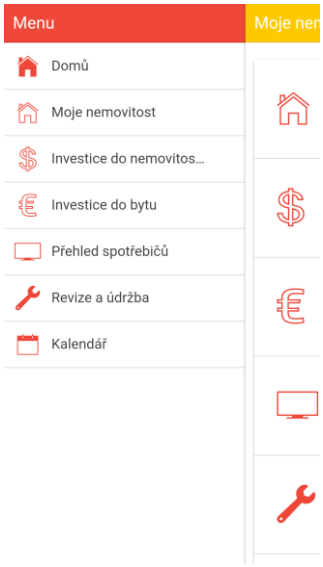
V souboru `app.js` je implementováno routování a přesměrování. Routování je popsáno v kapitole 4.2.2. Při spuštění aplikace je uživateli zobrazena výchozí stránka (stav) `homepage.html` s navigačními kartami aplikace.

Dále je na každé stránce uživateli zobrazeno navigační menu `menu.html` v pravém horním rohu. Menu lze zobrazit pomocí ikonky menu, popřípadě přejetím prstu po obrazovce zleva doprava.

Při detailním náhledu konkrétní položky se v levé časti hlavičky zobrazuje navigační šipka pro návrat zpět.



Obrázek 5.1: Homepage

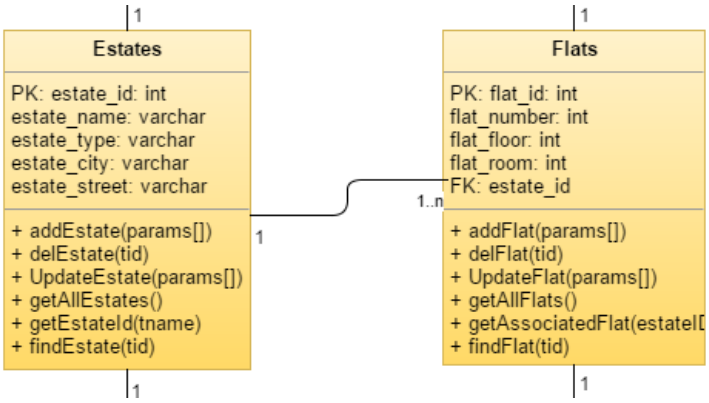


Obrázek 5.2: Postranní menu

5.2 Moje nemovitost

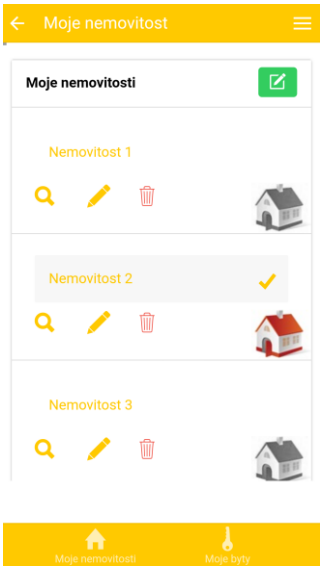
„Moje nemovitost“ hraje v aplikaci velmi důležitou roli. Zde uživatel vytváří nové nemovitosti a byty. Uživatel může přidat (vlastnit) více nemovitostí. Ke konkrétní nemovitosti pak potom přidává konkrétní byty. V souboru `controllers.js` se nachází aplikační logika pro tuto část a pro investice do nemovitosti a bytu. Jsou to controllery `MyEstateCtrl` a `MyFlatCtrl`. Je zde využívána služba `SQLService`, pro komunikaci s databází. Služba zajišťuje načtení dat z databáze do View.

Na obrázku 5.3 jsou zobrazeny tabulky databáze `estates` a `flats` uchovávající informace o nemovitosti a bytu. U nemovitosti je uchováván název nemovitosti, typ nemovitosti (rodinný dům, bytový, rekreační objekt), město a adresa. U bytu je to číslo bytu, podlaží, počet pokojů a nemovitost, které náleží.

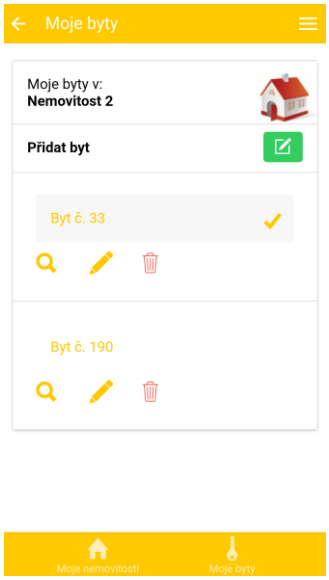


Obrázek 5.3: Tabulky databáze

Pro vytvoření a úpravu nemovitosti/bytu jsou použity modální okna, ve kterých se nachází formulář pro zpracování zadaných dat. *„Moje nemovitost“* obsahuje dvě záložky – moje nemovitosti a moje byty. Uživatel na každé kartě musí udělat jeden důležitý krok – zvolit aktivní nemovitost a byt. U těchto zvolených položek bude potom umožněno přidávání a zobrazování investic, revizí apod. Aplikace uživateli jasně dává najevo, který byt a nemovitost má zvolenou (checkbox u názvu). Pokud uživatel nemá zvolenou žádnou aktivní nemovitost nebo byt, aplikace uživatele upozorní o této skutečnosti. Ztráta aktivity je možná při prvním spuštění, kdy není přidána žádná nemovitost a byt, nebo jestliže uživatel smaže aktivní byt nebo nemovitost. Na obrázkách níže jsou zobrazeny záložky moje nemovitosti a moje byty.



