Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

BI aplikace pro iOS s napojením na existující ERP systém RIS2000

William Dupkala

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. William Dupkala

Informatika

Název práce

BI aplikace pro iOS s napojením na existující ERP systém RIS2000

Název anglicky

BI application for iOS connected to an existing ERP system RIS2000

Cíle práce

Cílem práce bude navrhnout funkční aplikaci pro operační systém iOS. Aplikace
bude vytvořena v jazyce SWIFT a bude vyvíjena v prostředí XCODE. Aplikace bude zaměřena na oblast Business Intelligence, aplikace bude napojena na existující ERP systém RIS2000 společnosti SAUL IS spol. s r. o.

Metodika

Metodika práce je založena na podrobné analýze existujících řešeni. Ta bude provedena na základě studia odborné literatury a již existujících softwarových řešení. Znalosti nabyté studiem budou zhodnoceny a na jejich základě bude definován současný stav řešení, jejich klady a nedostatky. Na základě výsledků rešerše bude aplikace vytvořena.

Doporučený rozsah práce

60

Klíčová slova

BI, Business Intelligence, SWIFT, aplikace, RIS2000, iOS

Doporučené zdroje informací

BASL, Josef; BLAŽÍČEK, Roman. Podnikové informační systémy : Podnik v informační společnosti. 3. aktualizované a doplněné vydání. Praha. 2012. Grada Publishing. s.128. ISBN 978-80-247-4307-3.

BRUCKNER, T. *Tvorba informačních systémů : principy, metodiky, architektury.* Praha: Grada, 2012. ISBN 978-80-247-4153-6.

KADLEC, V. *Agilní programování : metodiky efektivního vývoje softwaru.* Brno: Computer Press, 2004. ISBN 80-251-0342-0.

KOCH, M. et al. Management informačních systémů. Brno: CERM, 2010. 171 s. ISBN 978-80-214-4157-6.

MONK, Ellen F.; WAGNER, Bret J. Concepts in enterprise resource planning. 4. edition. Boston, United States of America. 2013. Course Technology. ISBN 978-1-111-82039-8.

POUR, J. – ŠEDIVÁ, Z. – GÁLA, L. *Podniková informatika : počítačové aplikace v podnikové a mezipodnikové praxi.* Praha: Grada Publishing, 2015. ISBN 978-80-247-5457-4.

Předběžný termín obhajoby

2016/17 LS – PEF

Vedoucí práce

Ing. Josef Pavlíček, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 28. 03. 2017

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "BI aplikace pro iOS s napojením na existující ERP systém RIS2000" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31.03.2017 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Josefu Pavlíčkovi, Ph.D. za připomínky a odborné vedení.

BI aplikace pro iOS s napojením na existující ERP systém RIS2000

Souhrn

Diplomová práce se zabývá vývojem aplikace pro iOS, napojený na existující ERP systém RIS2000. Práce se dělí na dvě části. V první části, v teoretické části, se práce věnuje pojmu ERP a BI a popisuje konkrétní ERP, software RIS2000 a BI nadstavbu nad tímto programem, TOP-RIS. Dále se práce zabývá operačním systémem iOS, co je potřeba pro vývoj a v jakém jazyce se vyvíjí aplikace pro tento OS. V druhé, praktické části, se práce zaměřuje na samotný vývoj aplikace. Popisuje implementace serverové vrstvy do provozu, napojení na ERP RIS2000 a tvorbu klientské aplikace v Xcode s následným testováním.

Klíčová slova: BI, Business Intelligence, SWIFT, aplikace, RIS2000, iOS

BI application for iOS connected to an existing ERP system RIS2000

Summary

The diploma thesis deals with development of an application for iOS, connected to an existing ERP system RIS2000. This thesis is divided into two parts. In the first part, the theoretical part, the thesis looks into the concept of ERP and BI and describes an example of an ERP, software RIS2000 and a BI connected this program, TOP-RIS. The thesis also describes the operation system iOS, what is needed for development and in what language are applications developed for this OS. The second part, the practical part, of this thesis focuses on the actual application development. It describes implementation of the server layer, connection to ERP RIS2000 and the making of the client application in Xcode with subsequent testing.

Keywords: Business Intelligence, iOS, BI, Swift, application, RIS2000

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době mobilní zařízení hrají důležitou roli v pracovním prostředí podniků. Pomocí těchto přístrojů jednotlivci snáze spolupracují a dosahují společných cílů podniku. S modernizací mobilních zařízení se navyšují výpočetní možnosti a nabízejí se nové způsoby s mobilními aplikacemi, které bývaly myslitelné pouze na desktopových počítačích.

V teoretické části této práce se budu věnovat nejdříve oblasti ERP (Enterprise Resource Planning). Vysvětlím pojem ERP a podle jakého členění tyto systémy budu dělit. Dále bych chtěl ukázat teorii procesu implementace ERP do podniku, vysvětlit tím, jak velký rozsah a důsledky má takový systém pro podnik. Jako konkrétní příklad takového ERP systému představím program RIS2000, modulární ERP od firmy SAUL IS spol. s r. o. Přes čtyři roky jsem pracoval v této české firmě nabízející tento podnikový systém. Při psaní této práce vycházím z praktických zkušeností a z teoretických poznatků. Tento systém bude jádrem dat, se kterým bude výsledná aplikace pracovat. Nadstavbou nad ERP jsou nástroje BI (Business Intelligence). Vysvětlím pojem BI a představím další software od SAUL IS spol. s r. o., BI nástroj TOP-RIS. Tento program je vzorem pro výslednou aplikaci v tom, jaké funkce by měla aplikace nabízet.

V další části teorie představím operační systém iOS od firmy Apple Inc. Pro tento operační systém budu vyvíjet příslušnou aplikaci. Pro vývoj aplikací pro iOS jsou potřeba určité nástroje, které postupně popíši. Cílová aplikace je vyvíjena v programovacím jazyce Swift. Jazyk Swift je poměrně nový a v další části práce ho představím.

Poslední část teorie je určena pro shrnutí celé teoretické části práce. Na základě tohoto shrnutí určím požadavky na vývoj takové aplikace a požadavky na samotnou aplikaci, co by měla umět.

Výsledkem této práce bude aplikace napojená na ERP systém RIS2000, která musí být schopna ukazovat smysluplná data. V praktické části této práce nejprve sestavím šablony pro výstupní sestavy a následně vytvořím tuto aplikaci. Výslednou aplikaci v závěru budu testovat, a to podle více testovacích scénářů, na základě kterých pak zjistím, zda aplikace funguje korektně.

1 Cíl práce

Cílem práce bude navrhnout funkční aplikaci pro operační systém iOS. Aplikace bude vytvořena v jazyce Swift a bude vyvíjena v prostředí Xcode. Aplikace bude zaměřena na oblast Business Intelligence, aplikace bude napojena na existující ERP systém RIS2000 společnosti SAUL IS spol. s r. o.

2 Metodika

Metodika práce je založena na podrobné analýze existujících řešeni. Ta bude provedena na základě studia odborné literatury a již existujících softwarových řešení. Znalosti nabyté studiem budou zhodnoceny a na jejich základě bude definován současný stav řešení, jejich klady a nedostatky. Na základě výsledků rešerše bude aplikace vytvořena.

3 ERP a BI

Podnikové informační systémy, v západních zemích známy jako ERP systémy, jsou dnes už nezbytnou součástí firem. Jejich cílem je dosažení lepšího firemního hospodářského výsledku - pomáhají urychlovat workflow firemních procesů. V této kapitole se budu nejdříve věnovat členění ERP systémů. Dále se budu věnovat tomu, co přesně ERP znamená a ukáži význam tohoto druhu informačního systému na konkrétní metodice implementace ERP do daného podniku. Konkrétním případem ERP je software RIS2000, který v další části popíši.

V druhé části této kapitoly se budu věnovat termínu BI a určitému příkladu BI, programu TOP-RIS, který je manažerskou nadstavbou ERP systému RIS2000.

3.1 ERP

Termínu ERP a aplikacím tohoto typu předcházelo několik vývojových stádií (kterým v této práci se detailněji nebudu věnovat) pro něž byl trend ke stále silnější provázanosti funkcí a tomu odpovídajících programových modulů. ERP software pokrývá rozhodující část podnikového řízení. V praxi ERP jsou nasazovány od počátků devadesátých let minulého století. Pojem ERP můžu tedy definovat jako typ aplikace, resp. aplikačního software, který umožňuje řízení a koordinaci všech disponibilních podnikových zdrojů a aktivit. Mezi hlavní vlastnosti ERP patří schopnost automatizovat a integrovat klíčové podnikové procesy, funkce a data v rámci celé firmy [1].

ERP systémy můžu definovat i jako transakční systémy, kam uživatelé zadávají a využívají data v reálném čase. Tyto systémy (resp. databáze) slouží pak jako zdroj pro další kategorie informačních systémů, zejména pro aplikace typu BI, které využívají data pro nejrůznější obchodní, marketingové, personální sestavy a reporty pro vyšší management firem.

3.1.1 Členění ERP systémů

Na trhu existuje mnoho firem, které jsou odlišné v mnoha ohledech. Pro účel této práce bude vhodné rozčlenit ERP systémy podle velikosti firmy, zde myšleno počet zaměstnanců. Takové členění ERP systému je velmi důležité pro určení formy implementace různých řešení a jejich provozování. V kontextu EU uvažuji toto členění [2]:

- malé podniky - do 50 zaměstnanců
- střední podniky - od 50 do 250 zaměstnanců

- velké podniky - nad 250 zaměstnanců

ERP pro malé podniky Menší podniky nedisponují kvalifikačním potenciálem k implementaci a provozu individuálního aplikačního software (IASW). Nejvhodnější pro tento typ podniků se jeví implementace typového aplikačního software (TASW), nebo-li parametrizovaný informační systém. Takový systém nabídne již naprogramované jednotlivé moduly ERP aktualizované dle právních předpisů dané země. Pro malé podniky v České republice se jeví nejideálnější modul účetnictví, neboť každý podnik má podle § 1, Zákona č. 563/1991 Sb. o účetnictví, vést účetnictví tak, aby účetní závěrka sestavená na jeho základě podávala věrný a poctivý obraz předmětu účetnictví a finanční situace účetní jednotky.

ERP aplikace lze členit i v závislosti na tom, jak pokrývají všechny klíčové oblasti řízení. Dle tohoto hlediska je možné vymezit další členění ERP systémů a to podle funkcionality. Jedna skupina, která je takto členěná, se nazývá Lite ERP. Jedná se o odlehčené verze vyšších ERP systémů a jsou určené především pro malé a střední podniky [1].

Nynější dlouhodobější trend dodavatelů ERP systému je pokrýt tento sektor menších podniků. Kolem roku 1998, dle žebříčku Fortune, 500 největších firem na světě již implementovalo ERP systémy [3]. Dodavatelé ERP systémů se zaměřili na středně velké podniky. Pro tento segment například společnost SAP vyvinula SAP Business All-in-one, jednoduchý balíček modulů ERP předdefinovaný pro jednotlivé odvětví průmyslu. Jelikož se jednalo o TASW speciálně parametrizovaný pro jednotlivá odvětví, implementace do podniků byla velmi rychlá.

V dnešní době se vyskytuje pojem SaaS (Software-as-a-service). Aplikační server a databázový server není přítomný u zákazníka, nýbrž u dodavatele ERP. Zde odpadá potřeba menšího podniku nakoupit si hardware potřebný k provozu ERP systému. Na základě předplatného s pravidelnými platbami dodavatel ERP má možnost poskytnout své služby levněji. Neodpadá však potřeba parametrizovat ERP dle požadavků firmy ani potřeba vyškolení uživatelů, kteří IS budou obsluhovat.

ERP pro střední podniky Zatímco v malém podniku by stačil jen jeden základní modul, pro zefektivnění podnikových procesů u podniku středního toto řešení nemusí stačit. Jako jádro ERP můžu vybrat modul účetnictví, ale pro podchycení ostatních podnikových procesů je potřeba na tento modul navázat moduly dalšími. Na rozdíl od podniku velkého, není potřeba mnoho navázaných modulů. Klasickým příkladem

navazujících modulů je: sklady a výroba, správa majetku nebo CRM (customer relation management).

Jak bylo výše uvedeno, i zde se dá uvažovat o řešení zvaném Lite ERP. Tato řešení jsou však vytlačována dodavateli ERP, kteří dodají celý svůj ERP, ale moduly, ke kterým nebyla zakoupena licence, jsou blokovány. Tudíž jedna z nevýhod Lite ERP, omezené možnosti následného rozšiřování systému, je tímto vyřešena.

ERP pro velké podniky Tento segment již potřebuje systém, který podporuje veškeré podnikové procesy. Pomocí členění ERP systémů dle funkcionality bych narazil na skupinu ERP systémů, které se nazývají All-in-one. Tyto systémy představují rozsáhlé aplikační software, které pokrývají celé podnikové řízení. Výhodou je komplexní funkcionalita, nevýhodou je však velmi vysoká složitost řešení a nároky na parametrizaci [1].

Pro lepší představu a pochopení pojmu ERP, popíši v následující kapitole proces implementace ERP do podniku, kde je vidět rozsah a důležitost takového systému v podnikové sféře.

3.1.2 Implementace ERP do podniku

V oblasti IT se uvažuje pojem implementace jako nejen pojem samotné fáze implementace ERP systému do podniku, ale často se chápe jako celý postup řešení aplikace, celý jeho životní cyklus - od fáze úvodních studií až po fázi samotného provozu na pracovních stanicích klienta. Nyní si rozeberu celý životní cyklus řešení ERP. Nadále je vhodné poznamenat skutečnost, že mnozí autoři pojmenovávají fáze životního cyklu řešení aplikace jinak, ale pro zjednodušení vezmu modifikaci MMDIS (Multidimensional Management and Development of Information System) při TASW[4].

I. fáze - úvodní studie Cílem této fáze je vytvořit jasné zadání výstupního produktu s využitím ERP. Definuje jeho vizi, rozsah a přístup k řešení. Naprosto mandatorní v této fázi je podpora ze strany vedení podniku, kde bude ERP využíván. S pomocí této podpory se v úvodní studii zjistí či ověří informace [4]:

- Jaké jsou byznys cíle projektu? Zde zjišťuji záměry vlastníků.
- Jakou hodnotu projekt přinese, vyplatí se investice? Naleznu strategické cíle podniku.
- Současný stav využívání IS/IT v podniku.

- Stavy procesů v podniku.
- Kvalifikační potenciál personálu v podniku.

K poznání potřeb podniku a uvědomění si vlastních možností jsou tyto otázky důležité pro další zavádění ERP. V této fázi se doporučuje spolupracovat i s poradenskou firmou, která připraví takový materiál umožňující upřesnit formulaci zadání pro následný výběr ERP. Úvodní studie musí být na závěr schválena ze strany investorů. Konkrétně rozsah řešení a požadavky na řešení, definované řídící orgány projektu a nákladový a časový harmonogram. A v neposlední řadě, v této fázi je vybrán konkrétní TASW a jeho implementátor.

Pod-fáze - vytvoření řešitelského týmu Podle Basla [2] se v první fázi životního cyklu nachází ještě krok vytvoření řešitelského týmu. Zde je nominován vedoucí projektu, který tento řešitelský tým řídí. Tento vedoucí má za úkol koordinovat znalosti i dovednosti jednotlivých členů týmu, aby byla na závěr splněna jeho zodpovědnost - dodržování základních termínů a limitů v rámci vyjednaného a již schváleného rozpočtu vedením firmy. Nadále by neměl zapomenout na důslednou dokumentaci všech kroků.

V průběhu implementace ERP existují dvě skupiny pracovníků, kteří musí úzce spolupracovat. Jedna skupina, ze strany dodavatele ERP, se věnuje vlastnímu nasazení ERP do podniku, správné nastavení a uvedení do provozu - IT konzultanti. Druhá skupina se zaměřuje na potřeby podniku - business konzultanti.

II. fáze - globální analýza a návrh V této fázi dochází k upřesnění požadavků investorů k dodavateli ERP systému. Účelem tohoto upřesnění je popsat nutné úpravy a doplnění funkcionality TASW (pokud je potřeba), aby vyhovovala potřebám podnikových procesů. Součástí této fáze je i upřesnění technologické architektury systému, rozdělení celého řešeného systému na samostatně realizovatelné části, či přírůstky. Věnujme se nyní, jakou náplň má tato fáze [4].

- Je potřeba specifikovat funkce. Vytvořit přehled případů užití mapovaných funkcí na moduly TASW.
- Vytvořit přehled procesních změn v podniku před nasazením TASW.
- Ohledně datových potřeb je třeba vymezit zdroje dat pro migraci ze stávajících aplikací. Přičemž se nesmí zapomenout na návrh způsobu zálohování a obnovení dat, pro případ nouze.

- Navrhuje se nasazení TASW do aplikační architektury podniku, navrhuje se architektura HW.
- Pokud je potřeba, dokončuje se celkový návrh úprav uživatelského rozhraní.
- Plánuje se školení budoucích uživatelů.

V této fázi se i nadále zapojuje vedení společnosti do plánování implementace. Zapojují se i klíčoví koneční uživatelé, jež svými připomínkami zasahují do rozhodování o změnách na straně podnikových procesů, tak na straně TASW. Velmi důležité je důsledné řízení změn TASW, neboť každá změna funkcionality je problémem pro nasazení budoucích upgradů TASW.

III. fáze - detailní analýza a návrh Tato fáze je důležitá v tom, že nyní se nasazuje testovací prostředí a jsou stanoveny testovací případy a postupy. Nadále se zde tvoří detailní specifikace definované části (nebo přírůstku) a detailní návrh jeho realizace. Pro každou jednotlivou část se tvoří tato specifikace [4].

- Popisuje se zde nastavení parametrů části TASW, která je předmětem realizace přírůstku. I pro každou použitou standardní funkci je vytvořen detailní popis nastavení parametrů systému.
- V oblasti dat se určí zdroj dat pro výchozí naplnění databáze, mapování na datový model TASW a návrh migračních postupů pro tato data.
- Popisuje se i nastavění systémových číselníků (např. systémový kalendář) a konstantních hodnot.
- Nasazuje se zde TASW do vývojového prostředí. Nastavují se zde hodnoty parametru dle funkční specifikace, ověřují se standardní funkcionality na prototypech.
- Ustanovuje se harmonogram instalace HW (pouze pokud je v požadavcích).
- Definuje se detailní plán školení a navrhují se školící materiály.

Potvrzují se zde kvality konzultantů, kteří musí posuzovat nutnost potřeb customizačních změn a případně projednat se zadavateli projektu dopady změn těchto customizací na TASW. Konzultant musí být schopen zasadit případnou změnu do TASW tak, aby budoucí upgrade nevyžadoval novou implementaci těchto změn.

IV. fáze - implementace Nejdůležitější fází v životním cyklu aplikace je implementace. Zde se už instaluje konkrétní TASW do prostředí podniku a realizuje se migrace dat a přizpůsobení byznys procesů podniku funkcím TASW. Dochází ke kompletaci veškeré dokumentace k projektu [4].

- V podniku se upravují byznys procesy podle detailního návrhu implementace TASW.
- Dokončuje se testování dat a zabezpečují se přístupová práva k datům.
- Parametrizuje se TASW, testuje se funkčnost parametrů v prostředí podniku.
- Zajišťují se a instalují se operační systémy, DBMS[1] a potřebné infrastruktury k bezchybnému fungování TASW.
- Předává se dokumentace k TASW pro administrátory a správce aplikačního SW.
- Přidělují se přístupová práva v TASW pro definované role.

K bezchybné implementaci dochází tehdy, pokud je důkladně odladěný a otestovaný systém, kde jsou včasně odhalené chyby a odchylky od předem dané specifikace. Záleží zde i mimo jiné na zkušenostech konzultantů, kteří implementaci provádí.

Způsoby implementace ERP

Postupů pro zavedení ERP je více, které se od sebe liší rychlostí či zaváděcí metodou. Mezi nejčastější zaváděcí strategie jsou podle Kocha [5]:

Souběžná strategie Informační systém je zaveden souběžně na všech pracovištích najednou. Tento postup je vhodné použít při zavádění jednodušších IS, které nevyžadují náběhovou fázi zavádění (složitá školení, konverzi dat z předchozích IS).

Pilotní strategie Informační systém se zavede na jednom pracovišti, které je na tuto činnost připraveno. Po zavedení probíhá ověřovací provoz a posléze zde probíhá školení pracovníků ostatních pracovišť. Tento způsob je vhodný pro zavádění odlišných IS, které vyžadují rozsáhlé testování nového IS v provozních podmínkách. Toto pilotní zavádění umožňuje postupnou transformaci dat z předchozích IS. V závěru pilotní fáze dochází k zavádění IS na ostatní pracoviště, které jsou již připraveny.

[1] database management system

Postupná strategie Zavádění IS na jednotlivá pracoviště probíhá postupně, bez pilotní fáze. Rychlost zavádění je závislá na připravenosti jednotlivých pracovišť (zaměstnanci a HW/SW) a na složitosti IS. Tento způsob se jeví vhodný pro takový systém, u kterého není nutné provozní ověřování (TASW).

Nárazová strategie Strategie zavádění, kde ukončíme činnost jednoho IS a po nezbytně nutné pauze spustíme nový informační systém. Tento postup je velmi riskantní, používá se tehdy, kde souběh IS není možný.

V. fáze - zavedení do provozu Výstupem této fáze by měl být úspěšný přechod na nové řešení implementovaného TASW. Probíhá intenzivní školení nových uživatelů ERP systému. Systém je již nasazen do ostrého provozu, jsou migrována data z původního systému (nebo vytvořeny počáteční zůstatky, pokud původní systém nebyl). Zahajuje se smluvní činnost podpory uživatelů - hot-line [4].

