VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE PRO VIZUALIZACI A ŘÍZENÍ VODOHOSPODÁŘSKÝCH STAVEB

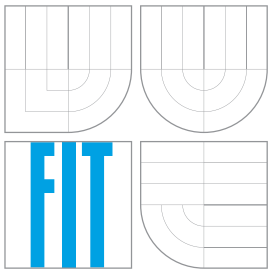
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN ŠVANCER

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE PRO VIZUALIZACI A ŘÍZENÍ VODOHOSPODÁŘSKÝCH STAVEB

MOBILE APPLICATION FOR VISUALIZATION AND CONTROL

OF WATER MANAGEMENT STRUCTURES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN ŠVANCER
AUTHOR

VEDOUCÍ PRÁCE — Ing. VÍTĚZSLAV BERAN, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Cílem práce je navrhnout a implementovat způsob univerzálního zobrazení dat z automatizovaných vodárenských objektů na mobilním zařízení. Vývoj je realizován spolu s firmou Speco Control s.r.o., která dané automatizace provádí. Výsledná aplikace je otestována a nasazena do reálného provozu.
Stěžejní částí je návrh univerzální struktury databáze, na základě níž je možné data zobrazit. V současné době se totiž ke každému objektu vyvíjí na míru řešený dispečerský program pro PC.

Abstract

The aim of this thesis is to design and implement a method of the universal displaying of data from automated water management structures on mobile devices. The development is realized along with the company Speco Control s.r.o., which conducts these automations. The final application is tested and used in actual operation.
The crucial part is the draft of the universal structure of database, thanks to which it is possible to display the data. Currently a made-to-measure dispatcher's program for PC is being develop for every object.

Klíčová slova

Mobilní aplikace, iOS, swift, uživatelské rozhraní, GUI, vizualizace, automatizace

Keywords

Mobile application, iOS, swift, user interface, GUI, visualization, automatization

Citace

Jan Švancer: Mobilní aplikace pro vizualizaci a řízení
vodohospodářských staveb, bakalářská práce, Brno, FIT VUT v Brně, 2015

Mobilní aplikace pro vizualizaci a řízení vodohospodářských staveb

Prohlášení

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

.....
Jan Švancer
12. května 2015

Poděkování

Děkuji vedoucímu práce Ing. Vítězslavu Beranovi, Ph.D., který mi umožnil pod jeho vedením tuto práci realizovat a poskytl mi odbornou pomoc. Děkuji firmě Speco Control s.r.o. za spolupráci a speciálně mému otci za poskytnutí odborných rad. V neposlední řadě děkuji svojí přítelkyni za pomoc s grafickým designem.

Obsah

1 Úvod **2**

2 Vývoj iOS aplikací a proces automatizace v zadávající firmě **3**
2.1 Uživatelské rozhraní 3
2.2 Úvod do iOS 6
2.3 Webové API 8
2.4 Princip automatizace a práce s daty v zadávající firmě 9
2.5 Vizualizační program pro PC 10

3 Vizualizace dat na mobilním zařízení **12**
3.1 Analýza a cíl práce 12
3.2 Návrh datového modelu 13
3.3 Analýza cílové skupiny uživatelů 16
3.4 Návrh uživatelského rozhraní 16
3.5 Komunikace se serverem 19

4 Mobilní aplikace pro vizualizaci a řízení vodohospodářských staveb **20**
4.1 Implementační prostředky 20
4.2 Implementace uživatelského rozhraní 20
4.3 Komunikace a práce s daty 27
4.4 Back-end aplikace 28

5 Testování **30**
5.1 Prvotní testování uživatelského rozhraní 30
5.2 Závěrečné testování 31
5.3 Výsledek testování 32

6 Závěr **33**

A Obsah CD **35**

B API manuál **36**

C Dotazník k testování **38**

Kapitola 1

Úvod

V dnešní moderní době, kdy již většina našeho průmyslu byla mechanizována, vyvstává do popředí nový trend a tím je průmyslová automatizace. Spousta procesů je automatizována a ustává tak potřeba velkého množství lidí na provoz. Automatizací je také dosaženo vyšší spolehlivosti a přesnosti. Právě takovou automatizací, především vodohospodářských objektů jako jsou např. ČOV, jezy, hráze, čerpací stanice, vodojemy a úpravny vod, se zabývá firma Speco Control s.r.o. ze Zlína. Ve spolupráci s touto firmou je práce vyvíjena.

Firma ke každému automatizovanému objektu dodává na míru vytvořený software pro PC, kde je objekt vizualizován, tedy rozkreslen na jednotlivé části tak, jak se fyzicky v objektu nachází a v nich jsou graficky zobrazeny veškeré zařízení a propoje. Díky tomuto softwaru je možné celý objekt nejen monitorovat ale také řídit a to dokonce vzdáleně z jakéhokoliv počítače s příslušným dispečerským programem a připojením k internetu. Program dále nabízí hlášení poruch, analytické informace a další věci spojené s chodem objektu. V dnešní době jsou nabízeny vizualizace také ve 3D.

Tato práce se zabývá návrhem a tvorbou mobilní aplikace, která by dokázala univerzálním způsobem zobrazit data z automatizovaných objektů. Aplikace nemá nahrazovat současné řešení pro PC, ale určitým způsobem jej doplnit, aby bylo možné kdykoliv a kdekoliv zjistit nejpodstatnější informace. Návrh univerzálního způsobu reprezentace dat a vytvořený back-end bude v budoucnu využit pro vytvoření webového rozhraní.

Celý text je členěn na kapitoly popisující jednotlivé části vývoje. Nejprve jsou teoreticky popsány principy tvorby aplikací pro mobilní platformu iOS, princip webové API, zásady návrhu uživatelského rozhraní a jeho testování. Také je zde popsáno, co se vlastně skrývá pod pojmem automatizace a jak tento proces funguje v zadávající firmě (2). V další kapitole (3) se nachází analýza problému, návrh univerzální databáze, návrh z toho vycházejícího uživatelského rozhraní a způsob komunikace s centrálním serverem. Poté je popsána implementace (4) navrženého rozhraní a serverové části API. Předposlední kapitola (5) se zabývá testováním vytvořené aplikace a na závěr (6) jsou v poslední kapitole shrnuty výsledky a přínos této práce pro zadávající firmu.

Kapitola 2

Vývoj iOS aplikací a proces automatizace v zadávající firmě

Tato kapitola je rozdělena na pět části popisující teoretické informace nutné pro návrh a implementaci výsledné aplikace.

2.1 Uživatelské rozhraní

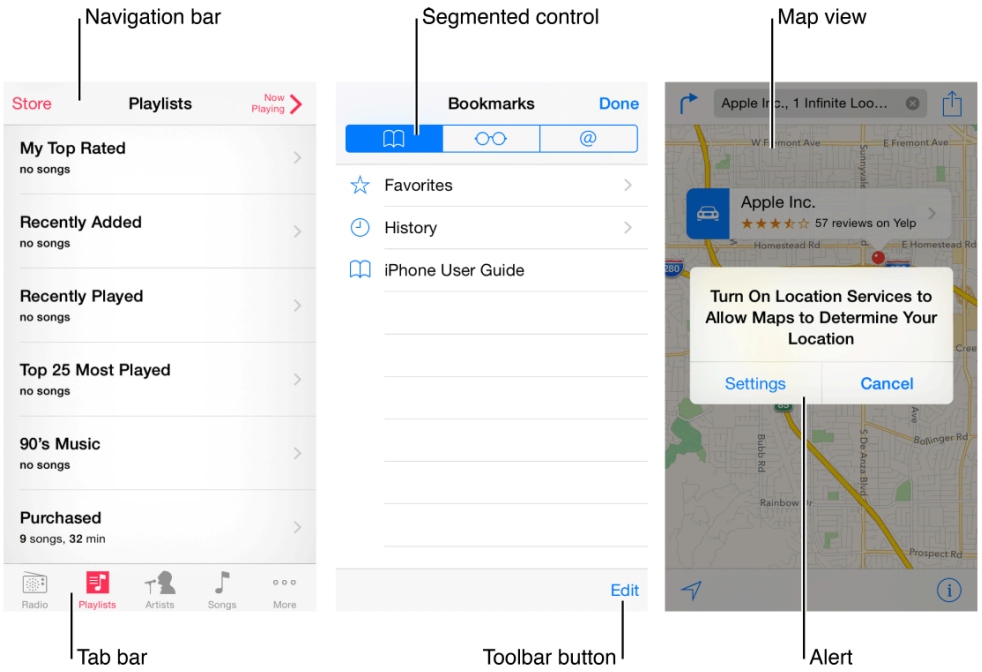
Uživatelské rozhraní můžeme chápat jako souhrn způsobů, jakými uživatelé ovlivňují chování systému. Systémem v tomto případě budeme předpokládat počítačový program či aplikaci. Návrh uživatelského rozhraní je pro výsledný úspěch klíčový. Pokud uživatel není schopen jednoduché a jasné interakce s aplikací, pravděpodobně se poohlédne po jiné alternativě, i kdyby aplikace jinak splňovala všechny požadavky. Uživatelské rozhraní je totiž právě ten poslední a mnohdy jediný článek, který uživatel vidí a se kterým pracuje.

Rozhraní je závislé na platformě, pro kterou je vyvýjeno. Tato práce se zabývá aplikací pro platformu iOS. Pro pochopení a návrh uživatelských rozhraní právě pro tento systém, vydal Apple publikaci s názvem *iOS Human Interface Guidelines* [4], která popisuje, jakým způsobem tvořit texty, ikonky a další prvky aplikace. Je zde mimo jiné popsán způsob, kterým přemýšlí uživatelé a proč některé aplikace mají úspěch už jen díky svému vzhledu.

Pravidla návrhu uživatelského rozhraní pro iOS

Níže uvedené zásady dobrého návrhu vychází mimo jiné z příručky *iOS Human Interface Guidelines*.

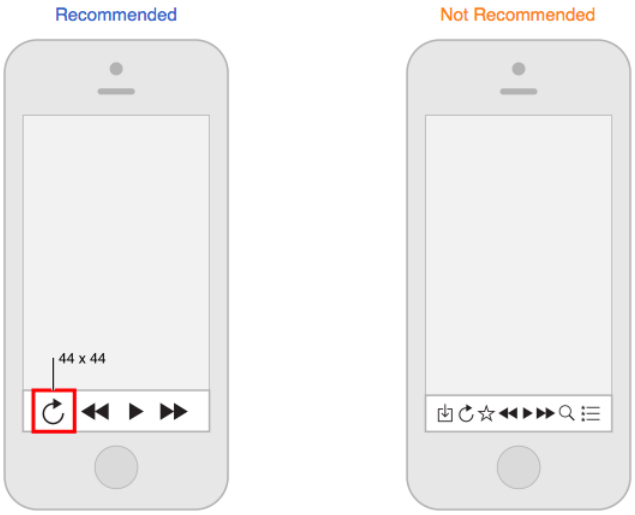
- **Integrita**- vzhled aplikace by měl odpovídat její funkčnosti,
- **zpětná vazba**- uživatel by měl mít vždy zpětnou vazbu o prováděných operacích, především o těch déle trvajících,
- **konzistence**- ovládací prvky by měli splňovat zažité vzory dané platformy,
- **jasnost**- text by měl být čitelný v každé velikosti, ikonky by měli být přesné a jasné a ozdoby pouze jemné a vhodné,
- **barvy**- každá barva má svůj význam, v iOS to můžeme vidět napříč všemi základními aplikacemi,



Obrázek 2.1: Některé základní prvky iOS (zdroj Apple.com)

- **využívejte prostor celé obrazovky,**
- **důležité prvky v horní polovině obrazovky,**
- **porozumění**- ujistěte se, že uživatel rozumí obsahu na první pohled, tzn. bez nutnosti posunutí obrazovky (scrollování),
- **změna velikosti písma**- design musí být uzpůsobem tak, aby byl schopen reagovat na změnu základní velikosti písma,
- **přehlednost**- využívejte volný prostor pro zajištění přehlednosti aplikace,
- **univerzální layout**- rozhraní aplikace by mělo být univerzální pro jakoukoliv velikost displeje,
- **možnost návratu**- uživatel by měl mít jednoduchou možnost návratu zpět v navigaci aplikace.

Pro iOS aplikace je typické použití tzv. *Navigation bar*, jak můžeme vidět na obrázku 2.1 vlevo. Jedná se vlastně o horní lištu, která uchovává titulek aktuálně zobrazované stránky, zpravidla ikonku pro návrat zpět v navigaci a případně další tlačítka. Dalším typickým prvkem je *Tab bar*, což je pro změnu dolní lišta, umožňující přepínání mezi jednotlivými částmi aplikace.



Obrázek 2.2: Ukázka návrhu velikosti ikonek, využití volného prostoru (zdroj Apple.com)

Testování uživatelského rozhraní

Testování uživatelského rozhraní je velice důležitou a nedílnou součástí vývoje aplikace. Aby byl tento postup efektivní, je vhodné se držet jistých zásad a testovat průběžně během samotného vývoje. Je mnohem méně nákladné předělat jednotlivé návrhy, než finální produkt. Tyto zásady a rady vycházejí ze článku Adama Fendrycha *Uživatelské testování návrhů webu* [2] na webovém serveru `lupa.cz`.

Existuje několik možných návrhů rozhraní, které lze rozdělit do tří hlavních kategorií:

- Náčrtek na papíře,
- wireframe,
- prototyp aplikace.

Náčrtek na papíře je velmi rychlá varianta. Stačí nakreslit na papír svou hrubou představu o výsledné podobě aplikace, nebo její části. Detaily zde nejsou tak důležité a také nejde o to, aby náčrtek odrážel podobu aplikace.
Výhoda: velmi jednoduché na vytvoření.
Nevýhoda: horší možnosti úprav. Většinou je třeba celý návrh nakreslit znovu.

Wireframe, neboli česky „drátěný model“ stránek aplikace, vytvořený v počítačovém editoru. Testování je v principu velmi podobné papírkovým náčrtkům. Wireframy jsou graficky uhlazenější a uživatel je vidí na displeji.
Výhody: velmi snadné provádění úprav. Možnost šířit elektronicky, kopírovat a tisknout.
Nevýhody: stále se jedná o oddělené stránky, které je třeba ukazovat uživatelům jednu po druhé.

Prototyp aplikace je jednoduchý prototyp výsledné aplikace, zaměřený pouze na funkčnost uživatelského rozhraní. Je zde možné simulovat přechody mezi stránkami a interakci jednotlivých tlačítek. Testování v této fázi probíhá formou zadávání úkolů uživateli a sledováním jeho chování.

Pro testování je potřeba vybrat vhodnou množinu potencionálních uživatelů. Pokud například vyvíjíme specializovanou aplikaci pro letectví, tak ji nebudeme testovat na náhodných kolemjdoucích, ale třeba na pilotech, kteří ví, o co se jedná. V případě, že vytváříme aplikaci obecnějšího charakteru, je naopak vhodné zahrnout do testování větší okruh různých uživatelů.

Před samotným zahájením testování je mimo jiné důležité si ujasnit, čeho chceme vlastně dosáhnout, co chceme testováním zjistit? Na tomto základě se pak sestaví otázky, které jsou uživatelům předloženy. Vývojář, který test provádí, by měl bedlivě sledovat chování uživatele a zapisovat si jeho výroky. Řeč těla může totiž prozradit víc, než samotné odpovědi testovaného.