Obrázek 5.4: Záložka „Moje nemovitosti“

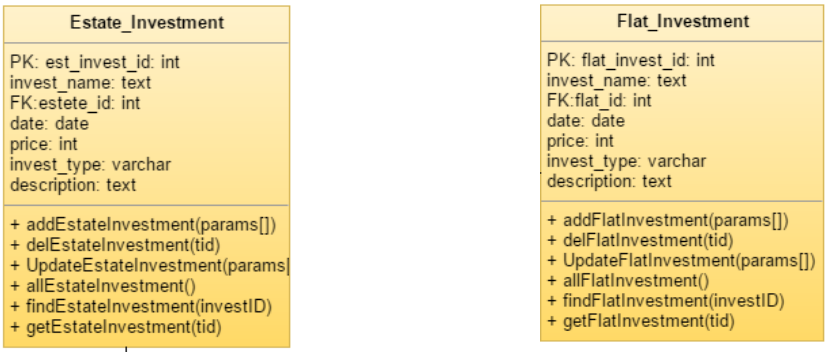


Obrázek 5.5: Záložka „Moje byty“

5.3 Investice do nemovitosti a bytu

V této sekci se uživateli zobrazují přidané investice v rámci aktivní nemovitosti či bytu. Typy investic nemovitosti mohou být například opravy, výměny, rekonstrukce nebo vlastní investice. Investice v bytu mohou zahrnovat také opravy a výměny v konkrétním bytě, ale mohou to být také typy investic jako pořizování nového vybavení bytu, doplňků, nových spotřebičů, elektroniky či vlastní typ investic.

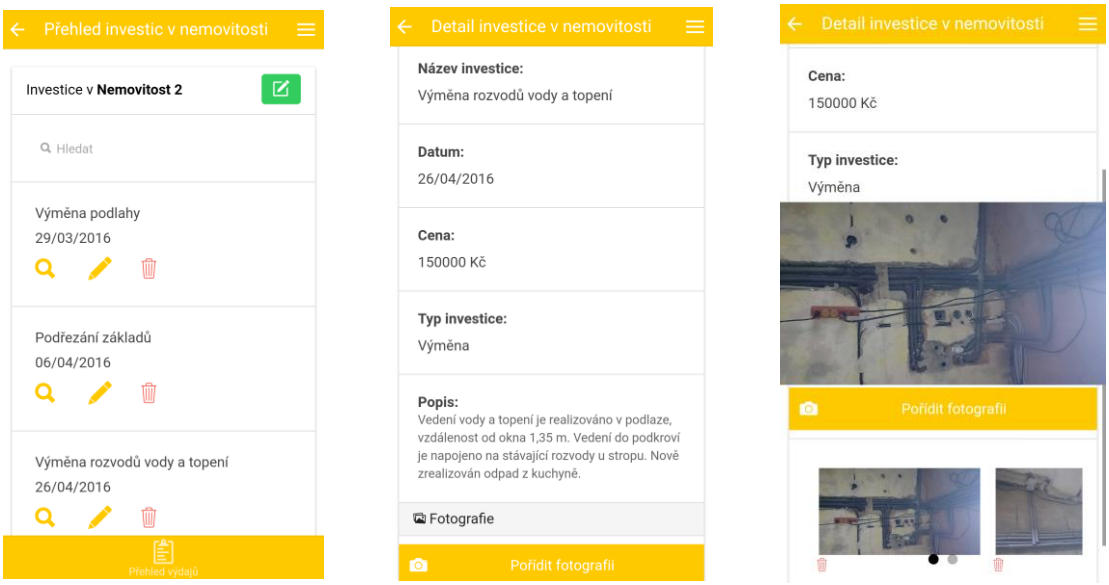
Controller pro investice do nemovitosti je `MyEstateInvestment` a pro investice do bytu `MyFlatInvestment`. Na obrázku 5.6 jsou zobrazeny tabulky databáze uchovávající investice nemovitosti `Estate_Investment` a bytu `Flat_investment`. Investice obsahuje název, datum, cenu, typ, popis a nemovitost nebo byt, ke které přísluší. Jestliže je přidávána investice jako typ nový spotřebič, automaticky se tento spotřebič vytvoří i v sekci pro správu spotřebičů.



Obrázek 5.6: Tabulky databáze

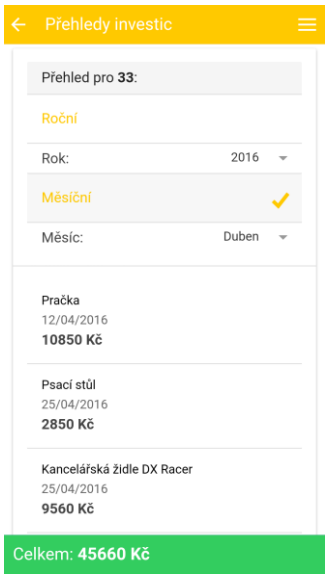
Uživatel zde může vyhledávat investice podle zadaného výrazu. Vyhledávání probíhá v názvu, datu, ceně, typu či popisu. Vyhledávání probíhá i v podvýrazech.

V detailním náhledu se nachází tlačítko pro přidávání fotografií ke konkrétní investici. Uživateli se po stisku tohoto tlačítka zapne fotoaparát. Pořízená fotografie se za využití služby `FileService` a `ImageService` uloží. Uživatel může obrázky odstraňovat nebo je může prohlížet pomocí slideshow.