- Podnik zahajuje fungování dle upravených byznys procesů.
- Dochází k naplnění databáze dle architektury TASW.
- Pokud bylo smluvně ošetřeno, zprovozňuje se rozhraní na ostatní aplikace.
- Zahajuje se uživatelská činnost s TASW s asistencí konzultantů.
- Na závěr se provádí konečná kontrola rozpočtu.

Pro zdárné ukončení této fáze je potřeba v úspěch proměnit mnoho proměnných. Je potřeba důsledně proškolit veškerý personál podniku, který bude TASW využívat a konzultant musí následně tento personál z nově nabytých znalostí ověřit. Velmi záleží na úspěšné migraci dat z původního systému, je potřeba si předem připravit a vyzkoušet převodové můstky pro data. Pokud neexistuje původní systém, příklad beru z nově vzniklého podniku, tak je potřeba ručně správně zadat veškerá data potřebné k úplnému provozu TASW - jedná se o počáteční zůstatky, číselníky systémové i číselníky speciální pro různé moduly a agendy. Při zavedení TASW do provozu je potřeba mít ze strany dodavatele TASW proškolené konzultanty, kteří se formou uživatelské podpory budou snažit odpovídat na případné dotazy k systému a ke specifickému chování TASW k podnikovým procesům.

VI. fáze - provoz a údržba Tato fáze je posledním v životním cyklu aplikace. Tato fáze časově trvá nejdéle a končí celkovým vyřazením TASW mimo provoz. Během této doby jsou dodavatelem TASW poskytovány informatické služby, záruční servisní

služby a servisní služby na základě předem schválené smlouvy. Je poskytováno personální, technické a materiálové zabezpečení provozu systému. V této fázi dochází i k implementaci nadstandardních změn do systému, ať již byly vytvořeny podle smluvní nabídky nebo z důvodu změn v právních předpisech státu [4].

- Databáze se již běžně užívá a aktualizuje prostřednictvím aplikace.
- Je vytvořena pravidelná záloha databáze, archivace dat.
- Běžné opravy chyb, realizace schválených změn a nasazování nových modifikací.
- Dobíhá školení stávajících uživatelů, jsou poskytnuta nová školení pro nově příchozí uživatele.
- Dodavatelem TASW jsou poskytovány pravidelné informace ohledně změn systémů, ať už v důsledcích změn právních předpisů.
- Probíhá fakturace služeb.

Fáze provoz a údržba končí rozhodnutím o odstranění systému z provozu. Ze strany dodavatele TASW z důvodu ukončení podpory již staré verze nebo ze strany odběratele z důvodu přechodu na jiný IS.

3.1.3 RIS2000

Podnikový informační systém RIS (systémové označení RIS2000) je určen pro komplexní vedení ekonomické a obchodní agendy firmy. Lze jej použít ve společnostech nejrůznějšího zaměření a velikosti. Systém RIS2000 je modulově řešený softwarový produkt, k jehož základním rysům patří zejména následující charakteristiky [6]:

- Víceletá databáze umožňující meziroční porovnávání.
- Důkladně propracovaný systém přístupových práv v několika úrovních.
- Uživatelské úpravy hotových sestav - součty, detail, třídění apod.
- Lze definovat vlastní výstupní sestavy a ukazatele.
- Export údajů resp. sestav do souborů v různých formátech (např. EXCEL) resp. do jiných aplikací.
- Veškeré operace lze provádět v cizích měnách, včetně pokladny.

- Všechny operace probíhají v reálném čase.
- Vyřešeno zpracování typu ústředí – pobočky s oboustranným předáváním dat (on-line i off-line).
- Systém je v kategorii TASW - vysoce parametrizovaný a umožňuje tak optimální konfiguraci podle potřeb konkrétního uživatele a maximální automatizaci prováděných operací.
- Všechny moduly jsou navzájem propojené a využívají společnou soustavu číselníků a parametrů.

ERP systém RIS2000 nabízí množství modulů. Pro potřebu problému tvorby aplikace budu používat pouze moduly ÚČETNICTVÍ.

RIS2000 umožňuje při dodržení určitých podmínek zpracovávat i data importovaná z jiných provozovaných aplikací obvykle prostřednictvím speciálně vytvořeného rozhraní.

Systém RIS2000 je instalován jako aplikace typu klient-server a to v kombinaci dvouvrstvé a třívrstvé architektury. Klientské stanice v tomto případě fungují jako terminály využívající terminálových služeb aplikačního serveru.

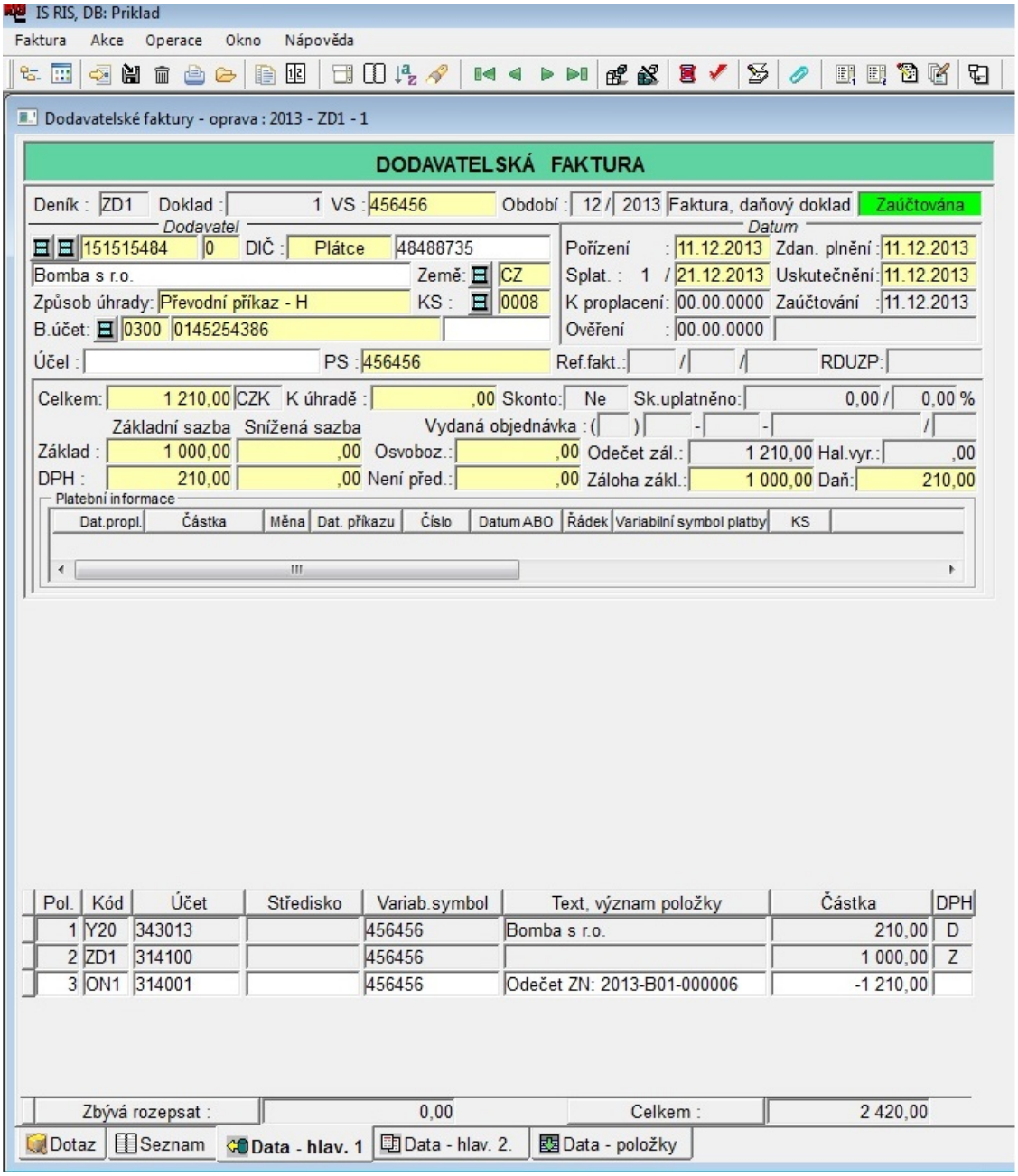
Na straně databázového serveru se předpokládá dostatečně výkonný server s relačním databázovým systémem a to buď SYBASE, ORACLE, INFORMIX nebo MS SQL Server. Jako operační systém může být použit minimálně Windows 2000 Server a výše, LINUX nebo další UNIXové OS resp. takové operační systémy, pro které jsou portovány příslušné DBMS.

Podporované verze DBMS pro RIS2000	
DBMS	Verze DBMS
Sybase	Sybase Anywhere - všechny verze od 9
	Sybase Enterprise 12.5
	Sybase Enterprise 15
Oracle	Oracle 10
	Oracle 11
Microsoft	MSSQL Server 2005
	MSSQL Server 2008
	MSSQL Server 2014
Informix (IBM)	Informix 10
	Informix 11

Tabulka 1: Podporované verze DBMS pro RIS2000 Zdroj: [6]

Jako pracovní stanice pro RIS2000 ve dvouvrstvé architektuře (tlustý klient) se předpokládá standardní PC v běžné konfiguraci odpovídající požadavkům na provoz operačních

systémů od Windows 2000, Windows XP Professional, Windows 7 po Windows 10. Vzdálené pracovní stanice, které budou připojeny prostřednictvím terminálových služeb k aplikačnímu serveru, mohou mít konfiguraci slabší (např. procesory PII, Celeron, menší paměť RAM apod.).



Obrázek 1: RIS2000 Zdroj: Vlastní

Modul ÚČETNICTVÍ V systému RIS2000 je používána celá řada číselníků a parametrů, pomocí kterých je řízen jednak chod celého systému (resp. varianty chování systému) dle požadavku uživatele a jednak vypovídací schopnosti celého systému (za-

vedení vnitropodnikového účetnictví apod.). Číselníky představují číselné pojmenování určitého prvku, jednotky, druhu v informačním systému. Obsah přiřazený k určitému číslu, či skupině čísel musí být vždy jednoznačný. Logicky zvolené číselníky umožňují nejen vyjadřovat ekonomické jevy a činnosti, ale i je efektivně zpracovávat.

Pro modul ÚČETNICTVÍ hrají základní úlohu následující číselníky: rozvrh účtů, číselník deníků a dále číselníky středisek, zakázek a činností, pokud je toto vnitropodnikové členění zavedeno. Ostatní číselníky jsou samozřejmě rovněž důležité a bez jejich naplnění by nebylo možné systém RIS2000 provozovat, ale z hlediska získávání informací z RIS2000 nehrají tak významnou úlohu. Vznikají buď jednorázově (kurzovní lístek), nebo průběžně přibývají např. při pořizování faktur (firmy, osoby) nebo jsou pořízeny přímo při instalaci a v zásadě jsou jednoznačně dané a nemění se (konstantní symboly, banky apod.).

Součástí modulu ÚČETNICTVÍ lze nalézt mnoho agend potřebných v podnikových procesech. Kniha pohledávek je agendou, které plní různé funkce. Pořizují se zde vydané faktury, které umožňují ruční pořízení, tisk a storno jednotlivých faktur na prodej materiálu, zboží a služeb v domácí i cizí měně a automatizované pořízení dobropisů. Nastavuje se zde hromadný tisk faktur dle výběru uživatele, vytváří se přehledy podle různých hledisek. Neméně důležité funkce jsou - upomínky, penalizace či plánovaná fakturace.

Kniha závazků umožňuje pořízení přijatých faktur a evidenci tuzemských faktur v Kč i cizí měně, kde rekapitulace DPH je v Kč, a zahraničních faktur v cizí měně. Eviduje se zde oběh faktur, průvodky faktury a ověřování faktur. Proplácí se zde faktury a tiskne se příkaz do banky - export ve formátu ABO[2].

Kniha analytické evidence bankovních výpisů je místem v programu, kde se pořizují a evidují bankovní výpisy v domácí i cizí měně. Zajímavou funkcí je párování předpisů a plateb dle parametrů přednastavených v systému, či podle kontační tabulky. Dnešní velmi důležitou funkcí je práce s formátem ABO, který umožňuje využívat moderní bankovní služby na základě výměny dat mezi bankou a klientem.

Pokladna umožňuje ruční pořízení a tisk příjmových a výdajových pokladních dokladů v domácí i cizí měně. Pořizují se zde příjmové pokladní doklady a tisknou se zde zjednodušené daňové doklady na prodej materiálu, zboží a služeb. Taktéž se zde párují předpisy a platby.

Většina dat vstupuje do systému jedním z dokladů z agend výše zmiňovaných a automaticky se z nich funkcí zaúčtování vytvářejí účetní doklady. Účetní doklad je po zaúčtování zařazen v tzv. souboru položek Hlavní knihy. Do tohoto souboru položek

[2]automatizované bankovní operace

Hlavní knihy lze pořizovat účetní doklady i ručně pomocí vnitřního dokladu. Položková hlavní kniha obsahuje záznamy o všech uskutečněných účetních operacích v účetním období, je hlavní část systému ÚČETNICTVÍ a dokládá se jí *průkaznost vedení účetnictví*. Při převodu do archívu Hlavní knihy se kontrolují pořízené položky a potvrzuje se správnost zaúčtování.

Jednou z nejvýznamnějších funkcí v ERP systému RIS2000 rozumíme saldokonto. Saldokonto je zvláštní evidencí nebo pohled na položky na účtu. Obvykle se týká účtů odběratelů a dodavatelů. Výstupem saldokonta je stav pohledávek a závazků. V systému RIS 2000 jsou k dispozici dva pohledy na saldokonto. Účetní saldokonto, s pohledem na má dáti / dal, které je určeno pro účetní podniku a údaje čerpá z hlavní knihy, tedy pouze ze zaúčtovaných dokladů. Obchodní saldokonto s pohledem na předpis / úhrada, je určeno pro obchodníky, vedení a další neúčetní pracovníky a čerpá z hlavní knihy a nezaúčtovaných dokladů.

3.2 BI

Podle Gály et al. Business Intelligence (BI) představuje specifický typ úloh informatiky, které téměř výlučně podporují analytické, plánovací a rozhodovací činnosti podniků a organizací a jsou postaveny na principech, které právě těmto činnostem nejvíce odpovídají [1]. Podstata spočívá v rozdílu analytických úloh a transakčních. Běžný ERP systém využívá transakčních úloh, kde uživatelé zadávají data v reálném čase. Pro určité skupiny lidí (management, prodejci apod.) jsou samozřejmostí reporty z takových systémů, které z entropické databáze vytáhnou údaje ve formě vhodné k prezentování. Pokud bych v jeden den vyrobil jeden report pro management a jeden trochu odlišný, který používá ale stejná data, pro obchodní oddělení v jiný čas téhož dne, může se stát (a stane se), že přestože obě skupiny si budou myslet, že mají stejná data, mohou učinit na základě těchto dat různá rozhodnutí, které podstatným způsobem ovlivní chod firmy.

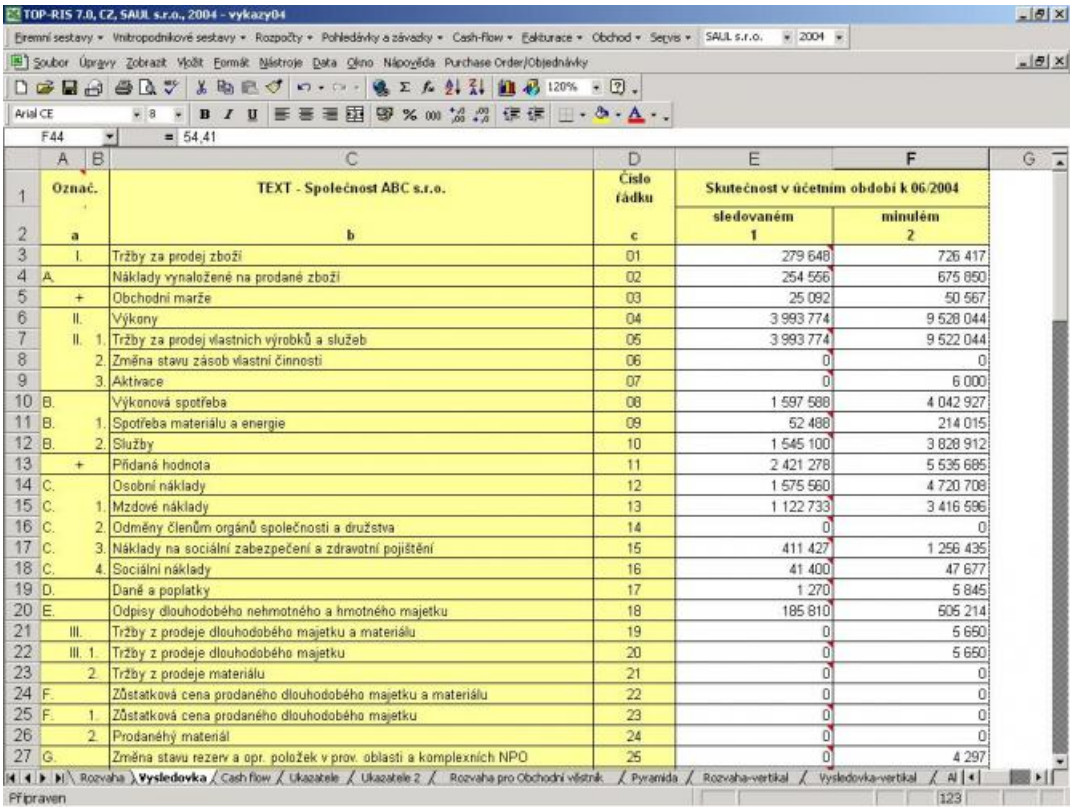
Další z důvodů potřeby systémů BI je ten, že v transakčních systémech je v databázi mnoho údajů konkrétních transakcí, v reportech jsou potřeba údaje z pohledů z jiných stran, vyhodnocení určitých ukazatelů. Oba druhy systémů plní nároky různých skupin uživatelů. Zatímco v transakčních systémech je prostředí upraveno pro nejefektivnější přístup k jednotlivým transakcím (např. v účetnictví si uživatel - účetní přeje rychle vyhledání a případně editace faktur), tak v analytických systémech uživatelé mají potřebu data vyhodnocovat na základě podnikových ukazatelů a aby byla poskytnuta možnost analyzovat tyto ukazatele podle různých dimenzí (hledisek) a jejich kombinací.

Zatímco transakční aplikace vytvářejí stále nová a nová data a stávající data se aktualizují, tím pádem lze generovat na základě těchto údajů dokumenty, analytické aplikace nová data nevytvářejí. Analytické aplikace využívají a transformují data z databází transakčních aplikací a podle potřeby je třídí tak, aby mohly být použity v reportech dle požadavků uživatelů.

Systémem BI si tedy lze představit jako sadu procesů, know-how, aplikací a technologií, jejichž cílem je účinně a účelně podporovat řídící aktivity ve firmě. Podporují analytické, plánovací a rozhodovací činnosti organizací na všech úrovních a ve všech oblastech podnikového řízení, tj. prodeje, nákupu, marketingu apod [1].

3.2.1 TOP-RIS

TOP-RIS je nástrojem business intelligence společnosti SAUL IS, který je určen zejména pro finanční manažery, ekonomy, vedoucí účetní a vrcholové vedení účetních jednotek.



Obrázek 2: TOP-RIS Zdroj: [7]

Program TOP-RIS poskytuje standardní účetní výkazy pro účetní závěrky - Rozvaha, Výkaz zisků a ztrát, Výkaz Cash-flow, a to jak v českém jazyce, tak i v slovenštině, angličtině nebo němčině. TOP-RIS podává též reálné informace i o vývoji libovolných

účetních veličin po měsících, čtvrtletích nebo po letech. Obsahuje též informace pro finanční analýzu (poměrové ukazatele a grafy). Tento software je používán i pro přípravu a tvorbu ročních účetních závěrek a výročních zpráv společností.

TOP-RIS zpracovává informace především z databází RIS2000 nebo případně i jiných software. Jednotlivé moduly TOP-RIS importují různé části databází. Pro modul výkazů a finanční analýzy i pro modul konsolidací je používán import účetní položkové hlavní knihy. Tento modul tedy představuje základ integrovaného softwaru pro finanční a ekonomické řízení firmy. Vedle toho jej lze využít jako nadstavbu nad jinými databázemi, pokud je možný export dat.

Jelikož je TOP-RIS psán v prostředí VBA nad programem Microsoft Excel, nevýhodou tohoto programu je závislost na politice firmy Microsoft - pokud dojde k aktualizaci knihoven VBA či MS Office, tato aplikace může ze dne na den přestat úplně fungovat všem, kteří umožnili takový update. Jelikož je mým cílem v této práci vytvořit mobilní zjednodušenou variantu tohoto software, bylo by tak i mimo jiné vytvořeno záložní řešení při nedostupnosti aplikace TOP-RIS.

4 iOS a Swift

Operační systém iOS je systémem vyvinutým společností Apple Inc. Tento operační systém je využíván v produktech této firmy - v produktové řadě iPhone, iPad apod. V této kapitole se zaměřím na operační systém iOS a historii tohoto operačního systému. Dále popíši, co bude potřeba k vývoji aplikací v jazyce Swift pro iOS a jaké jsou základní frameworky využívané vývojáři. V další kapitole představím samotný programovací jazyk Swift.

4.1 iOS

iOS je operační systém pro mobilní zařízení. Stejně jako moderní desktopové operační systémy, i tento využívá grafické uživatelské rozhraní. Interakce s operačním systémem je ale prováděno, na rozdíl od klasických desktopových OS pomocí klávesnice a myši, dotykovým ovládáním. Například aplikace mohou být otevřeny jedním dotykem na ikonu aplikace na obrazovce. Nebo přechod mezi různými obrazovkami je prováděno pomocí horizontálního přetažení prstem.