2.2 Úvod do iOS

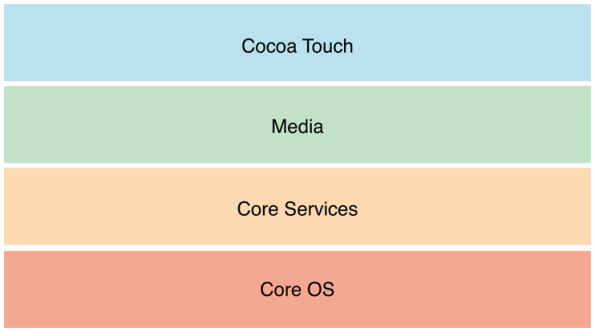
Jelikož výsledná aplikace bude implementována na mobilní platformu iOS, je vhodné si ji stručně představit a popsat principy vývoje právě pro tento mobilní systém.

iOS je mobilní operační systém vyvinutý společností Apple Inc. První verze tohoto systému byla představena v roce 2007, spolu s prvním telefonem iPhone, který v podstatě představoval revoluci v mobilních zařízeních. Systém byl původně určen pouze pro mobilní telefony iPhone a přehrávače iPod Touch, postupem času se však rozšířil i na další Apple zařízení, především tablet iPad. První verze systému nesla pojmenování *iPhone OS*, ale od roku 2010 se přešlo na dnes již známý název iOS. V době vývoje této práce existuje již iOS 8, kdy od verze 7 prodělal systém značné grafické změny, především přechod na flat design a celkově čistější vzhled.

Architektura iOS

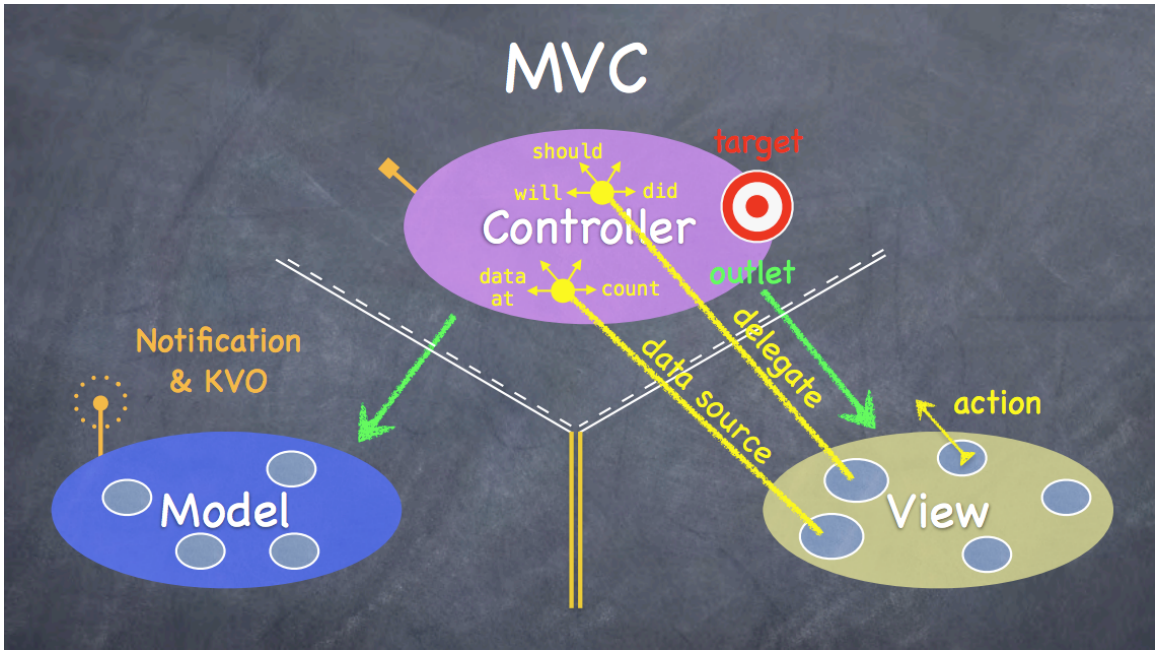
Operační systém iOS je systém UNIXového typy. Jedná se o odlehčenou verzi operačního systému Mac OS X, používaného v počítačích Apple. Jelikož je určen pro mobilní zařízení, neobsahuje veškerou funkcionalitu OS X, na druhou stranu ale přidává podporu dotykového ovládání.

Systém se dělí na čtyři základní vrstvy, které zajišťují funkčnost a poskytují vývojářům API a frameworky potřebné k vývoji aplikací. Jejich uspořádání lze vidět na obrázku 2.3.



Obrázek 2.3: Vrstvy iOS (zdroj Apple.com)

Vrstva Cocoa Touch – Jedná se o nejvyšší vrstvu v hierarchii, která obsahuje nejdůležitější frameworky pro vývoj aplikací. Tyto technologie poskytují infrastrukturu pro implementaci grafického rozhraní, interakci s uživatelem a vysoko úrovňové systémové služby jako



Obrázek 2.4: MVC v iOS [11]

např. multitasking, push notifikace, rozpoznávání gest a mnoho dalších. Při vývoji aplikace je vhodné začít právě s touto vrstvou a nižší vrstvy používat pouze v případě potřeby.

Vrstva Media – Tato vrstva obsahuje technologie pro audio a video, které lze v aplikacích využít. Tyto technologie umožňují plynulé přehrávání animací, videí a zvuků.

Vrstva Core Services – Vrstva obsahuje základní systémové služby pro všechny aplikace. Patří zde například služby pro práci s daty, lokalizační služby, databázové prostředky, služby pro multivláknové programování apod.

Vrstva Core OS – Jedná se o poslední vrstvu poskytující nízkoúrovňové funkce, na kterých jsou postaveny technologie vyšších vrstev. Většinou nebývá v aplikacích využívána přímo. Nachází se zde například prostředky pro práci s matematickými funkcemi, výpočty DSP, pro komunikaci s externími zařízeními, atd. [3]

Vývoj pro platformu iOS

K vývoji aplikací pro Apple (iOS, Mac OS X) je nutné stát se nejprve oficiálním vývojářem. K tomu je potřeba zakoupit developerskou licenci, bez níž není možné aplikace testovat na reálném zařízení a distribuovat prostřednictví obchodu App Store. K zakoupení licence a vůbec práci s Apple zařízeními, je potřeba mít své vlastní *Apple ID*, které slouží jako identifikace pro přístup ke službám společnosti Apple Inc.

Pro samotný vývoj jsou dále potřeba tři věci:

- Mac počítač se systémem OS X 10.8 (Mountain Lion) či novější,
- vývojové prostředí Xcode,
- iOS SDK. [1]

[1] Software Development Kit

Primárním vývojovým prostředím je program **Xcode**, který obsahuje editor zdrojových kódů, simulátor, interface builder [2], nástroje pro testování a ladění.

Vývoj aplikací je možný v jazyce **Objective-C** a **Swift**, který byl představen na WWDC v červnu 2014. Objective-C je objektově orientovaný jazyk, postavený na jazyce C, do kterého byl přidán systém zasílání zpráv z jazyka Smalltalk. Swift, který je zamýšlen jako alternativa k Objective-C, je také objektově orientovaný jazyk a umí pracovat s existujícími frameworky Cocoa a Cocoa Touch. Swift je z větší části obdobou Objective-C, ovšem za využití moderních konceptů a syntaxe. Byl zde například nahrazen Smalltalkový způsob volání metod za tečkovou notaci, hlavičové soubory (.h) nejsou vyžadovány, příkazy nemusí být ukončeny středníkem a další. Swift zachovává všechny klíčové vlastnosti Objective-C, často při zjednodušení syntaxe a neměl by tak dovolit tolik chyb programátora. Syntaxe jazyka je popsána v oficiální publikaci *The Swift Programming Language* [5].

Vývoj aplikací využívá koncepce **MVC**, Model View Controller. Jedná se o softwarovou architekturu, která rozděluje datový model aplikace, uživatelské rozhraní a řídící logiku do tří nezávislých komponent tak, že modifikace některé z nich má jen minimální vliv na ostatní. Model v architektuře iOS tvoří zdroje dat a operace s těmito daty. View převádí data reprezentovaná modelem do podoby vhodné k interaktivní prezentaci uživateli. View je nejčastěji vytvářen pomocí Storyboardu, což je prostředek programu Xcode pro grafický návrh rozhraní. Všechny komponenty lze vytvořit také programově, ale grafická reprezentace je pro vývoj jednodušší a přehlednější. Controller je vždy spojen s View a reaguje na události od uživatele. Následně provádí změny v Modelu nebo ve View. Část View a Model by spolu nikdy neměli přímo komunikovat [10]. Princip MVC v iOS aplikacích lze vidět na obrázku 2.4.

2.3 Webové API

Součástí této práce je vytvořit také back-end k výsledné aplikaci v podobě API s vhodnou databázovou strukturou. Pro napsání API je potřeba si nastudovat princip její činnosti a typický způsob práce s daty.

API označuje v informatice rozhraní pro programování aplikací. Jde o sbírku procedur, funkcí, tříd či protokolů nějaké knihovny, které může programátor využívat [13].

Web API představuje webovou službu, která poskytuje určité prostředky. Obvykle je založena na architektuře REST, což je cesta jak jednoduše vytvořit, číst, editovat nebo smazat informace ze serveru pomocí jednoduchých HTTP volání. Web API nejčastěji poskytuje odpovědi v XML nebo JSON formátu [12]. Parametry dotazu mohou být předány různými způsoby. Nejčastěji je to pomocí metod POST a GET. Jak správně navrhnout webové API, se lze dočíst v elektronické knize *Web API Design* [9].

Formát JSON je jednoduchý textový formát pro výměnu dat. Jeho výhodou je snadná čitelnost i zapisovatelnost člověkem a je také lehce analyzovatelný i generovatelný strojově. Způsob zápisu je nezávislý na počítačové platformě.

JSON je založen na dvou strukturách:

- Záznam- kolekce párů název/hodnota ({název : hodnota}). Páry jsou odděleny znakem , (čárka).
- Pole- seřazený seznam hodnot ([hodnota]). Hodnoty jsou opět odděleny znakem , (čárka) [1].

[2] nástroj pro grafický návrh uživatelského rozhraní

2.4 Princip automatizace a práce s daty v zadávající firmě

Kromě samotných nástrojů pro tvorbu této práce je důležité se podívat, co je to vlastně automatizace, jelikož tato aplikace má poskytovat nějaký vizualizační prostředek, který by dokázal zobrazit měřené hodnoty na automatizovaných objektech. Dále je potřeba pochopit celý systém komunikace, jaký firma, se kterou je tato práce vyvýjena, v současné době používá. A jakým způsobem jsou ukládány naměřená data.

Automatizací nebo automatickým řízením rozumíme použití různých řídících systémů pro provoz zařízení, jako jsou stroje, procesy v továrnách, výrobní linky apod. s minimálním zásahem člověka. Některé procesy již byly zcela automatizovány.

Největší výhodou automatizace je, že šetří práci, ale vede i k úsporám energie, materiálů a zlepšuje kvalitu, přesnost a preciznost.

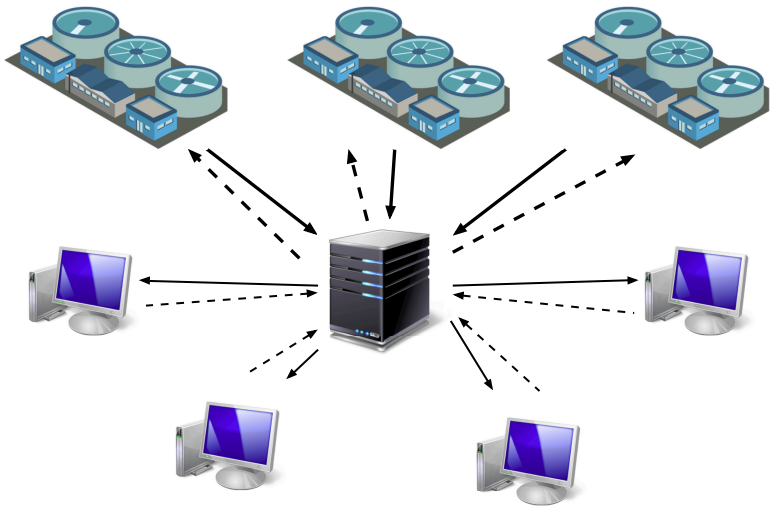
Firma Speco control s.r.o. se zabývá automatizací, řídícími systémy a dispečinky více než 10 let. Specializují se především na automatizace vodárenských objektů jako jsou ČOV, jezy, hráze, vodojemy, úpravny pitné vody a další.

Při procesu automatizace jsou do automatizovaného objektu nainstalovány veškeré potřebné elektrické zařízení. Patří sem například různé měřící sondy, výškoměry, digitální teploměry a čidla průsaku. Všechny tyto zařízení jsou řízeny přes PLC [3]. PLC je relativně malý průmyslový počítač používaný právě pro automatizaci procesů v reálném čase. PLC je charakteristický tím, že program se vykonává v tzv. cyklech. Převážnou část periferií tvoří digitální vstupy a výstupy a pro napojení na technologie jsou určeny analogové vstupy a výstupy [7].

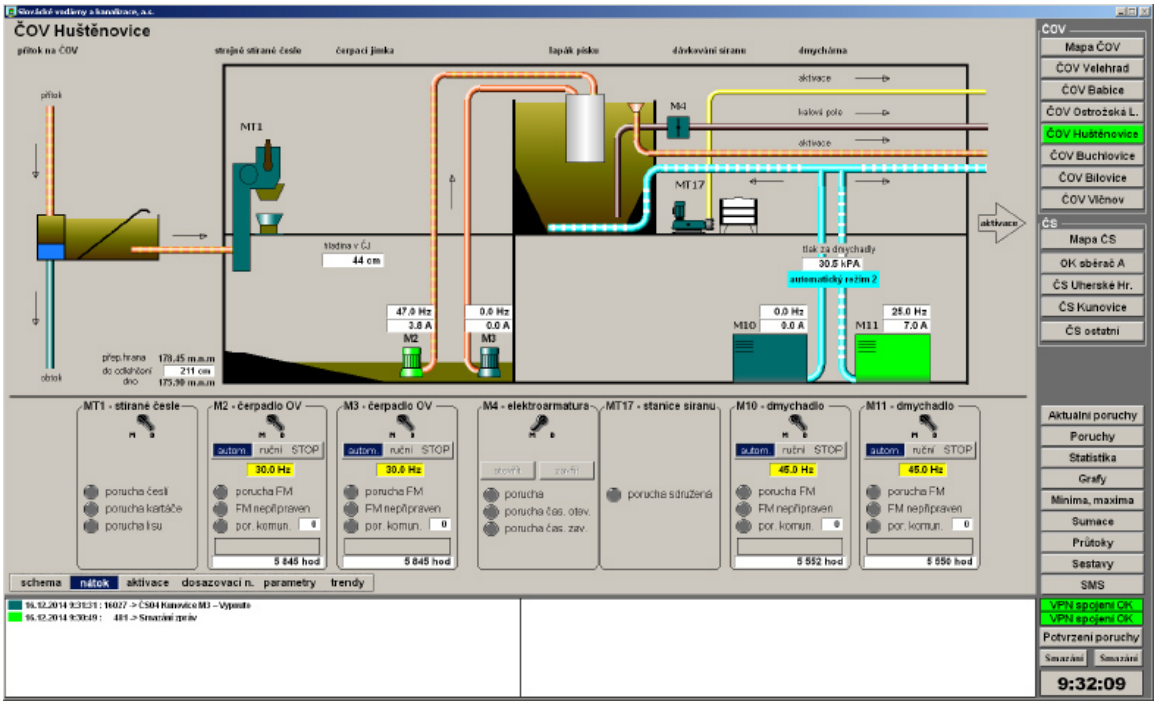
Přenos dat

Pro vizuální zobrazení měřených veličin na řízeném objektu dodává firma software pro PC, který bude důkladněji rozebrán v další části. Jak probíhá komunikace mezi počítačem a objektem, respektive řídícím PLC, je zobrazeno na obrázku 2.5.