Obrázek 5.7: Přehled investic do nemovitosti, detailní náhled a slideshow fotografií

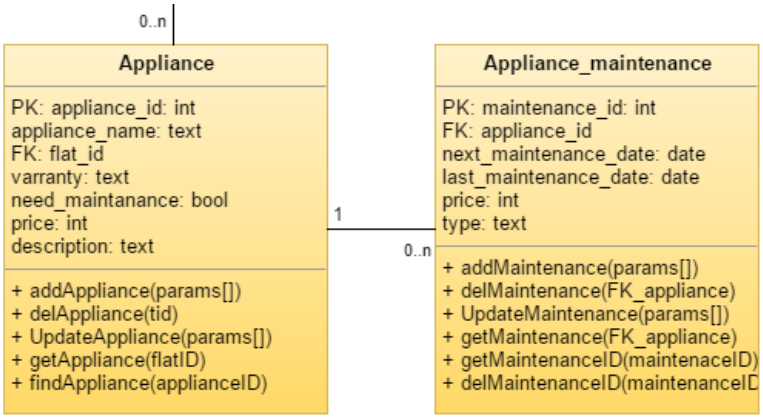
Další možností v sekci investic, ať už investic do bytu nebo do nemovitosti, je záložka přehled výdajů. Zde si uživatel zvolí přehled, který chce zobrazit. Má na výběr dvě možnosti. Buď roční přehled pro konkrétní rok, nebo měsíční přehled investic v konkrétním měsíci a v konkrétním roce. Aplikace potom sama vyfiltruje data podle zvolených parametrů a spočítá celkové výdaje. Suma je zobrazena v patičce aplikace. Na obrázku 5.8 je zobrazena tato záložka.



Obrázek 5.8: Přehled investic za určité období

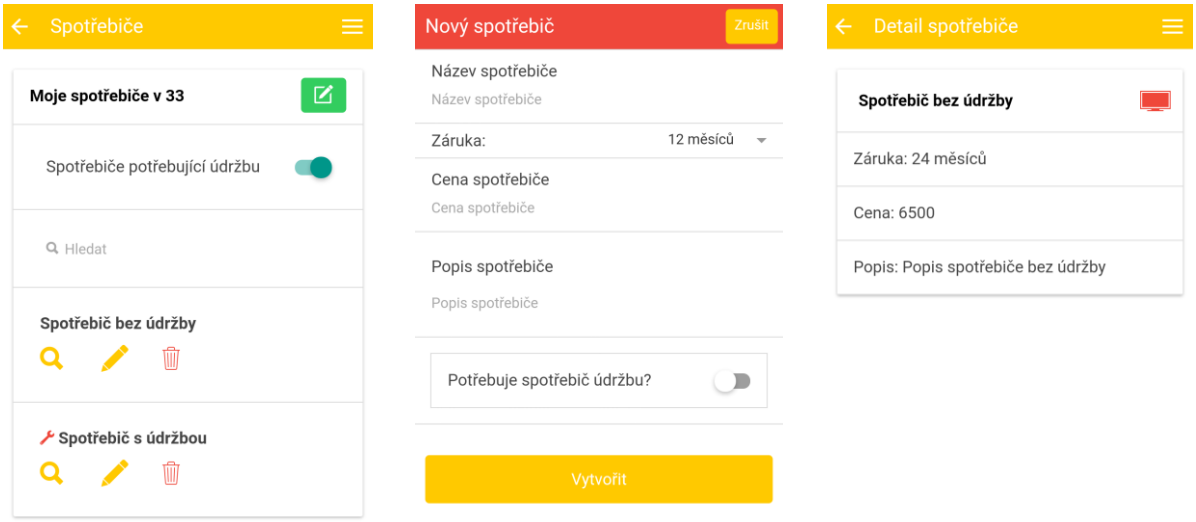
5.4 Přehled spotřebičů

V přehledu spotřebičů jsou evidovány konkrétní spotřebiče. Controllery `MyEstateAppliance` a `MyEstateApplianceDetail` pro tuto část se nacházejí v souboru `appliance.js`. Na obrázku 5.9 se nachází tabulky databáze pro přehled spotřebičů. U každého spotřebiče se eviduje byt, ve kterém se nachází, dále název spotřebiče, jeho záruka, cena, popis a nakonec zda spotřebič potřebuje údržbu. Údržbou je myšleno přidávání další evidence ke konkrétnímu spotřebiči. Lze tedy evidovat typ údržby, což může být servis, oprava atd., data posledních a plánovaných údržeb a jejich cena.



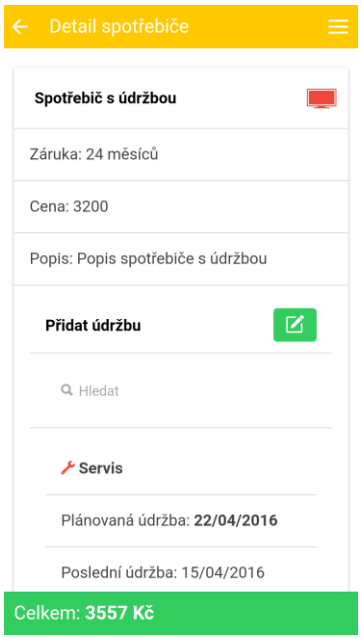
Obrázek 5.9: Tabulky databáze

Během přidávání nového spotřebiče uživatel zvolí, zda potřebuje spotřebič údržbu nebo ne (obrázek 5.10).