Protože iOS byl vyvinut pro snadné uživatelské ovládání [8], operační systém neumožní některé klasické funkce, které se dají nalézt v tradičních operačních systémech. Jedním z příkladů jsou operace se soubory a složkami. Uživatel má také omezený přístup ke zdrojům operačního systému. iOS nám sice umožní spustit více programů najednou, pracovat můžeme pouze jen s jedním aktivním programem naráz. Vývoj tohoto OS, který popíši v další kapitole, ale přidává postupně nové funkcionality, které nám postupem času může umožnit, alespoň v omezené míře, nahradit funkce klasického desktopového OS.

4.1.1 Historie iOS

Operační systém iOS byl nejdříve představen pouze pro první iPhone. Postupem doby se tento operační systém rozšířil nejen do nových modelů iPhone, ale i iPadů a iPodů.

iPhone OS 1 Když v roce 2007 představil Steve Jobs iPhone OS, mluvil o něm jako o upravené verzi systému OS X [8]. Prvotní literatura společnosti Apple Inc. ho nazývala iPhone OS, kdy byl na tu dobu velmi inovativní, chyběly mu v dnešní době neodmyslitelné funkce. Když iPhone OS v roce 2007 vyšel, obsahoval jen 16 předinstalovaných aplikací: Mail, iPod, Calendar, Photos, Clock, SMS, Safari, Notes,

YouTube, Calculator, Mapy, Settings, Photo, Weather, Actions a Telephone. Později přibyl v podobě aktualizace iTunes Store.

iPhone OS 2 Největší novinkou druhé generace iPhone OS a největší konkurenční výhodou před ostatními systémy byla možnost stahovat a vyvíjet aplikace pro App Store, který byl oficiálně spuštěn spolu s vydáním nového systému 8. července 2008. Dnes App Store stále roste a v současné době se může pochlubit více než 1,5 milionem aplikací, vytváří tak pro vývojáře miliardové zisky. V lednu roku 2016, App Store vygeneroval přibližně 40 miliard dolarů pro vývojáře. Zástupci Apple Inc. také řekli, že díky App Store bylo vytvořeno více než 1,9 milionu pracovních míst ve Spojených státech [8]. Dalšími novinkami v iPhone OS 2 byly Push Notifikace, které nově fungovaly i s aplikací Mail. Přibyla možnost otevírání dokumentů MS Office.

iPhone OS 3 iPhone OS 3.0 byl vydán 17. června 2009 a přinesl základní funkce známé z Windows jako kopírovat, vyjmout a vložit. Výrazně byla upravena přesnost GPS modulu a přibyl i kompas. Nový OS přinesl také funkci Spotlight, která slouží k prohledávání iPhonu. Uživatelé se dočkali i bezpečnostní aplikace Find my iPhone. 3. dubna 2010 byla zpřístupněna aktualizace iPhone OS 3.2, která byla určená především pro iPad který byl představen několik týdnů předtím [8].

iOS 4 Tento update OS byl pouze rok po uvolnění iPhone OS 3 [8], kdy došlo ke změně názvu systému z iPhone OS na iOS. V novém OS přibyl multitasking, který umožňoval přepínat se mezi otevřenými aplikacemi bez nutnosti ukládání aktuálních rozpracovaných úkolů. Mezi nové funkce iOS patřila služba FaceTime, která poskytovala uživatelům iPhonu komunikovat přes videohovory, vylepšená funkce Spotlight, která umožňovala kromě prohledávání iOS zařízení i procházení webu nebo Wikipedie, a dále AirPlay, který umožňoval sdílet obrazovku iPhone např. přes AppleTV s televizorem. Apple Inc. také představil aplikaci iBooks, která umožňuje uživatelům iOS číst elektronické knihy.

iOS 5 iOS 5.0 v roce 2011 přinesl přes 200 nových funkcí, za což ho můžeme označit jako největší update od vydání originálního iOS v roce 2007 [8]. Apple Inc. přinesl službu iMessage, který umožnil uživatelům posílat textové zprávy místo SMS. Významnou novinkou byla i možnost WiFi synchronizace, pomocí které se iOS zařízení stala nezávislými od PC. Kompletně přepracovaný byl design aplikace iPod, což je dnešní aplikace Hudba. Snad nejzajímavějším přídavkem celého systému iOS od jeho začátku bylo přidání inteligentní hlasové asistentky Siri.

iOS 6 S příchodem iOS 6 přišlo poprvé od představení originálního iPhone v roce 2007 ke zvětšení úhlopříčky displeje, což si vynutila doba obrovských úhlopříček, které v současnosti dosahují průměrně 5 palců. [8] Z původních 3,5 palce se úhlopříčka natáhla vertikálně na 4 palce, což má za následek rozložení obrazu 16: 9. V tomto update OS byla aplikace Mapy, která místo map od Googlu použila nově své vlastní, a dokázala lidi navigovat přímo na přistávací plochu letiště či do středu pouště. Dalším přírůstkem byla aplikace Passbook, do které mohou uživatelé ukládat letenky, kupony, lístky do kina a mnoho dalších položek. Uživatelé se také dočkali přepracovaného App Store.

iOS 7 Sedmá generace systému iOS přináší největší designovou změnu od původního iOS [8]. Úplnou změnou prošlo GUI, byly přidány nové funkce a přináší takzvaný plochý design. Součástí systému iOS 7 byl nový Control Center, což je rychlý způsob jak přepínat mezi celou řadu běžně používaných nastavení. Další nové funkce jsou vylepšený multitasking či přidání funkce AirDrop.

iOS 8 V době, kdy přišel iOS 8 si už většina lidí zvykla na nový design. [8] Proto Apple přestal pracovat na úpravě GUI, ale začal přidávat mnoho funkcí. iOS 8 zavedl automatické zálohování fotografií na iCloud Drive a HealthKit. Wi-Fi hotspot jde vytvořit pouhým přiblížením telefonu k Macu při zapnutém Bluetooth a Wi-Fi. Díky funkci Handoff bylo nově také možné začít práci na jednom zařízení a dokončit ji na jiném.

iOS 9 Apple při vývoji iOS 9 používal úplně jinou strategii, kdy pracoval hlavně na stabilitě celého systému. Všechny aplikace v iOS 9 začaly využívat technologii Metal, díky které jsou plynulejší. Aplikace Passbook se přejmenovala na Wallet. S příchodem Apple Pay totiž uživatelé získali možnost využít bezkontaktních plateb. Díky Wallet máte nejenom kupóny, letenky ale i kreditní karty na jednom místě. Asi největší změnu nový iOS přinesl pro iPad. Ten se naučil multitasking, který uživatelům umožňuje na boku svého iPadu zobrazit jinou aplikaci [8].

iOS 10 iOS 10 je v době psaní této diplomové práce nejaktuálnější verzí tohoto operačního systému. Tento iOS se z pohledu vývojáře třetích stran nejvíce zaměřuje na multitasking aplikací - nyní je možností svou aplikaci pomocí nových API připojit rovnou k hlavním systémovým aplikacím. Např. lze napsat aplikaci, která dodá nové emotikony do aplikace iMessage. Dále je kladen důraz na propojení aplikací se systémem HomeKit[3] a CarPlay.

[3] domácí síť spotřebičů

Aplikace vyvíjena v rámci této práci bude vyvíjena pouze pro iOS 10 a bude zároveň splňovat požadavky přednesené příručkou iOS Human Interface Guidelines.

4.1.2 iOS Human Interface Guidelines

Každá aplikace vyvíjená pro iOS by měla splňovat požadavky firmy Apple Inc. k tvorbě designu aplikace. Nejenom proto, že samotná firma Apple Inc. řídí schvalující proces o tom, jestli aplikace bude či nebude dostupná v App Store, odkud si ostatní uživatelé budou moci stáhnout novou aplikaci, ale i z důvodu kladné zpětné vazby od samotných uživatelů aplikace. Tyto požadavky lze nalézt v iOS Human Interface Guidelines, které jsou pravidelně aktualizovány s ohledem na vývoj nových verzí iOS (nyní pracuje s verzí iOS 10).

Vývojář byl měl v aplikaci plnit tři základní témata, které odlišují iOS od ostatních platforem [9]:

Clarity. Jasnost. V celém systému je text čitelný ve všech velikostech, ikony jsou ostré a přehledné a je kladen velký důraz na funkcionalitu propojenou s designem. Barvy, fonty či grafika samotná má zvýrazňovat důležitý obsah a prvky, které jsou interaktivní.

Deference. Váženost. Plynulé animace a krásný interface pomáhá uživatelům pochopit a pracovat s obsahem, místo aby s ním bojovali. Obsah aplikace obvykle plní celou obrazovku, zatímco pomocí průsvitných či rozmazaných prvků mohu naznačit další možnosti aplikace.

Depth. Hloubka. Výrazné vizuální vrstvy zprostředkovávají hierarchii a usnadňují porozumění. Doteky umožňují přístup k funkcím a dalšímu obsahu bez ztráty kontextu. Přechody mezi obrazovkami poskytují dojem z hloubky celé aplikace.

4.1.3 Vývoj v iOS

Vývoj pro iOS má velkou podporu ze strany Apple Inc. Na internetu můžeme najít veliké množství oficiálních dokumentů, či webových stránek zaměřené na vývoj aplikací. Tyto materiály jsou tvořeny jak pro experty v oboru, kteří přecházejí k iOS z jiných jazyků, tak i pro naprosté začátečníky. Existuje i ohromné množství web tutoriálů, youtube videí i online kurzů (například na Udemy) od třetích stran. K vývoji aplikace pro iOS je ale třeba mít k dispozici níže uvedené nástroje:

1. Počítač s nainstalovaným operačním systémem macOS. Tento bod je, podle mého názoru, největší překážkou k vývoji pro iOS. Je potřeba mít zakoupený počítač (iMac, MacBook, Mac Pro) od společnosti Apple Inc., které jsou nákladnější na pořízení. Emulace macOS (OSX) na běžném desktopu není oficiálně podporovaná [10].

2. IDE Xcode (aktuální verze k datu psaní práce je 8.1) + iOS SDK. Instalace Xcode je provedena přes App Store. iOS SDK se instaluje automaticky s Xcode a obsahuje nástroje, překladače kódu a frameworky potřebné k vývoji aplikací pro iOS. Hlavní programovací jazyk je Swift, ale z historických důvodů je zde i podpora Objective-C. Tento jazyk byl, předtím než byl vyvinut jazyk Swift, jediným podporovaným programovacím jazykem pro aplikace operačního systému iOS. Hlavními frameworky při vývoji aplikací pro iOS jsou Cocoa, který obsahuje Foundation Kit (např. manipulace se string hodnotami), Application Kit (kódy programů k interakci s GUI) [11] či Core Data (umožní datům v aplikaci být uloženy v SQLite) [12] a pak Cocoa Touch, který obsahuje rekognace gest či animace [11].

3. Registrace v Apple Developer Program. Tento bod je nepovinný, pokud není plánem nasadit aplikaci do reálného provozu do App Store. Registrace v tomto programu dále umožní instalaci, vývoj a testování betaverzí nových verzí OS. Umožní i přístup k pokročilejším možnostem vývoje pro aplikace (např. Apple Pay, Game Center). S tímto programem lze pozvat i ostatní uživatele k testování aplikace před tím, než bude uvedena na trh. Členství v tomto programu není zdarma, platí se 99 amerických dolarů ročně [13].

4.2 Swift

Programovací jazyk Swift je novým objektově orientovaným jazykem vyvinutý firmou Apple Inc. Veřejnosti byl tento jazyk představen v roce 2014 na konferenci vývojářů WWDC (Worldwide Developers Conference) od Apple Inc. Cílem vývoje tohoto jazyka je vytvořit jazyk pro tvorbu velké škály programů - od systémových programů, po mobilní aplikace i cloudové služby. Swift byl vyvinut, aby umožňoval snadnější vývoj [14]. Snaží se omezit množství chyb, kterých se vývojáři dopouštějí při kódování a usnadňuje údržbu programů pro vývojáře. Pomocí Swift se v dnešní době dá napsat program pro iOS, macOS, watchOS, tvOS a Linux. V době psaní této diplomové práce dochází ke změně verzí Swift z verze 2.0 na verzi 3.0. Od verze 2.0 je Swift open-source, lze jej upravit a využít i na jiných platformách [14].

Swift je z větší části obdobou Objective-C za využití moderních konceptů a syntaxe, zachovává klíčové vlastnosti Objective-C při zjednodušení syntaxe. Byl nahrazen způsob volání metod za tečkovou notaci, běžně známé z jazyků JAVA či C#. Následující příklad ukazuje, jak byla zjednodušena syntaxe Swift oproti Objective-C. V Cocoa a Cocoa Touch framework nalezneme mnohé třídy, např. NSString, NSArray, které jsou součástí Foundation Kit[15]. V Objective-C je umožněna tvorba objektů této třídy přímo v kódu. Jak ale tyto objekty byly vytvořeny, tak manipulace s nimi byla možná pouze přes volání těchto objektů. Příkladem je spojení dvou řetězců znaků NSString [16]:

```
NSString *str = @"hello,";
str = [str stringByAppendingString:@" world"];
```

V jazyce Swift mnoho těchto základních typů bylo přímo implementováno do jádra jazyka a je možná přímá manipulace s těmito typy. Výše uvedený kód v jazyce Swift:

```
var str = "hello,"
str += " world"
```

Dalším plusem ve Swift jsou Optionals, které fungují podobně jako v jazyce C, kde pointer odkazuje na hodnotu či je null. Tzn. proměnná označena jako optional může, ale i nemusí, obsahovat hodnotu.

4.2.1 Syntaxe

Pro základní testování jazyka Swift existuje program Playground. Tento program je vizuálně velmi prostý, obsahuje dvě části. Na levé straně uživatel zadává kód a na pravé straně se v reálném čase objeví výsledky kódu (zajímavostí je, že i ukazuje kolikrát byla například proběhnuta funkce for pro dané proměnné). V následujícím programu, který má pro danou proměnnou vypočítat zda je prvočíslo, ukáži, jak menší program v jazyce Swift vypadá.

Deklarace proměnných je pomocí dvou klíčových slov - let nebo var. Let se používá, pokud pro proměnnou bude hodnota konstantní v průběhu programu (podobně jako v JAVA final [17]). Var se používá u proměnných, kdy víme, že hodnota se bude v průběhu programu měnit. Ve Swift není třeba implicitně zadat datový typ, Swift si datový typ automaticky přiřadí z deklarované hodnoty. Řádky se nezakončují středníkem.

```
//deklarace proměnných
let number = 30 // bez datového typu
var isPrime:Bool = true // daný datový typ boolean
```

```
//výpočet prvočísla
if number == 1 {
        isPrime = false
}
if number != 2 && number != 1 {
        for i in 2 ..< number { //syntax Swift 3.0
                if number % i == 0 {
                        isPrime = false
                }
        }
}
//tisk výsledku
print(isPrime)
```

Výsledkem výše uvedeného programu pro hodnotu 30 je false.

5 Vývoj software

V této kapitole bych rád shrnul veškeré poznatky z teorie a na základě nich bych vybral vhodné řešení pro tvorbu zadané aplikace. V první části kapitoly ERP jsem popsal členění ERP podle velikosti podniků, pro které bývají vyvíjeny. Pro takové podniky je možná implementace dle metodiky MMDIS. Tato metodika je teoretická, ale ukazuje nástrahy při výběru ERP do podniku. Představení RIS2000 ukazuje možnosti takového ERP systému, jaké jsou požadavky pro chod takového systému a jaké jsou vlastnosti nejpoužívanějších modulů takového systému. Nadstavbou nad ERP jsou nástroje BI. BI používá své analytické databáze, které ale mají zdroj dat v produkční databázi ERP systému. Nadstavbou nad RIS2000 je program TOP-RIS, který slouží k analýze dat nad produkčními daty RIS2000.

Cíl práce, aplikace, kterou pojmenuji WiRIS, má být mobilním řešením pro stávající software TOP-RIS. Z praxe vím, že nejpoužívanější funkce, které zákazníci používající TOP-RIS využívají, jsou výpočty výkazů pro finanční úřad, konkrétně výkazy účetní závěrky. Nejpoužívanějšími výkazy jsou rozvaha a výkaz zisku a ztrát. Další funkce TOP-RIS, co jsou hodně používány, jsou výpočty saldokontních sestav, kde uživatel může sledovat stav pohledávek a závazků. Jelikož TOP-RIS je nadstavbou ERP RIS2000, budu v aplikaci využívat jako zdroj dat databázi systému RIS2000. TOP-RIS sice využívá své analytické databáze, ale ty jsou zabezpečené v MS Access. Snadnější přístup k datům bude přímo přes produkční databáze RIS2000. Další informací získané z teorie je, že systém RIS2000 je používán z velké většiny v podnicích malé či střední velikosti s maximálně 250 zaměstnanci. Takové podniky už mají naimplementovány ERP systémy minimálně s moduly účetnictví a tyto podniky nebudou využívat aplikaci WiRIS k intenzivní tvorbě různých denních, měsíčních apod. reportů, ale k rychlému určení denního stavu účetnictví podniku.

Z výše uvedených důvodů budu mít takové požadavky na aplikaci:

- Aplikace musí mít používaná data zabezpečená - je nutná autorizace uživatele před použitím aplikace.
- Aplikace musí umět vypočítat výkazy účetní závěrky - tzn. rozvahu a výkaz zisku a ztrát.
- Aplikace musí umět za konkrétní období vypočítat výsledný stav pro zadanou skupinu saldokontních účtů (odběratelské/dodavatelské).

5.1 Požadavky na vývoj

Aplikace bude mít třívrstvou architekturu - klientská vrstva, serverová vrstva a databázová vrstva. Klientská vrstva bude naprogramovaná v jazyce Swift, v IDE Xcode pro operační systém iOS. Databázová vrstva musí obsahovat databázi s daty obsažené v produkční databázi - je potřeba vytvořit převodový můstek dat. Dále je třeba mít serverovou vrstvu schopnu přijímat dotazy od aplikace a posílat výsledky do aplikace a zároveň musí komunikovat s databází. Pro serverovou vrstvu aplikaci jsem vybral službu Parse, která slouží jako BaaS (Backend as a Service).

Parse je aplikační platforma v cloudu, která přináší SDK a služby pro tvorbu mobilního backendu pro nejenom iOS, ale i např. Android či JavaScript [18]. Službami rozumím možnost využívat NoSQL dokumentovou databázi MongoDB[4], fileserver či cloud službu pro JavaScript kódy (myšleno zde to, že výpočetní logika některých operací nemusí být provedena v klientovi, ale na serveru). Parse tedy funguje například tak, že se aplikace připojí do Parse, zažádá o data, Parse procesuje požadavek a vrátí daná data do klienta.

V průběhu psaní této diplomové práce bohužel došlo k nepříjemnosti, startup firma provozující Parse oznámila konec činnosti a ke dni 28.1.2017 ukončila hostování Parse serverů [20]. Jelikož ale firma uvolnila zdrojové kódy k Parse backend, nyní je dostupný open source verze Parse. Naštěstí zde tedy byla možnost využít Parse backend pomocí služby Heroku, která jako PaaS (Platform as a Service) umožní hosting Parse serveru. Heroku je cloudová platforma vzniklá v roce 2007 pro ruby, nodejs či php aplikace [21]. Nabízi deploy pomocí GitHub a veliké množství služeb, které jsou nabízeny jako addon (např. MongoDB).

[4] místo tradičních relačních databází využívajících tabulky používá dokumenty podobné formátu JSON [19]

6 Praktické řešení

Na základě poznatků z předchozích kapitol sestavím výstupní sestavy aplikace. V další části této kapitoly uvedu v provoz serverovou část aplikace, pomocí převodového můstku převedu data z produkční databáze do databázové vrstvy tvořené aplikace. Pomocí návrhu pak napíši klientskou část aplikace v Xcode, která bude napojená na serverovou vrstvu spojenou s databázovou vrstvou aplikace.

6.1 Výstupní sestavy aplikace

V této části práce chci vytvořit šablony, podle kterých by aplikace měla počítat výstupní sestavy. V kapitole 5 jsem zjistil, že používá-li podnik ERP systém s moduly účetnictví, tak nejpoužívanější reporty z takového modulu jsou jak saldokontní sestavy, tak výkazy pro finanční úřad (výkazy účetní závěrky). BI nástroj by takové sestavy měl nabízet. Jelikož je tato aplikace koncipována jako aplikace typově BI, cílem bude ji naprogramovat tak, aby aplikace dokázala za uživatelem zadaných podmínek vypočítat tyto sestavy: rozvaha, výkaz zisku a ztrát a saldokonto odběratelské či dodavatelské.

Pro tyto sestavy budu potřebovat data z oblasti účetnictví. Tyto data jsou obsaženy v produkční databázi MSSQL, v databázových tabulkách pro účetnictví, konkrétně tabulky pro doklady z hlavní účetní knihy. Až tyto data převedu do analytické databáze, která bude zabudovaná v serverové části aplikace, můžu pomocí aplikace vypočítat dané sestavy podle šablon, které uvedu v další kapitole.

6.1.1 Sestavy

Zdrojem šablon sestav, které chci využít v aplikaci jsou výkazy pro finanční úřad, resp. rozvaha ve zjednodušeném rozsahu a výkaz zisku a ztrát ve zjednodušeném rozsahu. Důvod, proč vybírám konkrétně sestavy ve zjednodušeném rozsahu je ten, že ač určitý podnik by měl sledovat tyto výkazy v plném rozsahu, protože z důvodu právních předpisů je podnikem střední velikosti a takový podnik má za povinnost vykazovat tyto výkazy v plném rozsahu [22], je, že aplikace WiRIS je mobilní aplikace a jako taková by neměla být příliš algoritmicky složitá, tzn. program pro mobilní zařízení by neměl pomalu pracovat. Výkazy ve kráceném rozsahu sice mají stejný základ dat jako výkazy v plném rozsahu, ale algoritmicky jsou rychlejší ke zpracování - je tam menší detail.