[3] programovatelný logický automat



Obrázek 2.5: Princip přenosu dat z objektů do PC



Obrázek 2.6: Současná vizualizace pro PC (zdroj Speco control s.r.o.)

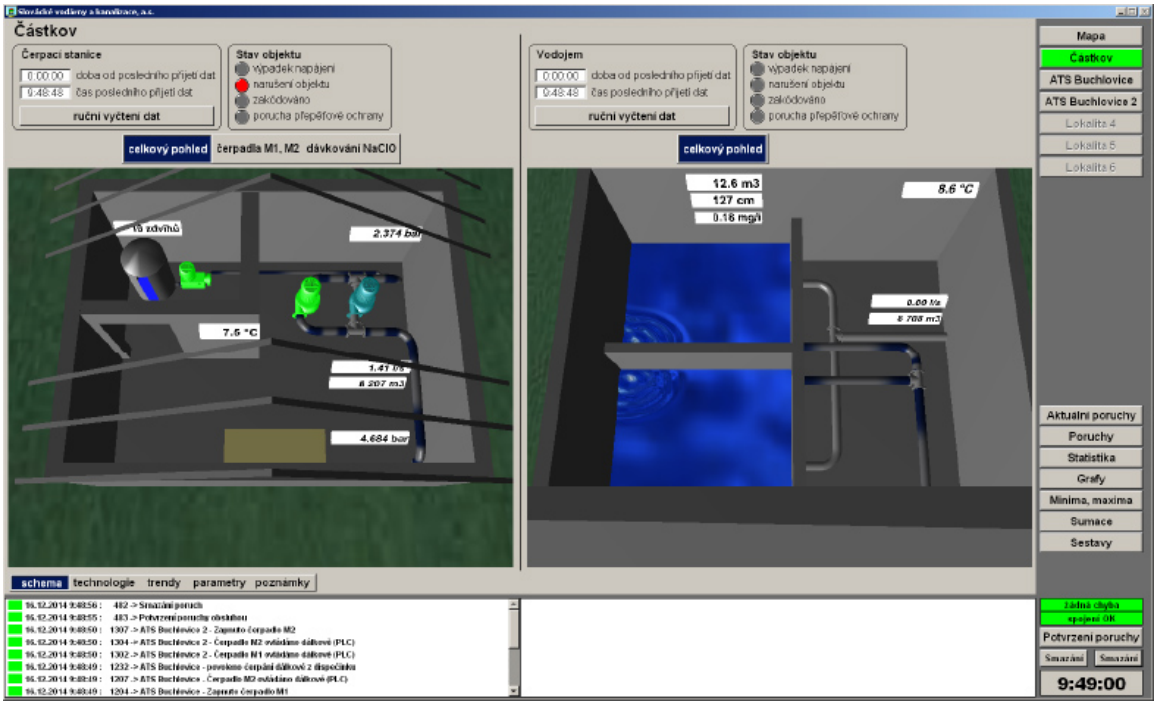
Data jsou přenášena z PLC na centrální server, kde běží příslušné obslužné programy. Zde jsou ukládány do předem připravených paměťových prostorů. Jedná se o surová data, bez jakýchkoliv informací okolo. Samotný přenos je realizován přes telekomunikační linky a internet.

V zásadě existují pouze dva datové typy, ve kterých se data mohou vyskytovat. Jsou to číselné hodnoty a logické hodnoty (boolean). Číselné hodnoty se ukládají do paměťového prostoru o velikosti jeden word (16b). Logické hodnoty jsou uloženy vždy na jednom bitu nějakého wordu. Dále jsou paměťové prostory rozděleny podle portů (na jeden port může být napojeno více objektů) a na jednom portu dále na 512 matic po 100 wordech. Pro komunikaci směrem PC – PLC je vyhrazeno dalších 512 matic po 10 wordech. Pokud je do tohoto prostoru zapsána hodnota, server ji zašle na daný PLC a ten na to reaguje vykonáním příslušné operace. Může se jednat o nastavení nové hodnoty, nebo třeba žádost o zaslání aktuálních dat.

Jelikož se data na serveru nachází bez jakýchkoliv popisujících informací, existuje ke každému objektu tabulkový dokument, který určuje mapování jednotlivých portů, matic a wordů na konkrétní hodnoty. Tento dokument vytváří programátor při implementaci řízení objektu a dále slouží pro vytvoření vizualizačního programu. Bez něj by nebylo možné určit co která hodnota v matici znamená.

2.5 Vizualizační program pro PC

Jak již bylo zmíněno, firma dodává ke každému automatizovanému objektu počítačový program pro vizualizaci a řízení. Tento program je tvořen vždy na míru pro každý objekt zvlášť. Jeho možnosti jsou velmi rozsáhlé a lze je individuálně přizpůsobovat konkrétním požadavkům zákazníka.



Obrázek 2.7: Současná vizualizace pro PC ve 3D (zdroj Speco control s.r.o.)

Objekt je rozdělen na jednotlivé sekce tak, jak se fyzicky vyskytují (samostatné budovy, místnosti, části procesu). V těchto sekcích jsou graficky zobrazeny samostatná zařízení (nádrže, čerpadla, čidla,...) a patřičné potrubní či jiné propoje mezi nimi. Jsou zde zobrazeny tzv. „volné“ hodnoty, tedy ty, které patří do dané sekce, ale nevážou se ke konkrétnímu zařízení. U samotného zařízení jsou zobrazeny všechny měřené veličiny, případně poruchy, které mohou nastat. Program také uchovává v databázi některá historické data a nabízí různé analytické informace, jako jsou sumace, grafy, minima, maxima, historické hodnoty apod. V případě, že se někde na objektu vyskytne porucha, je program schopen o této informaci okamžitě informovat a zaslat hlášení pomocí SMS zpráv vybranému okruhu osob. Ukázka programu je na obrázku 2.6.

Přes tento program je možné celý objekt nejen monitorovat, ale také řídit. Program může běžet na libovolném PC („dispečerský počítač“) s připojením k internetu. Vlastní řízení, hlídání poruchových stavů, snímání dat, přepočet analogových veličin, sumace analogových veličin, uchovávání parametrů, hlídání stykačů aj. provádějí samotné PLC. Pokud tedy dojde k vypnutí či poruše PC, řízení není nijak ovlivněno či narušeno.

Jednotlivá zařízení lze ovládat několika způsoby:

- místně pomocí tlačítek a potenciometrů na ovládacích skříňkách nebo na rozvaděči,
- dálkově ručně z dispečerského počítače,
- plně automaticky.

Řízený objekt tedy není na vizualizačním programu přímo závislý.

Jak je jistě patrné, vytvoření takového programu, není zrovna jednoduchá a levná záležitost. Celý objekt se musí nejdříve zmapovat, poté vhodně překreslit do PC, přidat měřené veličiny a nakonec odpovídající logiku.

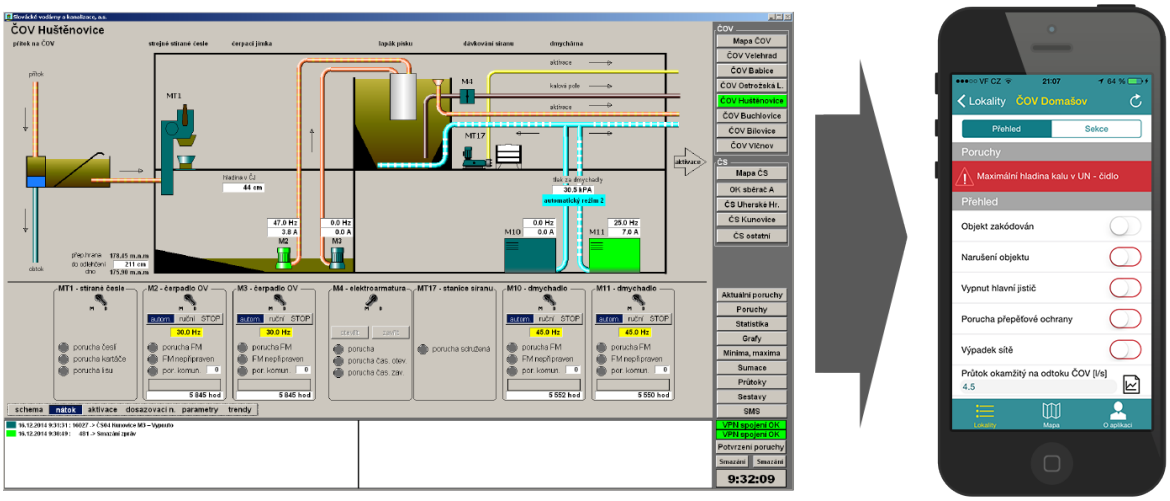
Kapitola 3

Vizualizace dat na mobilním zařízení

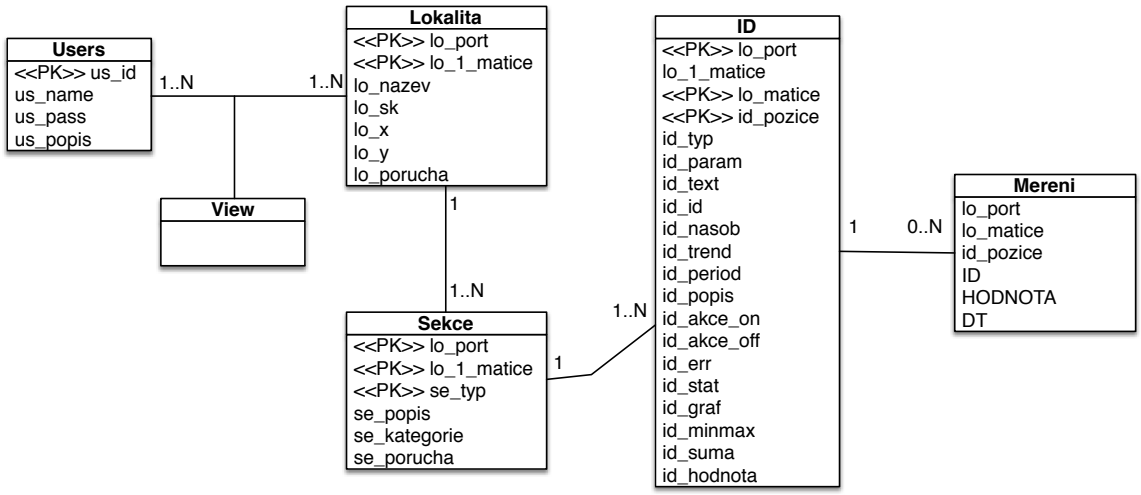
V této kapitole je popsána analýza zadání a ujasnění cíle práce. Důkladně je zde proveden návrh struktury univerzální databáze, podle které budou následně zobrazována data v aplikaci. Také je zde popsán kompletní návrh uživatelského rozhraní samotné aplikace a způsob komunikace s centrálním serverem.

3.1 Analýza a cíl práce

Cílem této práce je vytvořit mobilní aplikaci, která by byla schopná zobrazit stejné informace jako současný program pro PC. Tedy poskytnout informace o naměřených hodnotách na řízených objektech (lokalitách). Na první pohled je jasné, že není možné dosáhnout naprosto totožného řešení, protože aplikace by měla být univerzální pro jakýkoliv objekt. Informace tedy nemohou být zobrazeny graficky, ale bude se jednat pouze o tabulkové přehledy dělené do sekcí, jako v případě počítačového programu. Rozhraní je navíc omezeno velikostí displeje mobilních telefonů. I když v dnešní době jsou již telefony s velkou úhlopříčkou displeje, stále je potřeba počítat i s menšími. Dále by aplikace měla být schopná



Obrázek 3.1: Cíl práce



Obrázek 3.2: E-R diagram navržené databáze

zobrazit základní grafy, sumace a mít možnost provádět úpravy (nastavovat parametry) na řízeném objektu. Grafické znázornění cíle práce je vidět na obrázku 3.1.

Aplikace nemá nahrazovat současné řešení pro PC, ale jistým způsobem jej doplnit. Uživatel, v tomto případě zákazník, by tak měl u sebe neustále možnost zkontrolovat stav objektu a mít přístupná důležitá data okamžitě.

Během analýzy jsme došli s firmou k závěru, že jednou z nejdůležitějších funkcí bude poskytnout informaci o tom, zda se na objektu nachází nějaká porucha a kde. Dále zobrazit celkový přehled o objektu. Samotné zařízení a hodnoty na nich měřené jsou až druhotným úkolem, stejně jako zobrazení grafů a dalších analytických informací.

3.2 Návrh datového modelu

Jednou z nejdůležitějších částí vývoje je navrhnout vhodnou strukturu databáze, podle které by se aplikace dokázala sestavit. Databáze poběží na centrálním serveru a její naplnění bude v režii programátora.

Jak bylo řečeno v teoretické části, ke každému objektu existuje tabulkový dokument mapující jednotlivé měřené hodnoty do paměťového prostoru na serveru. Tento dokument je použit jako základ pro návrh databáze, protože již teď je v určité podobě nasazen na dispečerských počítačích, kde jsou do SQL databáze ukládány historické hodnoty. Dokument a z něj vycházející databázová struktura, je velmi dobře čitelný člověkem, ale skoro nevhodný pro automatické zpracování. Je tedy potřeba jej vhodně doplnit o určité sloupce (atributy) a přidat další upřesňující tabulky.

Struktura databáze vychází z návrhu uživatelského rozhraní, které je popsáno dále v této kapitole. Bylo potřeba vymyslet způsob, jakým určit, které měřené veličiny patří ke kterému zařízení, ve které části objektu se toto zařízení nachází a vůbec na jakém objektu (lokalitěú. Výsledný E-R diagram je na obrázku 3.2. Následuje popis jednotlivých tabulek, proč a za jakým účelem byly navrženy.

Tabulka *Users*

Tabulka *Users* uchovává přihlašovací informace o uživatelích a jejich hesla. Každý uživatel, zákazník, dostane přidělené unikátní 10–12ti místné ID, kterým budou později podepisovány všechny dotazy na API.

Tabulka *Lokalita*

Zde jsou uloženy informace o všech lokalitách (objektech). Každá lokalita je jednoznačně určena portem a první maticí, na které se nachází v paměti serveru. Dále je zde uložen název lokality, skupina pro řazení a souřadnice na mapě (zeměpisná šířka a délka). Poslední atribut `lo_porucha` značí, jestli se někde na lokalitě vyskytuje porucha. Informaci jsem se rozhodl přidat přímo k lokalitě, aby nebylo nutné si procházet vždy všechny hodnoty a hledat, zda se někde porucha vyskytuje.

Tabulka *View*

Tato tabulka přiřazuje jednotlivé lokality k uživatelům. Spojuje vazbu M:N (vazební tabulka). Atributy této tabulky jsou tedy primární klíče z tabulek *Lokalita* a *Users*.