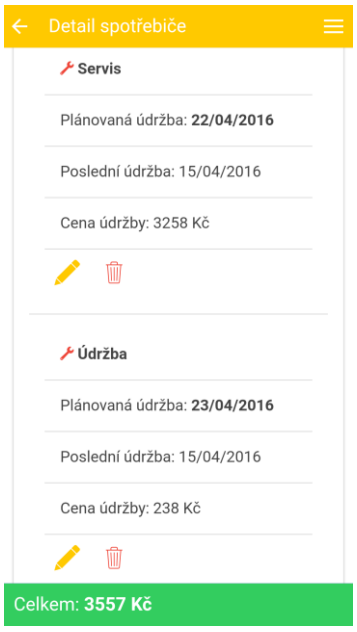


Obrázek 5.10: Přehled spotřebičů, přidání nového spotřebiče a detail spotřebiče bez údržby

Pokud spotřebič potřebuje údržbu, v detailním náhledu je mu zpřístupněna volba pro přidávání nové evidence, jako údržby spotřebiče, jeho servisy, opravy nebo přidávání vlastních typů (obrázky 5.11 a 5.12).



Obrázek 5.11: Detail spotřebiče s údržbou



Obrázek 5.12: Evidence

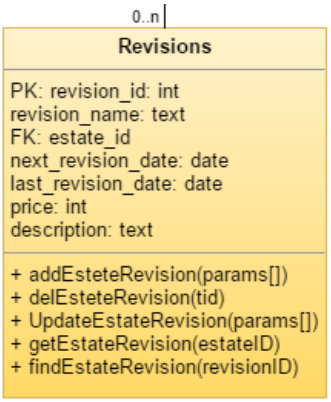
Jakmile uživatel přidá novou položku, systém vytvoří novou notifikaci (upozornění) podle plánovaného data, které uživatel zadal. Uživateli je potom notifikace zobrazena týden dopředu, že má nastavenou plánovanou údržbu nebo servis konkrétního spotřebiče.

Konkrétní položky u detailu spotřebiče může uživatel opět editovat, mazat a vyhledávat. V patičce spotřebiče s údržbou se uživateli počítají všechny výdaje spojené s údržbou. Uživatel tedy

vidí veškerou evidenci u spotřebiče. Dle těchto informací se potom může rozhodovat, zda je ještě výhodné investovat do stávajícího spotřebiče nebo raději koupit nový.

5.5 Revize

Na kartě revize uživatel spravuje nejrůznější revize, které se běžně provádějí u nemovitostí. Aplikační logika pro tuto část se nachází v souboru `revisions.js`. Odpovídající controllery k této části jsou `MyEstateRevisions`, `MyEstateRevisionsDetail` a `MyEstateRevisionsContacts`. Na obrázku 5.13 se nachází tabulka databáze uchovávající revize. U každé revize je evidován název revize, datum proběhlé revize a datum plánované revize, cena revize, popis k této revizi a nakonec nemovitost, ve které je revize prováděna.

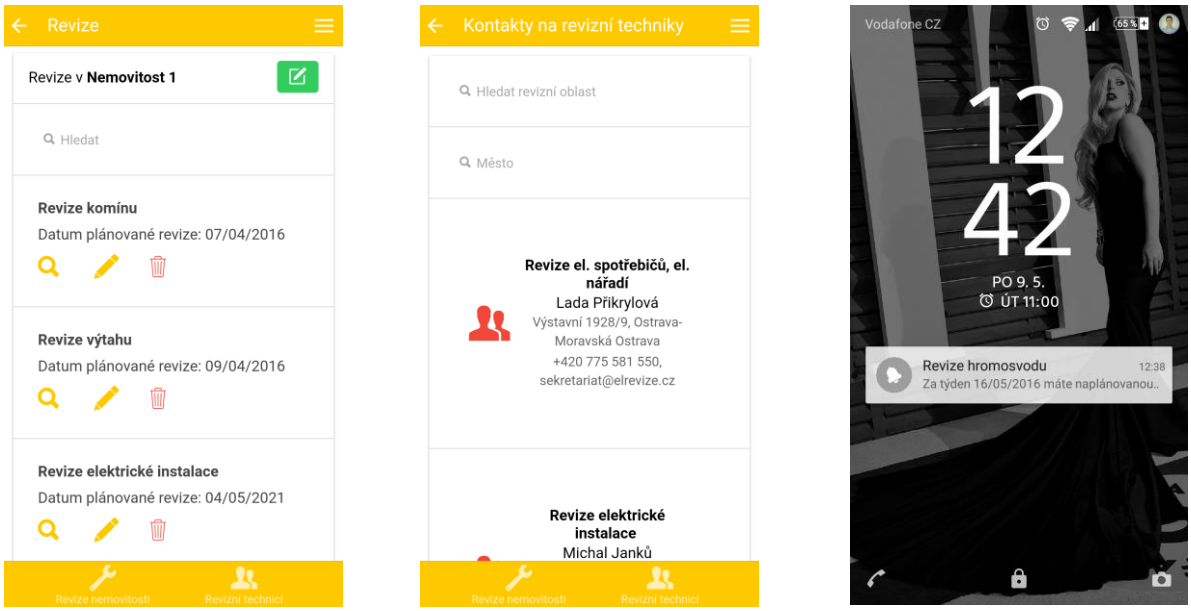


Obrázek 5.13: Tabulka databáze

Uživateli jsou při přidávání nové revize nabízeny nejběžnější prováděné revize. Uživatel musí zadat jen datum poslední revize. Aplikace už sama nastaví datum plánované revize. Většina revizí má datum plánované revize nastaveno na rok, některé na tři nebo pět let. Některé revize však mají plánovanou revizi po čtvrt roce. Poté, co uživatel přidá novou nebo upraví stávající revizi, je opět nastavena notifikace na plánované datum. Uživatel je upozorněn týden dopředu před plánovanou revizí.