Rozvaha ve zjednodušeném rozsahu Šablonu vezmu z programu TOP-RIS (aktuální výkaz z roku 2016) a upravím ji do formátu, který budu využívat v aplikaci. Pomocí

této šablony budu počítat výsledek sestavy v aplikaci, udělám select syntetického účtu z databáze a do každé buňky sestavy vložím výslednou hodnotu ze sumy všech výsledků dané syntetiky. Na obrázku níže je šablona, podle které budu definovat rozvahu v připravované aplikaci. Řádky s textem výpočet značí sumarizaci řádků jim příslušných. Například suma za oddíl B je sumou řádků B.I, B.II a B.III.

Označ.	AKTIVA	Číslo řádku	Číslo účtu	
			Brutto	Korekce
a	b	c	1	2
	AKTIVA CELKEM	001	Výpočet	Výpočet
A.	Pohledávky za upsaný základní kapitál	002	353*	
B.	Dlouhodobý majetek	003	Výpočet	Výpočet
B.I.	Dlouhodobý nehmotný majetek	004	012*,013*,014*,015*,019*,041*,051*	072*,073*,074*,075*,079*,093*
B.II.	Dlouhodobý hmotný majetek	005	021*,022*,025*,026*,029*,032*,031*,052*,042*,097*	081*,082*,085*,086*,089*,094*,098*
B.III.	Dlouhodobý finanční majetek	006	061*,066*,062*,067*,063*,065*,068*,069*,053*	
C.	Oběžná aktiva	007	Výpočet	Výpočet
C.I.	Zásoby	008	112*,111*,119*,121*,122*,123*,131*,132*,139*,124*,151*,152*,153*	191*,192*,193*,194*,196*,195*,197*,198*,199*
C.II.	Pohledávky	009	Výpočet	Výpočet
C.II. 1.	Dlouhodobé pohledávky	010	311*,315*, 481*,314*,388*,378*	
C.II. 2.	Krátkodobé pohledávky	011	335*, 336*, 341*, 342*, 343*, 345*	391*
C.III.	Krátodobý finanční majetek	012	251*,253*,254*,256*,257*	
C.IV.	Peněžní prostředky	013	211*,213*,221*,261*	
D.	Časové rozlišení aktiv	014	381*,382*,385*	

Označ.	PASIVA	Číslo řádku	Běžné uč.o.
a	b	c	9
	PASIVA CELKEM	015	Výpočet
A.	Vlastní kapitál	016	Výpočet
A.I.	Základní kapitál	017	411*, 252*, 419*
A.II.	Ážio a kapitálové fondy	018	412*, 413*, 414*, 416*, 417*, 418*
A.III.	Fondy ze zisku	019	421*, 422*, 423*, 427*
A.IV.	Výsledek hospodaření minulých let	020	428*, 431*, 429*, 431*, 426*
A.V.	Výsledek hospodaření běžného účetního období /+ -/	021	6*, 5*
A.VI.	Rozhodnuto o zálohách na výplatu podílu na zisku	022	432*
B + C	Cizí zdroje	023	Výpočet
B.	Rezervy	024	451*, 453*, 459*
C.	Závazky	025	Výpočet
C.I.	Dlouhodobé závazky	026	481*, 364*, 365*, 389*, 379*, 479*
C.II.	Krátkodobé závazky	027	231*, 232*, 461*, 324*, 475*, 321*, 325*, 361*, 362*, 395*, 398*, 249*, 331*, 333*, 335*, 336*, 341*, 342*, 343*, 345*, 346*, 347
D.	Časové rozlišení pasiv	028	383*, 384*

Obrázek 3: Šablona rozvahy ve zkráceném rozsahu Zdroj: Vlastní

Výkaz zisku a ztrát ve zjednodušeném rozsahu Výkaz zisku a ztrát a její šablonu tvořím stejným způsobem jako u rozvahy. Podle níže připravené šablony tedy definuji v aplikaci výkaz zisku a ztrát ve zjednodušeném rozsahu.

Označ. a	TEXT b	Číslo řádku c	Číslo účtu
I.	Tržby z prodeje výrobků a služeb	01	601*, 602*
II.	Tržby za prodej zboží	02	604*
A.	Výkonová spotřeba	03	501*,502*,503*,504*,511*,512*,513*, 518*
B.	Změna stavu zásob vlastní činnosti	04	581*,582*,583*,584*,621*
C.	Aktivace	05	585*,586*,587*,588*
D.	Osobní náklady	06	521*,522*,523*,524*,525*,526*,527*, 528*
E.	Úpravy hodnot v provozní oblasti	07	551*,557*,558*,559*
III.	Ostatní provozní výnosy	08	641*, 642*, 644*, 646*, 648*, 649*, 697*
F.	Ostatní provozní náklady	09	541*,542*,531*,532*,538*,552*,554*, 555*,543*,544*,545*,546*,547*,548*, 549*,597*
*	Provozní výsledek hospodaření	10	Výpočet
IV.	Výnosy z dlouhodobého finančního majetku - podíly	11	
G.	Náklady vynaložené na prodané podíly	12	
V.	Výnosy z ostatního dlouhodobého finančního majetku	13	
H.	Náklady související s ostatním dlouhodobým finančním majetkem	14	
VI.	Výnosové úroky a podobné výnosy	15	662*, 665*
I.	Úpravy hodnot a rezervy ve finanční oblasti	16	574*,579*
J.	Nákladové úroky a podobné náklady	17	562*
VII.	Ostatní finanční výnosy	18	663*, 664*, 666*, 667*, 668*, 669*, 698*
K.	Ostatní finanční náklady	19	563*,564*,565*,567*,568*,569*,598*
*	Finanční výsledek hospodaření	20	Výpočet
**	Výsledek hospodaření před zdaněním	21	Výpočet
L.	Daň z příjmů	22	591*,592*,595*,599*
**	Výsledek hospodaření po zdanění	23	Výpočet
M.	Převod podílu na výsledku hospodaření společníkům	24	596*
***	Výsledek hospodaření za účetní období (+ / -)	25	Výpočet
*	Čistý obrat za účetní období	26	Výpočet

Obrázek 4: Šablona výkazu zisku a ztrát ve zjednodušeném rozsahu Zdroj: Vlastní

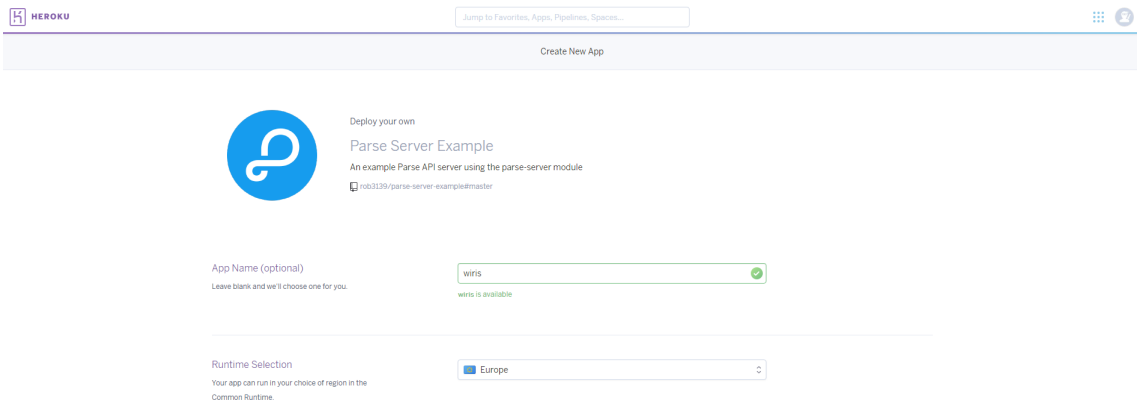
Saldokonto Saldokontní sestavy neřeším podle nějaké šablony. Sestavu pro odběratelské účty budu řešit selectem z databáze a pro zjednodušení beru účty v účtové skupině 31. To samé platí pro dodavatelské účty, budu brát select účtů pro účtovou skupinu 32. Za každou skupinu pak udělám list analytických účtů v dané účtové skupině a vypočtu jejich sumu pomocí hodnot Má dáti a Dal.

6.2 Implementace řešení server side

V této kapitole budu řešit serverovou stranu aplikace WiRIS. Data, které jsou využívány aplikací, pocházejí z produkčního systému RIS2000, která v této instanci používá technologii DBS MSSQL Server. Na cloudové službě Heroku nasadím Parse server s DBS MongoDB a tento deploy popíši. Posléze v klientské části aplikaci v Xcode upravím kód, aby se aplikace úspěšně připojila na server. Vyřeším problém přenosu dat z MSSQL do MongoDB a otestuji, zda se data správně přenese v plném počtu záznamů.

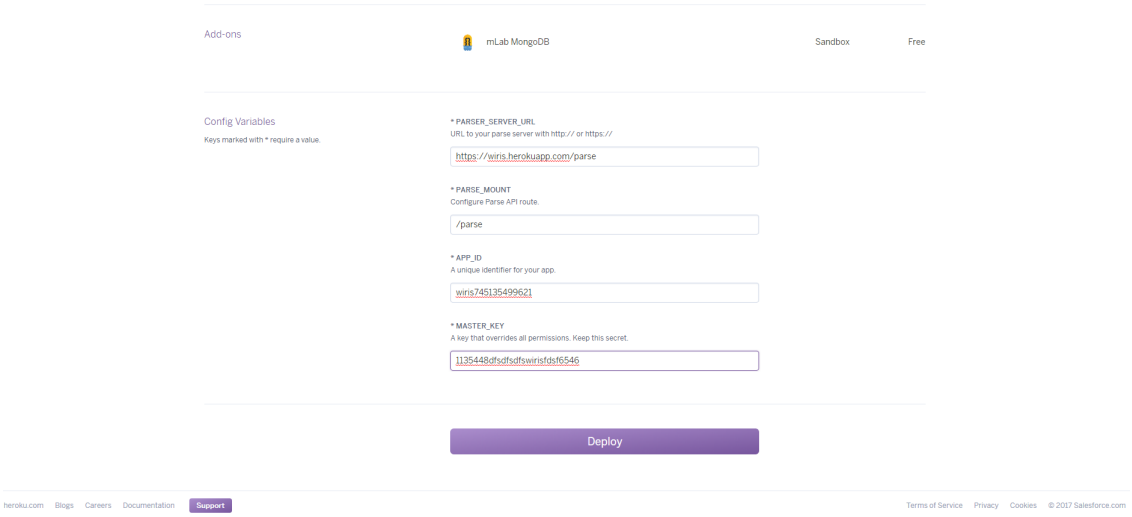
6.2.1 Heroku

Pro urychlení procesu nasazení Parse serveru byl použit již před-připravený default Parse server na GitHubu. Na jmenované stránce se nachází již připravený batch, který udělá deploy Parse na Heroku. Níže popíši deploy a nastavení serveru, aby aplikace byla schopna se připojit. Níže uvedený obrázek ukazuje první krok při deployi Parse serveru. Do "App Name" zadám název vyvíjené aplikace - tento název ale není nikde v produkci použit, slouží jako referenci. Runtime místo defaultních USA dáme Evropu.



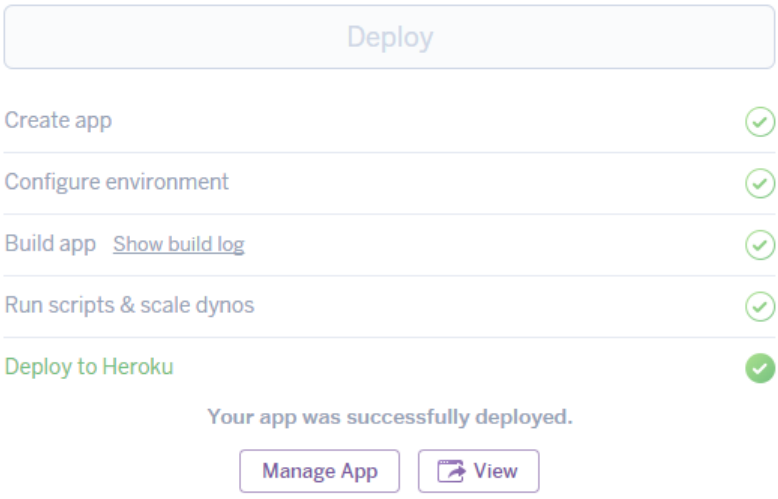
Obrázek 5: Konfigurace Parse Deploy Zdroj: Vlastní

Další obrázek ukazují další údaje, které musím konfigurovat pro zdárné dokončení deploy. Nejprve v add-ons je vidět MongoDB, tento databázový systém je využíván Parse serverem pro klasické operace s daty (a pro účely aplikace je využívána free verze). V sekci Config Variables jsou 4 důležité atributy. PARSER_SERVER_URL vyznačuje URL adresu Parse serveru - na tento URL se bude aplikace WiRIS připojovat. PARSE_MOUNT zobrazuje cestu k Parse API - zde ponechám defaultní hodnotu. APP_ID je unikátní klíč aplikace, společně s MASTER_KEY slouží k autorizaci přístupu na Parse server. Tyto dvě hodnoty jsou libovolně zvolitelné, ale doporučuje se obecně nepoužívat jiné znaky než alfanumerické.



Obrázek 6: Konfigurace Parse Deploy Zdroj: Vlastní

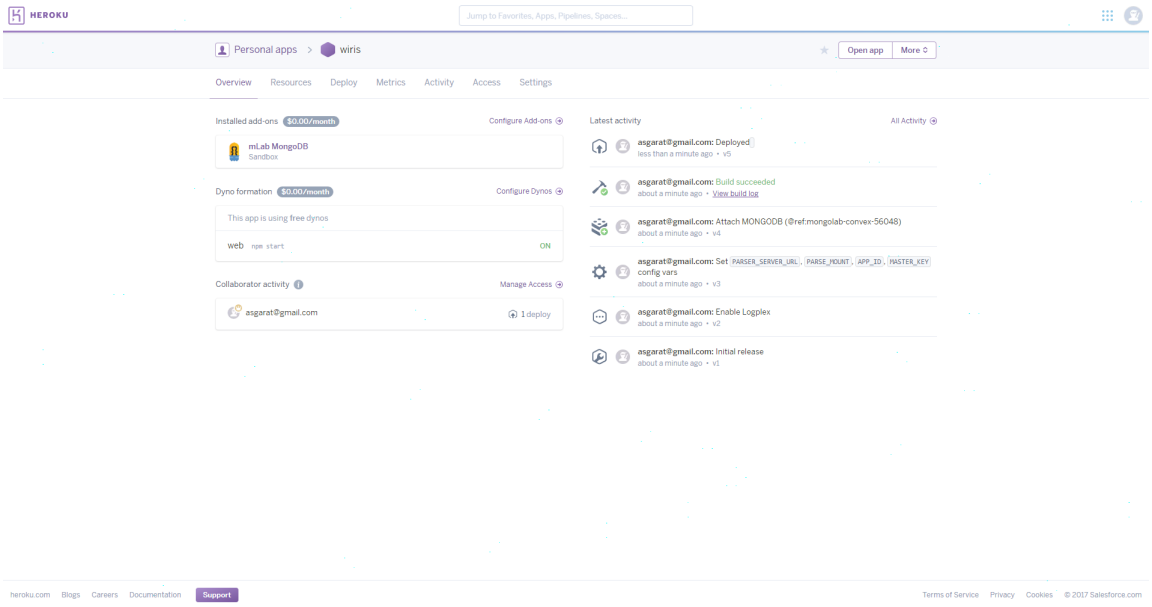
Zadám-li všechny povinné atributy, po zmáčknutí tlačítka se Parse server nasadí na cloud službě Heroku. Přestože je Heroku v základu službou zdarma, je potřeba pro validaci zadat informace platební karty. Z karty nebudou strženy peníze, pokud si ovšem neobjednám placené služby navíc - bez tohoto kroku se mi ale nepodaří provést úspěšný deploy.



Obrázek 7: Deploy Parse Serveru Zdroj: Vlastní

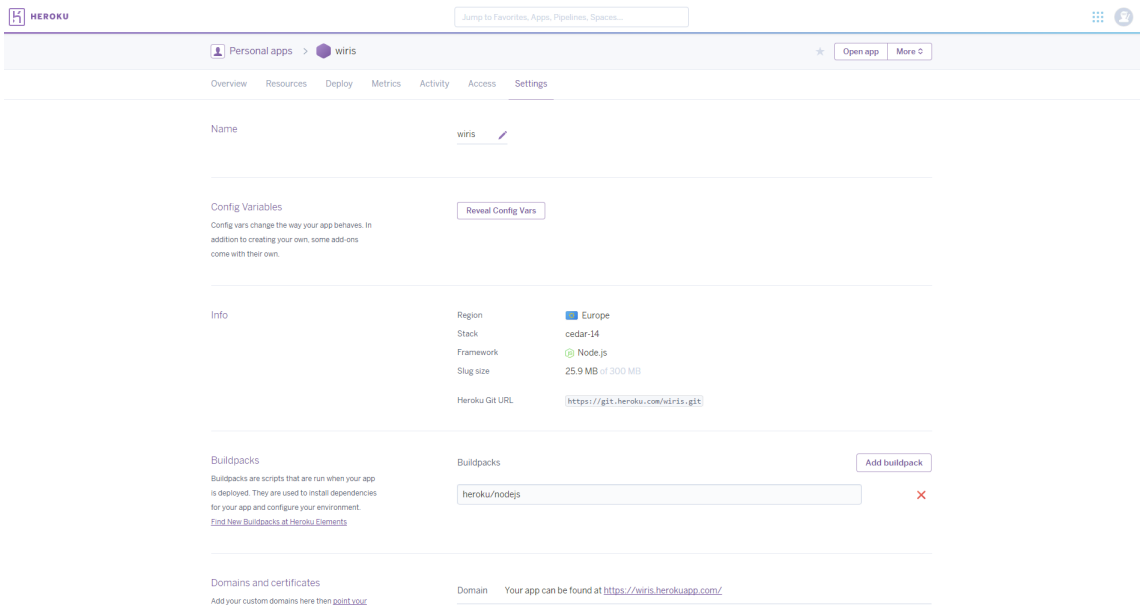
V následujícím obrázku je vidět základní dashboard Heroku. Pro účely této práce se budu zajímat o záložky Overview, kde je vidět souhrn informací a Resources, kde jsou

zdroje, jaké využívá aplikace. Naleznu zde i záložku Access, kde můžu přidat osoby, které se budou o chod serveru starat.



Obrázek 8: Heroku Dashboard Zdroj: Vlastní

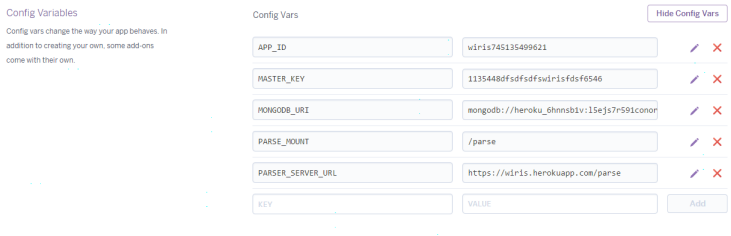
V záložce Settings naleznu důležitá nastavení serverové části aplikace WiRIS. Zde musím konfigurovat některé údaje. Další nastavení zobrazím tlačítkem Reveal Config Vars.



Obrázek 9: Záložka Settings Zdroj: Vlastní

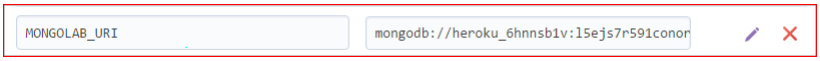
V sekci Config Variables vidím nastavení, které jsem provedl při deployi serveru.

Hodnoty APP_ID a MASTER_KEY v dalších krocích zadám do kódu aplikace, aby byla aplikace autorizována. Lze vidět ještě nový atribut MONGODB_URI, který značí umístění DB na serveru.



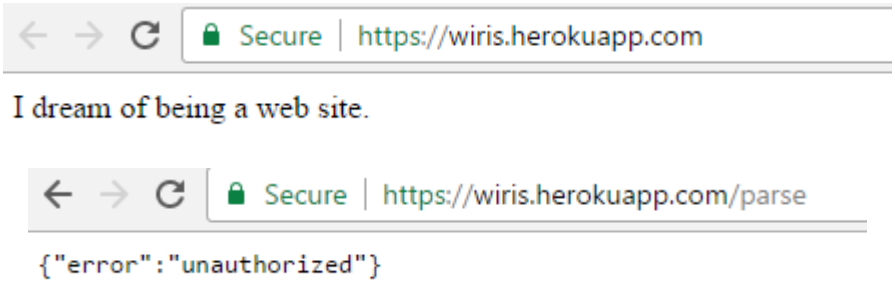
Obrázek 10: Config Variables Zdroj: Vlastní

Do konfiguraci musím přidat další řádek (obrázek viz níže), atribut MONGOLAB_URI se stejnou hodnotou jako MONGODB_URI. Tento string je taky využíván pro připojení Parse serveru k MongoDB.