Tabulka *Sekce*

Objekty jsou rozděleny do sekcí, stejně jako v případě PC programu. Tato tabulka obsahuje názvy právě těchto sekcí a dále obsahuje názvy jednotlivých zařízení v dané sekci. Lokality jsou opět určeny pomocí portu a první matice (složený primární klíč). Důležitý je zde atribut `se_typ`. Ten jsme se rozhodli použít pro univerzální identifikaci sekcí i zařízení a to následujícím způsobem:

- Každá tisícovka (tedy 1000, 2000, 3000,...) značí jednu sekci.
- V každé sekci jsou jednotlivé zařízení. Jejich názvy jsou vždy na násobcích padesáti (1050, 1100, 1150,...).
- Jednotlivé měřené veličiny jsou potom jednotky (1050, 1051, 1052,...). Tyto a následující informace jsou ovšem využity až v následující tabulce *ID*.
- Prvních 50 hodnot v sekci/tisícovce jsou tzv. „volné“ hodnoty. Tedy hodnoty, které se vztahují k dané sekci, ale nepatří k žádnému konkrétnímu zařízení.
- Prvních 1000 hodnot (0–999) slouží pro uchování obecných informací o objektu, které se zobrazí v úvodním přehledu.

K tomuto způsobu značení jsme dospěli s firmou z několika důvodů. Tabulka *Sekce*, stejně jako všechny ostatní doteď uvedené tabulky, se bude k současnému řešení přidělávat. Snažili jsme se najít způsob značení, který by byl jednoduchý na vytvoření. Nejlépe tak, aby se dal v tabulkovém editoru jednoduše rozkopírovat a upravovat pouze pár čísel. Navíc bylo zjištěno, že na jednom zařízení není nikdy měřeno 50 nebo více hodnot. Pokud by ovšem někdy v budoucnu takový případ nastal, není problém zařízení rozdělit. Stejně tak se v jedné sekci objektu nenachází více než 19 různých zařízení.

Dalším atributem tabulky je slovní popis dané sekce či zařízení a v případě, že se jedná o zařízení, také číselné označení kategorie, do které spadá (např. čerpadlo, nádrž, ventil,...).

Posledním atributem je opět informace o poruše někde v sekci nebo na konkrétním zařízení.

Tabulka *ID*

Tabulka *ID* obsahuje informace o jednotlivých měřených veličinách (výška hladiny, počet motohodin, teplota, ...) a mapuje je na paměťové místa na serveru. Právě tuto tabulku firma v současné době vytváří a používá. Její struktura je tedy částečně určena a musí být zachována.

Primární klíč je tvořen celkem třemi atributy – portem, maticí a dále pozicí wordu v matici. Pozice je dvojího typu. V případě číselné hodnoty je z rozsahu 0-99, pokud se jedná o logickou hodnotu je 100 a více. Logická hodnota typu boolean je totiž uložena pouze na jednom bitu některého wordu. Pro přesné určení tohoto bitu používá firma jednoduchý přepočetní vzorec. Na základě čísla pozice je tedy možné určit, o jaký typ měřené hodnoty se jedná.

Jelikož se jedna lokalita může vyskytovat na více maticích, je zde atribut `lo_1_matice`, který určuje první matici, na které se lokalita nachází a lze tak jednoznačně přiřadit matice k sobě.

Dále je zde důležitý atribut `id_typ`. Ten koresponduje s předchozí tabulkou a určuje kam konkrétní hodnota patří. Pokud je zde například číslo 1051, víme, že se jedná o sekci 1000 (např. Dmýchárna), zařízení 1050 (např. Čerpadlo M1) a druhou měřenou hodnotu (první má typ 1050). Typ také určuje řazení hodnot v aplikaci. Je tedy plně v režii programátora, který data do databáze vkládá, aby určil priority měřených veličin. Řazení se může lišit i podle přání zákazníka. Někdo by rád měl nejvýše zobrazené poruchové stavy, jiný počet motohodin u čerpadel nebo výšku hladiny u nádrží. Nad způsobem řazení jsme vedli s firmou diskuzi, ovšem nakonec jsme dospěli k tomuto závěru, který se jeví jako nejvhodnější jak z pohledu zákazníka, tak následné implementace.

Atribut `id_popis` určuje slovní popis měřené hodnoty. V původní tabulce je tento popis absolutně nevhodný pro strojové zpracování. Obsahuje totiž název zařízení i název měřené hodnoty (např. Čerpadlo M1- režim řízení). Způsob zápisu se navíc může lišit podle programátora, který daný dokument vytvářel. Proto byl přidán další sloupec `id_text`, kde je pouze název měřené hodnoty. Opět zde bylo myšleno na jednoduché vytvoření tohoto sloupce. Stačí pouze duplikovat sloupec předchozí a smazat informaci o názvu zařízení.

Pokud se jedná o logickou hodnotu, jsou zde uvedeny dva atributy, které určují slovní popis v případě nahozeného bytu (logická hodnota 1) `id_akce_on` a shozeného bitu (logická hodnota 0) `id_akce_off`.

Některé měřené hodnoty značí poruchový stav. Pro jejich rozpoznání je zde sloupec `id_err`. V případě, že je zde logická hodnota 1 a aktuální hodnota je také 1, jedná se o poruchu.

Poté je zde uvedeno několik atributů, které určují, zda lze hodnoty zanést do grafu, hledat minima a maxima, počítat denní nebo měsíční přírůstky (sumační tabulka) apod.

Atributem `id_param` je určeno, že se jedná o parametr, který lze nastavovat. Lze například změnit výšku vypínací hladiny, ale nelze nastavovat teplotu venkovního vzduchu.

Nakonec atribut `id_hodnota` určuje poslední naměřenou hodnotu, aby nebylo nutné tuto informaci hledat v následující tabulce *Mereni*.

Tabulka *Mereni*

Zde jsou uchovávány veškeré naměřené hodnoty. Primární klíč je shodný s tabulkou *ID* (tedy port, matice a pozice). Navíc je zde atribut určující datum a čas zapsání nové hodnoty a vlastní hodnota. Tato tabulka slouží pro sestavení grafů, sumačních tabulek a uchování historických dat.

3.3 Analýza cílové skupiny uživatelů

Před samotným návrhem uživatelského rozhraní je důležité si uvědomit, kdo bude koncovým uživatelem. Jelikož se nejedná o masivní produkt, ale aplikaci pro předem specifikovanou skupinu klientů, je vhodné ji analyzovat.

V první řadě jsou to **správci automatizovaných objektů**, kteří každý den pracují s desktopovým programem a perfektně znají svůj objekt. Dále to mohou být **vedoucí pracovníci** firem provozujících dané objekty. U nich se nedá očekávat taková detailní znalost objektů. Počítačový program znají a jsou s ním seznámeni, ale nepřicházejí s ním každý den do styku. Jde jim pouze o rychlé získání přehledových informací, případně grafů a statistik. Posledním typem uživatele jsou **pracovníci zadávající firmy**. U nich se předpokládá perfektní znalost jejich programu, ale minimální znalost objektů v porovnání například se správci. Jednoznačně lze tedy říct, že se jedná o uživatele, kteří potřebují mít rychlý a jednoduchý přístup k podstatným informacím a jsou zajisté technicky zdatní. Výsledné rozhraní by tedy mělo být spíše technického než estetického charakteru.

3.4 Návrh uživatelského rozhraní

Výsledkem analýzy zadání, uživatelů a ujasnění hlavních cílů aplikace je navržené uživatelské rozhraní. Návrh vychází ze současného řešení, zkušeností zaměstnanců firmy a připomínek potencionálních uživatelů.

Úrovně aplikace

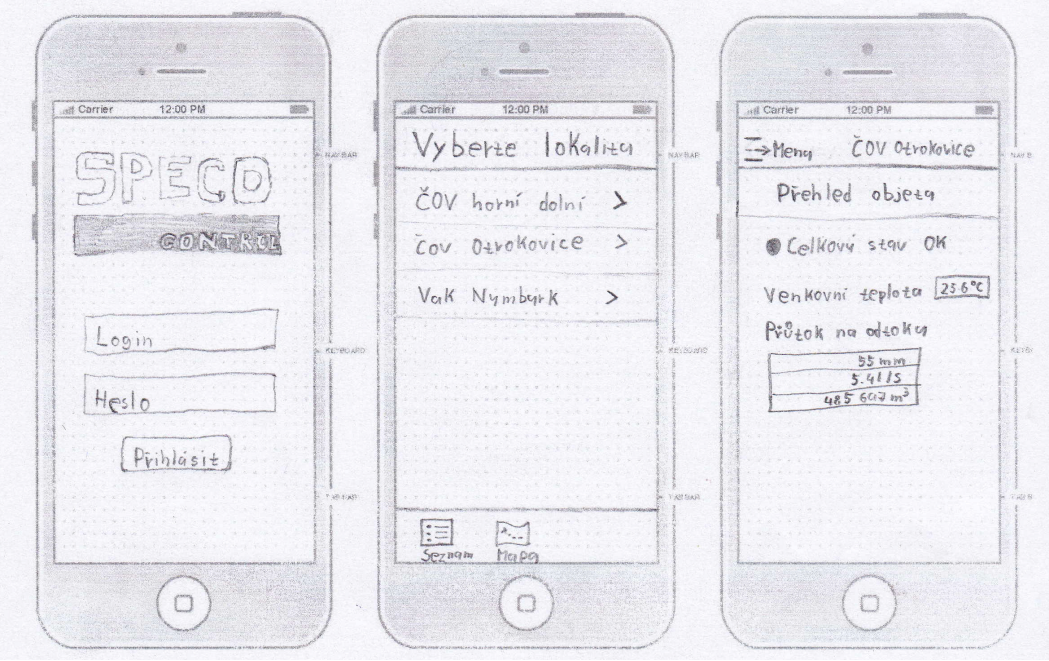
Struktura aplikace je rozdělena do několika úrovní:

- Přihlášení uživatele,
- výběr lokality (objektu) ze seznamu, případně z mapy,
- zobrazení seznamu aktuálních poruch a základního přehledu o lokalitě,
- výběr sekce,
- zobrazení seznamu zařízení ve vybrané sekci a možnost otevření detailu, zobrazení volných hodnot v sekci,
- detail zvoleného zařízení se všemi měřenými hodnotami.

Prvním krokem je **přihlášení uživatele**. Každý uživatel má svoje identické uživatelské jméno (login) a heslo. V původním návrhu jsem chtěl provést přihlášení pouze při prvním spuštění aplikace, případně po manuálním odhlášení. Uživatel by tak nemusel stále zadávat heslo. Ovšem po prvotním otestování ředitel firmy rozhodl o přihlašování při každém spuštění aplikace. Více v sekci testování 5.

Po přihlášení se zobrazí **seznam lokalit**, které k aktuálně přihlášenému uživateli patří. Jedním z přání firmy je možnost zobrazit všechny lokality také na **mapě**. K tomuto účelu jsem zde navrhl spodní lištu s ikonkami pro přepínání.

Jelikož hlavním úkolem aplikace je poskytnout informaci o poruchách, rozhodl jsem se ji zobrazit již od prvního možného okamžiku. Uživatel tedy ihned po přihlášení vidí podle barevného podkresu a příslušné ikonky, zda je na lokalitě porucha. Zatím ovšem není



Obrázek 3.3: Návrh přihlášení, seznamu lokalit a celkového přehledu o objektu

možné vyčíst jaká. Stejná informace se vyskytuje i na mapě. Zde jsou zase barevně odlišeny hlavičky „špendlíků“ značících jednotlivé lokality.

Po zvolení konkrétní lokality (objektu) je zobrazena **hlavní stránka**. Zde jsou zmiňované přehledové informace a v případě poruch také jejich názvy (tzn. na kterém zařízení a o jakou poruchu se jedná). Pokud existují poruchy, jsou vždy zobrazeny nejvýše. Po důkladné diskuzi s firmou jsme dospěli k závěru, že uživatel ve většině případů dojde právě do této úrovně.

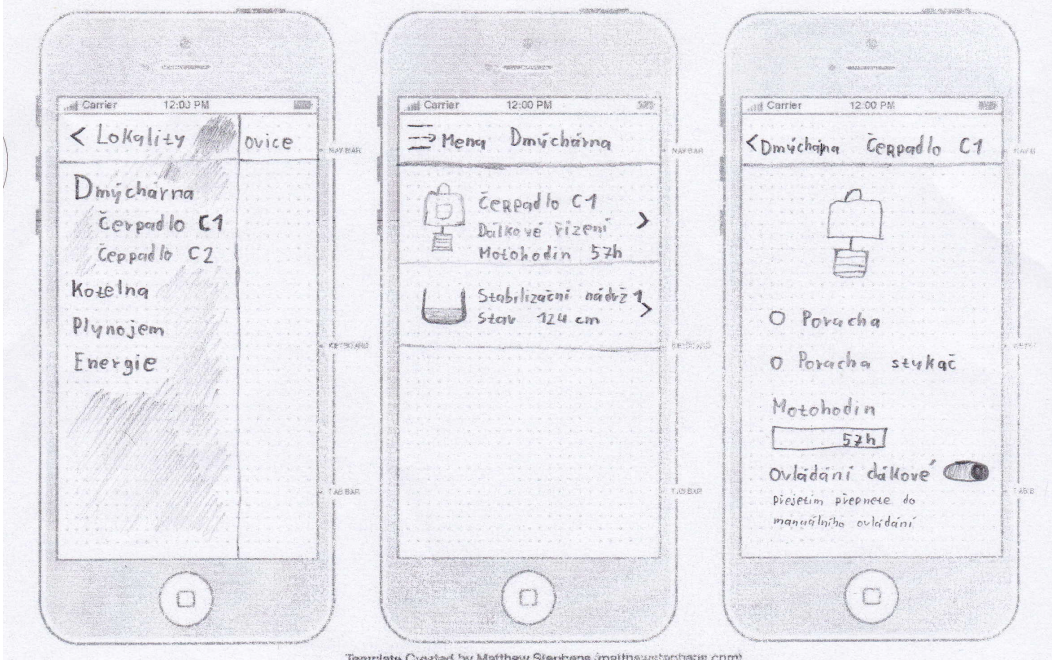
Všechny zařízení daného objektu jsou, stejně jako v případě desktopového programu, rozděleny do **sekcí**. V původním návrhu jsem chtěl využít bočního vyjížděcího menu pro seznam sekcí, jak lze vidět na obrázku 3.4 vlevo. Tento způsob menu je v dnešní době velmi populární i když sám Apple s ním úplně nesouhlasí. Podle přednášky Mike Sterna *Designing Intuitive User Experiences* na WWDC 2014 tento způsob menu nesplňuje základní principy intuitivní navigace. Tedy uživatel vždy ví, kde se nachází a zároveň vidí, kam jinam může přejít. Vyjížděcí menu (ang. též nazývané *Hamburger menu*) nesplňuje oba tyto předpoklady, protože normálně není na obrazovce vůbec vidět. [6]

Po částečné implementaci prototypu a předvedení, se zástupci firmy rozhodli pro jiný způsob řešení pomocí přepínače (*Segmented control*) na hlavní obrazovce. Je to údajně více technické řešení, při zachování stejného počtu kroků a navíc je podle zásad Applu více intuitivní.

Pokud se v sekci vyskytuje jedno či více zařízení s poruchou, je tato sekce, stejně jako název lokality, označena ikonkou a barevným podkresem.

Zvolením konkrétní sekce jsou zobrazeny **všechny příslušné zařízení** a případné **volné hodnoty**. Zařízení s poruchou je opět barevně označeno. Navíc, pro lepší přehlednost, jsou vedle názvu zařízení zobrazeny ikonky podle přidělené kategorie z databáze.