Nachází se zde také záložka revizní technici, kde uživatel má seznam revizních techniků na právě konkrétní revize, které vybírá z nabízeného seznamu při přidávání nebo editaci typu revize. Na této záložce může uživatel vyhledávat techniky podle revizní oblasti nebo města.

Na obrázku 5.14 jsou zobrazeny přehledy investic, záložka s kontakty na revizní techniky a dále screenshot zamykací obrazovky s upozorněním na plánovanou revizi příští týden.



Obrázek 5.14: Přehled revizí v nemovitosti, revizních techniků a upozornění na revizi v zamykací obrazovce

Seznam kontaktů je umístěn v souboru `kontakty.json` ve složce `/files`. Konkrétní položky jsou z tohoto souboru zpracovávány pomocí funkce `$http` v AngularJS, která je pro toto využití vhodná.

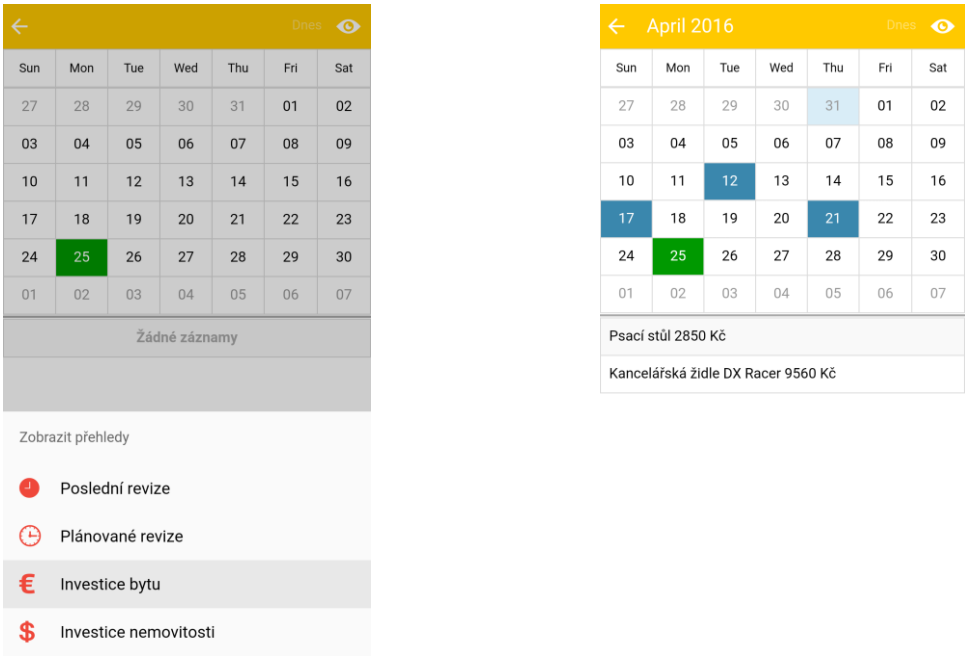
Kontakty, včetně typů běžně prováděných revizí a jejich data další pravidelné kontroly, mi byly poskytnuty a jedná se o reálné osoby či firmy zabývající se revizemi či servisními prohlídkami. Zdroj, který mi poskytnul tyto informace, si však přeje být neuveden.

5.6 Kalendář

Poslední funkcí informačního systému je kalendář s přehledy. V pravém horním rohu jsou umístěny dvě ikony. První je *„Dnes“*, která vrátí kalendář do aktuálního dne. Druhá ikona zobrazí okno pro výběr přehledů do kalendáře. Jakmile uživatel zvolí konkrétní přehled, načtou se data do kalendáře. Listování kalendářem probíhá standardním způsobem zleva doprava a naopak. Aktuální den nebo vybraný den je znázorněn zeleně.

Po kliknutí na konkrétní den, ve kterém je nějaká událost, jsou uživateli tyto události vypsány pod kalendář. Události v kalendáři jsou znázorněny tmavě modrou barvou. Události světle modrou barvou značí událost v jiném měsíci.

Na obrázku 5.15 je zobrazen příklad, který zobrazuje přehled investic bytu v kalendáři pro měsíc duben.



Obrázek 5.15: Načtení investic bytu do kalendáře

6 Testování

Instalace a stažení Cordovy a Ionic frameworku proběhla nejprve za pomocí konzolového nástroje `npm`. Následně po instalaci a vytvoření struktury byla aplikace otevřena a spuštěna jako nový projekt ve vývojovém prostředí Microsoft VisualStudio 2015, kde dále probíhal vývoj aplikace.

První část testování probíhala při vývoji, kdy byly postupně při implementaci kontrolovány části chování systému na zadávané vstupy. Výhodou VisualStudia byla možnost testovat aplikaci v simulátoru Apache Ripple simulator[24]. Tento simulátor je spuštěn v prohlížeči Google Chrome a zastupuje připojený telefon. Další výhodou tohoto simulátoru je fakt, že každá změna v kódu se ihned projeví v simulátoru a nemusí se znova celá aplikace zdlouhavě do zařízení instalovat. Simulátor umožňuje spuštění aplikace na virtuálním mobilním telefonu, který může být jak Android, tak i iOS. Na výběr je několik typů telefonů s různými verzemi operačního systému. Testování probíhalo na obou platformách virtuálního mobilního telefonu.