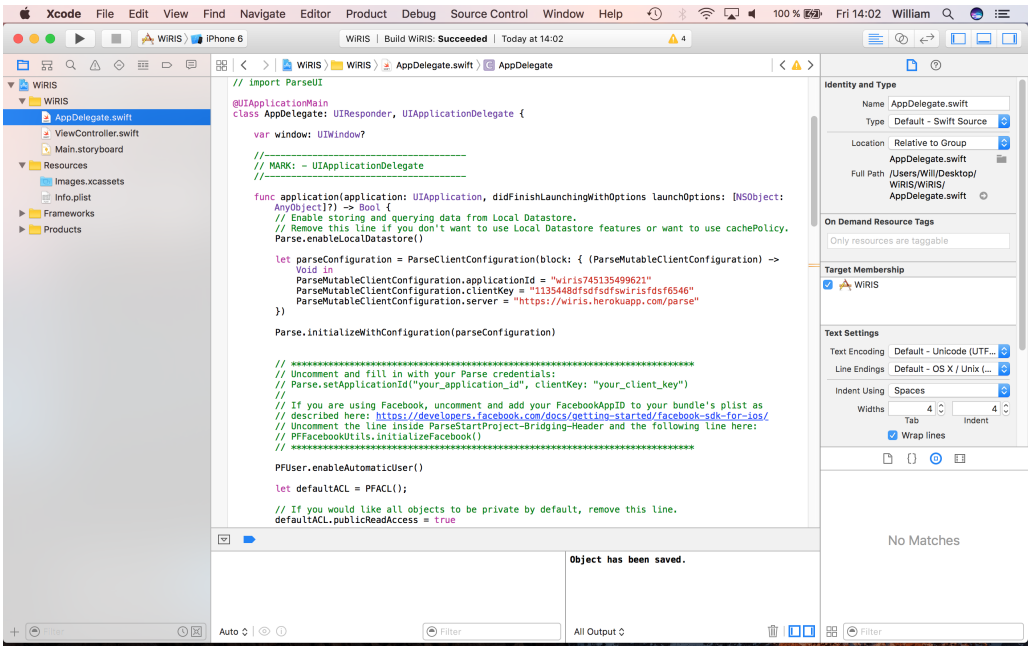
Obrázek 11: Atribut MONGOLAB_URI Zdroj: Vlastní

Pokud je vše správně zadáno, můžu otestovat, zda serverová část aplikace je v provozu. V dashboardu je tlačítko Open app, který zobrazí aplikaci na Heroku. Pokud je výsledek stejný jako obrázek níže, vše běží správně. Zadám-li na konci URL ještě string /parse, dostanu chybovou hlášku ve formátu JSON. Tato chyba se objeví, protože se pokouším přihlásit bez autorizačních údajů.



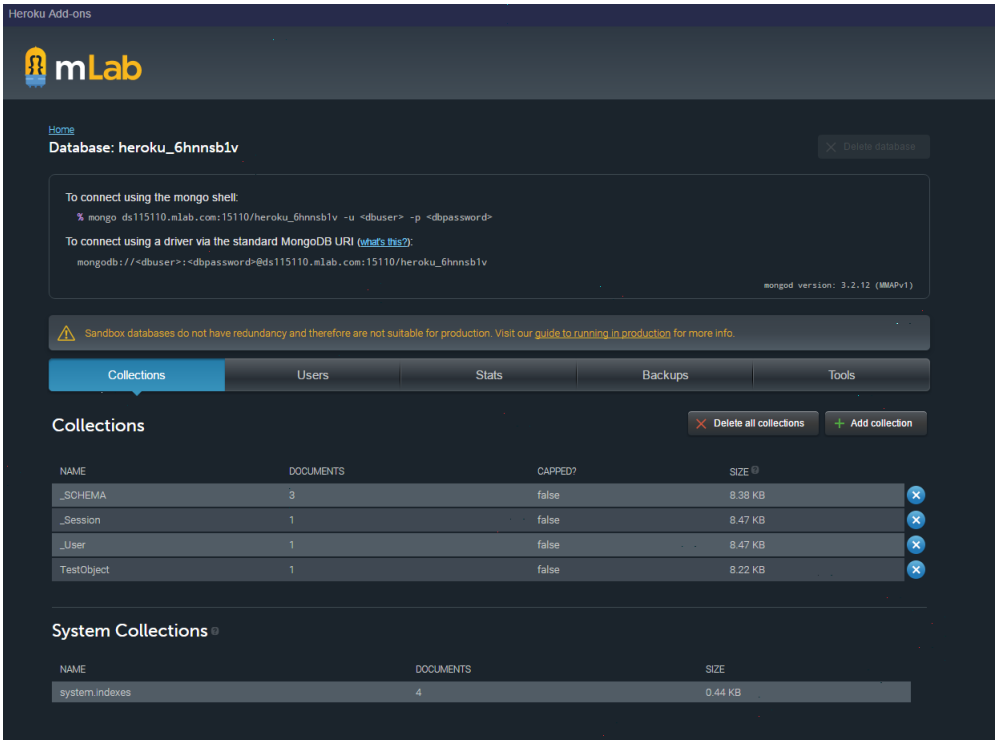
Obrázek 12: Test provozu Zdroj: Vlastní

Nyní propojím aplikaci v IOS s nově vytvořeným Parse serverem. Aplikaci v IOS budu stavět na základech default Parse projektu dodaný přímo vývojáři Parse projektu. Samotnou aplikací v IOS se budu zabývat dále, zde si jen ukáži propojení aplikace se serverem. V Xcode otevřu projekt WiRIS a v AppDelegate.swift se nachází blok kódu s konfigurací připojení k Parse serveru. Do applicationId dám hodnotu APP_ID, do clientKey hodnotu MASTER_KEY a hodnota server bude rovna hodnotě PARSER_SERVER_URL. V Xcode dám build + run a pokud se vše povedlo, objeví se v konzoli zpráva "Object has been saved." Tato zpráva značí, že aplikaci se úspěšně spojila s Parse serverem a vložila do MongoDB instanci testovací entity.



Obrázek 13: Xcode Parse Init Zdroj: Vlastní

Pro ověření, zda se z aplikace uložila data do databáze, si v Heroku Dashboard vyberu záložku Resources. Zde uvidím addon MongoDB, na který kliknu a objeví se UI databáze, kterou server a aplikace využívá. V části collection je entita TestObject s jednou instancí, která byla teď vytvořena. Kliknu-li na entitu TestObjects, zobrazí se mi seznam všech instancí této entity v JSON.



Obrázek 14: MongoLab UI Zdroj: Vlastní

6.2.2 Parse dashboard

Serverová část aplikace by s úspěšnou konfigurací Parse serveru na Heroku mohla být připravená a použita pro reálné nasazení celé aplikace, ale pro účely vývoje aplikace, testování aplikace a následné administrace aplikace provedu a popíši instalaci Parse dashboardu. Tento dashboard nám uživatelsky usnadní operace s databází.

Parse server využívá technologii node.js (lze vidět i na obrázku číslo 6), software vyvinut pro psaní internetových aplikací. Pro účely této diplomové práce nebudu nasazovat tento dashboard na nějaký server (ani na Heroku), ale pouze na localhost. Pro zprovoznění tohoto dashboardu si nainstaluji node.js na lokální stanici (zde na stanici s OSX, kde je vyvíjena aplikace v Xcode). Po nainstalování tohoto produktu dám v terminálu tento příkaz:

```
sudo npm install -g parse-dashboard
```

Po instalaci stačí pouze jeden příkaz (v jednom řádku) a dashboard se spustí:

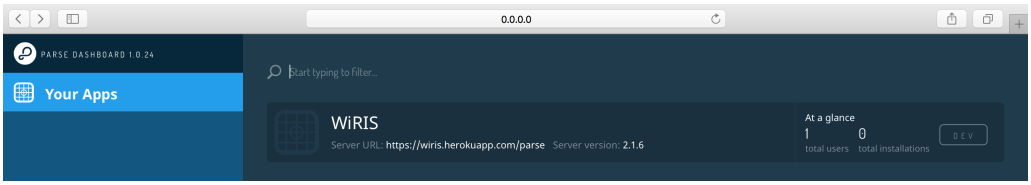
```
parse-dashboard --appId wiris745135499621 +
--masterKey 1135448dfsdfsdfswirisfdsf6546 +
```

```
--serverURL "https://wiris.herokuapp.com/parse" +
--appName WiRIS
```

Pokud se vše úspěšně podaří, objeví se oznámení, že dashboard je available na localhost.

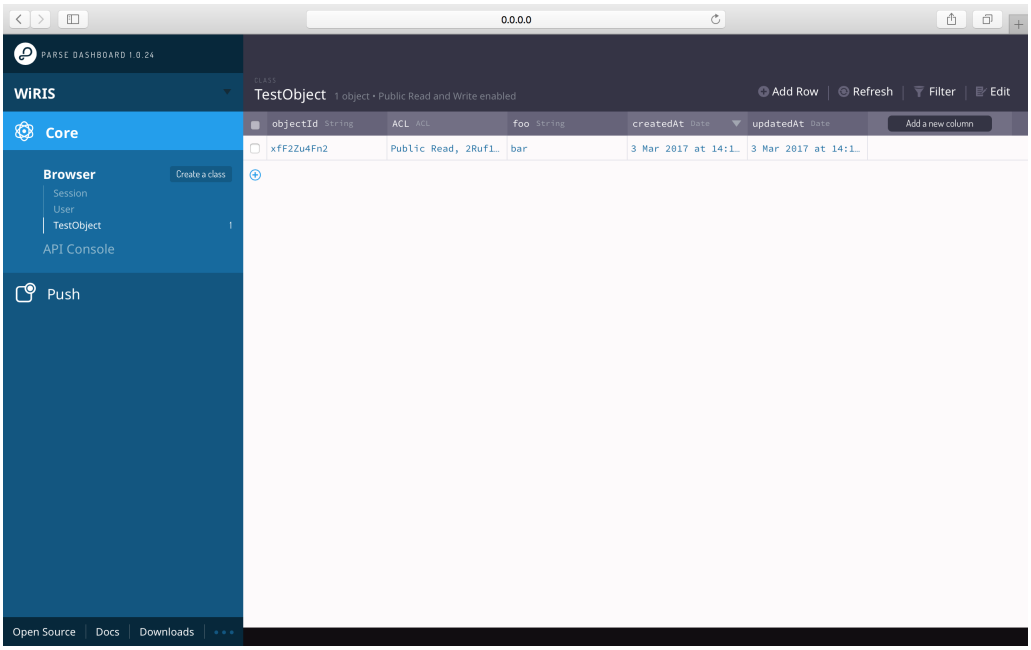
Obrázek 15: OSX terminal Zdroj: Vlastní

Nyní si v prohlížeči dám adresu zkopírovanou z obrázku číslo 12 - http://0.0.0.0:4040/. A objeví se mi Parse dashboard.



Obrázek 16: Parse Dashboard Zdroj: Vlastní

Kliknu na aplikaci WiRIS, objeví se mi dashboard přímo pro tuto aplikaci. Kde v části CORE lze najít UI k databázi, kterou aplikace využívá.



Obrázek 17: Parse dashboard CORE Zdroj: Vlastní

Zajímavostí je, že v části API Console se dá podle daných entit a atributů v nich sestavit query podle daných podmínek a získat výsledky takového query aniž bych musel použít k tomu aplikaci. A pokud jsou v aplikaci implementovány určité metody, tak část Push odešle Push notifikaci (zprávu) na všechny přístroje, které jsou připojeny k internetu a mají nainstalovanou tuto aplikaci.

6.2.3 Propojení produkční databáze MSSQL s Parse

V této kapitole bych chtěl vyřešit jeden z mandatorních podmínek pro úspěšný chod celého vyvíjeného systému. Z principu BI, na kterých principech je založena i aplikace WiRIS, nemůže být tato aplikace produkčním systémem, nýbrž systémem analytickým, nová data tato aplikace nevytváří. Produkčním systémem je v tomto případě dříve představený ERP systém RIS2000 a úkolem je získat data z tohoto systému a insertovat do databáze používaným WiRIS. Produkční databáze je v DBS Microsoft SQL Server 2008 R2 a na straně WiRIS je NoSQL DBS MongoDB. Vzhledem k odlišností databázového modelu obou systému se nebudu pokoušet o nějaký logický převodový můstek dat mezi oběma systémy. Nejjednodušší zde bude, pokud z MSSQL uděláme dump pár tabulek, které budeme potřebovat a posléze uděláme load těchto pár tabulek do MongoDB.

Tento proces zajisté lze provádět manuálně v případech nouze, ale vzhledem k tomu, že bude tímto procesem procesována větší kvantita dat, která může zpomalovat server,

ukazuje se jako vhodné řešení připravit automatizovanou dávku, která bude spuštěná přes noc. Takový přístup bude vhodný i z hlediska toho, že uživatele budou mít vždy čerstvá data k poslednímu dnu. Je zde možnost, že na produkčních systémech mohou vznikat data i mimo pracovní hodiny, a proto je vhodné dávku opakovat každý den.

SQL Server Pro účely této práce využiji testovací databázi s testovacími daty (test doklady). Potřebná data, které využijeme v aplikaci WiRIS budou obsaženy ve dvou tabulkách (jedny z největších v celé databázi) - položková hlavní kniha a hlavičky účetních dokladů. Položková hlavní kniha slouží jako souhrn všech dokladů v systému, lze tam nalézt tam hodnoty rok, deník, doklad a zároveň na jaké účty tyto doklady byly zaúčtovány. Pro potřeby saldokonta jsou pro saldokontní účty (zde pro zjednodušení uvažujme účty v účtové třídě 3* - odběratelské účty ve skupině 31* a dodavatelské účty ve skupině 32*) vyznačeny i data splatnosti dokladů či identifikace dodavatele/odběratele (většinou IČO). Údaje jako měsíc dokladu nebo datum zaúčtování dokladu lze najít v tabulce hlavičky účetních dokladů (zde pro zjednodušení uvažuju všechny doklady v systému jako zaúčtované). Jelikož je v položkové hlavní knize primárním klíčem rok, deník, doklad, radek pomocí tohoto složeného primárního klíče spojím obě tabulky.

```
select b.mesic, a.*
from saul.me10 a, saul.me17 b
where a.rok = b.rok
and a.denik = b.denik
and a.doklad = b.doklad
```

Samozřejmě pokud by se v budoucnu tato aplikace rozšiřovala, nic nebrání tomu získat i více údajů z více tabulek z databáze.

MongoDB V databázi MongoDB vytvořím 1 tabulku - v tomto DBS brány jako kolekce - spojením dvou tabulek z produkční databáze (z MSSQL) . V kolekci SCHEMA nadefinuji jednu novou kolekci doklad, který obsahuje data z tabulky me10 (položková hlavní kniha) a me17 (hlavičky účetních dokladů). Při použití oficiálního nástroje pro import CSV do MongoDB se bohužel neaktualizuje SCHEMA automaticky, musí se manuálně alespoň jednou předem nadefinovat. Bez definování kolekce v SCHEMA se tyto tabulky nezobrazí v Parse dashboardu.

Definice kolekce doklad ve SCHEMA:

```
{
"_id": "me10",
```

```
"mesic": "string",
"rok" : "string",
"denik" : "string",
"doklad" : "string",
"radek" : "string",
"ucet" : "string",
"protiucet" : "string",
"stred" : "string",
"cin" : "string",
"zak" : "string",
"zdroj" : "string",
"varsym" : "string",
"parsym" : "string",
"zamsym" : "string",
"sld" : "string",
"splatnost" : "string",
"ico" : "string",
"kz" : "string",
"kcm" : "string",
"kcd" : "string",
"kodb" : "string",
"typk" : "string",
"kku" : "string",
"datumk" : "string",
"iso" : "string",
"jedn" : "string",
"kurz" : "string",
"cm_kcm" : "string",
"cm_kcd" : "string",
"kzeme" : "string",
"text" : "string",
"typdph" : "string",
"zaklad_dph" : "string",
"uzakt" : "string",
"datakt" : "string"
}
```

Tato definice kolekce byla nadefinována s atributy s datovým typem string - důvodem

je, že aplikace pro svou definovanou funkčnost nepotřebuje jiný typ než string (přetypování může proběhnout přímo v kódu, pokud je potřeba). Automatickou funkčností importu CSV do MongoDB je nevhodné automatické přetypování atributů, co vypadají jako čísla, na datový typ integer. Problémem je to pro sloupec ucet, který je definován jako varchar v produkční databázi a v aplikaci WiRIS tento atribut bude považován jako string s příslušnými vyhledávacími funkcemi. Proto musím po importu dat do MongoDB pomocí shellu Mongo pro sloupec ucet napsat menší script v javascriptu, který tento atribut přímo v MongoDB přetypuje na string, konkrétně zde uzavře atribut do uvozovek. Následující kód uložím do souboru ucet.js, se kterým budu pracovat v konečné dávce.

```
var y = db.doklad.count();
var z = 0;
db.doklad.find({ 'ucet' : {$type: 16} }).forEach(function(x) {
x.ucet = x.ucet + "";
db.doklad.save(x);
z = z + 1;
print("Row " + z + " from total rows: " + y);
});
quit();
```

Pokud je vše správně nadefinováno, dalším krokem je napsat dávku, která přesype data z jedné DBS do druhé. Připravím batch a nastavím ji v OS automatické pravidelné spuštění v 23 hodin. Obsahem batch souboru bude export z MSSQL pomocí utility BCP do souboru typu CSV a posléze pomocí utility mongoimport import CSV do MongoDB.

```
@echo off
bcp "select b.mesic, a.* from wiris.saul.me10 a, +
wiris.saul.me17 b where a.rok = b.rok and +
a.denik = b.denik and a.doklad = b.doklad" +
queryout "C:\WiRIS\doklad.csv" -c -t, -T +
-SWILL-PC\WIRIS -Usaulis -Psaulis
"C:\Program Files\MongoDB\Server\3.4\bin\mongoimport.exe" +
-h ds115110.mlab.com:15110 -d heroku_6hnnsb1v -c doklad +
-u saulis -p saulis --drop --file "C:\WiRIS\doklad.csv" +
--type csv --fieldFile "C:\WiRIS\dokladtemp.txt"
"C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe" +
ds115110.mlab.com:15110/heroku_6hnnsb1v -usaulis -psaulis +
```

```
"C:\WiRIS\ucet.js"
REM PAUSE
```

V exportu je příkaz nadefinován tak, aby se pomocí selectu celé tabulky exportovala data do CSV (v parametrech je odkaz je jméno databáze, instanci databáze a přihlašovací údaje). Záludností při exportu pomocí BCP je absence údajů o názvech atributů (header). Mongoimport takové údaje vyžaduje, a proto je potřeba vytvořit 1 txt soubor s názvy atributů tabulky, kde každý řádek reprezentuje název jednoho atributu. V příkazu mongoimport jsou takové soubory označeny parametrem fieldFile. Dalšími parametry mongoimport příkazu je název serveru (-h), název databáze (-d), kolekce (-c) a přihlašovací údaje. User saulis (pwd: saulis) je vytvořen manuálně předem. Parametr drop udělá to, že před každým importem provede drop dané kolekce. Jelikož aplikace WiRIS data neupdatuje svou funkčností, drop kolekcí před importem vždy zajistí bezproblémovou aktualizaci dat. Po úspěšném uploadu databáze se provede script uložený v ucet.js, který přetypuje atribut ucet z int na string.

Jelikož se jedná o přenos jedné tabulky, v batch skriptu není řešeno logování procesu. Pro potřeby této práce byla databáze předem optimalizovaná (zde např. myšleno, že v textových sloupcích místo čárek, pomocí funkce replace, byly nahrazeny středníky), a proto se nepředpokládá chyby v převodu dat. V reálném provozu bych to vyřešil tak, že batch by skončil zápisem do logu a takový log se pošle operátorovi emailem.

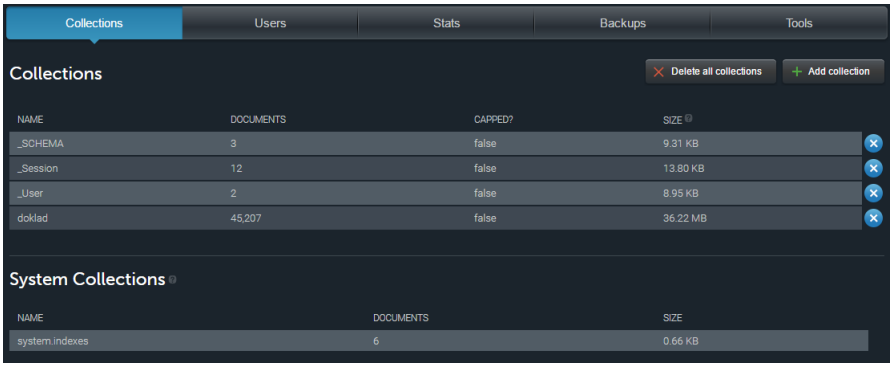
Výsledek přenosu Po spuštění dávky se v konzoli zobrazí dočasný log celého procesu. V tomto logů je záznam z BCP kolik záznamů z databáze MSSQL bylo vyexportováno a kolik záznamů bylo z CSV do MongoDB naimportováno. Pouhým porovnáním dvou hodnot si lze zkontrolovat úspěšné dané operace - zde je to tedy 100% úspěšné, přenesly se všechny záznamy. Úryvek z logu ukazuje danou skutečnost, první paragraf vyjadřuje hodnoty z tabulky položek hlavní knihy a druhý paragraf záznamy z tabulky hlaviček účetních dokladů.

```
45207 rows copied.
Network packet size (bytes): 4096
Clock Time (ms.) Total     : 907
Average : (49842.34 rows per sec.)
```

Následující úryvek logu ukazuje kolik záznamů se naimportovalo do MongoDB. Úplný výpis z logu je vložen do přílohy této diplomové práce.

```
heroku_6hnnsb1v.doklad 6.51MB/6.51MB (100.0%)
imported 45207 documents
```

Pro důslednější kontrolu se přihlásím na hlavní dashboard v rozhraní MongoDB a podívám se, zda se do nově vytvořené kolekce naimportovalo správný počet záznamů. Z obrázku níže je vidět, že do kolekce doklad naimportovalo 45207 záznamů - souhlasí to s počtem vyexportovaných záznamů z databáze MSSQL.



Obrázek 18: Import dat MongoDB Zdroj: Vlastní

6.3 Implementace řešení client side

V této kapitole popíši aplikaci v klientské části - aplikaci, co si uživatel nainstaluje z App Store. Aplikace byla vyvinuta v programovacím jazyce Swift 2.0. Vzhledem k tomu, že v době psaní práce došlo k uvolnění verze Swift 3.0, došlo k mírné úpravě syntaxe. Během vývoje jsem ale nedělal cílenou migraci kódu, který už byl napsán, na Swift 3.0. Pouze po aktualizaci Xcode (nyní verze 7.3.1) jsem, pokud se objevilo někde varování IDE, že se jedná o zastaralou syntax, automaticky upravil zvýrazněné části kódu. Tudíž je výsledný kód mezi verzemi Swift 2.0 a 3.0. Stále se ale jedná o funkční kód a nebrání to úspěšné publikaci na App Store.

Obsahem této kapitoly je popis samotného Storyboardu, jak vypadá. Dále popis jednotlivých obrazovek tzv. ViewController a logika každé obrazovky.

6.3.1 Storyboard

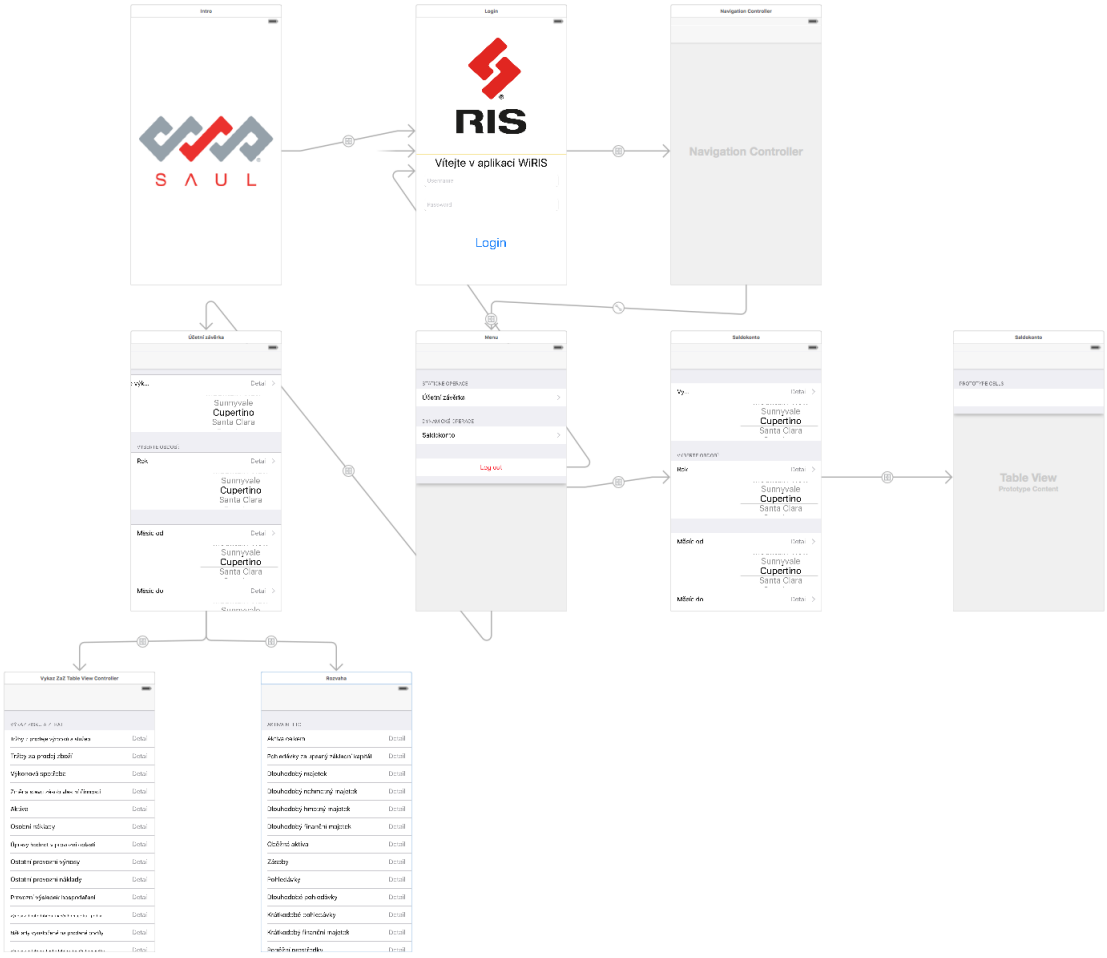
Storyboard v Xcode slouží pro tvorbu a design uživatelského rozhraní aplikace. Tato tvorba UI se může provádět dvojím způsobem, buď přímo v kódu se definuje rozměry a další atributy objektů v obrazovkách nebo pomocí grafických nástrojů ve Storyboardu. V této aplikaci budu objekty definovat hybridním způsobem, kde základ definuji ve Storyboardu a přesnější definice, pokud je potřeba, přímo v kódu. Dalším důležitým prvkem je tvorba aplikačního flow (cestu po obrazovkách kudy aplikace po sériích příkazů povede). Tyto cesty mezi obrazovkami (dále ViewController) se jmenují segue a značí

se šipkami. Pokud se segue pojmenuje, může se přímo v kódu vyvolat metoda performSegueWithIndentifier, který program přesune na další ViewController. Další možností je přímo ve Storyboardu definovat segue, kdy po kliknutí nějakého tlačítka se přesune aplikace na další ViewController. Tyto segue se musí vytvořit, až pokud je ViewController na druhé straně vytvořený. Pokud tedy níže popisuji vznik některého segue, tak ViewController na který segue odkazuje, je už vytvořený (k tomu stačí pouze čistý objekt ViewController k definici segue, bez vnitřních objektů).

Aplikace začíná na ViewController zvaný Login, který je definován atributem, že je Initial ViewController (zobrazeno malou šipkou). Zde vždy začíná aplikace po spuštění. Na obrázku níže je sice vidět ještě controller Intro, který úvodnímu controlleru předchází, ale slouží pouze pro zobrazení loga firmy SAUL IS spol. s r. o. Tento controller naběhne vždy jen na krátkou dobu před načtením celé aplikace. Úvodní ViewController Login slouží k zabezpečení dat, se kterými aplikace pracuje, před neautorizovanými osobami. Po zadání úspěšného loginu se aplikace přepne na ViewController Menu, který je vázán na Navigation Controller. Tento speciální Navigation Controller slouží k řízení hierarchických ViewControllerů, kde ViewController Menu je na vrcholu. Pokud tedy z Menu půjdu dále na další controllery, v horní části obrazovky se objeví speciální lišta tzv. Navigation Bar, kde budou tlačítka umožnující zpětný chod aplikací až do vrcholu této hierarchie ViewController (zde tedy Menu).

ViewController Menu slouží k rozdělení aplikace do dvou částí, jedna část slouží k práci s dynamickými reporty a druhá se statickými reporty. Dynamickou částí je zde Saldokonto. Ve ViewController Saldokonto uživatel si pomocí pickerů (seznam daných hodnot) vybere výběrovou podmínku a pomocí tlačítka výpočet se objeví v dalším ViewControlleru (kde je i logika výpočtu) výsledek. Všechny ViewControllery ve flow po controlleru Login jsou řešeny tabulkovým pohledem (TableView) a ze všech TableView jen zde se místo statických buněk používají dynamické buňky (prototype cells). Počet buněk zde tedy není pevně daná, ale přibývá/ubývá s výsledkem algoritmu.

Statické reporty se v této aplikaci dělí na Rozvahu a Výkaz zisku a ztrát. Výběr výkazu je na ViewController Účetní závěrka, kde podmínka výběru dat je podobná jako v controlleru Saldokonto. Po zmáčknutí tlačítka výpočet se zobrazí vybraný výkaz se statickou množinou předdefinovaných buněk.



Obrázek 19: Storyboard Zdroj: Vlastní

Tento storyboard je v projektu uložen pod názvem Main.storyboard. V tomto projektu existuje ještě třída, která není připojena k žádnému ViewController - třída AppDelegate. Tato třída je jádrem celé aplikace, zpracovává příkazy na úrovni aplikace, kde jedny z nejdůležitějších příkazů jsou applicationDidFinishLaunching (řídí start aplikace, konfigurace) a applicationWillTerminate (čištění při ukončení aplikace). V mém případě do této třídy přidám konfiguraci přístupu do Parse serveru. Kód v této třídě je default kódem dodaný ze stránek Parse serveru, kdy já jsem pouze změnil část s konfigurací připojení na moji speciální instanci Parse serveru. Konfigurace takové třídy je znázorněna na obrázku č. 14 výše, kdy jsem konfiguroval a testoval připojení aplikace k serveru.

V dalších částech této kapitoly podrobněji popíši tvorbu a logiku jednotlivých ViewController a tříd k nim připojených. Můj postup při vývoji této aplikace byl, že nejdříve ve Storyboard nadefinuji uživatelské rozhraní obrazovky a pak napíši logiku k vy-

tvořenému rozhraní.

6.3.2 Login ViewController

Tento ViewController má za úkol chránit data použita v této aplikaci před neautorizovanými osobami. Zde jsem tedy vytvořil nový ViewController, dal jsem mu název Login a hned nastavil rozměr obrazovky na 4.7 palců. Dále jsem do vytvořeného ViewController vložil několik objektů. ImageView, který bude statický objekt a ke kterému jsem hned přiřadil obrázek (logo systému RIS). Dalším statickým objektem je Label, kam jsem vložil vítací text. Dva další objekty jsou Text Field - do nich uživatel zadá své uživatelské jméno a heslo. Pro obě pole jsem vypnul automatickou korekci slova i kontrolu pravopisu. Pro pole pro heslo jsem také zapnul Secure Text Entry - tj. při zadávání textu se nezobrazí již zadaný text, ale místo toho se objeví tečky. Posledním objektem je Button, kterým uživatel pošle příkaz a zaloguje se do systému.

Důležitou částí tvorby UI v Xcode je tvorba constraint (omezení) ke každému objektu na obrazovce . Tato tvorba constraint je třeba z důvodu toho, že už je v nabídce iOS více telefonů s různými rozlišeními a zároveň je možné takové telefony obrátit do horizontálního módu. Pokud tedy nevytvoříme ke každému objektu určité constraint, tak je možné, že z prvního pohledu pěkného UI se spuštěním aplikace stane nepřehledná aplikace, kdy objekty budou překrývat samy sebe a uživatel nemá možnost s aplikací cokoliv dělat. Proto jsem vytvořil ke každému objektu v tomto ViewController zarovnání na střed a výšku k objektu nad ním (u obrázku to bylo k horní straně obrazovky). Xcode nabízí možnost kontroly takových constraint, pokud jsou správně zadané a v případě chyb automaticky opraví, či doplní chybějící. Tuto volbu jsem na závěr využil, kdy Xcode opravil můj ViewController, aby byla bez chyb při zobrazení v jiných rozlišeních.

Od tohoto ViewController vede segue do Navigation Controlleru s názvem Menu. Na tento segue odkazuje funkce login, popsaný níže.

Obrázek 20: Login ViewController Zdroj: Vlastní

K tomuto ViewController jsem v projektu vytvořil třídu ViewController.swift. Tato třída byla vytvořena jako delegate třídy UIViewController, a proto jsem tuto třídu mohl manuálně ve Storyboardu přiřadit k výše vytvořenému ViewController Login. Pokud takto správně vytvořím spojení, je nyní možnost pomocí klávesy Ctrl a přetáhnutí objektu přímo do kódu přiřazené třídy vytvořit referenci objektu v dané třídě. Tímto způsobem jsem vytvořil dva outlety (IBOutlet) pro TextField username a TextField password, pro které se vytvoří příslušné variable, se kterým můžeme v kódu pracovat. Zároveň i přidal akci (IBAction) pro tlačítko Login, který bude volat funkci login. Do této třídy jsem vytvořil celkem dvě metody. Metoda displayAlert slouží k vytvoření menší výstražné obrazovky, kam se v parametrech vloží název a text chyby. Tato metoda vrací speciální UIAlertController, okno, se zadaným názvem a zprávou. Druhá metoda, názvem login, je zde ta nejdůležitější, říká, co program má dělat po zmáčknutí tlačítka Login. Nejdříve je kontrola, zda je vyplněno pole username nebo login (pokud ne, tak program volá funkci displayAlert), pokud ano, vytvoří se activity indicator[5] a zároveň se zabrání interakci s aplikací uživatelem. Jelikož je do třídy implementován

[5]indikátor aktivity - animovaný objekt, co znázorňuje aktivitu programu v pozadí

framework Parse, můžu pomocí třídy PFUser() vyvolat metodu logInWithUsername-InBackground s parametry uživatelem zadaným loginem a heslem. Po vyvolání metody se zastaví indikátor aktivity a uživateli je umožněna kontrola nad aplikací. Pokud uživatel je v kolekci Users v databázi, je mu umožněn přístup do aplikace a provede se metoda performSegueWithIndentifier s identifikátorem Menu.

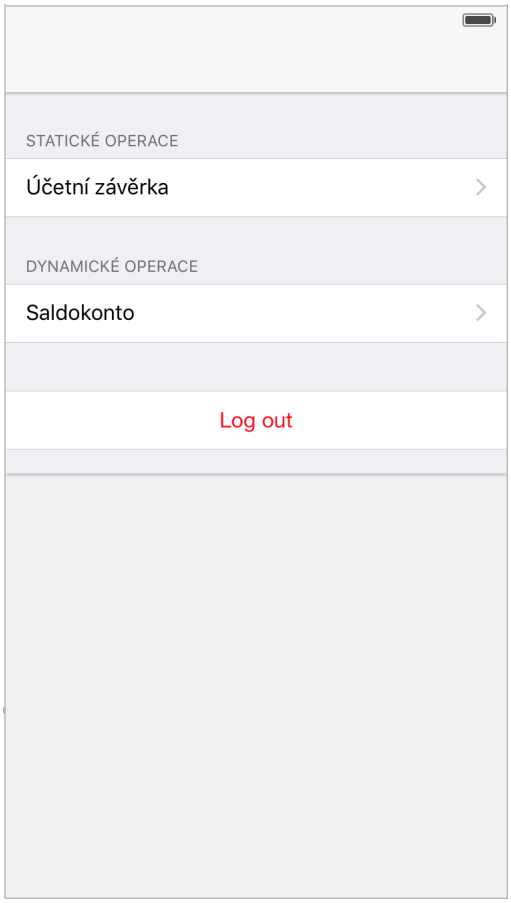
V této aplikaci nebude mít uživatel možnost se zaregistrovat jako nový uživatel, v praxi se oprávnění uživatelé budou přidávat manuálně do databáze.

6.3.3 Menu ViewController

Tento ViewController jsem vytvořil jako Table ViewController a pojmenoval ho Menu. Bude sloužit jako rozcestník mezi statickými reporty (Účetní závěrka) nebo dynamickými reporty (Saldokonto) a možností se odlogovat se z aplikace. Nejdříve jsem tuto tabulku definoval, že má mít statické buňky a mají být rozděleny do skupin. Dále jsem vytvořil tři takové skupiny, každá po jedné buňce. Pro buňky v prvních dvou skupinách jsem nadefinoval "basic" styl s accessory "Disclosure Indicator", výsledkem je, že v buňce na levé straně je text a na pravé je šipka. Třetí buňkou je pouhý text - styl je "basic". Všechny tři buňky slouží jako tlačítka pro pohyb aplikací dále.

Zároveň jsem při vybraném Menu ViewController v nabídce Editor -> Embed In dal volbu Navigation Controller. Tato volba vytvořila před tímto ViewController speciální controller, který řídí hierarchickou strukturu aplikace od toho Menu ViewController. Tzn. od tohoto obrázku dále, se objeví v horní části aplikaci speciální lišta, pomocí které můžeme automaticky zpět do předešlého okna. Segue z Login ViewController vede do Navigation Controller.

Dále jsem z buňky Účetní závěrka vytvořil segue do Účetní závěrka ViewController a z buňky Saldokonto segue do Saldokonto ViewController.



Obrázek 21: Menu ViewController Zdroj: Vlastní

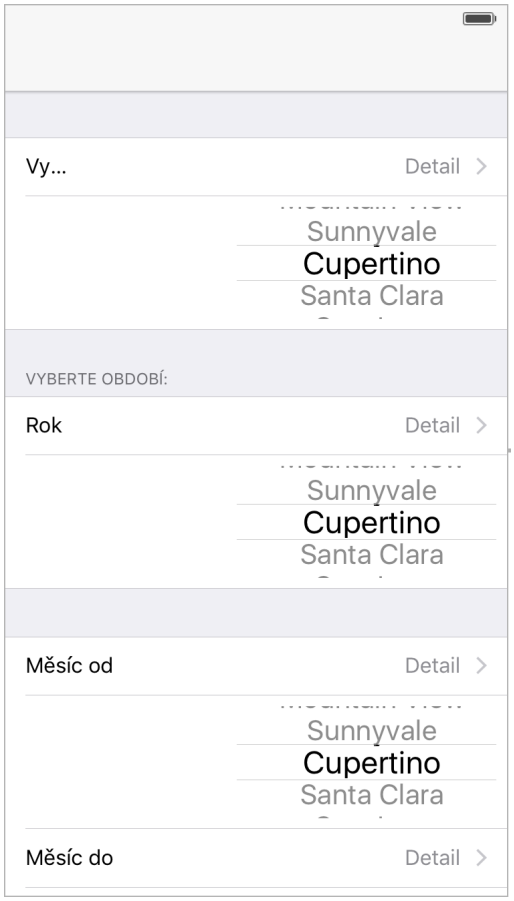
Navigation Controller nemá žádný mnou napsaný kód, používá pouze základní metody pro přechod mezi obrazovkami. Třída MenuViewController využívá jen pouze jednu základní metodu tableView s parametry tableView a didSelectRowAtPath. Pokud uživatel zmáčkne buňku v sekci dva s nultým indexem, program uživatele odloguje pomocí třídy PFUser a aplikace přes segue se vrátí zpět na obrazovku Login.

6.3.4 Saldokonto ViewController

ViewController Saldokonto jsem vytvořil jako Table ViewController s názvem Saldokonto. Tato tabulka je definovaná se statickými buňkami ve skupinách. Vytvořil jsem celkem čtyři skupiny buněk, první dvě skupiny mají každá dvě buňky, třetí skupina má buněk čtyři a poslední skupina má jen jednu buňku. První skupina slouží k výběru typu saldokontních účtů, které chceme sledovat. První buňka v této skupině je definovaná se stylem "Right Detail", kdy vlevo je definovaný text a vpravo je text získaný z jiných vstupů. Zde vstupem je hodnota z pickeru, který je vložen v druhé buňce. Tento picker je rolovací seznam předdefinovaných hodnot, které definujeme jako array přímo v

kódu. Všechny buňky s pickerema v tomto ViewController zamýšlím jako skrývatelné, kde po zmáčknutí buňky se stylem "Right Detail" se zobrazí buňka s pickerem a po opětovném zmáčknutí buňky se buňka s pickerem skryje. V dalších dvou skupinách buněk je podobný styl buněk, kdy si uživatel může vybrat podmínky pro danou sestavu: rok, měsíc od a měsíc do. Ve spodní části je definována buňka Vyhledat, která vrátí výsledný report.

Z tohoto ViewController vychází segue nazvaný Saldo a končí v dalším ViewController zvaným také Saldokonto (ve storyboardu názvy ViewController nemusí být unikátní).



Obrázek 22: Saldokonto ViewController Zdroj: Vlastní

Pro tento ViewController jsem vytvořil třídu SaldoParmViewController. Pro všechny label objekty s názvem Detail jsem vytvořil outlety a dále jsem vytvořil outlety i pro všechny čtyři pickery. Pomocí těchto outletů si definuji proměnné, se kterými budu pracovat. Tato třída má taktéž metodu displayAlert, funkčnost je stejná jako výše popsané. Metoda viewDidLoad se provede vždy při prvotním načtení ViewController. V této metodě tedy nejdříve nadefinuji array dat pro všechny pickery ucet, rok, mesicOd a mesicDo a jako defaultní hodnotu proměnných, se kterými bude dále pracovat, nechám

nultý index z konkrétních array. Tímto se tedy nemůže stát, aby uživatel zadal hodnotu, se kterým program nepočítá.

Metody pickerView slouží k definici atributů jednotlivých pickerů, kde identifikaci řeším pomocí tagování. První picker ve ViewController má tag 1 atd. První metoda numberOfComponentsInPickerView vrací počet sloupců v seznamu, zde je potřeba pouze jeden sloupec. Metoda pickerView s parametrem numberOfRowsInComponent vrací počet řádků v seznamu a tady to řeším count daného array. Další metoda pickerView s parametrem titleForRow vrací pro každý picker název řádku pro každý řádek v seznamu. Poslední pickerView s parametrem didSelectRow pro každý picker přiřadí hodnotu vybraného řádku k dané proměnné (ucet, rok, mesicOd, mesicDo) a zároveň provede update této hodnoty na label konkrétní buňky přes outlet.

Dále v metodě didViewLoad jsou dvě metody pro tableView. První tableView metoda s parametrem heightForRowAtIndexPath slouží pro skrývání buněk s pickery, konkrétně se zde nastavuje výška dané buňky. Když se ViewController poprvé načte, všechny buňky s pickery jsou skryté. Když uživatel zmáčkne buňku se šipkou, rozbalí se pod ní buňka s pickerem. Nejenom tuto činnost, rozbalování buněk, ale i výpočet zadaného reportu, umožní další metoda tableView, ale s parametrem didSelectRowAtPath. Pro buňku s nultým indexem v třetí sekci se po zmáčknutí buňky zobrazí další ViewController, pomocí segue Saldo.

Tento ViewController slouží pro výběr podmínek vypočítané sestavy. Po vybrání hodnot ucet, rok, mesicOd a mesicDo, potřebuji hodnoty těchto proměnných dostat do dalšího ViewController, kde proběhne samotný výpočet sestavy. Pro toto slouží metoda prepareForSegue. V této metodě si vytvořím instanci třídy kam chci proměnné předat, tedy SaldoTableViewController, a pak do této instance tyto proměnné předám. Aby toto předání hodnot proměnných fungovalo, musím v třídě SaldoTableViewController definovat stejné proměnné se stejným datovým typem se kterým je předávám.

Výsledek Saldokonta Tento ViewController byl také vytvořen jako Table ViewController. Ponechal jsem zde vlastnost buňky, že je dynamická (prototype) a tento prototyp buňky jsem nazval identifikátorem “cell”. Pro tento ViewController je to z hlediska nastavení ve Storyboardu vše, formátování buněk se definuje přímo v kódu.

Obrázek 23: SaldoTable ViewController Zdroj: Vlastní

Pro tento ViewController vytvořím třídu SaldoTableViewController. Tato třída využívá metod z frameworku Parse. Logiku třídy zabuduji do metody viewDidLoad, která se provede hned po načtení viewController. Nyní vysvětlím, jak fungují query do MongoDB přes Parse. V kódu vytvořím instanci query třídy PFQuery s parametrem názvem tabulky, ze které budu selectovat. Posléze pomocí metody whereKey zúžím dataset, který budu zpracovávat. Do těchto podmínek budu právě dávat proměnné z předchozího ViewController, kde jsem si zadával podmínky reportu - rok, mesic a ucet. Metoda findObjectsInBackgroundWithBlock vrátí celý set dat podle zadaných podmínek. MongoDB je NoSQL databáze, nelze zde použít relační techniky pro selectování dat, a proto zde nejde udělat group by. Jelikož ale výsledkem tohoto selectu je seznam účtů a výsledná suma Má Dáti (v databázi atribut kcm) - Dal (zde kcd), musím celou logiku zabudovat do for funkce. Nejdříve vytvořím dictionary sumUcet, kde klíčem bude účet a hodnotou suma. Každý řádek ve výsledném datasetu prohledám a pokud naleznu atribut ucet, který v dictionary není, tak tento účet do dictionary přidám se sumou Má Dáti - Dal. Pokud atribut ucet je už v dictionary, pouze vezmu hodnotu sumy k tomuto účtu a přičtu sumu Má Dáti a odečtu Dal. Na závěr zavolám metodu

reloadData, která udělá refresh tabulky.

Tabulka se řídí podle výsledné dictionary, kterou jsem použil v selectu. Metoda numberOfSectionsInTableView vrací hodnotu sloupců v tabulce, zde tedy jeden sloupec tabulky. Metoda tableView s parametrem numberOfRowsInSection vrací kolik řádku tabulka má. Zde vrací počet řádků rovno hodnotě count dictionary sumUcet. Metoda tableView s parametrem cellForRowAtIndexPath slouží k definici buňky, kdy množství buněk je dynamická hodnota. Já zde do každé buňky dám text a odkaz do dictionary sumUcet přes array, aby každý řádek (buňka) měl hodnotu rovnu jednomu klíči ze sumUcet. Výsledkem je seznam účtů a jejich suma.

6.3.5 Účetní závěrka ViewController

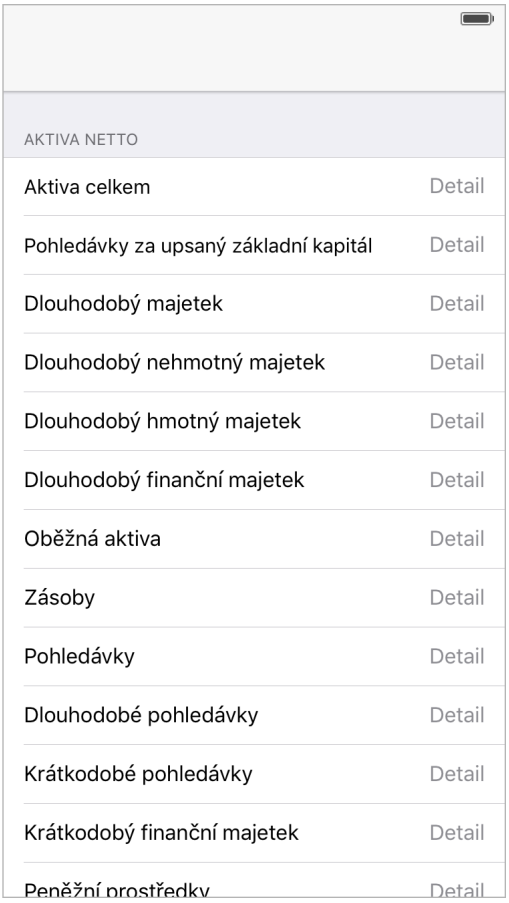
ViewController pro Účetní závěrka je definována podobně jako pro Saldokonto. Je zde vytvořen Table ViewController, jsou zde statické buňky rozdělené do čtyř skupin. První dvě skupiny mají každá dvě buňky, třetí skupina má čtyři buňky a poslední skupina má jen jednu buňku. Zde jsem udělat stejným způsobem, jako u ViewController Saldokonto definované buňky s pickery. Jediný rozdíl je, že uživatel nebude vybírat ze dvou druhů účtů, ale ze dvou druhů výkazů.

Z tohoto ViewController vedou dva segue. Jeden z názvem VykazZaZ do Vykaz ZaZ ViewController a jeden s názvem Rozvaha do Rozvaha ViewController.

Obrázek 24: Závěrka ViewController Zdroj: Vlastní

Pro tento ViewController jsem vytvořil třídu ZaverkaParmViewController. Kód je opět téměř shodný s třídou SaldoParmViewController. Největším rozdílem je to, že tato třída slouží jako rozcestník do dvou různých ViewController. V kódu jsem to jednoduše vyřešil tak, že kdy přes if else se proměnná rovná vybranému výkazu, aplikace půjde po zadaném segue. To samé jsem musel udělat u metody prepareForSegue, podle zadaného výkazu jsem vytvořil instanci výsledného ViewController a do ní jsem přesměroval vybrané proměnné.

Rozvaha ViewController ViewController s názvem Rozvaha je založena jako Table ViewController se statickým počtem buněk. Vzorem pro tuto tabulku byla Rozvaha ve zjednodušeném rozsahu. Buňky jsou rozděleny do dvou skupin, z nichž první jsem nadepsal jako Aktiva netto a druhou jako Pasiva. Skupina Aktiva Netto má 14 řádek a skupina Pasiva má také 14 řádek. Každá buňka je definovaná se stylem "Right Detail", kde na levé straně je informační text a na pravé straně bude text s proměnlivou hodnotou.

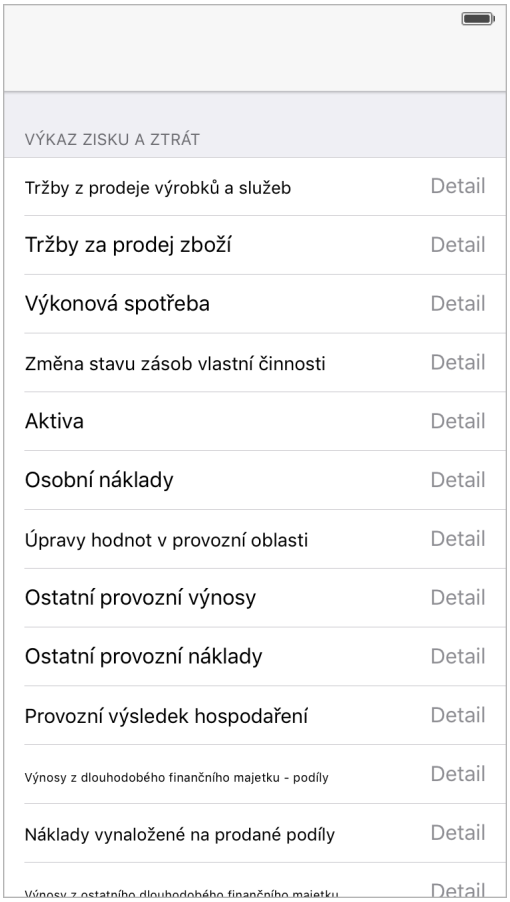


Obrázek 25: Rozvaha ViewController Zdroj: Vlastní

Vytvořil jsem třídu RozvahaTableViewController. Pro každý label Detail jsem ve třídě vytvořil outlet s příslušnou proměnnou (28 celkem). Pro každou proměnnou jsem nastavil default hodnotu 0, abych měl jistotu, že budu vždy pracovat s čísly. Dále jsem si vytvořil metodu query, která přijímá parametry ucet a speciální příznak p (značí způsob výpočtu sumy hodnot ve výsledku). Tato metoda query volá metodu třídy PFQuery s parametry pro výběr dat definované v Závěrka ViewController. Metoda findObjectsInBackgroundWithBlock vrací výsledek selectu. Zde je ovšem problém, že tento select probíhá v pozadí, mimo hlavní thread aplikace. V metodě query jsem musel tedy přidat completionHandler, který říká, že tato metoda vrací hodnotu až tehdy, kdy doběhne zadaný select.

Po načtení této obrazovky se postupně zavolá metoda query pro každou skupinu účtů a vrací se suma této skupiny. Podle šablony Rozvahy v kapitole 6.1 se do každé buňky vloží suma skupiny účtů, které do dané buňky patří. Součtové řádky jsem přidal do posledních metod query, kdy mám jistotu, že tyto metody budou zavolány jako poslední a omylem nezapomenu nějakou hodnotu přičíst či odečíst.

Výkaz zisku a ztrát ViewController Poslední vytvořený ViewController s názvem Výkaz zisku a ztrát byl také vytvořen jako Table ViewController se statickým počtem buněk. Tyto buňky jsou v pouze v jedné skupině a s nadpisem skupiny "Výkaz zisku a ztrát". Styl buněk je stejný jako ve ViewController pro Rozvahu. Zde jsem vložil 26 řádků do tabulky. Pro tuto tabulku byl vzorem Výkaz zisku a ztrát ve zjednodušeném rozsahu, vytvořený podle šablony v kapitole 6.1.



Obrázek 26: Výkaz zisku a ztrát ViewController Zdroj: Vlastní

VykazZaZTableViewController je názvem poslední přidané třídy. Logika je stejná jako v třídě RozvahaTabelViewController. Jen jsem zde použil jiný algoritmus pro výpočet buněk podle zadané šablony Výkaz zisku a ztrát ve zjednodušeném rozsahu.

6.4 Závěr

V první části této kapitoly jsem nejdříve připravil výstupní sestavy k aplikaci, kterou budu vyvíjet. Navrhl jsem, co by aplikace měla dělat a podle jaké šablony by se sestavy měly vypočítat. V druhé části jsem vytvořil serverové prostředí pro moji aplikaci,

nejdříve jsem na server Heroku udělal deploy Parse server s databázovým systémem MongoDB. Po úspěšné konfiguraci jsem v aplikaci nakonfiguroval spojení aplikace se serverem. A v závěru této části jsem připravil převodový můstek dat mezi MSSQL a MongoDB a posléze jsem databázi na Parse naplnil daty. V poslední, třetí, části jsem popsal klientskou část aplikace WiRIS, kterou jsem napsal v jazyce Swift v IDE Xcode.

7 Testování řešení

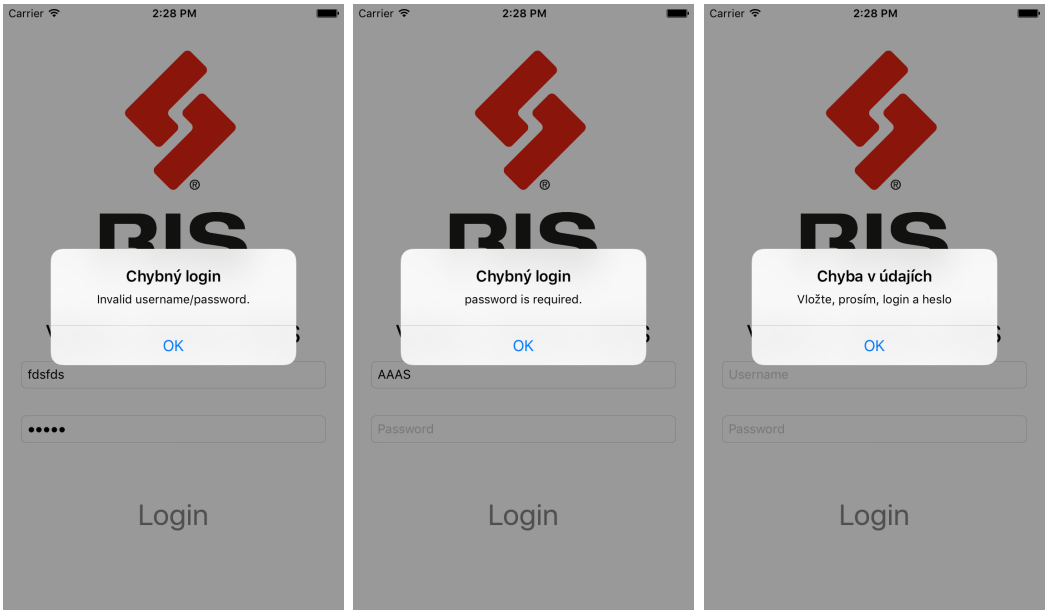
Po úspěšné kompilaci aplikace WiRIS v Xcode se nyní pokusím o testování aplikace. Připravím čtyři testovací scénáře, pomocí kterých bych chtěl otestovat všechny části aplikace, zda korektně fungují jak z programového hlediska, aby během běhu aplikace nedošlo k pádu, tak i z logického hlediska, aby výsledky aplikace byly použitelné.

První testovací scénář se bude týkat přihlášení se do aplikace. Potřebuji otestovat, zda se uživatel dostane do aplikace. Zároveň potřebuji otestovat chybové stavy - špatně zadaný login, či uživatel není autorizován pro přístup do aplikace. Další tři scénáře otestují výsledky reportů, zda fungují a vrací správná data. Správnost dat ověřím select přímo z produkční databáze MSSQL. Druhý scénář bude testovat rozvahu, třetí výkaz zisku a ztrát a poslední, čtvrtý, testuje výsledky reportu saldokontních účtů.

7.1 První scénář - Login

Zadání Otestuj, jestli aplikace správně odmítne špatně zadaný login a pokus o přístup neautorizovanou osobou. Dále, zda se uživatel William úspěšně přihlásí do aplikace.

Výsledek Nejdříve otestuji přihlášení pod vymyšleným uživatelem. Aplikace správně vyhodnotí neautorizovaný přístup, uživatel fdsfds není v databázi uživatelů na serveru. Pokud zapomenu při přihlášení zadat heslo, aplikace upozorní na nezadané heslo. Pokud vůbec nezadám údaje a pokusím se přihlásit, objeví se varovné okno. Důvod, proč jsou některé varování v anglickém a některé v českém jazyce je ten, že chybové zprávy o špatné autorizaci pocházejí přímo z Parse a ten není lokalizovaný do českého jazyka. Možným řešením je získat seznam chybových hlášek z Parse a přes if else přeložit danou chybovou zprávu.



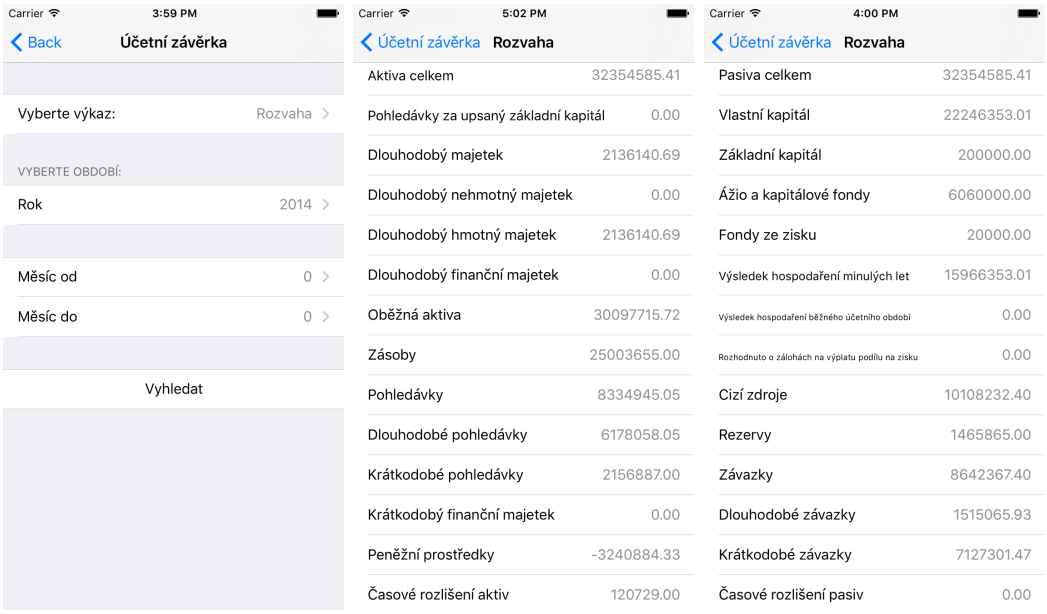
Obrázek 27: Výsledky testování loginu Zdroj: Vlastní

Při zadání správných přihlašovacích údajů - login: William heslo: dupkala - se uživatel po dotazu aplikace do Parse serveru autorizuje a objeví se hlavní menu aplikace. Výsledek testu splňuje zadání.

7.2 Druhý scénář - Rozvaha

Zadání Otestuj, zda počáteční zůstatky v nultém účetním období roku 2014 jsou správně zadané. Zjisti to pomocí rozvahy, kdy aktiva se budou rovnat pasivům. Dále ověř správnost výsledků reportu.

Výsledek Na obrazovce Účetní závěrka vyberu výkaz Rozvaha, rok 2014 a pro měsíc od a měsíc do stejnou hodnotu 0. Na první pohled vidím, že se aktiva celkem a pasiva celkem rovnají. Z pohledu účetnictví tato sestava vypadá správně.



Obrázek 28: Testování rozvahy Zdroj: Vlastní

Pro ověření správnosti výsledku udělám z databáze MSSQL export dat v podobném formátu, jaký je dělán v aplikaci WiRIS. Použiji výsledné hodnoty a dosadím hodnoty pro dané účty do vzorové šablony pro Rozvahu ve zjednodušeném rozsahu. Výsledkem je mi stejná sestava, která se mi vygenerovala z aplikace. Zadání bylo splněno bez chyb.

```
select substring(a.ucet, 1, 3), SUM(a.kcm) - SUM(a.kcd)
from saul.me10 a, saul.me17 b
where a.rok = b.rok and a.denik = b.denik and a.doklad = b.doklad
and a.rok = 2014 and b.mesic = 0
group by substring(a.ucet, 1, 3)
order by substring(a.ucet, 1, 3) asc
```

Results | Messages

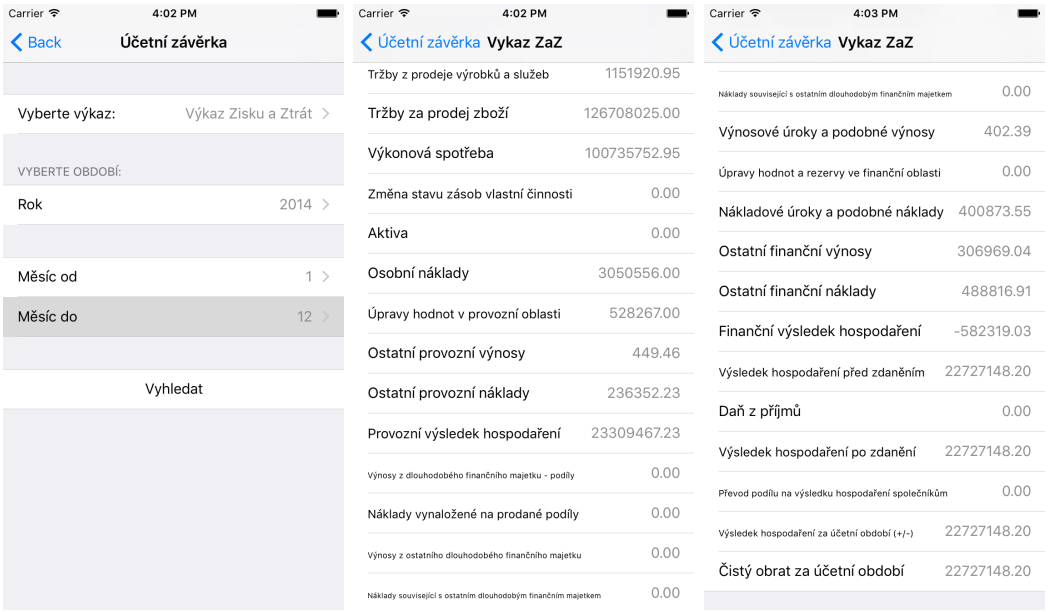
	(No column name)	(No column name)
1	022	2436495.69
2	082	-300355.00
3	132	25003655.00
4	211	3176833.28
5	221	-6417717.61
6	249	-2000000.00
7	311	4042788.65
8	314	153669.40
9	315	25600.00
10	321	-4979922.47
11	331	-96419.00
12	336	-49496.00
13	341	495720.00
14	342	-1464.00
15	343	1661167.00
16	365	-124234.96
17	378	1956000.00
18	381	120729.00
19	389	-268182.03
20	411	-200000.00
21	413	-6060000.00
22	421	-20000.00
23	428	-11333708.79
24	431	-4632644.22
25	459	-1465865.00
26	479	-1122648.94
27	701	0.00
28	710	0.00

Obrázek 29: Select účtů v období 0 Zdroj: Vlastní

7.3 Třetí scénář - Výkaz zisku a ztrát

Zadání Otestuj, aby data vypočtena aplikací pro výkaz zisku a ztrát, při podmínkách rok rovná se 2014 a období 1 (leden) až 12 (prosinec) a ověř správnost výsledků reportu.

Výsledek Na obrazovce Účetní závěrka zadám podmínku rok rovná 2014 a měsíce od 1 (leden) do 12 (prosinec). Ve výkazu zisku a ztrát je poslední řádek "Čistý obrat za účetní období" kontrolní. Výpočet tohoto řádku je součet zůstatků všech výnosových účtů mínus součet všech zůstatků nákladových účtů (účty třídy 6 mínus účty třídy 5). Hodnota tohoto řádku se musí vždy rovnat hodnotě řádku předchozího - "Výsledek hospodaření za účetní období (+/-)", který je výsledkem sestavy. Jelikož hodnoty obou řádek se rovnají, zadání scénáře bylo provedeno bez chyb.

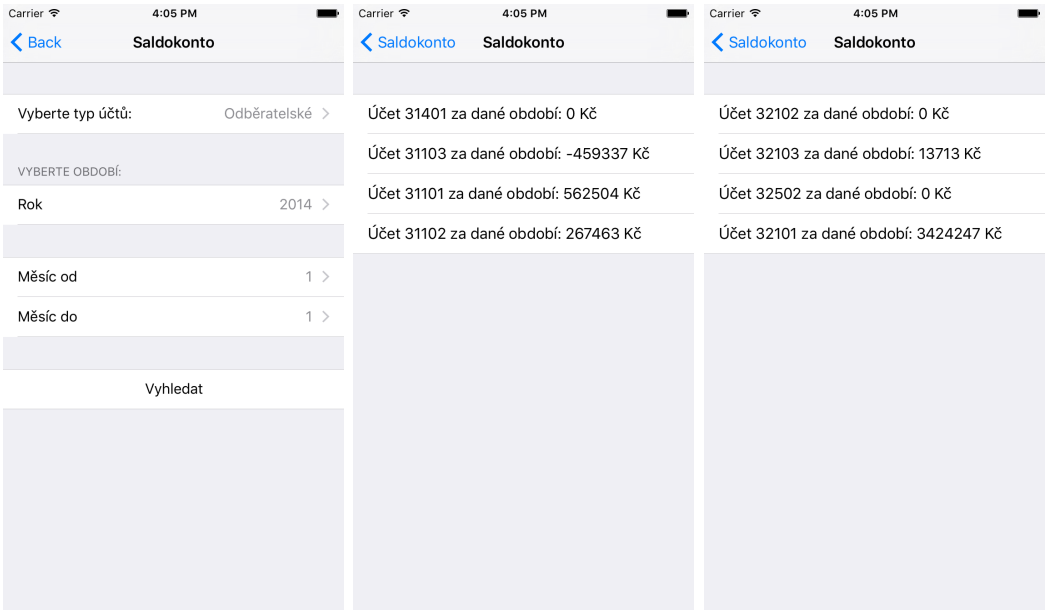


Obrázek 30: Testování výkazu zisku a ztrát Zdroj: Vlastní

7.4 Čtvrtý scénář - Saldokonto

Zadání Otestuj, zda se liší sestava pro rok 2014 a období 1 (leden) pro odběratelské a dodavatelské účty. Dále ověř správnost výsledků reportu.

Výsledek Na obrazovce Saldokonto vyberu rok 2014 a pro měsíc od a měsíc do stejnou hodnotu 1 (leden). Výsledek pro odběratelské a dodavatelské účty se vizuálně liší, což je správně. Pro kontrolu správnosti výsledku využiji produkční databázi MSSQL.



Obrázek 31: Testování saldokonta Zdroj: Vlastní

Z databáze MSSQL udělám select, kde vyberu odběratelské saldokontní účty (začínají 31) a dodavatelské saldokontní účty (začínají 32). Pokud srovnám výsledky selectu (zaokrouhlím na celé číslo) a výsledky sestavy, hodnoty se rovnají. Zadání scénáře bylo splněno.

```
select a.ucet, SUM(a.kcm) - SUM(a.kcd)
from saul.me10 a, saul.me17 b
where a.rok = b.rok and a.denik = b.denik and a.doklad = b.doklad
and a.rok = 2014 and b.mesic = 1
and a.ucet like '31%'
group by a.ucet
UNION
select a.ucet, SUM(a.kcm) - SUM(a.kcd)
from saul.me10 a, saul.me17 b
where a.rok = b.rok and a.denik = b.denik and a.doklad = b.doklad
and a.rok = 2014 and b.mesic = 1
and a.ucet like '32%'
group by a.ucet
```

Results | Messages

ucet	(No column name)
31401	0.00
31103	-459335.65
31102	267467.75
31101	562504.00
32101	3424247.98
32502	0.00
32102	0.00
32103	13712.50

Obrázek 32: Select saldokontních účtů období 1 Zdroj: Vlastní

7.5 Závěr

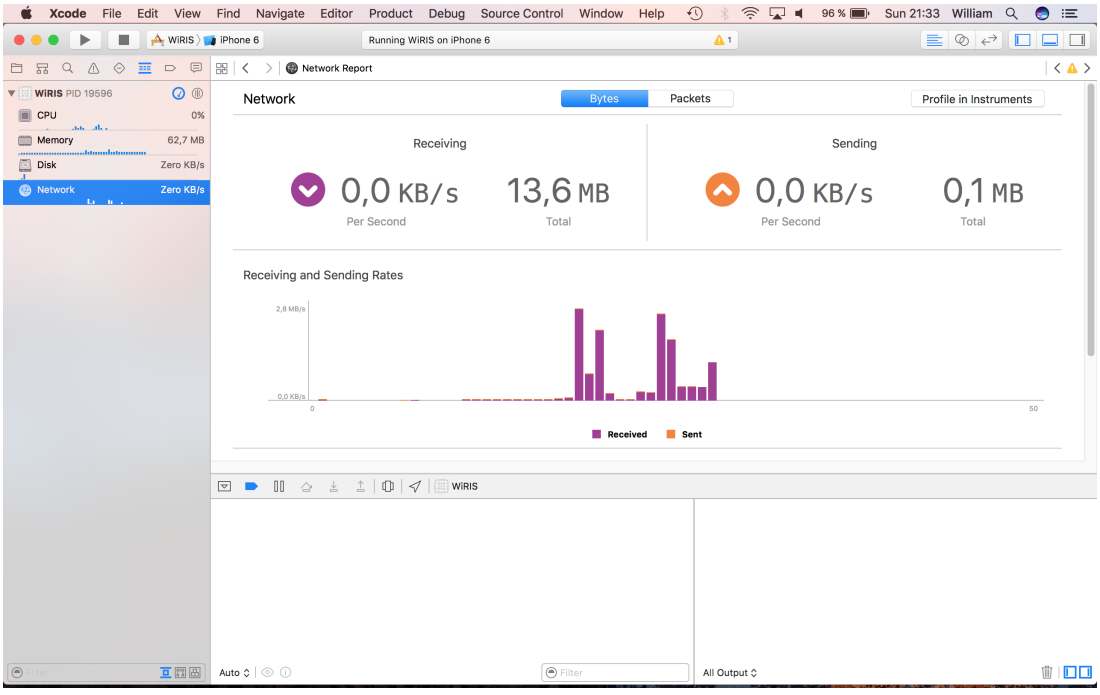
Pro testování aplikace jsem vytvořil čtyři testovací scénáře. Všechny scénáře byly splněny bez chyb z programového hlediska, nedošlo ani jednou k chybě či pádu programu, a také nebyly nalezeny chyby z logického hlediska. Výsledky sestav byly shodné s výsledky získané přímo z produkční databáze.

8 Diskuze

Vytvořená aplikace pomocí testovacích scénářů otestovala hlavní části aplikace. Výsledky testů neukázaly chyby v programu.

Aplikace je nyní ve stavu, aby byla nasazena do provozu, tzn. je nyní možné požádat o umístění aplikace do App store. Ve vybraném řešení, třívrstvé architektury, bych nic neměnil, Parse server na Heroku funguje. Pokud bych ale měl aplikaci nasadit do ostré produkce, prodiskutoval bych pár věcí ve spolupráci s dodavatelem systému RIS2000 a TOP-RIS, firmu SAUL IS spol. s r. o., které bych upravil či přidal. Na straně klientské části aplikace bych přidal možnost, aby šlo ve výsledných sestavách, po zmáčknutí konkrétní buňky uživatelem, otevřít detail dané buňky - z jakých položek se buňka skládá (jaká data byla použita k výpočtu dané buňky). Tato funkce je složitější k implementaci, ale zvýšil by uživatelský komfort.

Provedl jsem diagnostiku aplikace při práci v síti. Na obrázku uvedeném níže jsem obdržel výsledky při pokusu o výpočtu rozvahy za celý rok 2014 (tzn. období 0 - 12). Při objemu databáze 45 207 záznamů, aplikace stáhla ze serveru jednu sestavu - rozvahu ve velikosti 13.6 MB.



Obrázek 33: Diagnostika aplikace Zdroj: Vlastní

Uvažuji, že tuto aplikaci budou většinou používat uživatelé, co využívají služební telefony, a proto výsledná velikost stažené rozvahy by neměla ničemu vadit. Na základě

těchto informací bych ale se pokusil o jiný přístup k získávání dat do aplikace. Parse server umožňuje některé metody z aplikace, co by případně zbytečně čerpaly zdroje mobilního přístroje, umístit do služby Parse Cloud. Metoda z aplikace z jazyka Swift by se přeložila do JavaScriptové metody na Parse. Aplikace by tedy místo volání metody přímo v programu, volala metodu, přes API Parse Cloud, v JavaScriptu na serveru a data by se stejnou cestou vracela zpět do aplikace. Nevýhoda je však ta, že v případě takto složitějších metod je třeba ještě dobře umět JavaScript.

Závěr

Cílem této diplomové práce bylo vytvořit funkční BI aplikaci pro operační systém iOS, které je napojené na ERP systém RIS2000 od společnosti SAUL IS spol. s r. o. Osoba, která chce vytvořit takovou aplikaci a rozumět tomu, musí znát širokou škálu vědomostí z různých oblastí. Vzhledem k tomu, že některé oblasti zkoumání jsou poměrně nové, nelze k nim nalézt mnoho zdrojů. Tato práce popisuje tyto oblasti a dává je dohromady do jednoho celku.

Ve své práci jsem čtenáři představil pojem ERP, jak se může členit a popsal jsem teorii implementace ERP do podniku. Jako reálný příklad ERP jsem popsal systém RIS2000, kde jsem se prioritně věnoval nejpoužívanějšímu modulu - účetnictví. Nadstavbou nad ERP jsou nástroje BI, programy, pomocí kterých uživatelé analyzují data z produkční databáze. Nadstavbou nad RIS2000 je BI nástroj TOP-RIS, který dokáže analýzy a reporty nad účetními daty. Aplikaci, kterou jsem napsal, je podobným nástrojem jako TOP-RIS, je to mobilní BI řešení pro RIS2000 a také dělá analýzy a reporty nad účetními daty z RIS2000.

Čtenář se seznámil i s prostředím, do kterého aplikace byla vyvíjena, s operačním systémem iOS od firmy Apple Inc. Pro vývoj v iOS je potřeba splnit několik požadavků. Tyto požadavky, co je potřeba k vývoji a jaké jsou rady pro vývojáře od firmy Apple, byly sepsány do jedné kapitoly. Dále jsem čtenáře seznámil s moderním objektově-orientovaným programovacím jazykem Swift, také vyvinutým firmou Apple Inc. Pokusil jsem se ukázat syntaxi jazyka na malém příkladu, co vyhodnocuje, zda číslo či není prvočíslo.

Analýzou všech přednesených dat jsem dospěl k seznamu požadavků, které by měla vyvíjená aplikace splňovat. Aplikaci jsem rozdělil do tří vrstev a vyhledal jsem způsob, jak realizovat každou vrstvu. Klientská vrstva je realizována v jazyce Swift pro iOS, serverová vrstva je obsloužena službou Parse, která musela být hostována na cloudové službě Heroku. Databázovou vrstvou je NoSQL MongoDB, která je napojená na Parse.

Parse server jsem úspěšně nasadil na cloudovou službu Heroku, popsal jsem co bylo potřeba ke zprovoznění takového serveru. V klientské vrstvě, v aplikaci, kterou jsem nazval WiRIS, jsem úspěšně aplikaci napojil na server a otestoval jsem i spojení s databází. Pro aplikaci jsem i připravil zdroje sestav, které aplikace bude vypočítávat z dat z databáze.

Naprogramovaná aplikace je na CD přiloženém k této diplomové práci. Popsal jsem vývoj této aplikace v Xcode, uživatelské rozhraní a logiku všech tříd v programu.

Výslednou aplikaci jsem otestoval pomocí daných testovacích scénářů. Veškeré testy doběhly s pozitivním výsledkem. Na závěr jsem otestoval aplikaci v síťovém provozu a dospěl jsem k názoru, že pokud bych aplikací uvedl do reálného provozu, algoritmus pro výběr dat bych umístil místo z klienta na server.

Reference

[1] GÁLA, Libor; POUR, Jan; ŠEDIVÁ, Zuzana. *Podniková informatika*. 2. přepracované a aktualizované vydání. Praha. 2011. Grada Publishing. ISBN 978-80-247-2615-1.

[2] BASL, Josef; BLAŽÍČEK, Roman. *Podnikové informační systémy : Podnik v informační společnosti*. 3. aktualizované a doplněné vydání. Praha. 2012. Grada Publishing. s.128. ISBN 978-80-247-4307-3.

[3] MONK, Ellen F.; WAGNER, Bret J. *Concepts in enterprise resource planning*. 4. edition. Boston, United States of America. 2013. Course Technology. ISBN 978-1-111-82039-8.

[4] BRUCKNER, Tomáš; VOŘÍŠEK, Jiří; BUCHALCEVOVÁ, Alena. *Tvorba informačních systémů : Principy, metodiky, architektury*. 1. vydání. Praha. 2012. Grada Publishing. ISBN 978-80-247-4153-6.

[5] KOCH, Miloš. *Management informačních systémů*. 1. vydání. Brno. 2006. Akademické nakladatelství CERM. ISBN 80-214-3262-4.

[6] SAUL IS. *RIS2000*. Praha. 2016.

[7] SAUL IS. *TOP-RIS* [online]. Praha [cit. 2016-09-24]. Dostupné z WWW: <`http://www.saul.cz/foto/TOP-RIS`>.

[8] GREBEŇ, David. *Kompletní historie iOS* [online]. Praha [cit. 2017-01-10].Zavřel Media. Dostupné z WWW: <`https://www.letemsvetemapplem.eu/2016/03/06/kompletni-historie-ios`>.

[9] APPLE INC. *iOS Human Interface Guidelines* [online]. Cupertino [cit. 2017-01-14]. Dostupné z WWW: <`https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles`>.

[10] APPLE INC. *Software Licence Agreements* [online]. Cupertino [cit. 2017-01-20]. Dostupné z WWW: <`http://images.apple.com/legal/sla/docs/OSX1011.pdf`>.

[11] APPLE INC. *Cocoa (Touch)* [online]. Cupertino [cit. 2017-02-21]. Dostupné z WWW: <`https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/Cocoa.html#//apple_ref/doc/uid/TP40008195-CH9-SW1`>.

[12] APPLE INC. *Core Data* [online]. Cupertino [cit. 2017-02-21]. Dostupné z WWW: <https://developer.apple.com/library/content/documentation/Cocoa//Conceptual/CoreData/index.html>.

[13] APPLE INC. *Apple Developer Program* [online]. Cupertino [cit. 2017-02-21]. Dostupné z WWW: <https://developer.apple.com/programs/how-it-works>.

[14] APPLE INC. *Swift and oper source* [online]. Cupertino [cit. 2017-02-24]. Dostupné z WWW: <https://swift.org/about/#swiftorg-and-open-source>

[15] NUTTING, Jack, Dave MARK a Jeff LAMARCHE. *Cocoa: průvodce programováním pro Mac*. Brno: Computer Press, 2011. ISBN 978-80-251-2883-1.

[16] APPLE INC. *Strings and Characters* [online]. Cupertino [cit. 2017-02-25]. Dostupné z WWW: <https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/StringsAndCharacters.html>.

[17] HEROUT, Pavel. *Učebnice jazyka Java*. 3., rozš. vyd. České Budějovice: Kopp, 2007. ISBN 978-80-7232-323-4.

[18] MAROTTO, Fosco. *What is Parse Server* [online]. [cit. 2016-12-20]. Dostupné z WWW: <http://blog.parse.com/announcements/what-is-parse-server/>

[19] MONGO DB INC. *What is MongoDB* [online]. [cit. 2016-12-20]. Dostupné z WWW: <https://www.mongodb.com/what-is-mongodb>

[20] PARSE. *Open Source* [online]. [cit. 2017-02-02]. Dostupné z WWW: <http://parseplatform.org>.

[21] STACK EXCHANGE. *Heroku : About Heroku* [online]. [cit. 2017-02-02]. Dostupné z WWW: <https://stackoverflow.com/tags/heroku/info>.

[22] Zákon č. 563/1991 Sb., o účetnictví. Dostupné z WWW: <http://www.mfcr.cz/assets/cs/media/Zak_1991-563_UZ-zakona-o-ucetnictvi-bez-vyznaceni-zmen.pdf>

Příloha A

Použitá loga v aplikaci. Zdroj z WWW: <http://www.saul.cz/>

Příloha B

Úplný výpis z logu převodu dat mezi MSSQL a MongoDB.

Starting copy...

1000 rows successfully bulk-copied to host-file. Total received: 1000

1000 rows successfully bulk-copied to host-file. Total received: 2000

1000 rows successfully bulk-copied to host-file. Total received: 3000

1000 rows successfully bulk-copied to host-file. Total received: 4000

1000 rows successfully bulk-copied to host-file. Total received: 5000

1000 rows successfully bulk-copied to host-file. Total received: 6000

1000 rows successfully bulk-copied to host-file. Total received: 7000

1000 rows successfully bulk-copied to host-file. Total received: 8000

1000 rows successfully bulk-copied to host-file. Total received: 9000

1000 rows successfully bulk-copied to host-file. Total received: 10000

1000 rows successfully bulk-copied to host-file. Total received: 11000

1000 rows successfully bulk-copied to host-file. Total received: 12000

1000 rows successfully bulk-copied to host-file. Total received: 13000

1000 rows successfully bulk-copied to host-file. Total received: 14000

1000 rows successfully bulk-copied to host-file. Total received: 15000

1000 rows successfully bulk-copied to host-file. Total received: 16000

1000 rows successfully bulk-copied to host-file. Total received: 17000

1000 rows successfully bulk-copied to host-file. Total received: 18000

1000 rows successfully bulk-copied to host-file. Total received: 19000

1000 rows successfully bulk-copied to host-file. Total received: 20000

1000 rows successfully bulk-copied to host-file. Total received: 21000

1000 rows successfully bulk-copied to host-file. Total received: 22000

1000 rows successfully bulk-copied to host-file. Total received: 23000

1000 rows successfully bulk-copied to host-file. Total received: 24000

1000 rows successfully bulk-copied to host-file. Total received: 25000

1000 rows successfully bulk-copied to host-file. Total received: 26000

1000 rows successfully bulk-copied to host-file. Total received: 27000

1000 rows successfully bulk-copied to host-file. Total received: 28000

1000 rows successfully bulk-copied to host-file. Total received: 29000

1000 rows successfully bulk-copied to host-file. Total received: 30000

1000 rows successfully bulk-copied to host-file. Total received: 31000

1000 rows successfully bulk-copied to host-file. Total received: 32000

1000 rows successfully bulk-copied to host-file. Total received: 33000

1000 rows successfully bulk-copied to host-file. Total received: 34000

1000 rows successfully bulk-copied to host-file. Total received: 35000

1000 rows successfully bulk-copied to host-file. Total received: 36000

1000 rows successfully bulk-copied to host-file. Total received: 37000

1000 rows successfully bulk-copied to host-file. Total received: 38000

1000 rows successfully bulk-copied to host-file. Total received: 39000

1000 rows successfully bulk-copied to host-file. Total received: 40000

1000 rows successfully bulk-copied to host-file. Total received: 41000

1000 rows successfully bulk-copied to host-file. Total received: 42000

1000 rows successfully bulk-copied to host-file. Total received: 43000

1000 rows successfully bulk-copied to host-file. Total received: 44000

1000 rows successfully bulk-copied to host-file. Total received: 45000

45207 rows copied.

Network packet size (bytes): 4096

Clock Time (ms.) Total : 1843

Average : (24529.03 rows per sec.)

2017-03-11T14:31:08.545+0100 connected to: ds115110.mlab.com:15110

2017-03-11T14:31:08.586+0100 dropping: heroku_6hnnsb1v.doklad

2017-03-11T14:31:11.164+0100 [####.....] heroku_6hnnsb1v.doklad 1.14MB/6.51MB (17.6%)

2017-03-11T14:31:14.182+0100 [#########.....] heroku_6hnnsb1v.doklad 2.61MB/6.51MB (40.0%)

2017-03-11T14:31:17.165+0100 [#############.....] heroku_6hnnsb1v.doklad 4.05MB/6.51MB (62.3%)

2017-03-11T14:31:20.164+0100 [####################....] heroku_6hnnsb1v.doklad 5.50MB/6.51MB (84.5%)

2017-03-11T14:31:23.164+0100 [####################...] heroku_6hnnsb1v.doklad 5.91MB/6.51MB (90.8%)

2017-03-11T14:31:24.901+0100 [########################] heroku_6hnnsb1v.doklad 6.51MB/6.51MB (100.0%)

2017-03-11T14:31:24.902+0100 imported 45207 documents

MongoDB shell version v3.4.2

connecting to: mongodb://ds115110.mlab.com:15110/heroku_6hnnsb1v

MongoDB server version: 3.2.12

WARNING: shell and server versions do not match

Press any key to continue . . .