Poslední základní úrovní je **samotné zařízení** se všemi měřenými hodnotami. Jednotlivé řádky jsou stylizovány podle zobrazované informace. Logické hodnoty obsahují název



Obrázek 3.4: Návrh zobrazení sekcí a zařízeních v ní, vpravo detail jednoho zařízení

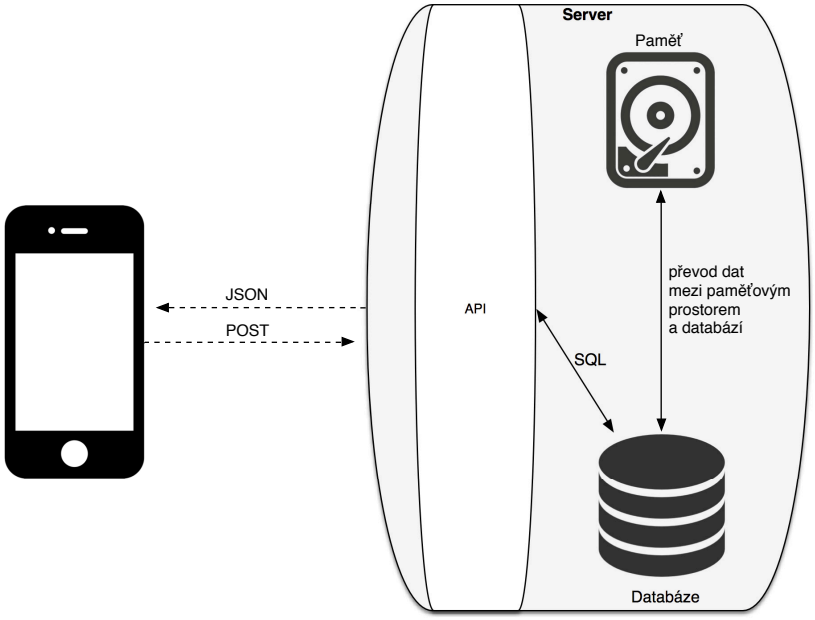
měřené veličiny a ikonku přepínače značící aktuální stav (ano – ne). Číselné hodnoty obsahují název a dekadickou hodnotu. Fyzikální veličina měřené hodnoty je zapsána v databázi hned za popisem, většinou v hranatých závorkách. Původně jsem ji chtěl odfiltrovat a v aplikaci zobrazit vždy až za konkrétní číselnou hodnotou. Ovšem stejný způsob značení je použit i v desktopovém programu a uživatelé jsou na něj zvyklí. Navíc se nedá spoléhat na to, že bude veličina vždy zapsána v hranatých závorkách.

Na řádek, značící konkrétní měřenou hodnotu kdekoliv v aplikaci, je možné kliknout a dostat se tak na obrazovku s **historickými hodnotami**. Rozsah hodnot se dá nastavit pomocí stránky s nastavením. Pro rychlou volbu jsou zde navrženy 3 tlačítka k nastavení posledního dne, týdne nebo měsíce. V případě, že je hodnota v databázi označena jako parametr, je zde možné zapsat nebo zvolit novou hodnotu a odeslat ji na server. Tímto způsobem lze měnit parametry na řízeném objektu a tím pádem jej řídit. Sekce s historickými hodnotami byla navržena až po testování v zadávající firmě. Viz. sekce 5.

Některé hodnoty mohou být zaneseny do **grafu**, případně vytvořena **sumační tabulka** s denními nebo měsíčními přírůstky. Pro přechod na tyto informace se na daném řádku vykreslí příslušná ikonka. Rozsah grafu, stejně jako sumační tabulky, lze opět nastavit.

Aktualizace hodnot

Aktualizace hodnot se provádí vždy po spuštění aplikace. Jak je ale dobrým zvykem, mělo by být možné si kdykoliv vynutit aktualizaci manuálně. Pro iOS aplikace je typické „zatažení“ hlavního obsahu okna dolů, čímž se provede stažení nových dat. Tento způsob jsem navrhl také v této aplikaci. Ovšem nedá se předpokládat, že všichni uživatelé zmíněný mechanismus znají, především pokud používají jiné mobilní platformy. Z tohoto důvodu jsem se rozhodl, přidat na většinu obrazovek do pravého horního rohu tlačítko (ikonku), pro manuální aktualizaci dat.



Obrázek 3.5: Princip komunikace

Grafický návrh

Grafický návrh uživatelského rozhraní vychází ze stávajících firemních barev, ikonky jsou navrženy tak, aby korespondovaly se zavedeným stylem iOS aplikací a jednoznačně určují svůj význam. Jelikož se jedná o aplikaci technického charakteru, snažil jsem se o co nejčistší grafický návrh bez zbytečných okrasných elementů.

3.5 Komunikace se serverem

Posledním krokem návrhu je způsob komunikace mobilní aplikace se serverem. Současné desktopové programy využívají protokol Modbus TCP. Ten komunikuje přímo s paměťovým prostorem na serveru, kde jsou uložena veškeré data. Firma si nejdříve představovala, že aplikace bude komunikovat stejným způsobem. Je zde ovšem jeden problém. Aplikace by musela současně komunikovat také s databází, aby si byla schopná spojit data s konkrétním popisem. Z tohoto důvodu jsem se rozhodl raději rozšířit úlohu serveru a naměřené hodnoty ukládat přímo do databáze. Ušetří se tím komunikační i výpočetní režie na straně aplikace. Pro ukládání dat do databáze na straně serveru má firma navíc již vyvinuté a otestované mechanismy, které lze s velkou výhodou použít.

Jelikož iOS aplikace nedokáží pracovat přímo s databází typu SQL, je potřeba na serveru vytvořit API, které na základě dotazu vytáhne určité data z databáze a zabalená ve formátu JSON odešle přes síť do aplikace.

Princip komunikace je zobrazen na obrázku 3.5.

Kapitola 4

Mobilní aplikace pro vizualizaci a řízení vodohospodářských staveb

Následující kapitola popisuje vlastní implementaci jednotlivých částí projektu. Nejprve jsou uvedeny technologie a prostředky, které jsem se rozhodl pro vytvoření využít. Následně je zde rozvedena realizace uživatelského rozhraní, komunikace s databází a nakonec serverová část API s databází.

4.1 Implementační prostředky

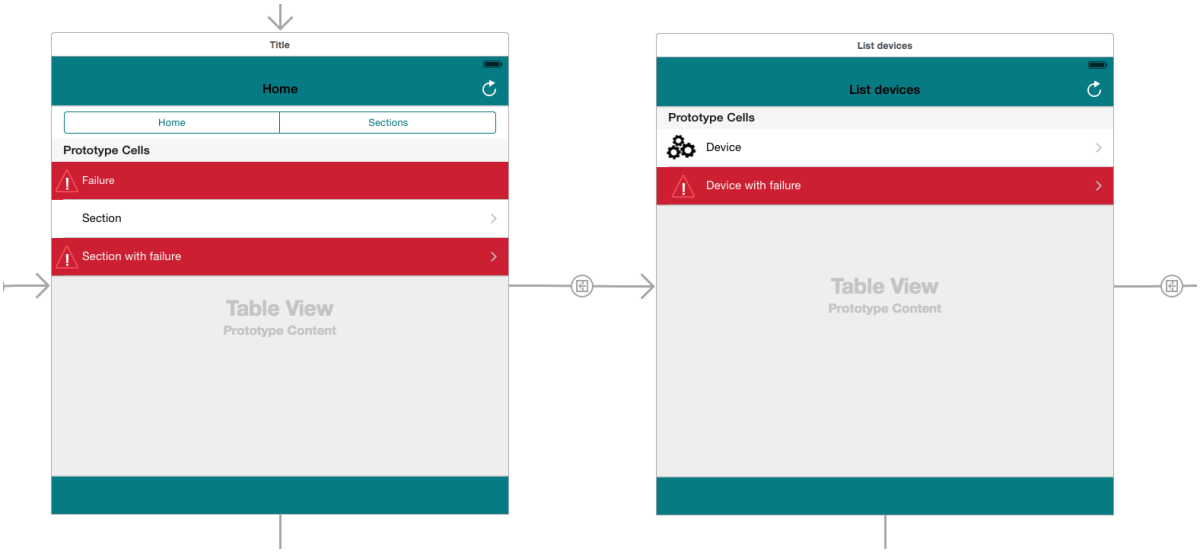
Jako implementační jazyk jsem se rozhodl použít nově představený **Swift**. I když v době implementace byl jazyk stále ve vývoji, opravovali se chyby a na internetu se k němu vyskytovalo minimum návodů a tutoriálů, přesto jsem si práci s ním chtěl vyzkoušet. V průběhu roku dokonce vyšla aktualizace Swift 1.2. která přinesla větší změny a celý projekt musel být převeden podle nových pravidel jazyka. Dále jsem v implementaci využil framework **Cocoa Touch**, který tvoří základ pro iOS aplikace. Samotný vývoj probíhal ve vývojovém prostředí **Xcode**. Také jsem se rozhodl využít knihovny 3. strany pro kreslení a práci s grafy, kterou jsem ovšem musel pro vlastní potřeby mírně přepsat.

Pro část API jsem použil jazyk **PHP**, jelikož s ním mám již zkušenosti a nemusím se tak učit další. Jako databázi jsme s firmou nakonec zvolili **MySQL**, protože je její použití nejvíce rozšířené. Firma sice v současné době používá databáze od Microsoftu, ale byli ochotni se přizpůsobit a vyvinout prostředky pro možnost komunikace s databází typu MySQL.

4.2 Implementace uživatelského rozhraní

Uživatelské rozhraní je vytvořeno pomocí nástroje Storyboard. Jedná se o grafický editor, kde si lze poskládat jednotlivé obrazovky aplikace (*View*) a přechody mezi nimi (*Segue*). Některé grafické prvky jsou dále upraveny programově v závislosti na stavu, ve kterém se aplikace právě nechází.

Všechny obrazovky, kromě přihlášení, jsou zabaleny do společného *Tab Bar controlleru*. Jedná se o dolní přepínací lištu zmíněnou v části návrhu. Dále jsou jednotlivé části (větve) zasazeny do *Navigation controlleru*, to je pro změnu horní navigační lišta. Díky ní je možný jednoduchý přechod zpět. Všechny nově zobrazené obrazovky jsou vkládány do zásobníku tak, jak jsou postupně otevřeny a je umožněno jejich jednoduché navracení. Do horní lišty



Obrázek 4.1: Ukázka grafického rozhraní ve storyboardu

je možné umístit také titulek aktuální obrazovky a další elementy. Právě tento prostor jsem využil na umístění ikonky pro manuální aktualizaci dat a dalších tlačítek.

Většina obrazovek je tvořena pomocí tabulkového zobrazení (*Table view*). Mnoho iOS aplikací, i když se tak na první pohled nemusí zdát, je tvořena právě tabulkami. Každá buňka totiž může být jinak stylizována, čímž lze dosáhnout rozmanitého vzhledu. Různého formátu buněk jsem se rozhodl využít i já. Vytvořil jsem si několik vzorových buněk podle zobrazovaného obsahu a z nich následně vybíral. Namátkou například buňka pro název lokality s poruchou a bez poruchy, buňka pro číselnou hodnotu, logickou hodnotu, poruchový stav atd. Ukázku grafického rozhraní, vytvořeného pomocí storyboardu, lze vidět na obrázku 4.1. Jak si můžete podle obrázku všimnout, velikost obrazovky neodpovídá rozměrům, dokonce ani poměru stran, displeje telefonu iPhone. Apple totiž zavedl tzn. *Auto layout*, pomocí kterého lze nadefinovat jeden vzhled aplikace, který bude stejný na zařízeních s různou velikostí displeje. Toho lze dosáhnout pomocí *constraints*, což jsou různé limity nebo předpisy pro daný prvek. Nejlepší je ukázka na příkladu. Obrázku lze například nadefinovat, že má být vždy 10px od levého okraje a má mít šířku 30px. Vedle něj umístíme textové pole. Jeho šířku ale nebudeme specifikovat, zato nastavíme odsazení od obrázku a druhého okraje třeba na 10px. Šířka tohoto prvku se dopočítá automaticky v závislosti na velikosti displeje. Tímto způsobem jsou implementovány všechny prvky v aplikaci. Prostředí Xcode si dokáže již během tvorby samo hlídat, zda jsou constraints vhodně zvolené a nedochází k nějakým konfliktům.

Každá obrazovka má svůj controller. Ve swiftu jsou to třídy odvozené od *UIViewController*, případně *UITableViewController*. Controller vkládá data do svojí řízené obrazovky a reaguje na zprávy od uživatele. Například v případě, že bylo kliknuto na tlačítko, rozpoznáno gesto apod.

Přihlášení

Přihlašovací obrazovka je první, co se po spuštění aplikace zobrazí. Je tvořena logem firmy, dvěma políčky (*UITextField*) pro zadání přihlašovacích údajů a tlačítko (*UIButton*) k přihlášení. Snažil jsem se myslet na detaily a proto například pokud uživatel klikne do po-

líčka pro zadání přihlašovacího jména, tak se mu na klávesnici místo klávesy *Enter* zobrazí *Dále* (ang. *Next*) pro jednoduchý přechod na políčko s heslem. Dále po prvním přihlášení je uloženo uživatelské jméno do vnitřní proměnné, konkrétně *NSUserDefaults* a při opětovném spuštění je již předvyplněno. Snažil jsem se tak kompenzovat jeden z požadavků firmy, kdy si přejí neustále přihlašování uživatele do aplikace.

Jednotlivé elementy jsou propojeny s příslušným controllerem této obrazovky. Tzn. je zde uvedena reference na obě textové políčka a také implementována metoda reagující na stisk tlačítka (*Touch Up Inside*). Stejným způsobem jsou propojeny i všechny ostatní obrazovky a jejich aktivní elementy s příslušným controllerem.

Lokality

Seznam lokalit je umístěn v tabulce (*UITableView*). Jednotlivé řádky jsou stylizovány podle toho, zda se na lokalitě vyskytuje porucha či nikoliv. Který styl řádku (buňky) se použije, stejně jako jeho obsah, je určeno v controlleru tohoto view.

Jelikož je většina obrazovek v aplikaci tvořena pomocí tabulek, následuje popis jak se s nimi a hlavně jejich controllery pracuje.

UITableViewController

Každá tabulka (*UITableView*) musí být spojena s *UITableViewControllerem*, respektive s třídou odvozenou právě od této nadtřídy. Ta obsahuje několik povinných metod pro nastavení tabulky a jejího obsahu.

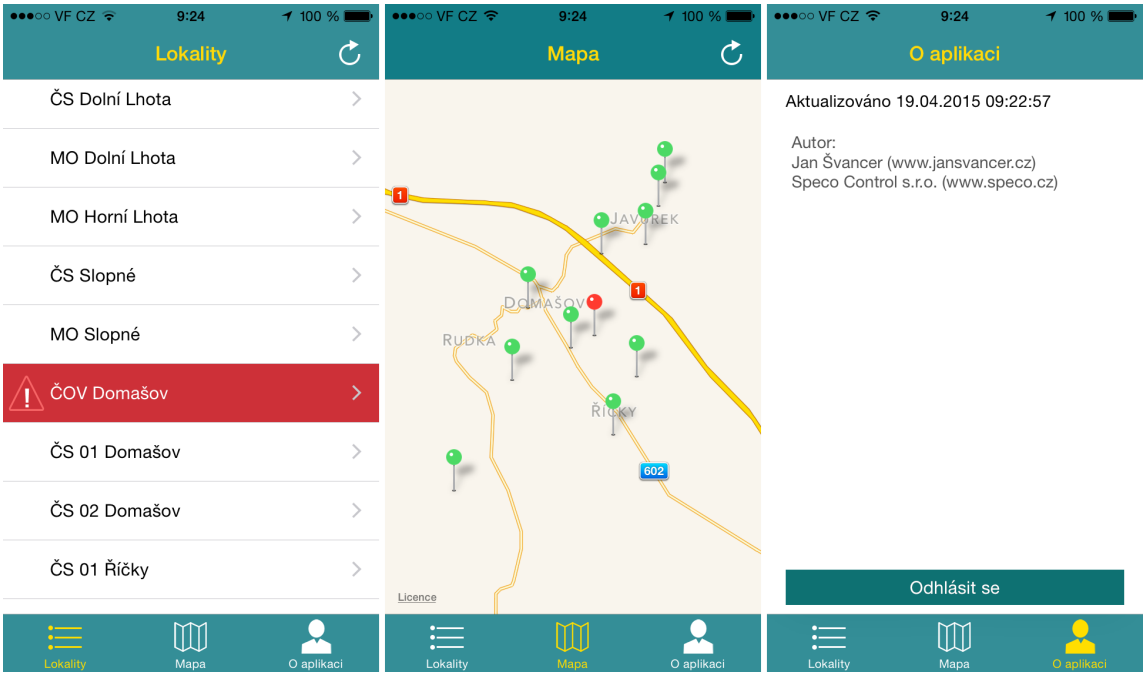
Třída každého controlleru obsahuje metodu `ViewDidLoad()`, která se volá před samotným vykreslením obsahu na obrazovku a je zde možné prvky před zobrazením upravit, případně přidat. V této metodě si obvykle načtu obsah, který se bude následně v tabulce zobrazovat do proměnné typu pole.