Vývojové prostředí VisualStudia také nabízí kvalitní debugger[25]. Aplikace byla vždy v místech pro ukládání, mazání nebo čtení z databáze pozastavena a kontrolována, zda pracuje s validními daty.

Nevýhodou simulátoru je, že obsahuje jen základní funkce telefonu, a to pouze pro geolokaci a orientaci zařízení. Aplikace byla tedy v pozdějších fázích vývoje testována na reálném zařízení Sony Xperia Z2 s operačním systémem Android 5.1.1, na které se potom mohly testovat i funkce telefonu jako notifikace nebo fotoaparát. Aplikace byla také díky debuggovacímu nástroji VisualStudia postupně debuggována i pro připojený telefon. Postupně byly ověřovány všechny možné stavy systému, jak systém reaguje při zadávání extrémních či nevalidních hodnot.

Dalšími testovanými telefony byly Lenovo K3 Note s operačním systémem Android 6.0.0 a Sony Xperia M. Testování probíhalo na těchto zařízeních stejně.

Posledním testovaným zařízením bylo zařízení Apple iPad s verzí sytému 9.2.1. Zde však nastal problém při instalaci aplikace. Protože aby mohla být aplikace nahrána na reálné zařízení se systémem iOS, je k tomu potřeba nahrát aplikaci z Mac počítače a navíc být členem Apple developer programu. Školní Mac počítač jsem z těchto důvodu a omezených instalačních práv nemohl použít. Aplikace tedy byla nahrána za použití firemního Mac počítače do firemního iPadu. Aplikace byla úspěšně nahrána a spuštěna bez problémů. Firemní iPad jsem však nemohl otestovat stejným způsobem jako na telefonech se systémem Android, protože jsem iPad nemohl mít k dispozici po celou dobu vývoje.

[24] https://taco.visualstudio.com/en-us/docs/run-app-ripple-simulator/
[25] Softwarový nástroj, který se používá pro hledání chyb při vývoji software

Poslední částí byla aplikace testována na 3 uživatelích, kteří se s aplikací ještě nesetkali. Bylo jim pouze vysvětleno, na co aplikace bude sloužit. Všichni aktéři neměli s orientací v aplikaci a jejím používání žádný vážnější problém. Výtka byla na volbu aktivní nemovitosti a bytu, kdy je změna možná pouze v možnosti *„Moje nemovitost“*. Očekávali by výběr nemovitosti a bytu i v konkrétních přehledech investic apod. S přidáváním položek, editací, mazáním nebo zobrazením detailního náhledu neměli problém. V detailu investice nemovitosti si někteří nevšimli možnosti pro pořizování fotografií a také v kalendáři v pravém horním rohu ikonky pro zobrazení přehledů do kalendáře.

7 Závěr

Cílem této práce bylo vytvořit informační systém pro společnost GetONE s.r.o, který bude spravovat nemovitosti obdobně jako servisní knížka.

Výsledkem je mobilní hybridní aplikace Moje Nemovitost. Jedná se o webovou aplikaci využívající HTML5, CSS a AngularJS. Aplikace je následně zabalena pomocí nástroje Apache Cordova pro konkrétní mobilní platformu. Vzhled nativní aplikace je dosažen díky použití Ionic frameworku.

Uživatel v aplikaci může spravovat více nemovitostí, systém tyto nemovitosti eviduje včetně jejich bytů, které k ní přísluší. Dále systém eviduje výdaje a investice v konkrétní nemovitosti a v konkrétním bytu podle toho, jaký aktivní byt a nemovitost uživatel zvolí a chce u nich zobrazovat či spravovat přehledy. Systém dále umožňuje ke konkrétním investicím pořizovat fotografie. Pořízené fotografie najdou uplatnění například při rekonstrukcích nemovitosti. Systém uživateli zobrazuje přehledy investic buď za vybraný rok, nebo za konkrétní měsíc v konkrétním roce. Nedílnou součástí systému je také správa revizí. Podle typu revize systém automaticky nastaví upozornění na týden dopředu před plánovanou revizí. Systém také obsahuje kontakty na revizní techniky pro dané revizní oblasti. Umožňuje vyhledávání těchto kontaktů buď podle revizní oblasti, nebo podle města. Obdobou je i správa spotřebičů, která také upozorňuje na plánované údržby či servisy konkrétního spotřebiče. Správa spotřebičů nejen eviduje spotřebiče samotné včetně jejich záruk, ale umožňuje i u konkrétního spotřebiče vedení dodatečné evidence oprav, servisů či údržeb včetně vynaložených investic k tomuto spotřebiči. Uživatel má tedy přehled výdajů ke konkrétnímu spotřebiči. Tyto informace mu mohou pomoci při rozhodování, zda se ještě vyplatí investovat do tohoto spotřebiče nebo raději koupit nový. Informační systém dále obsahuje kalendář, který přehledně zobrazuje výdaje nemovitosti, bytu nebo plánované a proběhlé revize v kalendáři.