Následují povinné metody pro nastavení parametrů tabulky.
První z nich `numberOfSectionsInTableView` vrací, jak název napovídá, počet sekcí v tabulce. Sekce jsou jednotlivé části tabulky oddělené od sebe hlavičkou. Může se jednat například o rozdělení adresáře do sekcí podle abecedy apod. V mojí aplikaci je většího počtu sekcí využito na několika místech. Například na úvodní obrazovce v případě, že se na lokalitě vyskytují poruchy, jsou vytvořeny dvě sekce. První pro seznam poruch a druhá pro přehledové hodnoty. Dále je povinná metoda, která určuje počet řádků tabulky. Ve většině případů je to počet prvků pole s daty.

Velmi důležitou a povinnou metodou každého *UITableViewControlleru* je metoda pro určení obsahu jednotlivých buněk. Zde si vždy vyberu aktuální řádek tabulky podle příslušného indexu a typ zobrazované buňky podle vykreslovaného obsahu. Každá buňka má svůj originální identifikátor a třídu. Pokud tedy chci například vykreslit název lokality s poruchou, vyberu si podle identifikátoru právě tento typ buňky. Následně nastavím obsah, většinou voláním příslušné metody dané buňky, která je definována v její třídě a vrátím buňku v návratové hodnotě. Všechny buňky jsou děděny od třídy *UITableViewCell*.

Mapa

Zajímavou implementační částí je **mapa**. Lépe řečeno její správné nastavení. Pro vykreslení mapy jsem využil framework *MapKit*, který je v základu součástí iOS SDK. Jelikož na mapě můžou být zobrazeny libovolné lokality, je důležité vhodně nastavit její střed a zobrazovaný



Obrázek 4.2: Hlavní sekce aplikace

rozsah tak, aby byly vždy po spuštění všechny viditelné. K tomuto účelu jsem implementoval metodu `fitAndShowAnnotations()`. Ta projde postupně všechny lokality, respektive souřadnice bodů na kterých se lokality nacházejí a hledá krajní hodnoty. Pro určení středu a přiblížení mapy jsou od sebe krajní hodnoty odečeteny a vynásobeny vhodně zvoleným koeficientem. Zároveň při hledání krajních hodnot jsou body představující lokality přidávány postupně na mapu. Aby bylo možné bod zanést do mapy, musí se jednat o objekt dědený od třídy *MKAnnotation*. Lokalita na mapě je potom představována typickým špendlíkem.

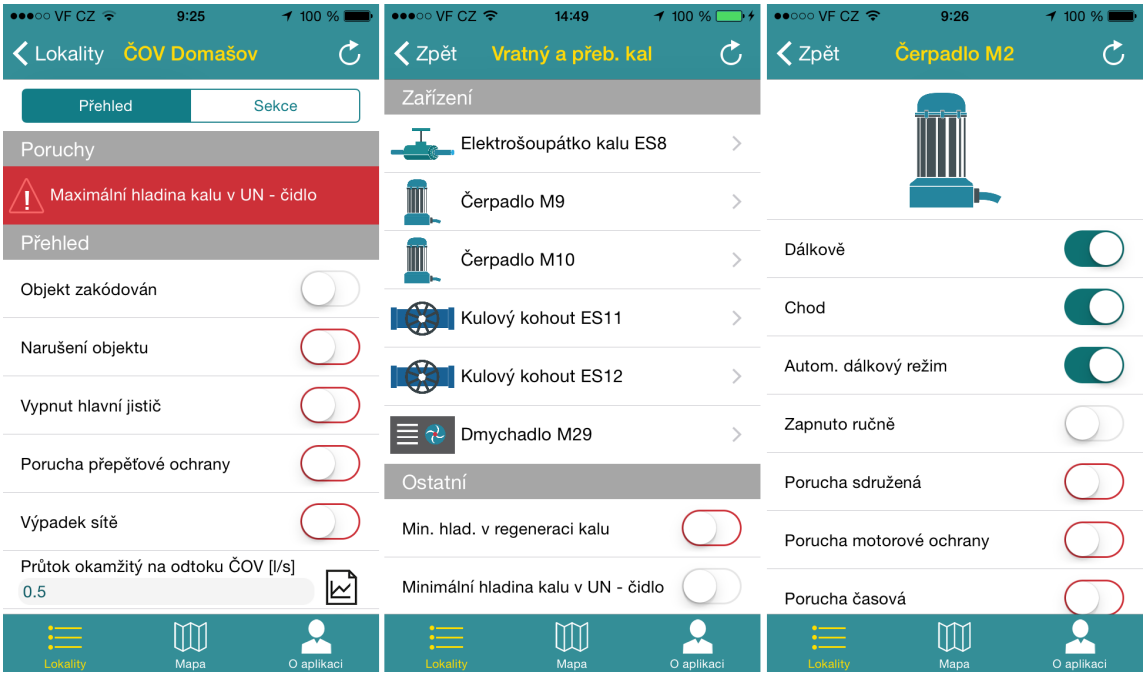
Aby bylo i v mapě na první pohled patrné, na které lokalitě se vyskytuje porucha, rozhodl jsem se hlavičku špendlíku obarvit v případě poruchy na červeno, jinak na zeleno. Toho lze dosáhnout v metodě `viewForAnnotation`, která je součástí protokolu *MKMapViewDelegate*. Kromě barvy hlavičky si zde nastavuji také titulek a podtitulek vztahující se ke konkrétní lokalitě.

O aplikaci

Oproti návrhu jsem přidal jednu obrazovku obsahující základní informace o aplikaci, informace o poslední aktualizaci dat a také tlačítko pro manuální odhlášení. Přechod na tuto obrazovku je možný z dolní přepínací lišty (*Tab Bar*).

Data v lokalitě

Implementace všech ostatních obrazovek, po zvolení konkrétní lokality, je velmi podobná seznamu lokalit. Opět je zde využito tabulkového zobrazení s různým formátováním buněk. Jelikož 4 základní typy buněk se vyskytují skrz celou aplikaci, rozhodl jsem se jejich vzhled nadefinovat zvlášť (v souborech typu .xib). V příslušném controlleru jsou potom pouze zaregistrovány, aby je bylo možné ve view používat. O tom, který formát buňky se použije, rozhodují vždy konkrétní hodnoty, které mají být vykresleny.



Obrázek 4.3: Detail lokality: úvodní přehled, sekce s volnýma hodnotama a detail zařízení

Kliknutí na buňku je detekováno funkcí `didSelectRowAtIndexPath` a na základě typu buňky jsou provedeny příslušné přechody na jiné obrazovky.

Jelikož délka zobrazovaných textů v buňce je dosti rozdílná, někdy například pouze jedno slovo, jindy zase pomalu celá věta, rozhodl jsem se namísto hledání optimální výšky buňky zvolit její automatické nastavení pouze s omezením na minimální velikost. Toto řešení se nakonec ukázalo jako nejvhodnější, i když jsem se původně snažil najít vhodný kompromis, ale rozdílu v délce textů jsou mnohdy obrovské.

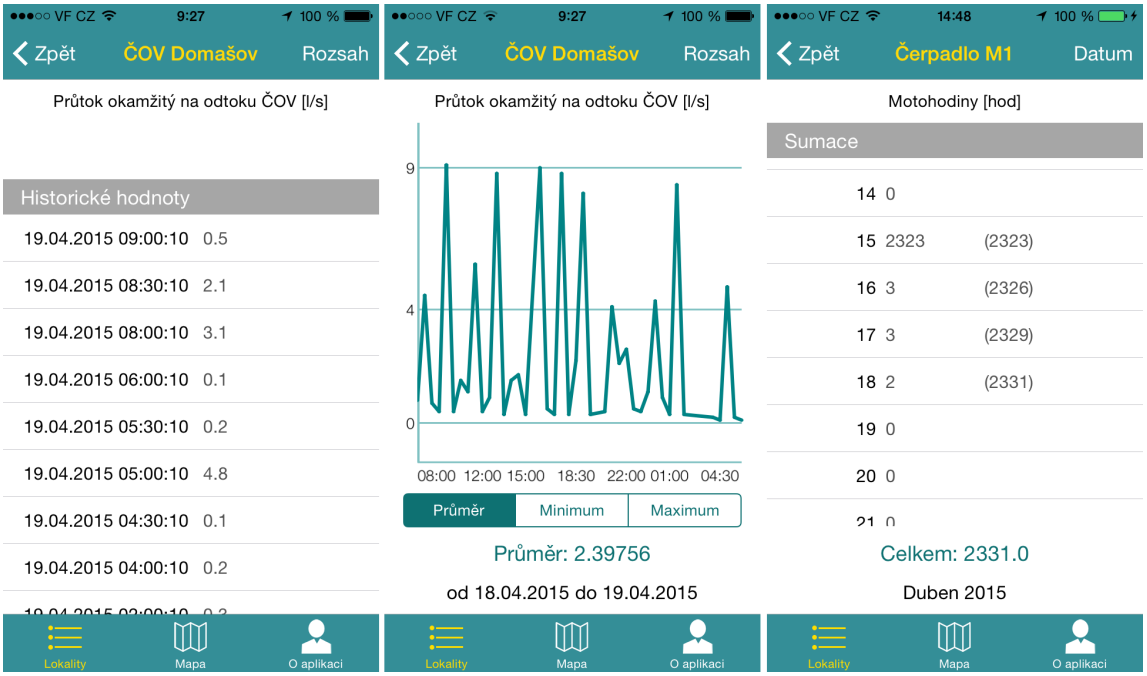
Grafy

Pro vykreslení grafů jsem použil knihovnu třetí strany *BEMSimpleLineGraph*. Ačkoliv knihovna sama o sobě je dosti propracovaná, musel jsem si její funkčnost rozšířit. Základním nedostatkem byla nemožnost zadávat konkrétní hodnoty na ose X. Hodnoty vždy rostly po celých jednotkách. V aplikaci jsou grafy využity pro zanesení historických hodnot v čase. Může se ale stát, že v některý časový okamžik není hodnota v DB zapsaná a měřítko osy je potřeba přizpůsobit, tedy vykreslit větší mezeru odpovídající skutečné vzdálenosti dvou bodů od sebe.

Do knihovny jsem implementoval metodu, která čte hodnoty na ose X ve formátu *NSDate* (datum) a následně vypočítá celkový časový rozsah aktuálně zobrazovaný v grafu, který odpovídá 100% šířky osy X. Jednotlivé body jsou potom proporcionálně zakresleny do grafu, aby odpovídaly svojí skutečné časové hodnotě.

V této části aplikace jsem musel využít staršího programovacího jazyka Objective C, jelikož původní knihovna pro práci s grafy je napsána právě v tomto jazyce. Apple pro zpětnou kompatibilitu poskytuje tzv. Bridging Header, tedy „most“ skrz nějž lze použít části kódu v Objective C v jazyce Swift.

Pro nastavení rozsahu grafu slouží samostatná obrazovka. V horní části jsem předpřipra-



Obrázek 4.4: Historie, graf a sumační tabulka

vil 3 tlačítka pro rychlé nastavení na poslední den, týden a měsíc, jak lze vidět na obrázku 4.5 vpravo. Defaultně po vykreslení grafu je rozsah nastaven na poslední týden. Toto nastavení by se ale mohlo do budoucna optimalizovat v závislosti na hodnotě v grafu. Jelikož některé hodnoty jsou zapisovány 1x denně, ale jiné například každých 30 minut nebo při změně. Pro přesné nastavení rozsahu slouží dvojice řádků s *DatePickery*, což jsou otočné válce pro nastavení data a času, typické pro iOS.

Sumace

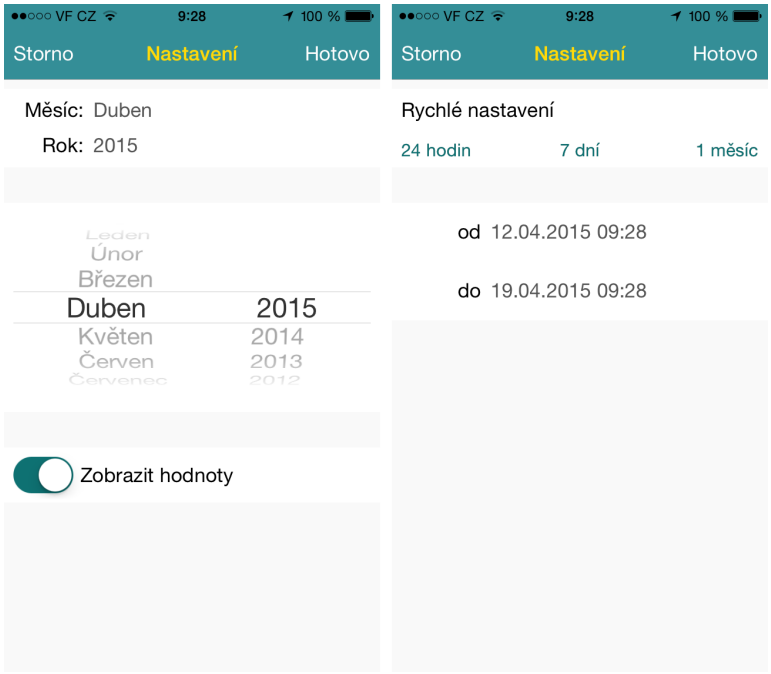
Další částí aplikace jsou sumační tabulky. Ty poskytují přehled o denních nebo měsíčních přírůstcích u stále rostoucích hodnot (např. motohodiny). Pro správné vypočtení hodnot, musí být ze serveru získány příslušné hodnoty. Pro denní přírůstky se čte z databáze vždy poslední hodnota ve dne, pro měsíční zase poslední hodnota v daném měsíci. Jakmile takto získám správné hodnoty není výpočet nikterak složitý, pouze se odečtou dvě čísla a ošetří neočekávané stavy.

Aktuálně zobrazovaný měsíc, případně rok, lze opět nastavit na vlastní obrazovce. Také lze zvolit, pomocí přepínače *UISwitch*, zda se má zobrazovat i původní hodnota, ze které byly přírůstky počítány (viz. obrázek 4.5 vlevo).

Historické hodnoty a zápis

Tato část byla do aplikace přidána až na základě testování v zadávající firmě. Slouží pro zobrazení historických hodnot ve zvoleném rozsahu. Pro jeho nastavení slouží stejná obrazovka jako pro rozsah grafu. V tabulce jsou zobrazeny všechny historické hodnoty, které se v databázi nacházejí.

U hodnot, které jsou označeny v databázi sloupcem `id_param`, se navíc nad tabulkou vykreslí buď políčko pro zápis nové hodnoty, nebo přepínač mezi stavy zapnuto – vypnuto



Obrázek 4.5: Zvolení měsíce u sumační tabulky, nastavení časového rozsahu

podle typu hodnoty a tlačítko pro odeslání této nové hodnoty na server. Před samotným odesláním je uživateli zobrazeno ještě potvrzovací okno pomocí *UIAlertControlleru*. Jedná se o standartní komponentu v systému iOS určenou právě pro tyto účely. Tímto způsobem lze na objektu upravovat určité parametry stejně jako v případě desktopového programu.

Jazykové mutace

Jelikož aplikace má být určena nejenom pro české zákazníky, ale především pro uživatele z Ruska, byl jsem požádán firmou abych, se zaměřil i na způsob jednoduchého převodu do různých jazykových mutací.

Hledáním a studiem jsem zjistil, že existuje přímo nativní způsob jak psát mezinárodní aplikace v prostředí Xcode. Tento nástroj se nazývá *Localization*. V nastavení projektu lze přidávat jazyky, ve kterých chceme aplikaci poskytovat. Pokud zvolíme více než jeden jazyk, prostředí Xcode automaticky vytvoří soubor `.strings`, který obsahuje všechny texty, které jsou uvedeny ve Storyboardu. Nemusíme tedy ručně vše vypisovat, ale samotný program to provede za nás. Problém nastává ve chvíli, když něco ve Storyboardu změníme, nebo přidáme. Takovou změnu musíme do souborů s překlady zanést manuálně.

Pro texty, které jsou v aplikaci přidávány v kódu, je potřeba použít speciálního datového typu *NSLocalizedString*. Ten obsahuje identifikátor a slovní komentář textu, který má na toto místo přijít. Samotné texty jsou potom uloženy v souborech *Localizable.strings*. V těch je uveden zvolený identifikátor a text v požadovaném jazyce. Pro každý jazyk existuje právě jeden soubor *Localizable.strings*.

Aplikaci jsem nakonec přeložil celkem do 3 jazyků – angličtina, čeština a ruština. Která jazyková mutace se použije, pozná systém podle systémového nastavení, přičemž výchozí hodnotou je angličtina.

4.3 Komunikace a práce s daty

Komunikace s centrálním serverem probíhá na principu dotaz – odpoveď. Uživatel tedy skrz aplikaci posílá požadavky na API a ta vrací odpoveď zakódovanou ve formátu JSON. Parametry dotazu jsou předávány metodou POST.

Uložení dat v aplikaci

Stažené data jsou uloženy ve třídě *DataManager*, která obsahuje pole objektů s hodnotami, sekcemi a lokalitami. Také uchovává informace o poslední aktualizaci aktuálně uložených dat a číslo portu a matice právě stažené lokality.

Pro přístup k těmto hodnotám jsem implementoval třídu *DataAPI*, která je vytvořena jako singleton (jedináček) [8]. Obsahuje gettery do DataManageru a také metody pro stažení dat pomocí API. Metody pro stažení se vykonávají asynchronně. Nejprve je vytvořen dotaz, odeslán na API metodou třídy *ApiCommunication* a následně zpracován výsledek. Tedy buď uložen do DataManageru, nebo vrácena chyba. Před zahájením a po ukončení činnosti jsou vyslány notifikační zprávy, které budou rozebrány dále.

Jedna konkrétní hodnota je v aplikaci uložena jako objekt. Implementovány jsou tedy třídy *Lokalita*, *Sekce*, *Id* a *Hodnota*. Ty obsahují většinou stejné atributy jako jim odpovídající tabulky v databázi. Pouze v případě Id (měřené hodnoty) jsou uloženy jen atributy potřebné pro správný chod aplikace, zbytek slouží pro potřeby firmy a není tedy nutné je všechny uchovávat.

Stahování dat

Po spuštění a přihlášení do aplikace se uživateli stáhne seznam lokalit a to konkrétně v metodě `viewDidLoad`. Pokud se uživatel znovu vrátí na seznam lokalit, jsou v pozadí staženy nové aktuální data. Toto se děje v metodě `viewWillAppear`, která je volána vždy předtím, než je daná obrazovka vykreslena uživateli.

Jakmile uživatel zvolí konkrétní lokalitu, jsou staženy všechny měřené hodnoty a sekce v ní. Původně jsem stahoval data pro všechny lokality po přihlášení do aplikace, ale jelikož se jedná o veliké množství hodnot (někdy až 1000 záznamů v jedné lokalitě), rozhodl jsem se vždy stahovat pouze aktuálně prohlíženou lokalitu. Historické hodnoty jsou stahovány až v případě zobrazení příslušné obrazovky pro historii, graf nebo sumace. Navíc jsou staženy záznamy pouze ve zvoleném časovém rozsahu. Historických hodnot na serveru můžou být s postupem času řádově statisíce.

Pokud nejsou data stahována na pozadí, je uživateli vykresleno malé okénko ve středu obrazovky, obsahující *UIActivityIndicator*, tedy točící se kolečko značící nějakou aktivitu v pozadí. Po dokončení stahování je příslušná obrazovka informována pomocí notifikační zprávy, aby se překreslila.

Aktualizace dat

Pro manuální aktualizaci hodnot jsem v aplikaci zvolil dvojí způsob, jak bylo zmíněno v části návrhu 3. První z nich, tedy „zatažení“ obrazovky dolů, je vytvořeno pomocí *UIRefreshControl*. Tento způsob aktualizace je pro iOS natolik typický, že Apple poskytuje již předem připravené prostředky, které lze s velkou výhodou využít. Druhý způbob aktualizace je pomocí klasického tlačítka umístěného v horní navigační liště. Jeho stisk je odchytáván v příslušném controlleru.

Po dokončení aktualizace se uživateli po dobu 1s zobrazí na displeji zpráva oznamující úspěšné stažení, případně okno s příslušnou chybovou hláškou.

Notifikace

Jelikož stahování dat probíhá asynchronně, je potřeba o jeho dokončení informovat zbytek aplikace. V iOS existuje systém notifikací, který by se dal přirovnat k návrhovému vzoru *Observer* [8].

Při zahájení stahování a po jeho dokončení se vyšle příslušná notifikační zpráva. Obrazovky které mají zájem, tedy jsou zaregistrovány jako observer pro příslušný typ zprávy, jsou potom informovány. Navíc lze spolu s notifikační zprávou předat i parametry. V aplikaci je využívám pro identifikování stavu. Posílá se tedy vždy stejná notifikační zpráva (*DataDownloadNotification*), pouze s jinými parametry, které znamenají začátek nebo konec stahování (*status*), informaci zda se má zobrazit activity indicator (*showActivityIndicator*) a informace zda se má vykreslit zpráva o dokončení stahování (*showDone*). Po přijetí takové notifikace jsou potom vytaženy z *DataManageru* nové, aktuální, hodnoty a zavolán refresh nad příslušnou tabulkou. Zobrazování *activity indicatoru* a zprávy o dokončení stahování dat má na starosti nadřazený controller pro TabBar, který zastřešuje celou hlavní část aplikace.

4.4 Back-end aplikace

Posledním krokem implementace je část API a databáze běžící na centrálním serveru. Samotné plnění měřených hodnot do databáze provádí firma a tato část tedy není součástí dokumentace.

API

API je napsána v jazyce **PHP**. Obsahuje jednoduché dotazovací skripty do databáze a výsledek převádí pomocí funkce `json_encode` na formát JSON. Pro připojení pomocí *mysqli* a práci s databází, jsem si napsal objektovou třídu, která dotazy provádí. Všechny hodnoty, předané skriptu metodou POST, jsou vždy nejdříve escapovány a až poté použity jako parametry SQL dotazu.

V příloze lze nalézt manuál k vytvořené API (B). Obsahuje názvy všech skriptů, jejich povinné parametry, slovní popis funkce a návratových hodnot. Z důvodu bezpečnosti zde neuvádím adresu serveru.

Databázové triggery

Samotná databáze byla vytvořena v prostředí *phpMyAdmin* a data jsou do ní nahrávány ze souborů typu .csv, které jsou vygenerovány z excelových tabulek.

Zajímavější částí jsou triggery, které slouží pro aktualizaci tabulek. Nově naměřené hodnoty jsou zapisovány pouze do tabulky *Mereni*, aby si aplikace, případně API, nemusely vytahovat vždy poslední hodnotu nebo hledat, zda existuje v dané lokalitě/sekci porucha, rozhodl jsem se napsat triggery, které budou tyto informace rozkopírovávat přímo k příslušným řádkům v ostatních tabulkách.

V první řadě je to tabulka ID. Pokud je do měření zapsána nová hodnota, je zkopírována do atributu `id_hodnota` v příslušnému řádku tabulky ID podle shodujícího se portu, matice a pozice.

Složitější je ovšem hlídání poruch v sekci, u konkrétního zařízení a na celé lokalitě. Zde si vždy procházím všechny hodnoty, které mohou nabývat poruchového stavu, jsou označeny jako `id_err` v tabulce ID a spočítám si všechny takové, které mají aktuální hodnotu 1. Je potřeba procházet pouze hodnoty vztahující se k danému zařízení, sekci nebo lokalitě. Proto provádím zaokrouhlení dolů na násobky 50 v případě zařízení v sekci a celé tisícovky pro konkrétní sekci.

Část kódu pro aktualizaci poruch lze vidět na následující ukázce:

```
...
UPDATE sekce
s JOIN (
        SELECT lo_port, lo_1_matice, FLOOR(id_typ/50.)*50 AS typ
        FROM id
        GROUP BY lo_port, lo_1_matice, typ
) si
ON s.lo_port = si.lo_port
AND s.lo_1_matice = si.lo_1_matice
AND s.se_typ = si.typ
SET s.se_porucha = false;

UPDATE sekce
s JOIN (
        SELECT lo_port, lo_1_matice, FLOOR(id_typ/50.)*50 AS typ
        FROM id
        WHERE id_err = 1 AND id_hodnota = 1
        GROUP BY lo_port, lo_1_matice, typ
        HAVING COUNT(*) >= 1
) si
ON s.lo_port = si.lo_port
AND s.lo_1_matice = si.lo_1_matice
AND s.se_typ = si.typ
SET s.se_porucha = true;
...
```

Kapitola 5

Testování

V této kapitole jsou shrnuty metody testování navržené aplikace a zjištěné výsledky. Testování probíhalo ve dvou fázích, kterému odpovídají i dvě hlavní sekce kapitoly. Na závěr je zde uvedeno zhodnocení celého testování.

5.1 Prvotní testování uživatelského rozhraní

Aplikace byla během vývoje testována průběžně. První větší test a dizkuze probíhala mezi zaměstnanci zadávající firmy. Cílem bylo otestovat základní princip zobrazení dat a jejich jasné a rychlé pochopení. V tomto testování jsem se chtěl zaměřit především na tři hlavní otázky:

1. **Přihlašování**- jelikož se jedná o aplikaci běžící na tak osobním zařízení jako je telefon, rozhodl jsem se implementovat přihlášení pouze při jejím prvním spuštění, případně po manuálním odhlášení. V rámci testování jsem si chtěl ověřit, zda je tento způsob podle uživatelů bezpečný a vhodný. Ukázaly se ovšem celkem rozdílné názory. Závěrečné slovo měl pan ředitel, který jasně a nekompromisně rozhodl o přihlašování při každém spuštění. Proto jsem se rozhodl alespoň realizovat pomocné mechanismy v podobě předvyplněného uživatelského jména, jak je popsáno v části implementace 4.

2. **Vyhledávání**- při implementaci jsem uvažoval, zda jsou uživatelé schopni rychle nalézt požadované zařízení. Tzn. jak dobře znají svoje objekty a ví kde se co nachází. Nakonec zde nemuselo být nikterak složité testování a dotazování, ale spíše ujasnění si, že uživatel bude v minimálním počtu případů hledat jedno konkrétní zařízení a pokud bude, tak se maximálně napodruhé „trefí“ do správné sekce. Zvažoval jsem implementaci vyhledávání, ale díky tomuto zjištění jsem byl ujištěn, že je to zbytečné.

3. **Seznam zařízení**- po otevření jedné sekce se zobrazí uživateli seznam všech zařízení v ní. U každého zařízení byla, kromě názvu, uvedena první měřená hodnota, aby uživatel bez nutnosti otevřít si detail zařízení viděl nejpodstatnější informaci hned. Jak se ale na reálných datech ukázalo, některé zobrazované texty byly příliš dlouhé a celkově se tak řádky stávaly nepřehledné. I když byl tento nápad zpočátku chválen, nakonec se vedení firmy rozhodlo tuto informaci zde neposkytovat.

5.2 Závěrečné testování

Závěrečné testování probíhalo ve dvou fázích. Nejdříve byla finální aplikace poskytnuta zaměstnancům zadávající firmy a po jejich schválení a provedených úpravách byla aplikace předvedena vybraným koncovým zákazníkům. Pro potřeby testování, bylo nutné připravit všechny podklady, patřící k jednotlivým zákazníkům. Tedy zadat do databáze informace o jejich lokalitách. Dále bylo nutné zadat tyto lokality do programu pro převod naměřených hodnot do databáze. Tento převod musel po několik dní na serveru běžet, abych získal relevantní množství naměřených dat. Samotné testování rozhraní by bylo možné i bez konkrétních naměřených hodnot, ale chtěl jsem zákazníkům poskytnout již plně funkční aplikaci.

Kromě testování uživatelského rozhraní byla aplikace také testována na její funkčnost. Především aktuálnost zobrazovaných informací.

Testování v zadávající firmě

Pracovníkům firmy byl poskytnut mobilní telefon s nainstalovanou aplikací. Oproti prvotnímu testování, kdy jim aplikace byla víceméně pouze krátce představena, teď měli možnost s ní déle pracovat a zkoušet všechny její funkce. V průběhu celého testování jsem byl přítomen, abych jim mohl odpovídat na případné otázky. Také jsem sledoval jejich reakce a chování. Posléze jim byl poskytnut dvou stránkový dotazník, který lze najít v příloze (C). Dotazník obsahuje 4 typy otázek. Otázky se slovní odpovědí, s možností volby *a*no – *n*e, s možností volby více odpovědí a hlavně s volbou na stupnici 1 – 5, kde 1 je nejlepší.

Z výsledků zjištěných pomocí dotazníku jsem našel jako nejslabší místo aplikace způsob nastavování rozsahu grafu. Jelikož mi ale většina pracovníků označila jako preferovaný mobilní systém Windows Phone, tak tento výsledek plně chápu. Zvolil jsem totiž způsob *Date pickeru*, který je typický právě pro iOS. V další verzi bych se zaměřil pravděpodobně na jiný způsob výběru data, nejspíše pomocí kalendáře. Pro technickou aplikaci je to vhodnější. Dalším výsledkem této části testování byl požadavek na část aplikace zobrazující historické hodnoty. Tento závěr jsem ale již prezentoval v části implementace 4.

Z chování pracovníků při práci s aplikací jsem si všiml, že se například snažili klikat na části, které nejsou interaktivní. Třeba celkovou sumu u sumační tabulky, což je pouze textová informace. Toto chování mě mírně mátlo, ale domnívám se, že pouze zkoušeli zda to „něco neudělá“.

Celkově byly ohlasy pozitivní a práce s aplikací jim připadala intuitivní.