Jedním z možných vylepšení aplikace je správa nájemníků, kdy uživatel vlastnící bytový dům bude mít přehled o svých nájemnících včetně pravidelných plateb za nájem. Dalším vylepšením může být synchronizace s pravidelnými platbami například za elektřinu, plyn, vodu, telefon, internet apod. Dále je v plánu realizovat cloudové úložiště, ve kterém budou uložena všechna data, ke kterým by mohl uživatel přistupovat odkudkoliv prostřednictvím internetu. Uživatel by měl vytvořen účet, pomocí kterého by se přihlašoval. Aplikace by už však vyžadovala trvalé připojení k internetu.

Literatura

[1] Zakas, N.Z.: *JavaScript pro webové vývojáře - Programujeme profesionálně*, Computer Press, 2009

[2] Naramore, E. et al.: *PHP5, MySQL, Apache: Vytváříme webové aplikace*, Computer Press, 2009

[3] Hruška, T.; Křivka Z.: *Informační systémy IIS: Studijní opora*, FIT VUT v Brně, 2012. Dostupné z: https://www.fit.vutbr.cz/study/courses/WAP/private/opory/OporaIIS2PISPojemDataProcesyTransakce.pdf

[4] Budiu, R.: *Mobile: Native Apps, Web Apps, and Hybrid Apps*. Nielsen Norman Group. [online] 14. 9 2013. Dostupné z: http://www.nngroup.com/articles/mobile-nativeapps/

[5] Saccomani, P.: *Native, Web or Hybrid Apps? What's The Difference?*. [online] [cit. 2016-04-10]. Dostupné z: http://www.mobiloud.com/blog/2012/06/native-web-or-hybrid-apps/

[6] Xamarin Inc.: *Xamarin Platform*. [online]. 2016. Dostupné z: https://www.xamarin.com/platform

[7] Webová aplikace. In: *Wikipedia: the free encyclopedia* [online]. 14. 2. 2016 [cit. 2016-04-15] Dostupné z: https://cs.wikipedia.org/wiki/Webov%C3%A1_aplikace

[8] Model – View – Controller. *MSDN: the microsoft developer network* [online]. 2016 [cit. 2016-04-12]. Dostupné z: https://msdn.microsoft.com/en-us/library/ff649643.aspx

[9] Čápka, D.: *Návrhové vzory - MVC architektura* [online][cit. 2016-04-12]. Dostupné z: http://www.itnetwork.cz/navrhove-vzory/mvc-architektura-navrhovy-vzor/

[10] Bernard, B.: *Úvod do architektury MVC* [online]. 7. 5. 2009 [cit. 2016-04-12]. Dostupné z: https://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[11] HyperText Markup Language. In: *Wikipedia: the free encyclopedia* [online]. 14. 2. 2016 [cit. 2016-04-15] Dostupné z: https://cs.wikipedia.org/wiki/HyperText_Markup_Language

[12] Hanák, D.: *Úvod do AngularJS* [online]. 7. 8. 2012 [cit. 2016-04-17]. Dostupné z: http://www.itnetwork.cz/javascript/angularjs/javascript-tutorial-uvod-do-angularjs

[13] Mrozek, J.: *Začínáme s AngularJS* [online]. 30. 11. 2012 [cit. 2016-04-17]. Dostupné z: https://www.zdrojak.cz/clanky/zaciname-s-angularjs/

[14] Step 2 -Angular Templates. In: *AngularJS Documentation* [online]. 2016 [cit. 2016-04-17] Dostupné z: https://docs.angularjs.org/tutorial/step_02

[15] AngularJS - Services. In: *Tutorialspoint* [online]. 2016 [cit. 2016-04-17] Dostupné z: http://www.tutorialspoint.com/angularjs/angularjs_services.htm

[16] Adobe Announces Agreement to Acquire Nitobi, Creator of PhoneGap. In: *Adobe Systems Incorporated* [online]. Los Angeles 3. 9. 2011 [cit. 2016-04-19] Dostupné z: http://www.adobe.com/aboutadobe/pressroom/pressreleases/201110/AdobeAcquiresNitobi.html

[17] Introduction. In: *Apache Cordova documentation* [online]. 2016 [cit. 2016-04-19] Dostupné z: https://cordova.apache.org/docs/en/latest/guide/overview/

[18] Platform Support. In: *Apache Cordova documentation* [online]. 2016 [cit. 2016-04-19] Dostupné z: https://cordova.apache.org/docs/en/latest/guide/support/index.html

[19] Hujer, M.: *Cordova a Sencha Touch aneb mobilní aplikace pomocí webových technologií* [online]. 15. 1. 2014 [cit. 2016-04-19]. Dostupné z: https://www.zdrojak.cz/clanky/cordova-sencha-touch-mobilni-aplikace/

[20] Václavík, J.: *Vyvíjíme hybridní aplikace v Ionicu: Úvod a instalace* [online]. 20. 7. 2015 [cit. 2016-04-20]. Dostupné z: https://www.zdrojak.cz/clanky/vyvijime-hybridni-aplikace-v-ionicu/

[21] Ionic Documentation.: *Start building with Ionic!* [online]. 2016 Dostupné z: http://ionicframework.com/docs/

[22] Zendulka, J.; Rudolfová, I.: Databázové systémy IDS: Studijní opora. 2006.

[23] Pospíchal, V.: *SQLite: Databáze pro váš web* [online]. 2. 9. 2013 [cit. 2016-04-21]. Dostupné z: https://www.zdrojak.cz/clanky/sqlite-databaze-pro-vas-web/

[24] SQLite – Data Type. In: *Tutorialspoint* [online]. 2016 [cit. 2016-04-25] Dostupné z: http://www.tutorialspoint.com/sqlite/sqlite_data_types.htm

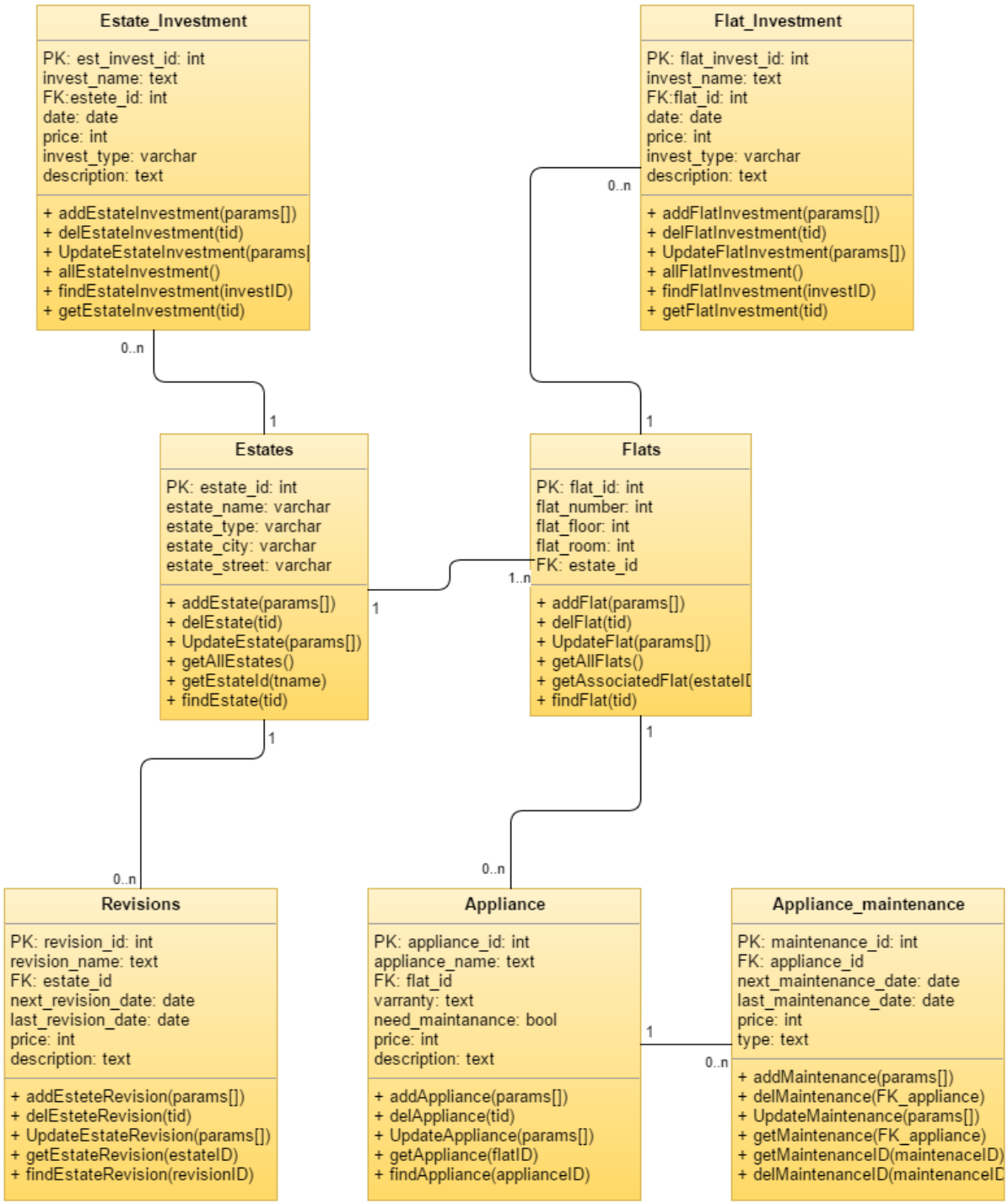
[25] SQLite [online]. 2000 [cit. 2016-04-25]. Dostupné z: https://www.sqlite.org/

Seznam příloh

Příloha 1: Struktura adresáře www

```
└── www               # Hlavní adresář webové aplikace (zobrazené ve WebView)
    ├── index.html
    ├── css
    ├── img
    ├── files
    │   └── kontakty.json
    ├── js            # Adresář s logikou aplikace
    │   ├── app.js
    │   ├── controllers.js
    │   ├── services.js
    │   ├── appliance.js
    │   ├── revision.js
    │   ├── notification.js
    │   └── calendar.js
    ├── lib               # Adresář s externími JS knihovnami a pluginy
    └── templates         # Adresář se šablonami v aplikaci
        ├── appliance-detail.html
        ├── appliance.html
        ├── calendar.html
        ├── contacts.html
        ├── detail-revision.html
        ├── estate-investment.html
        ├── estate-investment-detail.html
        ├── estate-investment-summary.html
        ├── flat-investment.html
        ├── flat-investment-detail.html
        ├── flat-investment-summary.html
        ├── homepage.html
        ├── menu.html
        ├── my-estate.html
        ├── my-estate-detail.html
        ├── my-flat-detail.html
        └── revisions.html
```

Příloha 2: ER diagram



Příloha 3. Obsah CD.

/src/ – Složka obsahující zdrojové soubory (složku www)

/doc/ – Složka obsahující tuto zprávu ve formátu .docx a .pdf

readme.txt – Soubor s manuálem pro instalaci a spuštění