Testování u zákazníků

Před testováním u koncových zákazníků jim byla aplikace krátce představena a poté poskytnuta k hlubšímu zkoumání. Během této části se mohli ptát na různé informace a nejasnosti. Na závěr jim byl poskytnut dotazník, stejně jako v případě testování ve firmě.

Jak již bylo řečeno v úvodu, zákazníci pro testování byli předem pečlivě vybráni a následně osloveni s žádostí. S připravenými podklady, tedy v databázi byly uloženy všechny jejich lokality, jsem se vydal postupně k těmto zákazníkům:

- Mikroregion Domašovsko
- Svazek obcí aglomerace Dolní Lhota
- Slovácké vodárny a kanalizace a.s.

Pan **starosta obce Domašov** byl první testovaný, který používá mobilní platformu iOS a to, jak telefon iPhone, tak také tablet iPad. S ovládáním aplikace tedy neměl žádné problémy a výhrady což se projevilo i v dotazníku kdy většinu částí týkající se rozhraní označil známkou 1, tedy nejlepší. Při testování u něj v kanceláři jsme zároveň porovnávali zobrazované informace v mobilní aplikaci s desktopový programem. Také jsem při vzájemné diskuzi zjistil, že se přes tablet každý večer připojuje vzdálenou plochou na dispečerský počítač s dispečinkem. Mobilní aplikaci by tedy ocenil. Výsledkem setkání byl mimo jiné poznatek aby nastavování parametrů řízených objektů bylo podmíněno heslem, což jsem i dříve navrhoval já, ale firma s tím úplně nesouhlasila. Navíc jsme se shodli, že by se nemuselo jednat o stejné heslo jako pro přihlášení, ale jiný potrvzovací kód. Tím by bylo umožněno některým uživatelům data pouze prohlížet a jiným také zapisovat, bez nutnosti rozlišovat na úrovni aplikace různé role uživatelů. Pan starosta totiž uváděl jako příklad jeho objekty, kdy nastavení parametrů provádí pouze on, nikoliv správci na konkrétních lokalitách.

Za **svazek obcí aglomerace Dolní Lhota** jsem mluvil se starostou obce Slopné, který má na starosti ČOV, měrné objekty a čerpací stanice. Pan starosta, coby starší člověk, nepoužívá a podle jeho slov se ani nechystá používat chytrý telefon. Na dispečerský počítač se připojuje ze svojí kanceláře přes vzdálenou plochu, případně na notebooku. Nicméně mobilní aplikaci se nebrání a pro správce, kterého na ČOV má, by ji určitě využil. Rozhraní chytrého telefonu pro něj bylo víceméně novinkou, ale dokázal velice rychle vše pochopit a v aplikaci se zorientovat. Podle jeho slov aplikace přesně kopíruje data z dispečerského programu pro PC, což bylo účelem této práce. Stejně jako pan starosta Domašova byl pro návrh ošetřit nastavení změn na řízeném objektu pod jiným heslem, jelikož všechny tyto změny provádí pouze on. Odpovědi v dotazníku se velmi podobají ostatním testovaným. Horší známku dostaly části typické pro platformu iOS, konkrétně nastavení rozsahů a data sumační tabulky a informace o poslední aktualizaci. Jinak neměl žádné větší připomínky.

Posledním testovaným byl pan **Mgr. Jan Skryja**, člen dozorčí rady Slováských vodáren a kanalizací a hlavně vedoucí ČOV Uherské hradiště. Pan vedoucí byl aplikací velice nadšen, jelikož jak jsem později zjistil, je nakloněn novým věcem a technologiím. Ačkoliv sám používá mobilní platformu Android nebylo ovládání pro něj nikterak složité a neznámé. Tento fakt se projevil také v dotazníku, kdy neměl k rozhraní aplikace žádné výhrady. Zvláště se mu líbily grafy a do budoucna navrhl, zda by bylo možné kombinovat více křivek z měřených hodnot v jednom grafu. Vyvstává zde ovšem problém, že rozhraní aplikace, tak jak je teď navrženo, toto jednoduchým způsobem nedovoluje, nicméně jako nápad a podnět do další verze je to přínosná informace.

5.3 Výsledek testování

Celkově testování dopadlo, k mé radosti, velmi pozitivně. Všichni uživatelé se shodli na tom, že by si na tabulkové zobrazení hodnot rychle zvykli a jako okamžitý nástroj k zjištění potřebných informací by to pro ně tedy mělo význam.

Nejslabším místem uživatelského rozhraní se ukázalo nastavování rozsahu data u grafů a nastavení data u sumační tabulky. Někteří uživatelé měli také problém s vyčtením času poslední aktualizace dat na serveru. Všechny tyto věci jsou řešeny stylem typickým pro zvolenou platformu iOS a neznalost těchto zvyklostí u uživatelů používající jiné platformu se dá tedy předpokládat.

Na základě testování byla do aplikace přidána sekce zobrazující historické hodnoty zapsané v databázi. Také testování přesvědčilo firmu ošetřit zápisy nových hodnot na server heslem, různým od přihlašovacího.

Kapitola 6

Závěr

Cílem této práce bylo vymyslet způsob univerzálního zobrazaní dat z automatizovaných vodárenských objektů pro firmu Speco control s.r.o., navrhnout všechny nutné prostředky pro realizaci a následně vytvořit mobilní aplikaci na platformě iOS.

K dosažení výsledku bylo potřeba nastudovat princip automatizace a řízení vodárenských objektů, způsob ukládání měřených hodnot na serveru firmy a v neposlední řadě také činnost současného dispečinku pro stolní PC. Ze získaných znalostí jsem dokázal navrhnout vhodné univerzální řešení pomocí tabulkových zobrazení.

Pro následné vytvoření mobilní aplikace jsem musel nastudovat principy tvorby uživatelských rozhraní na platformu iOS, správně pochopit standartní funkce systému a zažité vzory. Výsledkem je uživatelské rozhraní plně respektující zvyklosti této mobilní platformy, což se ukázalo při testování, kdy uživatelé používající jiné mobilní systémy měli s některými funkcemi problémy. Zde je vidět, proč není vhodné psát aplikace univerzálně. Původně to byl jeden z požadavků firmy, ale právě argumenty, že každý uživatel je zvyklý na jiný systém ovládání a je potřeba aby rozhraní tento fakt respektovalo, je přesvědčilo o vhodnosti raději navrhnout aplikace pro každou platformu zvlášť.

Součástí této práce bylo také vytvořit back-end v podobě API a databáze na straně serveru pro správnou funkčnost aplikace. Jelikož API není vázána na konkrétní jazyk či systém, je možné ji využít v budoucnu pro mobilní aplikaci na jiné platformy, ale také pro webové rozhraní, které by firma chtěla vytvořit.

Závěrečné testování dokázalo, že mobilní aplikace plní svoji funkci, coby doplněk k současnému dispečinku pro stolní PC. Umožňuje si zjistit nejpodstatnější informace kdykoliv a kdekoliv. K navrženému rozhraní nebyly zjištěny žádné větší nedostatky, pouze neznalost určitých zvyklostí zvolené platformy. Především nastavování data pomocí „otočných válců“ *d*atepicker a *p*icker. Dále manuální aktualizace dat „zatažením“ obrazovky směrem dolů a s tím související vyčtení informace o poslední aktualizaci dat na serveru, která se zde nachází. Pro uživatele to byl úplně nový způsob interpretace měřených hodnot oproti dispečinku pro PC, ale dokázali si na něj velice rychlo zvyknout a pochopit.

Přínos práce pro zadávající firmu je především ve vytvořené API a obecnému způsobu jakým univerzálně zobrazovat stejné data, jako ve složitě a ručně vytvářeném dispečinku pro stolní PC. Dalším krokem bude vytvořit mobilní aplikaci pro platformu Android a také webovou variantu.

Literatura

[1] *Úvod do JSON* [online]. Dostupné z: <http://www.json.org/json-cz.html>.

[2] Adam Fendrych: *Uživatelské testování návrhů webu* [online]. 2009. Dostupné z: <http://www.lupa.cz/clanky/uzivatelske-testovani-navrhu-webu/>.

[3] Apple Inc.: *About the iOS Technologies* [online]. 2014. Dostupné z: <https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html>.

[4] Apple Inc.: *IOS Human Interface Guidelines* [online]. 2014. Dostupné z: <https://itunes.apple.com/us/book/ios-human-interface-guidelines/id877942287?mt=11>.

[5] Apple Inc.: *The Swift Programming Language* [online]. 2014. Dostupné z: <https://itunes.apple.com/us/book/swift-programming-language/id881256329?mt=11>.

[6] Apple Inc.: *WWDC 2014 Videos* [online]. 2014. Dostupné z: <https://developer.apple.com/videos/wwdc/2014/>.

[7] Collins, K.: *PLC Programming for Industrial Automation.* [b.m.]: Triangle Research International, 2006. Dostupné z: <http://runplc.com/wp-content/uploads/Books/plcprogramming.pdf>.

[8] Gamma, E.: *Design Patterns: Elements of Reusable Object-Oriented Software.* USA: Addison- Wesley, 1994. ISBN 0-201-63361-2.

[9] Mulloy, B.: *Web API Design: Crafting Interfaces that Developers Love.* [b.m.]: apigee, 2012. Dostupné z: <https://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf>.

[10] Paul Hegarty: *[youtube kanál] Stanford University - Developing iOS 7 Apps for iPhone and iPad* [online]. 2014. Dostupné z: <https://www.youtube.com/playlist?list=PL9qPUrlLU4jSlonxFqhWKBu2c_sWY-mzg>.

[11] Paul Hegarty: *[iTunes] Developing iOS 8 Apps with Swift* [online]. 2015. Dostupné z: <https://itunes.apple.com/us/course/developing-ios-8-apps-swift/id961180099>.

[12] Wikipedia: *API* [online]. Dostupné z: <http://cs.wikipedia.org/wiki/API>.

[13] Wikipedia: *Representational State Transfer* [online]. Dostupné z: <http://cs.wikipedia.org/wiki/Representational_State_Transfer>.

Příloha A

Obsah CD

- **/README.txt** – soubor s informacemi o obsahu CD
- **/SpecoApp/src** – projekt programu Xcode se zdrojovými soubory
- **/SpecoApp/Icons** – ikonky a další grafické prvky aplikace
- **/SpecoAPI/Scripts** – skripty API
- **/SpecoAPI/DB** – struktura SQL databáze + testovací data
- **/Testovani** – oskenované vyplněné dotazníky z testování
- **/TZ** – technická zpráva ve formátu PDF
- **/TZ/src** – zdrojové soubory technické zprávy v LaTeXu
- **/Video** – prezentační video
- **/Plakat** – plakát k aplikaci

Příloha B

API manuál

Všechny skripty vrací výsledek ve formátu JSON. Jedná se o pole záznamů (`[{název : hodnota}, {název : hodnota}, ...]`), případně prázdné pole (`[ ]`).

`login.php`

Skript pro přihlášení vrací v případě správně zadaného uživatelského jména a hesla příslušný záznam z tabulky *u*sers. Jinak vrací prázdné pole ve formátu JSON.

Parametr	**Popis**
`us_name`	uživatelské jméno
`us_pass`	heslo

`usersLocations.php`

Vrací seznam lokalit z tabulky *l*okalita, které patří k uživateli se zadaným identifikačním číslem.

Parametr	**Popis**
`us_id`	identifikační číslo uživatele

`locationSections.php`

Vrací názvy sekcí a zařízení v sekci z tabulky *s*ekce patřící do zvolené lokality. Kontroluje se také, zda lokalita patří k uživateli se zadaným identifikačním číslem.

Parametr	**Popis**
`us_id`	identifikační číslo uživatele
`lo_port`	port na kterém se lokalita nachází
`lo_1_matice`	číslo první matice na které se lokalita nachází

`locationIds.php`

Vrací seznam měřených veličin z tabulky *i*d patřící do zvolené lokality. Kontroluje se také, zda lokalita patří k uživateli se zadaným identifikačním číslem.

Parametr	Popis
us_id	identifikační číslo uživatele
lo_port	port na kterém se lokalita nachází
lo_1_matice	číslo první matice na které se lokalita nachází

lastUpdate.php

Vrací datum posledního zápisu nové hodnoty do databáze. V případě, že je zadáno pouze identifikační číslo uživatele, hledá se poslední zapsaná hodnota na libovolné lokalitě patřící k danému uživateli. Pokud je zvolena konkrétní lokalita pomocí čísla portu a první matice, je hledán poslední záznam patřící k této lokalitě.

Parametr	Popis
us_id	identifikační číslo uživatele
lo_port	port na kterém se lokalita nachází
lo_1_matice	číslo první matice na které se lokalita nachází

historicValues.php

Vrací historické záznamy z tabulky *m*ereni pro konkrétní měřenou veličinu ve zvoleném časovém rozsahu.

Parametr	Popis
lo_port	port na kterém se hodnota nachází
lo_matice	číslo matice na které se hodnota nachází
id_pozice	pozice bitu na kterém se hodnota nachází
datum_od	počáteční datum (yyyy-MM-dd HH:mm:ss)
datum_do	koncové datum (yyyy-MM-dd HH:mm:ss)

lastHistoricValues.php

Vrací poslední záznam v měsíci z tabulky *m*ereni pro konkrétní měřenou veličinu ve zvoleném časovém rozsahu.

Parametr	Popis
lo_port	port na kterém se hodnota nachází
lo_matice	číslo matice na které se hodnota nachází
id_pozice	pozice bitu na kterém se hodnota nachází
datum_od	počáteční datum (yyyy-MM-dd HH:mm:ss)
datum_do	koncové datum (yyyy-MM-dd HH:mm:ss)

writeValueToServer.php

Zapsání nové hodnoty na server. Lze měnit pouze u veličin označených v databázi atributem id_param.

Příloha C

Dotazník k testování

MOBILNÍ APLIKACE SPECO CONTROL

1) **Struktura aplikace**
   - Srozumitelnost tabulkového zobrazení dat?

     Nejlepší 1 2 3 4 5 *nejhorší*

2) **Informace o poruchách**
   - Jsou informace o poruchách dostatečné a včasné?

     Nejlepší 1 2 3 4 5 *nejhorší*

   - Slovní názor na zvýraznění názvu lokality/sekce/zařízení na kterém se vyskytuje porucha:

   - Setřídit poruchové hodnoty nahoru? *ano* *ne*

3) **Dělení hodnot a zařízení do sekcí**
   - Odpovídá rozdělení ve stávajícím dispečinku pro PC?

     Nejlepší 1 2 3 4 5 *nejhorší*

4) **Aktualizace**
   - Zjištění informace o poslední aktualizaci dat na serveru.

     Nejlepší 1 2 3 4 5 *nejhorší*

   - Srozumitelnost manuální aktualizace dat.

     Nejlepší 1 2 3 4 5 *nejhorší*

5) **Grafy**
   - Přehlednost grafu, schopnost vyčíst požadované informace.

     Nejlepší 1 2 3 4 5 *nejhorší*

   - Srozumitelnost nastavení rozsahu grafu.

     Nejlepší 1 2 3 4 5 *nejhorší*

6) **Sumace**
   - Přehlednost a srozumitelnost sumační tabulky.

     Nejlepší 1 2 3 4 5 *nejhorší*

   - Srozumitelnost nastavení data.

     Nejlepší 1 2 3 4 5 *nejhorší*

7) Řazení lokalit podle vzdálenosti od aktuální polohy? *ano* *ne*
8) Sekce (část aplikace) s historickými hodnotami jednotlivých měřených hodnot, především poruch? *ano* *ne*

9) iOS Android Windows phone

10) Telefon Tablet

11) Ostatní:

Jméno:

Lokalita:

Datum:

Podpis: