

Západočeská univerzita v Plzni  
Fakulta aplikovaných věd  
Katedra informatiky a výpočetní techniky

## **Diplomová práce**

### **Docházka a výkazy práce pro systém IMIS na platformě Android**

# Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 7. července 2013

Maxipes Fík

# Abstract

Text of abstract.

# Obsah

<b>1</b>	<b>Úvod</b>	<b>1</b>
<b>2</b>	<b>Současný systém</b>	<b>2</b>
2.1	Vybraná funkcionalita . . . . .	2
2.1.1	Evidence docházky . . . . .	2
2.1.2	Vykazování odvedené práce . . . . .	3
2.1.3	Motivace . . . . .	3
2.2	Použitá technologie . . . . .	4
2.2.1	Oracle Forms . . . . .	4
2.2.2	Architektura . . . . .	5
2.2.3	Komponenty formuláře . . . . .	6
2.2.4	Uživatelské rozhraní . . . . .	9
2.3	Důležité formuláře . . . . .	9
2.3.1	Zápis příchodů a odchodů . . . . .	10
2.3.2	Výkaz práce . . . . .	11
<b>3</b>	<b>Analýza</b>	<b>12</b>
3.1	Architektura . . . . .	12
3.1.1	Přímé připojení k databázi . . . . .	12
3.1.2	Oracle Database Mobile Server . . . . .	12
3.1.3	Webová služba . . . . .	13
3.1.4	Zvolená architektura . . . . .	14
3.2	Výběr typu webové služby . . . . .	15
3.2.1	Representational State Transfer . . . . .	15
3.2.2	Simple Object Access Protocol . . . . .	16
3.2.3	Odůvodnění výběru . . . . .	18
3.3	Datová vrstva . . . . .	18
3.3.1	Datový model . . . . .	18
3.3.2	Práce s datumem a časem . . . . .	19
3.3.3	Výhrady k datové vrstvě . . . . .	20
3.4	Business logika . . . . .	21

3.4.1	Triggery . . . . .	21
3.4.2	Databázové balíčky a uložené procedury . . . . .	21
3.4.3	Forms knihovny . . . . .	22
3.5	Synchronizace dat . . . . .	22
3.5.1	Obousměrná synchronizace . . . . .	22
3.5.2	Obousměrná synchronizace s úpravou databáze . . . . .	25
3.5.3	Řešení kolizí . . . . .	25
3.5.4	Srovnání . . . . .	26
3.6	Uživatelské rozhraní . . . . .	26
<b>4</b>	<b>Zabezpečení</b>	<b>27</b>
4.1	Autentizace a autorizace . . . . .	27
4.2	VPN pro vzdálený přístup . . . . .	28
4.3	Autentizace proti databázi . . . . .	29
4.4	Autentizace proti LDAP . . . . .	30
4.4.1	LDAP . . . . .	30
4.5	Shrnutí . . . . .	32
<b>5</b>	<b>Webová služba</b>	<b>33</b>
5.1	REST . . . . .	33
5.2	Filtry . . . . .	36
5.3	web.xml . . . . .	36
<b>6</b>	<b>Android aplikace</b>	<b>37</b>
6.1	Funkcionalita . . . . .	37
6.1.1	Nastavení a konfigurovatelnost . . . . .	38
6.1.2	Uživatelská přívětivost . . . . .	38
6.1.3	Návrhy na vylepšení . . . . .	38
6.2	Komponenty Android aplikace . . . . .	38
6.2.1	Aktivita . . . . .	38
6.2.2	Fragment . . . . .	41
6.2.3	Služba . . . . .	41
6.2.4	Content provider . . . . .	41
6.2.5	Broadcast receiver . . . . .	41
6.3	Komponenty pro synchronizaci . . . . .	41
6.4	Uživatelský účet . . . . .	42
6.5	Ukládání dat . . . . .	44
6.6	SQLite . . . . .	45
6.7	REST . . . . .	46
6.8	Struktura projektu . . . . .	46
6.9	Manifest + oprávnění . . . . .	46

6.10	9png grafika . . . . .	47
6.11	GPS . . . . .	47
6.12	Zpětná kompatibilita . . . . .	47
6.13	Budoucí rozšiřitelnost . . . . .	47
6.14	Vytváření grafů . . . . .	47
6.15	Chybové reporty . . . . .	47
6.16	Distribuce . . . . .	47
<b>7</b>	<b>Testování</b>	<b>48</b>
7.1	O čem psát... . . . .	48
<b>8</b>	<b>Závěr</b>	<b>49</b>
	<b>Seznam zkratk</b>	<b>50</b>
	<b>Literatura</b>	<b>51</b>
<b>A</b>	<b>Uživatelská dokumentace</b>	<b>52</b>
<b>B</b>	<b>Manifest?</b>	<b>56</b>

# 1 Úvod

## 2 Současný systém

Integrovaný manažerský informační systém (IMIS) je součástí informačního systému Ramses ERP vyvinutého společností CCA Group a.s. Jedná se modulový ERP systém zabývající se oblastmi podnikových financí, kontrolováním nákladů, personalistikou a činnostmi podporující obchod. Společnost tento systém sama využívá pro svoje interní potřeby.

### 2.1 Vybraná funkcionalita

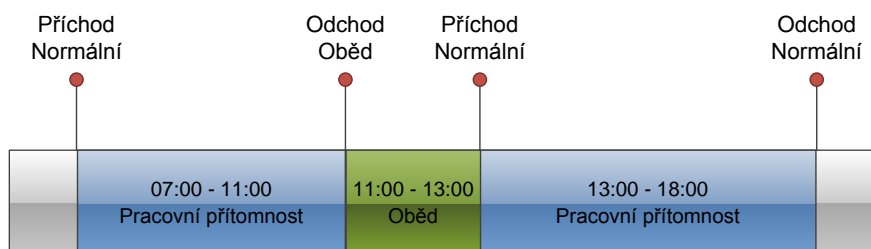
V této práci se zaměřuji na funkcionalitu z oblasti systému věnující se personalistice. Konkrétně se jedná o moduly pro zápis příchodů a odchodů na pracoviště a vykazování provedené práce. Jedná se o činnosti, které zaměstnanec provádí jako každodenní rutinu a zároveň jsou to činnosti s nejširší skupinou uživatelů v rámci podniku.

#### 2.1.1 Evidence docházky

Docházkový systém slouží k evidenci docházky zaměstnanců, která se následně využívá k přípravě podkladů pro zpracování mzdové agendy.

Eviduje se každý příchod i odchod z pracoviště společně s účelem události. Účel události je důležitý, protože délka pracovní přestávky či odchod z pracoviště k lékaři jsou legislativně ošetřené záležitosti.

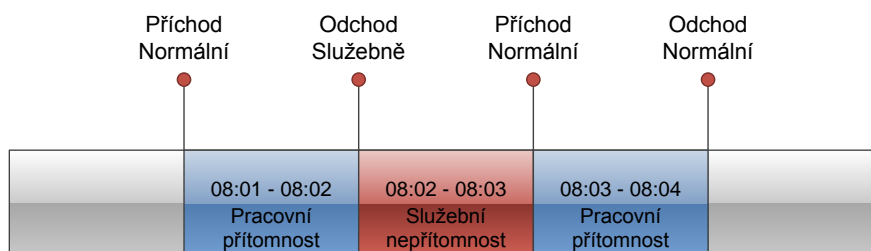
Každá událost se zaznamenává s přesností na minuty. Časová osa s průběhem běžného pracovního dne je zobrazena na obr. 2.1.



Obrázek 2.1: Časová osa běžného pracovního dne



Na obrázku 2.2 je znázorněn pracovní den se služební cestou. Zaměstnanec nejprve zadá příchod do normální pracovní doby a s minutovým odstupem následuje služební odchod. Podobně musí zadat i návrat, nejprve zadá příchod do normální pracovní doby a s minutovým odstupem následuje normální odchod.



Obrázek 2.2: Časová osa pracovního dne se služební cestou

### 2.1.2 Vykazování odvedené práce

Vykazování odvedené práce je jedním ze způsobů pro průběžnou kontrolu aktivit zaměstnanců. Ve firmě probíhá současně více projektů a bez výkazů by bylo velmi těžké sledovat průběžně náklady jednotlivých projektů. Díky evidenci je možné sledovat produktivitu jednotlivých zaměstnanců stejně jako nalézt slabá místa v pracovním procesu.

Výkazy práce jsou propojeny s docházkovým systémem a je možné porovnávat výstupy z těchto systémů.

### 2.1.3 Motivace

Motivací pro vznik této práce bylo vytvořit mobilní aplikaci umožňující provádět každodenní agendu - zadávat příchody, odchody a výkazy práce pro zaměstnance, kteří často cestují a působí mimo sídlo organizace. Také snaha o využití možností mobilního zařízení. Aplikace umožní pohodlné zadávání údajů a přinese možnost mít požadované informace po ruce.

Výsledná mobilní aplikace nemá nahradit vybrané části používaného systému, ale poskytnout efektivnější alternativu ve vybraných činnostech.

## **2.2 Použitá technologie**

Současný systém je postaven na Oracle technologii. Jako uživatelské rozhraní používá Oracle Forms a data jsou ukládány v Oracle databázi.

### **2.2.1 Oracle Forms**

Oracle Forms[4] je softwarový produkt vyvinutý společností Oracle. Slouží k vytváření formulářů, které interagují s Oracle databází. Jako programovací jazyk využívá PL/SQL. Produkt byl původně používán jako terminálové rozhraní pro komunikaci se serverem. Později byl přepracován do architektury klient-server.

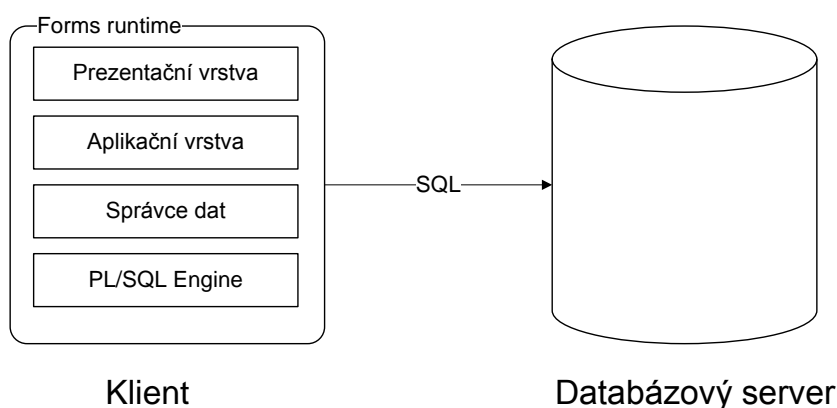
Prostředí běhu zajišťuje defaultní správu transakcí. Díky tomu je Oracle Forms silný nástroj pro efektivní vývoj aplikací, jejichž primárním cílem je přístup k datům uložených v databázi.

### **PL/SQL**

PL/SQL (Procedural Language/Structured Query Language) je procedurální nadstavba jazyka SQL od firmy Oracle založená na programovacím jazyku Ada.

## 2.2.2 Architektura

Oracle Forms používá client-server architekturu. Klient funguje jako tlustý klient, který se kromě zobrazení dat stará o business logiku aplikace. Serverem je myšlen databázový server. Architektura je znázorněna na obr. ??



Obrázek 2.3: Klient-server architektura Oracle Forms aplikace

### Forms prostředí běhu

- **Prezentační vrstva**  
Zobrazuje informace pro uživatele formou grafického uživatelského rozhraní. Kontroluje zadávané vstupy.
- **Aplikační vrstva**  
Stará se o provedení aplikační logiky.
- **Správce dat**  
Stará se o zpracování dat se kterými formulář pracuje. Řídí databázové transakce.
- **PL/SQL Engine**  
Komponenta která zpracovává PL/SQL kód. Stará se o provedení procedurálního (PL) kódu a SQL kód předává ke zpracování databázi.

**Databáze** Databáze obsahuje data a kód, který s těmito daty pracuje (triggery, procedury, funkce).

### 2.2.3 Komponenty formuláře

Z hlediska architektury se Oracle Forms aplikace skládá z těchto celků:

#### Moduly

**Modul formuláře** Modul formuláře je hlavní komponenta aplikace. Poskytuje kód nezbytný pro interakci s úložištěm a uživatelským rozhraním. Data poskytovaná databází jsou reflektovaná v prvcích uživatelského rozhraní jako jsou textová pole, zaškrťovací políčka, přepínače, tlačítka atd. Formulář je logicky organizován do bloků. Existují dva typy bloků:

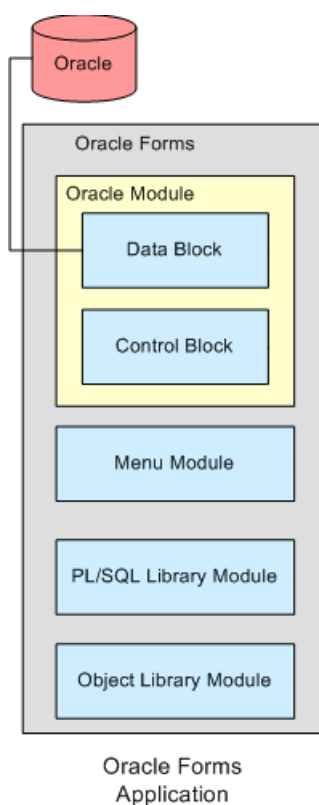
- **Datový blok**  
Datový blok zobrazuje zdrojová data a poskytuje abstrakci pro způsob jakým jsou tato data získávána. Blok může být asociován s databázovou tabulkou, databázovým pohledem, uloženou procedurou, dotazem do databáze nebo transakčním triggerem. Asociace datového bloku a databázových dat standartně umožňuje přístup k těmto datům a jejich modifikaci.  
Datové bloky mohou být navzájem svázány vztahem *rodič - potomek*. Takový vztah představuje relaci 1:N databázových tabulek. Oracle Forms zajišťuje to, že při spojení mezi master a detail bloky se zobrazí pouze ty detail bloky, které jsou vázány na master blok přes cizí klíč.
- **Řídící blok**  
Představuje blok, který nemá vztah k databázové tabulce. Řídící blok může obsahovat jakékoli prvky uživatelského rozhraní. Prvky mohou sloužit k uložení dočasných proměných nebo k zobrazení dat, které nemají přímou vazbu s databází.

**Modul menu** Modul obsahuje hierarchii menu. Každé menu obsahuje zvolitelné položky. Každý formulář obsahuje defaultní menu obsahující příkazy pro základní DML operace s databází CRUD.

**Modul PL/SQL knihovny** Modul obsahuje znovu využitelný kód, který může být využit jinými formuláři, menu či knihovnami. Programové jednotky knihovny mohou být funkce, procedury a balíčky. Programové jednotky jsou

spouštěny na straně klienta. Mohou obsahovat business logiku. Knihovny jsou nezávislé na formuláři, jsou zaváděny dynamicky a mohou být zároveň využívány více formuláři.

**Modul knihovny objektů** Modul obsahuje znovu využitelné objekty. Řeší uskladnění, správu a distribuci těchto objektů, které mohou být využity jinými formuláři, menu či knihovnami. Využívání tohoto modulu přináší přínosy v podobě úspory paměti při běhu aplikace.



Obrázek 2.4: Komponenty Oracle Forms aplikace[3]

## Triggery

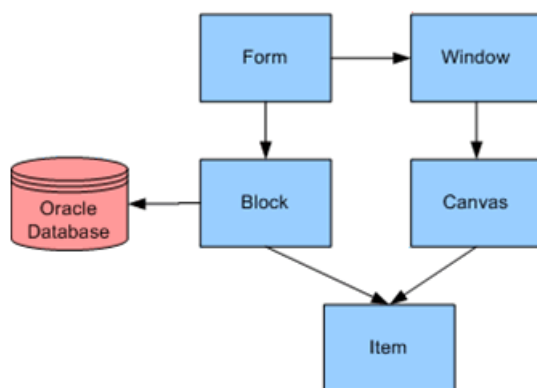
Aplikace v Oracle pracuje s následujícími typy triggerů:

- Block-processing triggers - jsou spouštěny při události na položce patřící tomuto bloku.
- Interface event triggers - jsou spouštěny při události v uživatelském rozhraní formuláře.
- Master-detail triggers - jsou spouštěny při události související se vztahem *rodič - potomek* na daných blocích. Např. při změně položky rodiče příslušný trigger zobrazí příslušné položky v bloku potomka.
- Message-handling triggers - zpracovávají zobrazení chybových či informačních zpráv.
- Navigational triggers - jsou spouštěny při navigaci po položkách formuláře.
- Query-time triggers - jsou spouštěny na úrovni bloku před a po dotazu do databáze.
- Validation triggers - jsou spouštěny při validaci záznamu v položce.
- Transactional triggers - vyvolají se při různých událostech související s interakcí s datovým úložištěm.

Pokud se jedná o datový blok, který je svázan s tabulkou v databázi, prostředí běhu automaticky zajišťuje DML pro tyto bloky. Pokud vývojář požaduje nestandardní akci při těchto úkonech, provede překrytí těchto triggerů s vlastní definovanou akcí.

## 2.2.4 Uživatelské rozhraní

Při pohledu na uživatelské rozhraní se Oracle Forms aplikace skládá z těchto objektů:



Obrázek 2.5: Objekty uživatelského rozhraní Oracle Forms aplikace

Plátno je objekt, na který je nakresleno celé GUI formuláře, tedy všechny viditelné objekty. Okno ohraničuje plochu plátna, která bude zobrazena. Plátno je zobrazeno v okně. Blok sdružuje jednotlivé položky, které se vztahují ke stejnému databázovému objektu.

## Seznam hodnot

Seznam hodnot je prvek uživatelského rozhraní, který uživateli nabízí výběr hodnot. Výběr může být na základě pevně daných dat či dotazem do databáze.

## 2.3 Důležité formuláře

V této jsou popsány formuláře pro vybranou funčnost systému. Konkrétně se jedná o formuláře pro zápis příchodů a odchodů na pracoviště a vykazování

provedené práce.

### 2.3.1 Zápis příchodů a odchodů

Obrázek 2.6: Formulář pro zápis příchodů a odchodů

- Položky bloku pro zápis docházky  
**Kód** - identifikátor zaměstnance.  
**Heslo** - heslo zaměstnance. Pokud heslo nemá tak prázdné pole.  
**Druh** - druh události, příchod či odchod. Kód události.  
**Popis** - doplní se automaticky podle kódu události.  
**Poznámka** - volitelná poznámka.  
**Datum** - datum události.  
**Čas** - čas události.
- Pole *Denní záznamy* - zobrazuje denní docházku formou strukturovaného řetězce.
- Pole s tlačítky - umožňuje akce, které nejsou pro potřeby cílové aplikace relevantní.



### 2.3.2 Výkaz práce

**VX42 - Výkaz práce (01.03.2012)**

**Uživatel**

Zaměst. 1520 MARTIN KADLEC BC Kód KDA Stav : V P 15 S 0 Z 0

Vedoucí 1429 JANSÁ JIŘÍ Střed PRG (počet záz.) 0 15 0 0

Celkem za období z docházky (vyhodn.) 056:40 z docházky 060:54 z výkazu 056:40

**Výkaz práce - hodiny**

Datum	S	Zakázka	Prac.	Hodin	Požad.	Hláš.	Organizace	Popis činnosti
PÁ 01/06/2012	P	R-CCA-FIRMA-	1 1	KDAREZC	02:00			sm
SO 02/06/2012	P	K-VV-N-2012	4 1	KDAREZC	01:00			firemni smernice
NE 03/06/2012								
PO 04/06/2012								
ÚT 05/06/2012								
ST 06/06/2012								
ČT 07/06/2012								

**Výkaz práce - Km**

Datum	S	Zakázka	Auto	Km	Organizace	Popis činnosti
PÁ 08/06/2012	V					
SO 09/06/2012						
NE 10/06/2012						
PO 11/06/2012						
ÚT 12/06/2012						

Celkem z doch. Km Hodin Rozdíl

03:00 A 03:00 00:00 14:00(P)-15:00(P),16:59(P)-17:00(P),

Alt1=PRÍSTUPY Alt2=ZMĚNA OBDOBÍ Alt3=ODMĚNY OSV Alt4=SUMA Alt5=POTVRZENÍ

Alt6=IMP.KNIHY J. Alt7=IMP.VÝKAZU ^P=TISK

Obrázek 2.7: Formulář pro pracovní výkazy

- Položky bloku *Uživatel* zobrazují identifikaci uživatele. Dále jsou zde statistiky stavů, ve kterých se nachází výkaz. V posledním řádku je zobrazen měsíční součet odpracovaného a vykázaného času.
- V bloku *Datum* je zobrazen měsíční kalendář.
- V bloku *Výkaz práce - hodiny* jsou výkazy pro zvolený den:  
**Stav výkazu** - vypsáný, potvrzený, schválený, uzavřený.  
**Zakázka** - identifikace zakázky.  
**Pracoviště** - identifikace pracoviště.  
**Hlášení** - chybové hlášení, které bylo podnětem pro tento úkol.  
**Organizace** - identifikace organizace.  
**Popis činnosti** - zaměstnancův popis odvedené činnosti.
- V bloku *Výkaz práce - Km* jsou výkazy jízd pro zvolený den:  
**Auto** - identifikace vozidla.  
**Km** - počet ujetých km.
- Pole se statistikami zobrazuje denní součet odpracované doby, vykázané doby, ujetých km.
- Pole s tlačítky - jediná relevantní akce je změna zobrazovaného měsíce.

## 3 Analýza

TODO doplnit

### 3.1 Architektura

Při návrhu architektury jsem se rozhodoval mezi třemi variantami: přímé spojení Android aplikace ke vzdálené databázi pomocí JDBC, synchronizaci dat se vzdálenou databází pomocí Oracle Database Mobile Server a nakonec s využitím webové služby, která by sloužila jako rozhraní mezi klientskou aplikací a databázovým serverem.

#### 3.1.1 Přímé připojení k databázi

Přestože přímé připojení k Oracle databázi pomocí JDBC je možné, tuto variantu jsem zamítl. Připojení pomocí JDBC je primárně určeno pro stabilní síťové připojení, které má malou odezvu a nízkou ztrátu paketů. Využití JDBC by přineslo problémy v podobě špatné odezvy aplikace, kvůli znovu navazování spojení a vytváření nových databázových relací, které musely být v důsledku ztráty konektivity ukončeny.

Vzhledem k tomu, že původní Forms aplikace funguje jako tlustý klient, provádí veškerou bussines logiku. Tato logika je zapotřebí ke správné funkci systému. Bylo by tedy nutné přenést tuto logiku na stranu klienta a potřeba komunikace se vzdálenou databází by byla větší než k pouhému přenesení dat.

#### 3.1.2 Oracle Database Mobile Server

Oracle Database Mobile Server 11g[7] je server zajišťující synchronizaci dat mezi Oracle databází a mobilními zařízeními. Klíčovou vlastností tohoto produktu je synchronizační jádro, které je schopné zajistit synchronizaci velkého počtu mobilních zařízení se vzdáleným databázovým systémem. Přestože bylo toto synchronizační jádro navrženo pro stabilní připojení, je schopné zajistit spolehlivou funkci i při nestabilním připojení. V případě, že je spojení

přerušeno synchronizace je pozastavena a po navázání spojení pokračuje v místě přerušení. Dále umožňuje šifrování dat, jak pro přenos tak i pro jejich persistenci.

Tato varianta byla zamítnuta protože řeší pouze synchronizaci dat a neumožňuje zajistit provedení business logiky. Dalším důvodem je skutečnost, že její použití by vyžadovalo zakoupení licence pro tento server.

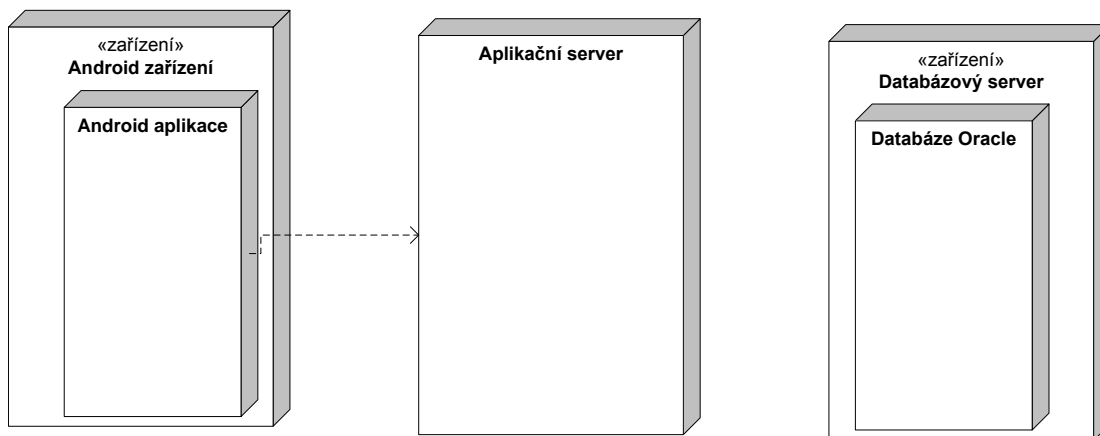
Server je možné spustit na serverech Oracle WebLogic Server a Oracle Glassfish. Mobilní klient, který běží na straně mobilního zařízení zajišťuje správu zařízení nutnou k synchronizaci. Tento klient je dostupný pro platformy Java, Android, Blackberry, Windows a Linux.

### 3.1.3 Webová služba

Jako použitou architekturu jsem zvolil použití webové služby, která bude fungovat jako rozhraní mezi klientskou aplikací a databázovým serverem. Android klient v této architektuře funguje jako tenký klient spravující jen část funkčnosti z původního tlustého klienta. Business logika je umístěna na straně webové služby. Díky tomu že, webová služba bude umístěna v blízkosti firemní databáze, dojde k minimalizaci odezev při zajištění business logiky systému. Mezi klientem a webovou službou se přenášejí pouze data, která jsou opravdu nutná.

Z pohledu rozšiřitelnosti systému o další mobilní platformy se toto řešení jeví rovněž výhodně. Business logika by nebyla implementována ani na klientských aplikacích jiných platforem. Při změně logiky bude potřeba úpravy v kódu pouze na straně webové služby. Cílová architektura je vyobrazena na obrázku 3.1.

### 3.1.4 Zvolená architektura



Obrázek 3.1: Diagram nasazení

#### TOTO Deployment diagram

Android aplikace funguje jako tenký klient, který se připojuje k webové službě. Webová služba používá REST architekturu a přistupuje k samotné databázi.

- Webová služba - Java EE 6, aplikační server GlassFish
- Databáze - Oracle 10g, obsahuje navíc databázové procedury, které se používají v současných formulářích
- Android - obsahuje persistentní úložiště, obsahuje záznamy o docházce, úložiště se bude automaticky synchronizovat ve stavu online s databázovým serverem prostřednictvím webové služby

TODO prepsat srozumitelněji TODO schema komunikace -HHTTP, JDBC

## 3.2 Výběr typu webové služby

### 3.2.1 Representational State Transfer

Representational State Transfer[1] (REST) je architektonický styl pro distribuovaný systém. Architektura využívá těchto vlastností:

**Klient-server** Využívá se architektura klient-server z důvodu rozdělení zodpovědností pro komponenty systému (*Separation of concerns*). Klient je zodpovědný za uživatelské rozhraní, díky tomu je získána větší přenositelnost na jiné platformy. Na straně serveru, který je zodpovědný za datovou vrstvu, se získá výhoda větší škálovatelnosti.

**Bezestavost** Komunikace probíhá bezestavově. Každý požadavek musí obsahovat všechny informace nutné k jeho vyřízení. Pokud je nutné pamatovat si stav, je to zodpovědností klienta. Díky tomu se zvyšuje spolehlivost systému, protože to usnadňuje zotavení se ze stavu částečného selhání. Také se zvyšuje škálovatelnost, protože server si nemusí udržovat informaci o stavu a díky tomu potřebuje méně zdrojů. Rovněž je snazší takový server implementovat.

**Mezipaměť** Data přijímaná jako odpověď ze serveru musí být označena, zda mohou být uložena do mezipaměti klienta či nikoli. Pokud jsou uložena, klient je může využít opakovaně. Díky tomu je snížen počet interakcí.

**Jednotné rozhraní** Zdroje jsou identifikovány pomocí URI, které ale nezávisí na tom jakým způsobem budou data vrácena klientovi. Každý požadavek na server obsahuje informaci jakým způsobem mají být data zpracována (pomocí MIME hlavičky HTTP požadavku).

**Vícevrstevný systém** Klient nemusí získávat data přímo ze serveru na který se obrací. Server může fungovat jako prostředník, který je sám v roli klienta vůči jinému uzlu systému. Rozdělení systému do více vrstev může být využito k zapouzdření zastaralých služeb, k ochraně nových služeb před zastaralými klienty či zjednodušení komponent díky sdílení zřídka využívaných služeb. Jiným přínosem může být zlepšení škálovatelnosti díky rozložení zátěže mezi více uzlů na stejné vrstvě systému.

**Kód na požádání** Server může klientovi poskytovat kód, který je spustitelný na straně klienta a přinést tak další funkcionalitu.

## RESTful web API

Systém který využívá principy REST se označuje jako *RESTful*. RESTful webové API je webové API využívající HTTP a REST principy. Jedná se o kolekci zdrojů s těmito definovanými aspekty:

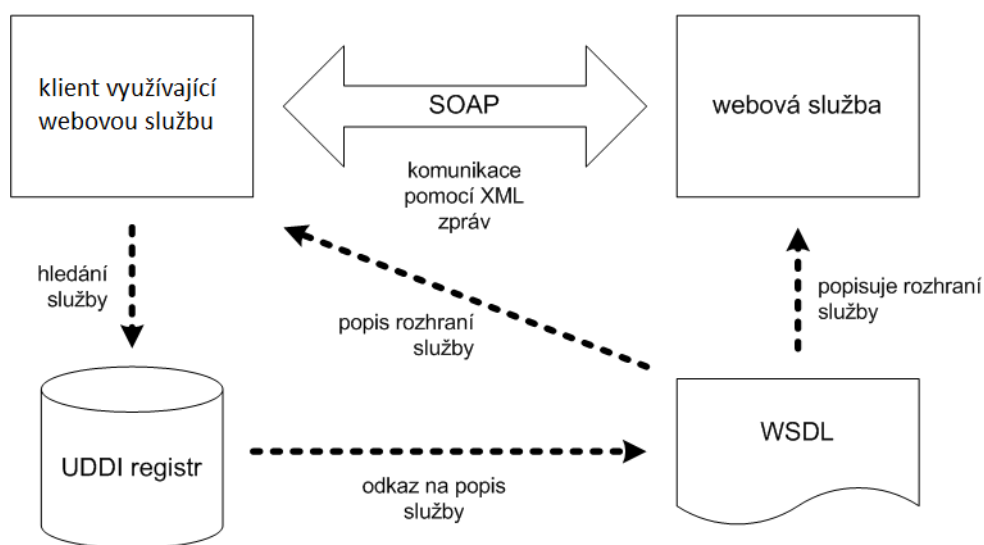
- základní URI pro webové API
- typ internetového média dat poskytovaných API (JSON, XML nebo jiný splňující hypertextové standardy)
- množina operací podporovaná webovým API pomocí HTTP method
  - k získání zdroje se používá GET metody
  - k aktualizaci zdroje se používá PUT metoda
  - k vytvoření zdroje se používá POST metoda
  - ke smazání zdroje se používá DELETE metoda
- API musí být řízeno hypertextově

### 3.2.2 Simple Object Access Protocol

Simple Object Access Protocol (SOAP) je standardizovaný protokol pro výměnu dat mezi webovými službami. Protokol řídí výměnu zpráv mezi poskytovatelem a konzumentem služby. Komunikace probíhá většinou pomocí HTTP protokolu, který je použit z důvodu lepší prostupnosti přes zabezpečující síťové prvky.

Existuje několik různých druhů šablon pro komunikaci na protokolu SOAP. Nejznámější z nich je RPC šablona, kde jeden z účastníků komunikace funguje jako klient a druhý jako server. Server odpovídá na požadavky klienta.

Na obr. 3.2 je znázorněn vztah základních technologií používaných u webové služby využívající SOAP protokol.



Obrázek 3.2: Vztah tří základních technologií (SOAP, WSDL a UDDI) webových služeb[5]

**Webová služba** Poskytovatel služby. Ke službě se přistupuje pomocí *endpoint* URL. *Endpoint* URL definuje umístění služby, kde poskytovatel očekává příchozí požadavky.

**Klient** Konzument služby, který si v UDDI registru vyhledá požadovanou službu. Poté sestaví zprávu podle specifikace a kontaktuje poskytovatele.

**UDDI[6]** (Universal Description, Discovery, and Integration) registr je místo, které poskytuje informace o dostupných službách a jejich poskytovatelích.

**WSDL[8]** (Web Services Description Language) je jazyk, kterým je popisována funkcionality poskytovaná webovou službou. WSDL soubor poskytuje strojově čitelný popis toho, jak může být služba volána, jaké parametry očekává a jaké datové struktury vrací.

which states what service methods are available, which objects are used as return types and method parameters, and what is the structure of such objects.

### 3.2.3 Odůvodnění výběru

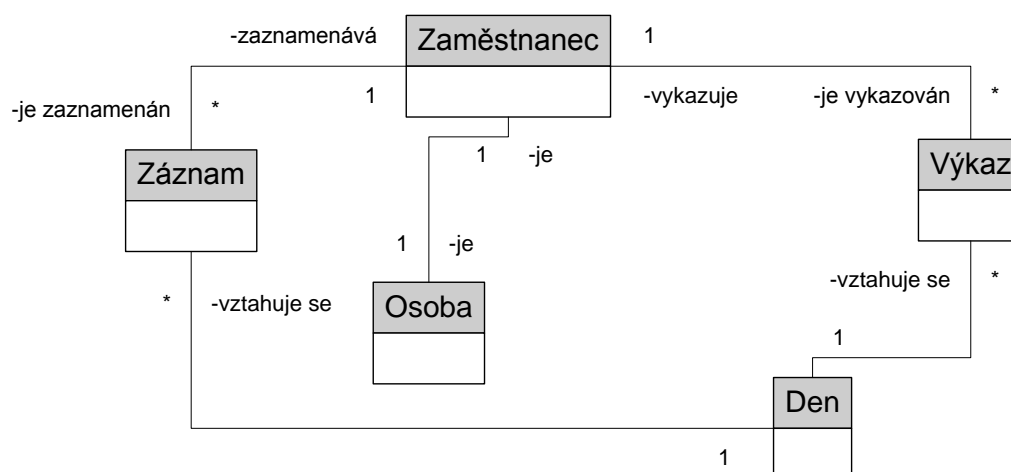
Vzhledem k tomu, že webová služba má sloužit ke správě vzdálené databáze, jeví se REST se svojí CRUD maticí operací jako jasná volba. Při použití REST společně s přenosem dat ve formátu JSON bude objem přenesených dat malý, což je důležité, protože klientskou aplikací má být mobilní zařízení. Webovou službu využívající REST bude ve srovnání se SOAP snadnější implementovat. Jedinou nevýhodou bude bezstavovost komunikace, která přináší nutnost provádění procesu autentizace pro každý požadavek.

## 3.3 Datová vrstva

V této kapitole analyzuji datovou vrstvu systému.

### 3.3.1 Datový model

V následujícím diagramu (obr. 3.3) zobrazuji entity nacházející se v systému, které jsou relevantní pro zkoumanou část systému. Další tabulky a číselníky jsou pro jednoduchost vypuštěny.



Obrázek 3.3: Diagram relevantních entit nacházejících se v systému



Jednotlivé entity a jejich popis:

- **Záznam**  
Příchod nebo odchod na pracoviště. Obsahuje informaci o čase, datu, účelu a případně zaměstnancův krátký popis události.
- **Výkaz**  
Pracovní výkaz. Obsahuje informaci o čase, datu, osobě zadavatele, osobě řešitele, vykazované době, vazbu na zakázku či chybové hlášení, stav výkazu, zaměstnancův popis činnosti.
- **Zaměstnanec**  
Zaměstnanec a jeho příslušnost k pracovnímu oddělení, funkce, vedoucí pracovník a typ úvazku.
- **Osoba**  
Podrobnější informace o osobě zaměstnance, jméno, pracovní zkratka.
- **Den**  
Den ke kterému se váže výkaz či záznam. Slouží k rozlišení pracovních dnů a svátků.

### 3.3.2 Práce s datumem a časem

Při návrhu datového modelu jsem řešil problém pomocí jakého datového typu vyjadřovat údaj o čase či datu. V Oracle databázi je použit datový typ Date. SQLite databáze nabízí tři způsoby jako ukládat informaci o čase:

- **TEXT** podle ISO8601 normy ve formátu "YYYY-MM-DD HH:MM:SS.SSS".
- **REAL** podle Juliánského kalendáře, počet dní od poledne 24. Listopadu roku 4714 před Kristem (Greenwichského času).
- **INTEGER** jako Unix Time, počet sekund 1970-01-01 00:00:00 UTC.

Pro uložení v SQLite databázi jsem zvolil typ INTEGER. V aplikaci (Android klient, webová služba) jsem se rozhodl reprezentovat časový údaj pomocí primitivního typu long. Měl jsem k tomu řadu dobrých důvodů:

- odpadá starost s formátem datumu při serializaci a deserializaci JSON řetězce
- snadné porovnávání hodnot pomocí relačních operátorů
- sníží se počet konverzí v aplikaci (např. pro výpočet pozice pro vykreslení komponenty v UI)

Také jsem se ujistil, že rozsah typu long je pro potřeby aplikace dostačující. Srovnání použitých datových typů je znázorněno v tabulce 3.1.

Datový typ	Minimální hodnota	Maximální hodnota	Přesnost
Oracle Date	January 1, 4712 BCE	December 31, 4712 CE	sekundy
SQLite INTEGER			sekundy
Java long	2.12.292269055 BC	17.8.292278994 AD	milisekundy

Tabulka 3.1: Datové typy reprezentující časový údaj

### 3.3.3 Výhrady k datové vrstvě

#### Tabulka bez primárního klíče

Tabulka pro evidenci docházkových událostí neobsahuje primární klíč. To je velký problém, protože klient měnící data potřebuje jednoznačně identifikovat záznam. Absenci primárního klíče lze vyřešit indetifikací záznamu pomocí pseudosloupce ROWID. Hodnota ROWID se skládá z těchto hodnot:

- číslo datového objektu
- číslo datového bloku v souboru, kde se záznam nachází
- pozice řádku v datovém bloku
- číslo datového souboru, kde se záznam nachází. Číslo souboru je relativní vzhledem k *tablespace*

Pomocí ROWID lze tedy jednoznačně identifikovat záznam. Bohužel kvůli k tomu, že jeho hodnota je relativní vzhledem k *tablespace*, tak jeho použití jako unikátního identifikátoru může selhat v těchto situacích:

- migrace
- smazání

Vhodným řešením by byla změna databázového schématu tabulky pro záznam docházkových událostí a to přidáním primárního klíče.

### **Schéma pro docházku**

+ snazší zamezit chybnému vstupu

**Datum a čas odděleně**

**Čas jako reálné číslo**

## **3.4 Business logika**

existuje nějaká možnost převodu formsů do javy - oracle adf - co to je, co to resi, proč to neresi můj problém  
přijde to do webové služby - důvody

### **3.4.1 Triggery**

jen ty, jejichž funkčnost bude muset být implementována.

- On-Delete, On-Insert, On-Update, Pre-Delete, Pre-Insert, Pre-Update
- When-Validate-Item

### **3.4.2 Databázové balíčky a uložené procedury**

ukázky volání z javy

### 3.4.3 Forms knihovny

## 3.5 Synchronizace dat

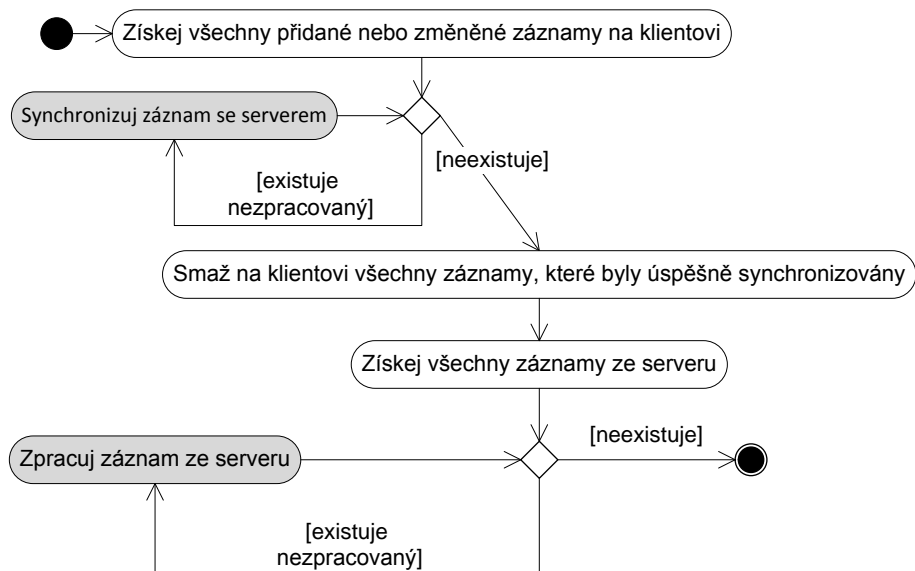
V současném systému uživatel zadává data prostřednictvím příslušného formuláře. Změny jsou aplikovány bezprostředně po uložení během databázové transakce. Mobilní klient přináší nový způsob použití - data lze zadat i v režimu offline, kdy mobilní klient není v dosahu webové služby. Tyto data jsou uložena persistentně na straně klienta a jsou synchronizována až ve chvíli kdy je možná komunikace s webovou službou.

Synchronizace se týká pouze dat pro docházku uživatele. Ostatní data jsou prostřednictvím mobilního klienta pouze zobrazována. Je třeba počítat s tím, že záznamy přidané na straně klienta v režimu offline nemusí být přijaty při synchronizaci z důvodu porušení business pravidel a uživatel by měl být o této skutečnosti vhodně informován.

### 3.5.1 Obousměrná synchronizace

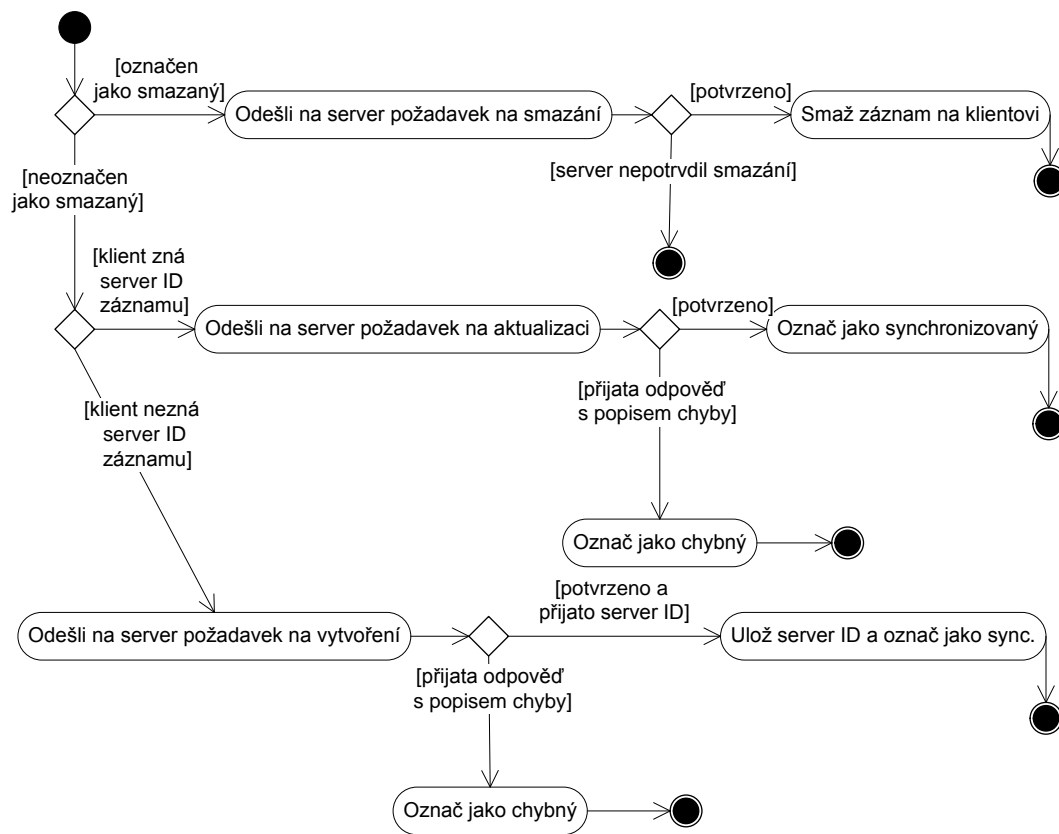
Při obousměrné synchronizaci se odesílají data ze strany klienta na server tak i opačným směrem ze serveru na klienta. Klienta lze navrhnout tak, aby si uchovával informaci o změnách na svojí straně. Při analýze databázového schématu pro docházku v současném systému jsem zjistil, že databáze neuchovává informaci o změnách na svojí straně. Beze změny této skutečnosti není možné sledovat změny na straně databáze. Výsledkem je poněkud neefektivní způsob synchronizace, kdy klient odesílá na server pouze změny, zatímco ze serveru stahuje všechna data pro daného uživatele a období.

Celkový průběh synchronizace je znázorněn v diagramu 3.4. Klient nejprve odešle všechny svoje změny na server. Poté smaže všechny úspěšně odeslaná data. Bez smazání by nebylo možné zjistit, že na serveru došlo ke změně či smazání dat jiným klientem. Poté už zbývá pouze stažení aktuálních dat ze serveru. Data která na klientovi nebyla smazána z důvodu neúspěšného odeslání na server, zůstávají do té doby, než uživatel tyto data upraví tak aby vyhovovali bussines pravidlům. Dalším důvodem pro neúspěšnou synchronizaci může být přerušení spojení. Data zůstávají na klientovi, až do doby úspěšného pokusu o synchronizaci.



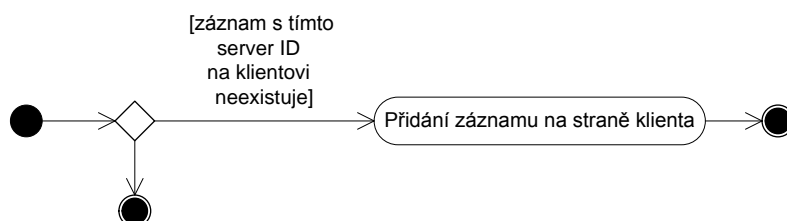
Obrázek 3.4: Diagram aktivit pro průběh synchronizace

Diagram 3.5 podrobněji rozepisuje průběh odeslání požadavku na server. Pokud klient nemá server ID záznamu, znamená to, že záznam byl vytvořen na straně klienta a odesílá se požadavek na vytvoření. Pokud klient zná server ID může požadovat smazání nebo aktualizaci záznamu.



Obrázek 3.5: Diagram aktivit pro odeslání požadavku na server

Diagram 3.6 podrobněji rozepisuje průběh přijetí záznamu ze serveru.



Obrázek 3.6: Diagram aktivit pro přijetí záznamu ze serveru

### 3.5.2 Obousměrná synchronizace s úpravou databáze

Jiná varianta řešení problému synchronizace dat, která se snaží eliminovat nedostatky předchozí varianty, by vyžadovala změny v databázovém schématu současného systému. U každého záznamu by byla přidána informace o poslední změně záznamu s vhodnou časovou přesností. Pokud by došlo k požadavku na smazání záznamu, nebyl by záznam skutečně smazán, ale pouze nastaven příznak smazaného záznamu. Při použití tohoto řešení by bylo možné synchronizovat oběma směry pouze změny ze strany klienta i serveru.

Klient který iniciuje synchronizaci nejprve odešle na server požadavek ke kterému připojí údaj o času provedení poslední synchronizace. Server odesílá ke klientovi pouze ty data, která se změnila po tomto termínu. Poté klient odesílá svoje změny na server.

### 3.5.3 Řešení kolizí

Kolize teoreticky nastane vždy, když se v době od poslední synchronizace změnila stejná data jak na serveru, tak v zařízení. Vzhledem k tomu, že server neukládá informaci o čase poslední synchronizace a není tedy možné zjistit že vůbec došlo ke změně dat, tak mobilní klient vždy přepíše záznam na serveru.

### **3.5.4 Srovnání**

V obou případech řešení je iniciátorem synchronizace klient. Druhá varianta by oproti první přinesla úsporu množství přenesených dat. Vzhledem k tomu, že druhá varianta by vyžadovala změnu v databázovém schématu současného systému, zvolil jsem první variantu i přesto, že z hlediska efektivity synchronizace je to horší řešení.

## **3.6 Uživatelské rozhraní**

zarizeni zna identitu uzivatele, nezadava zbytecnosti



## 4 Zabezpečení

V následující kapitole se zabývám zabezpečením aplikace. Popisuji několik možných variant z hlediska ověřování identity uživatele. Na závěr vysvětluji výběr zvoleného řešení.

### 4.1 Autentizace a autorizace

Při analýze současného systému jsem zjistil, že informace o docházce a výkazech zaměstnanců jsou dostupné všem ostatním uživatelům (údaje týkající se nadřízených pracovníků jsou dostupné i podřízením). Dalším zajímavostí je, že heslo používané k zadání docházky je pro uživatele nepovinné (má ho jen ten uživatel, který si ho nastavil).

#### Autentizace

Autentizace je proces ověření proklamované identity subjektu. Uživatel se identifikuje pomocí svého uživatelského jména a hesla.

#### Autorizace

Autorizace je proces získávání souhlasu s provedením nějaké operace. Uživatel musí zadávat svoj přístupové údaje při zadání každého záznamu docházky.

#### Riziko poškození systému

Webová služba umožňuje čtení a úpravu docházkových dat a dále čtení dat o výkazech práce a zaměstnancích. Dále používá některé databázové objekty jako jsou procedury a funkce, které pracují s těmito daty. Je vhodné aby aplikace měla přístup pouze k těm databázovým objektům, které jsou relevantní pro navrženou funkčnost aplikace. To je vhodné pro maximální zabezpečení

okolního systému a minimalizaci případných rizik při zneužití či chybě v aplikaci. Toto je zodpovědností databázového administrátora organizace a v této práci se touto problematikou dále nezabývám.

## HTTP Basic autentizace

Klient posílá autentizační hlavičku jako součást HTTP požadavku na server. Jméno a heslo je zasláno jako jeden textový řetězec oddělený dvojtečkou. Výsledný řetězec je poté zakódován metodou Base64. Uživatelské jméno a heslo se tedy posílá v zakódované podobě. Nejedná se ale o kryptografické zabezpečení přihlašovacích údajů. Použití této metody předpokládá použití zabezpečeného komunikačního kanálu mezi klientem a serverem.

## 4.2 VPN pro vzdálený přístup

Virtuální privátní síť je prostředek pro propojení počítačů v prostředí nedůvěryhodné sítě. Díky VPN spojení mohou počítače komunikovat tak, jako by byly součástí důvěryhodné sítě.

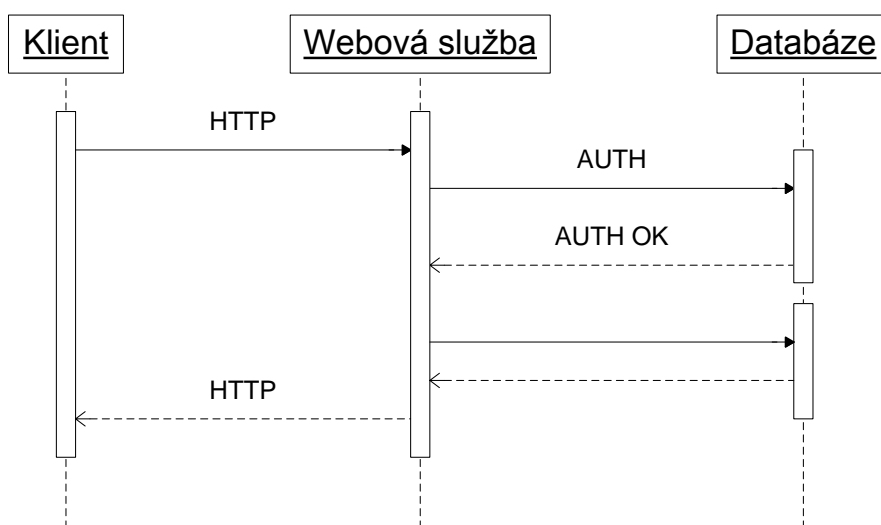
### Vlastnosti připojení VPN

- Zapouzdření  
Při použití technologie VPN jsou data zapouzdřena pomocí hlavičky obsahující směrovací informace, které umožňují průchod dat přes tranzitní síť.
- Ověřování  
Klient a VPN server se vzájemně ověřují na úrovni počítače. Android má integrovanou podporu pro VPN využívající protokoly PPTP, L2TP a IPSec.
- Šifrování dat  
Pro utajení dat během jejich přenosu sdílenou nebo veřejnou tranzitní sítí jsou data na straně odesílatele zašifrována a na straně příjemce dešifrována. Šifrování a dešifrování je založeno na tom, že odesílatel i příjemce používají společný šifrovací klíč.

Webová služba je tedy umístěna na serveru uvnitř firemní sítě. Pokud klient chce komunikovat s webovou službou musí tak činit prostřednictvím sítě VPN.

### 4.3 Autentizace proti databázi

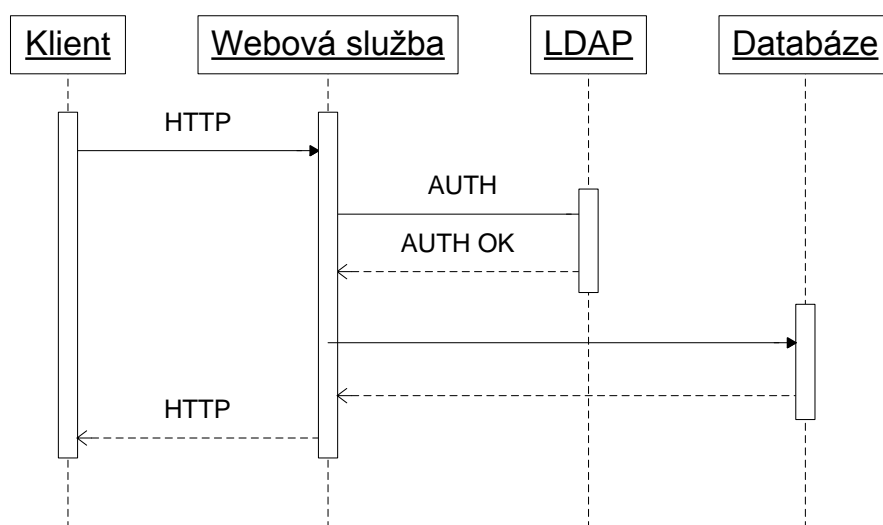
Klient i server jsou součástí jedné VPN sítě, která zajišťuje důvěryhodný komunikační kanál. Klient posílá na server HTTP požadavek jehož součástí je autentizační hlavička nesoucí uživatelské jméno a heslo. Webová služba ověří jméno a heslo pomocí databázové procedury. Jméno a heslo uživatele je uloženo v databázi. Implementované řešení je znázorněno na obr. 4.1 který zobrazuje sekvenční UML diagram v případě úspěšné autentizace.



Obrázek 4.1: Diagram aktivit při úspěšné autentizaci

## 4.4 Autentizace proti LDAP

Alternativním řešením by bylo ověřování uživatelů pomocí LDAP adresáře. Organizace již LDAP používá v některých dalších firemních systémech. Toto řešení se liší v tom, že dotaz na ověření identity uživatele probíhá k LDAP adresáři a nikoli k databázi. Implementované řešení je znázorněno na obr. 4.2, který zobrazuje sekvenční UML diagram v případě úspěšné autentizace.



Obrázek 4.2: Diagram aktivit při úspěšné autentizaci

### 4.4.1 LDAP

Directory Access Protocol (LDAP) je internetový protokol definující přístup k distribuované adresářové službě. Podle tohoto protokolu jsou jednotlivé položky na serveru ukládány formou záznamů a uspořádány do stromové struktury. Protokol LDAP je byl navržen v souladu se sadou standartů X.500 vyvinutých pro adresářové služby v počítačových sítích. Protokol LDAP je jejich odlehčenou verzí.

Aplikace funguje na bázi klient-server. Klient se při komunikaci se serverem autentizuje. Prostřednictvím klienta lze přidávat, modifikovat a mazat

záznamy na serveru.

## Schéma

Úkolem informačního modelu LDAP je definovat datové typy a informace, které lze v adresářovém serveru ukládat. Data jsou uchovávána ve stromové struktuře pomocí záznamů. Záznam představuje souhrn atributů (dvojice jméno - hodnota). Atributy nesou informaci o stavu daného záznamu. Záznamy, uložené v adresáři, musí odpovídat přípustnému schématu. Schéma představuje soubor povolených objektových tříd a k nim náležících atributů. Ukázka schématu definující strukturu záznamu zaměstnance:

```
objectclass ( 1.1.2.2.2 NAME 'zamestnanec'
              DESC 'zamestnanec firmy'
              SUP osoba
              MUST ( jmeno $ identifikacniCislo )
              MAY zkratkaZamestnance )
```

Objekt popisující zaměstnance dědí od objektu osoba, vyžaduje povinný atribut 'jmeno' a 'identifikacniCislo' a nepovinný atribut 'zkratkaZamestnance'.

## Funkční model

Funkční model umožňuje pomocí základních operací manipulovat a přistupovat k záznamům v adresáři a měnit či zjišťovat tak jejich stav.

- Autentizační operace: Slouží k přihlášení a odhlášení uživatele pro komunikaci s adresářovým serverem. Jsou jimi míněny především operace bind a unbind. Na úspěšném provedení operace bind závisí výsledky aktualizacích a dotazovacích operací nad adresářem.
- Aktualizační a dotazovací operace: Každý adresářový server podporuje základní operace s daty, jako je vyhledávání, přidávání, mazání, porovnávání a modifikace záznamů. Tyto operace bývají často spjaté s nastavením bezpečnostního modelu.

## LDAP URL

Umístění zdroje je v LDAP specifikováno pomocí URL, které má následující tvar:

`ldap://host:port/DN?attributes?scope?filter?extensions`

- host - doména nebo IP adresa
- port - síťový port (defaultně 389)
- DN - význačné jméno použité jako základ pro vyhledávání
- attributes - seznam atributů
- scope - specifikuje vyhledávací rozsah
- filter - filtrovací kritérium
- extensions - rozšíření

## 4.5 Shrnutí

Hlavním prvkem zabezpečení je VPN přístup. Uživatel bez přístupu do firemní VPN nemůže komunikovat s webovou službou. Od uživatele se očekává, že si nakonfiguruje VPN připojení v nastavení Android mobilního zařízení.

Použil jsem první řešení protože je shoduje se způsobem ověřování v současném systému. Stávající řešení pomocí Oracle Forms aplikace používá rovněž ověření uživatele dotazem k databázi tzn. ověření se děje na aplikační vrstvě.

Je nutné dívat se na pravidlo dobrovolného hesla jako na firemní pravidlo, které může být kdykoli zrušeno. V případě zrušení tohoto pravidla by se z největší pravděpodobností uplatnilo ověřování proti LDAP adresáři.

## 5 Webová služba

JAX-RS citace

### 5.1 REST

Při návrhu REST služby jsem nejprve identifikoval všechny zdroje, které webová služba zpřístupňuje. Jedná se o údaje o docházce zaměstnanců, výkazech práce a zaměstnancích samotných. Posledním zdrojem je možnost testovat komunikaci s webovou službou.

Zdroj pro docházku zaměstnanců tzn. jejich příchody a odchody (označované jako události) a poskytované služby zobrazuje tabulka 5.1. Služba umožňuje operace CRUD na datovém zdroji zaměstnanců a dále zjištění součtu doby v zaměstnání.

GET	events/{icp}?from={from}&to={to}
	Získá všechny události zaměstnance za dané období Parametry: <ul style="list-style-type: none"> <li>• icp - identifikátor zaměstnance</li> <li>• from - datum začátku období</li> <li>• to - datum konce období</li> </ul>
DELETE	events/{rowid}
	Smaže danou událost Parametry: <ul style="list-style-type: none"> <li>• rowid - identifikátor události</li> </ul>
POST	events
	Vytvoří událost, používá se bez parametrů protože identifikátor pro událost vytváří server
PUT	events/{rowid}
	Aktualizuje danou událost Parametry: <ul style="list-style-type: none"> <li>• rowid - identifikátor události</li> </ul>
GET	events/sum/{icp}?from={from}&to={to}
	Získá součet přítomnosti zaměstnance za dané období Parametry: <ul style="list-style-type: none"> <li>• icp - identifikátor zaměstnance</li> <li>• from - datum začátku období</li> <li>• to - datum konce období</li> </ul>

Tabulka 5.1: Služby pro události docházky

Zdroj pro údaje o zaměstnancích



GET	employees/{icp}
	Získá údaje o zaměstnanci identifikováno pomocí parametru Parametry: <ul style="list-style-type: none"> <li>• icp - identifikátor zaměstnance</li> </ul>
GET	employees/all/{icp}
	Získá seznam všech zaměstnanců, kteří jsou aktuálně v zaměstnaneckém poměru, obsahuje informaci zda jsou tito zaměstnanci podřízeni, vzhledem k zaměstnanci identifikováno pomocí parametru Parametry: <ul style="list-style-type: none"> <li>• icp - identifikátor zaměstnance</li> </ul>
GET	employees/lastevents
	Získá poslední událost v docházce všech zaměstnanců, kteří jsou aktuálně v zaměstnaneckém poměru
GET	employees/lastevents/{icp}
	Získá poslední událost v docházce zaměstnance identifikováno pomocí parametru

Tabulka 5.2: Služby pro zaměstnance

Zdroj výkazy práce

GET	records/{kodpra}?from={from}&to={to}
	Získá všechny výkazy práce zaměstnance za dané období Parametry: <ul style="list-style-type: none"><li>• kodpra - identifikátor zaměstnance (zkratka)</li><li>• from - datum začátku období</li><li>• to - datum konce období</li></ul>
GET	records/sum/{icp}?from={from}&to={to}
	Získá součet vykázaného času zaměstnance za dané období Parametry: <ul style="list-style-type: none"><li>• icp - identifikátor zaměstnance</li><li>• from - datum začátku období</li><li>• to - datum konce období</li></ul>

Tabulka 5.3: Služby pro výkazy práce

## 5.2 Filtry

## 5.3 web.xml

## 6 Android aplikace

### 6.1 Funkcionalita

[TODO ještě upřesnit+UML use case?] Na základě analýzy současného systému a potřeb zaměstnanců byla vybrána k implementaci následující funkčnost:

#### Docházka

- Přehledné zobrazení událostí docházky daného zaměstnance
- Uživatel má možnost přidávat, ediovat a mazat svoje události
- Aplikace zajišťuje automatickou synchronizaci těchto údajů s firemní databází
- Zobrazení poměru typů docházkových událostí za dané období

#### Aktuální přítomnost na pracovišti

- Zobrazení seznamu zaměstnanců aktuálně přítomných na pracovišti
- Uživatel má možnost spravovat seznam svých "oblíbených" zaměstnanců a tento seznam zobrazovat přednostně

#### Výkazy práce

- Zobrazení poměru typů zakázek za dané období
- Zobrazení vývoje vývoje daného typu zakázky v daném období
- Možnost zobrazení těchto údajů i za jiné zaměstnance

### 6.1.1 Nastavení a konfigurovatelnost

Aplikace si musí pamatovat údaje nutné pro snadnou obsluhu tzn. uživatelské jméno a heslo, adresu umístění webové služby a tyto údaje jsou konfigurovatelné.

Dále aplikace umožní uživateli konfigurovat vzhled některých komponent, jako je barva typu události v docházce a typu záznamu ve výkazech.

### 6.1.2 Uživatelská přívětivost

Uživatelské rozhraní aplikace klade důraz na přehlednost, ergonomii a časově efektivní obsluhu.

### 6.1.3 Návrhy na vylepšení

[TODO pozn. původní ambice byla možnost zadávat i výkazy práce prostřednictvím mobilního klienta, vzhledem k tomu, že by bylo nutné provést úpravy v současném systému se od tohoto upustilo, kniha jízd]

## 6.2 Komponenty Android aplikace

### 6.2.1 Aktivita

Aktivita představuje jednu obrazovku s uživatelským rozhraním. Každá aktivita umožňuje nějakou funkčnost. Příkladem aktivit použitých v implementované aplikaci může být zobrazení denní docházky uživatele, kalendář umožňující výběr data, přidání a editace docházkové události nebo uživatelské nastavení.

#### Životní cyklus aktivity

Aktivita se může nacházet ve třech stavech:

- **Běžící**  
Aktivita je na popředí a umožňuje interakci s uživatelem.
- **Pozastavený**  
Na popředí se dostala jiná aktivita. Pozastavená aktivita je stále viditelná, ale je překrytá novou aktivitou. Pozastavená aktivita nemůže provádět interakci s uživatelem. Objekt aktivity zůstává v paměti, je zachována veškerá stavová informace. Při nedostatku systémové paměti může být aktivita zničena.
- **Zastavený**  
Aktivita není viditelná. Objekt aktivity zůstává v paměti, je zachována veškerá stavová informace. Při nedostatku systémové paměti může být aktivita zničena.

## Metody životního cyklu aktivity

Při přechodech aktivity mezi výše uvedenými stavy dochází k volání callback metod. Tyto metody jsou volány systémem. Každá z metod má defaultní chování. Programátor může libovolnou z metod překrýt v případě, že požaduje jinou funkcionalitu, ale musí mít na paměti, že většina z nich vyžaduje volání metody rodiče.

- **onCreate()**  
Metoda je volána v okamžiku, kdy je aktivita poprvé vytvořena. Zde se připraví prvky uživatelského rozhraní a nastaví se zdroje dat. Pokud existuje uložený stav aktivity z dřívější doby, může být využit.
- **onRestart()**  
Volána pokud se aktivita před tím nacházela ve stavu *Zastavený* a má se dostat do stavu *Běžící*.
- **onStart()**  
Volána před tím, než se aktivita stane viditelnou.
- **onResume()**  
Volána před tím, než je uživateli umožněna interakce s aktivitou. Aktivita se dostává na vrchol zásobníku aktivit aplikace.
- **onPause()**  
Volána ve chvíli kdy se má jiná aktivita dostat na popředí. V této

metodě by měla být uložena veškerá rozpracovaná data a ukončeny úkoly, které si zabírají výpočetní výkon. Nová aktivita není spuštěna, dokud tato metoda není dokončena.

- **onStop()**

Po volání metody není aktivita nadále viditelná. K volání dochází v situaci, kdy je aktivita ukončována nebo se jiná aktivita dostala do popředí.

- **onDestroy()**

Volání před tím než je aktivita ukončena. K volání dochází v případě, že aktivita končí svoji činnost nebo v situaci, kdy se systém dožaduje systémových prostředků.

## Ukládání stavu aktivity

Ukládat stav aktivity je vhodné protože systém může zničit aktivitu z důvodu nedostatku paměti. Další důvodem je situace, kdy uživatel změní orientaci obrazovky. V takové situaci nezůstává v paměti objekt aktivity, ale aktivita musí být znovu vytvořena. Uživatel samozřejmě očekává, že aktivitu nalezne ve stejném stavu v jakém ji opustil. K těmto účelům jsou určeny dvě callback metody:

- **onSaveInstanceState(Bundle)**

Metoda je volána před tím než je aktivita zničena (před provedením *onStop()*). Do objektu *Bundle* by měla být uložena všechna data, která jsou nutná pro obnovu předchozího stavu aktivity. Data jsou ukládána ve formátu klíč-hodnota. Metoda není volána v situaci, kdy uživatel explicitně ukončil aktivitu. Na začátku metody je nutné zavolat metodu rodiče, aby mohl být uložen stav prvků UI pomocí defaultní implementace.

- **onRestoreInstanceState(Bundle)**

Metoda je volána (po provedení *onStart()*) v případě, že aktivita byla znovuobnovena z předchozího uloženého stavu. Na začátku metody je nutné zavolat metodu rodiče, aby mohl být obnoven stav prvků UI pomocí defaultní implementace.

### **6.2.2 Fragment**

### **6.2.3 Služba**

+ restart zařízení?

### **6.2.4 Content provider**

### **6.2.5 Broadcast receiver**

1. komponenty pro sync a auth, provazani s android ucetm
2. CursorLoader
3. Async task
4. nestandartni UI
5. modifikace adapterview
6. cutom UI - viewgroup
7. widgety
8. alarmanager
9. handler - hlavni vlakno
10. activity - dedicnost

+ nejaka ukazka konkretniho pouziti

## **6.3 Komponenty pro synchronizaci**

1. sync architektura - komponenty

## 6.4 Uživatelský účet

Android poskytuje správu účtů pro online služby. Uživatel zadá svoje přihlašovací údaje při vytvoření účtu a dává tak aplikaci svolení k využívání tohoto účtu.

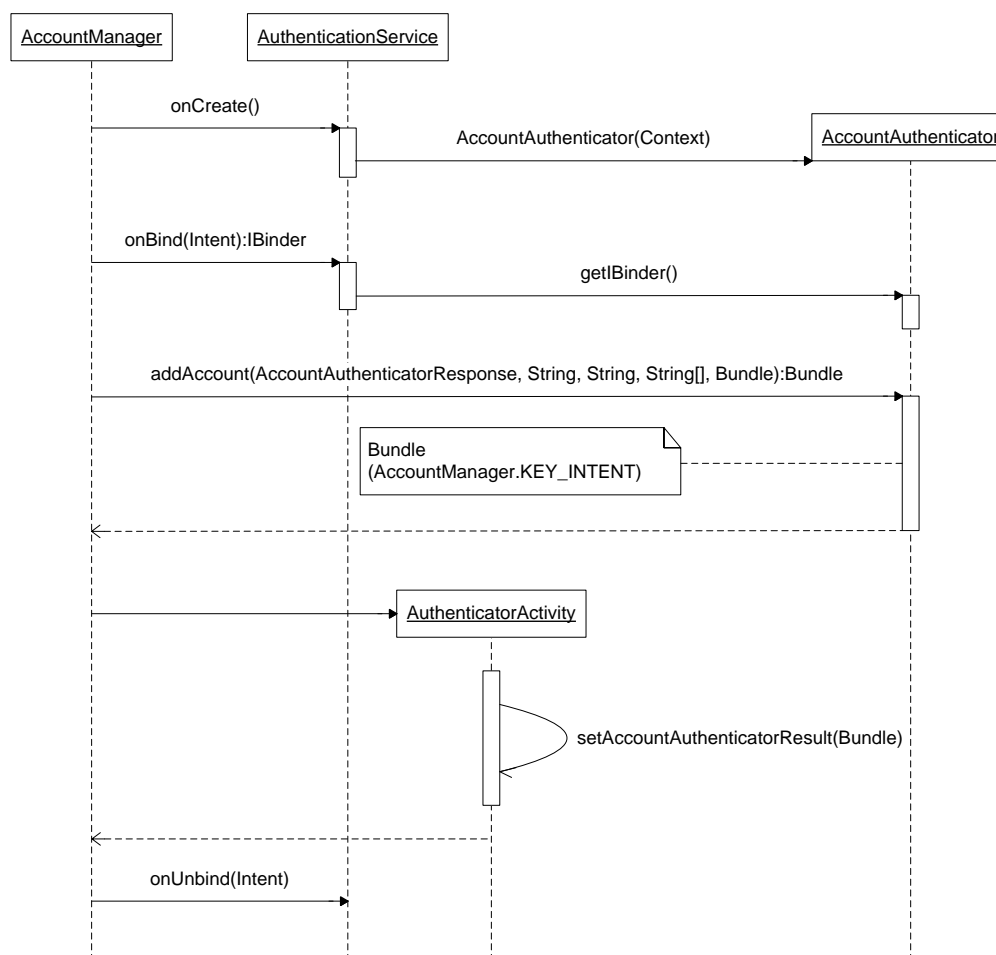
Různé online služby mohou využívat odlišný způsob pro autentizaci. Android manažer účtů využívá komponentu *authenticator* [2], která je obvykle poskytována třetí stranou - poskytovatelem dané služby. Příkladem služby, která poskytuje vlastní *authenticator* je např. Google, Facebook a Microsoft Exchange.

Nejdůležitější třída pro práci s účty je *AccountManager*. Zde je výčet nejdůležitějších metod:

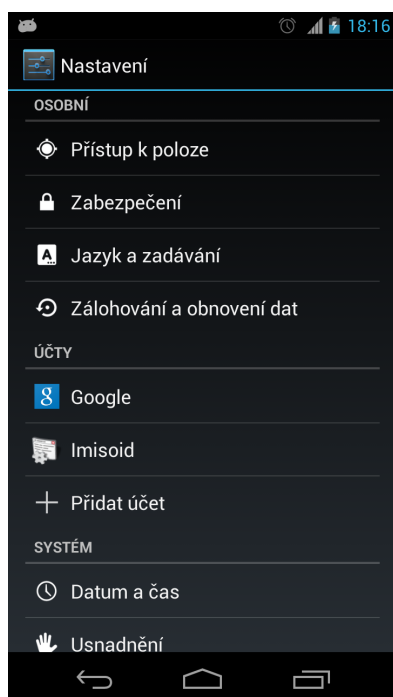
- **addAccountExplicitly(Account account, String password, Bundle userdata)**  
Vytvoří nový účet.
- **Account[]: getAccountsByType(String type)**  
Vrátí seznam všech účtů daného typu.
- **setAuthToken(Account account, String authTokenType, String authToken)**  
Přidá autentizační token do cache paměti pro daný účet.
- **String:getPassword(Account account)/setPassword(Account account, String password)**  
Získání uloženého hesla pro účet./Nastavení hesla.

Na diagramu 6.1 je zobrazen průběh vytvoření účtu, který je použit v implementované aplikaci. Jakmile uživatel v sekci *Nastavení->Účty* (obr. 6.2) zvolí přidání nového účtu a následně zvolí typ účtu (obr. 6.3), manažer účtů spustí službu *AuthenticationService*. Služba vytvoří instanci třídy *AccountAuthenticator* - *authenticator* komponenty. *AccountManager* následně volá metodu *addAccount()*, která zkontroluje zda již účet daného typu existuje. Pokud ne, vrátí objekt *Bundle* obsahující *Intent* s klíčem *AccountManager.KEY\_INTENT* označující, že bude nutná interakce s uživatelem. Následně je spuštěna aktivita *AuthenticatorActivity* vyzývající uživatele, aby zadal svoje přihlašovací údaje. Po potvrzení je vytvořen účet (obr. 6.2).

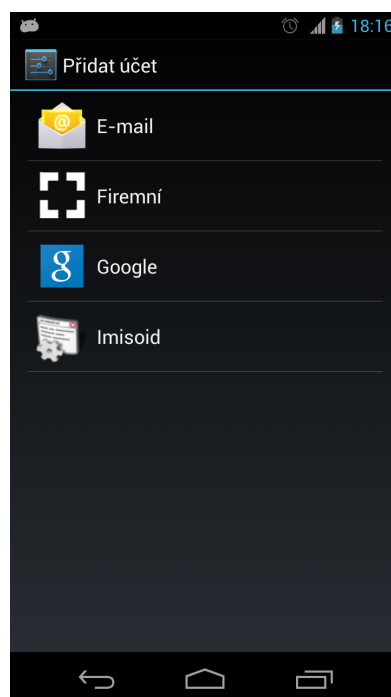




Obrázek 6.1: Sekvenční diagram zobrazující průběh vytvoření účtu



Obrázek 6.2: Nastavení



Obrázek 6.3: Přidání účtu

## 6.5 Ukládání dat

Android umožňuje několik způsobů jak persistentně ukládat data aplikace. Programátor by měl vzít v úvahu, zda data mají být soukromá či dostupná i ostatním aplikacím a také velikost těchto dat.

### Sdílené preference

Ukládá primitivní datové typy ve tvaru klíč-hodnota. Slouží k uložení nastavení specifických pro aplikaci. K těmto se přistupuje pomocí *SharedPreferences* rozhraní. Data jsou ukládána persistentně.

V aplikaci používám toto úložiště pro nastavení síťového připojení, barevného nastavení pro typy docházkových událostí a další uživatelsky měnitelné hodnoty.

Objekty se získávají pomocí příslušné get metody:

```
SharedPreferences settings = PreferenceManager.  
    getDefaultSharedPreferences(context);  
int color = settings.getInt("color", defaultColor);
```

Změny nastavení se provádějí pomocí *SharedPreferences.Editor* rozhraní, které se postará aby data zůstala konsistentní a řídí transakční zpracování.

```
SharedPreferences settings = PreferenceManager.  
    getDefaultSharedPreferences(context);  
SharedPreferences.Editor editor = settings.edit();  
editor.putInt("color", userColor);  
editor.commit();
```

## Interní úložiště

Soubory lze ukládat v interní paměti zařízení. Tyto soubory jsou defaultně přístupné pouze pro aplikace, která je vytvořila a při odinstalování jsou automaticky smazány.

## Externí úložiště

Další možností pro ukládání souborů je externí úložiště (např. SD karta). Toto úložiště je sdílené a data mohou být editována i mimo aplikaci.

## SQLite databáze

Data lze ukládat persistentně pomocí SQLite databáze. Vytvořená databáze je dostupná jakékoli třídě v aplikaci, ale není přístupná mimo aplikaci, která jí vytvořila. SQLite databázi detailněji popisují v kapitole 6.6.

## 6.6 SQLite

[TODO]je treba resit delku dat napriklad stringu?, dynamic typing

V knihovnách pro Forms aplikace se nachází další kód, který bude nutné přepsat do webové služby.

## **6.7 REST**

RestTemplates - springframework

1. REST operace - davkove vs jednotlivé
2. REST, tabulka URI,

## **6.8 Struktura projektu**

(jen ty použite)

## **6.9 Manifest + oprávnění**

persmission v manifestu, vypsát a vysvětlit

## **6.10 9png grafika**

## **6.11 GPS**

## **6.12 Zpětná kompatibilita**

## **6.13 Budoucí rozšiřitelnost**

## **6.14 Vytváření grafů**

knihovny, cloudové řešení, vlastní komponenty

## **6.15 Chybové reporty**

## **6.16 Distribuce**

# 7 Testování

[TODO testovací scenare synchronizace]

## 7.1 O čem psát...

1. pripraveno webove sluzby na dalsi mobilni platformy
2. cinnost aplikce online/offline
3. flow diagramy pro ruzne cinnosti
4. pristupova prava
5. uspora pesistentni pameti na strane androida
6. chybove reporty a opravy na aplikaci v ostrem prostredi, obrazek + ukazka
7. perioda automatickeho mazani dat
8. datovy model schema

## 8 Závěr

# Seznam zkratek

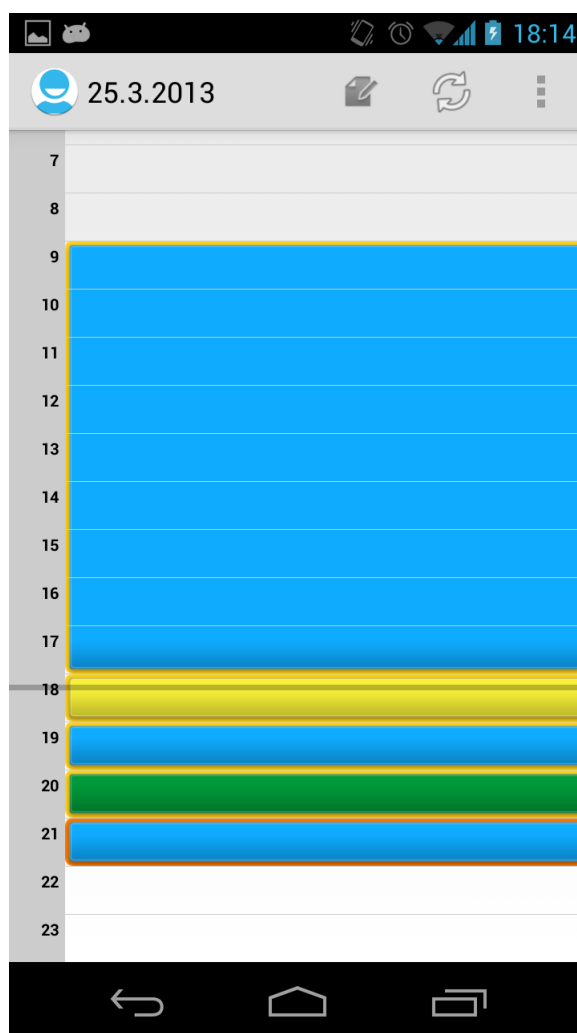
IMIS	Integrovaný manažerský informační systém
ERP	Enterprise Resource Planning
PL/SQL	Procedural Language/Structured Query Language
DML	Data manipulation language
CRUD	Create, read, update and delete
UDDI	Universal Description, Discovery, and Integration

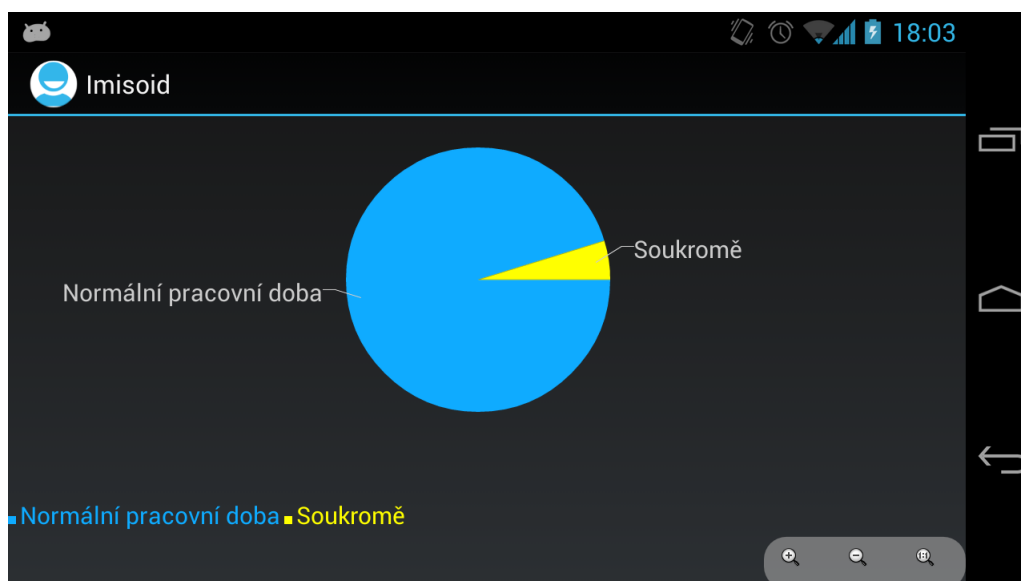


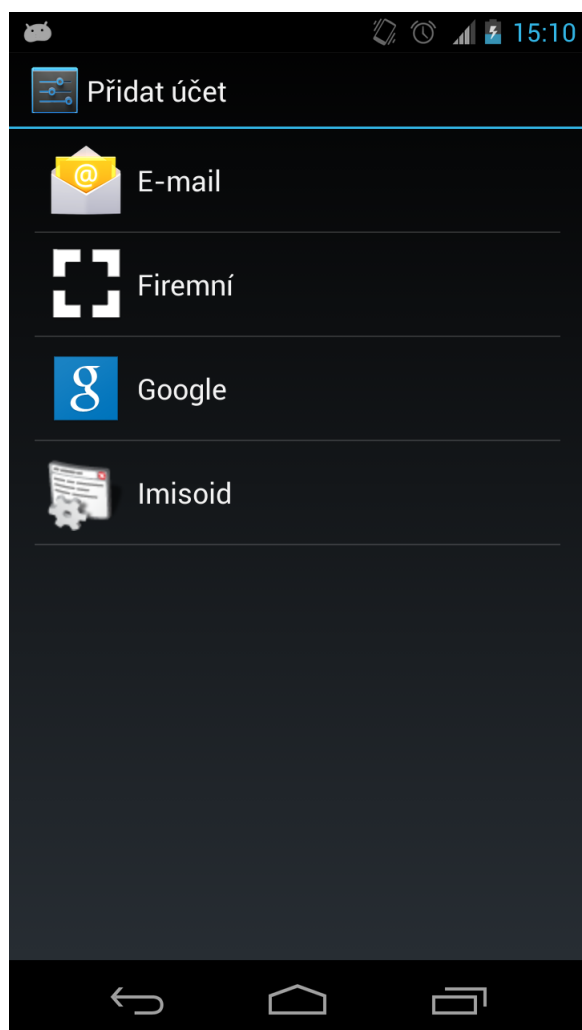
# Literatura

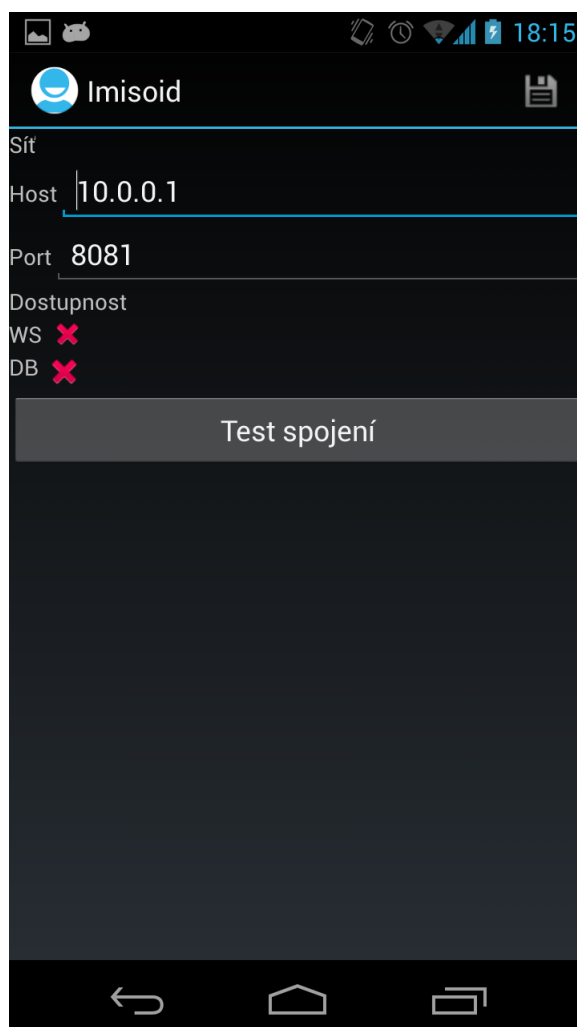
- [1] Fielding, R. T.: Architectural Styles and the Design of Network-based Software Architectures. [online], [cit. 2013-7-6].  
URL <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- [2] <http://developer.android.com/>: The Developer's Guide. [online], [cit. 2013-7-2].  
URL <http://developer.android.com/reference/android/accounts/AccountManager.html>
- [3] <http://technet.microsoft.com/>: Migrating Oracle Forms. [online], [cit. 2013-7-5].  
URL <http://www.oracle.com/technetwork/developer-tools/forms/overview/index.html>
- [4] <http://www.oracle.com/>: Oracle Forms. [online], [cit. 2013-7-5].  
URL <http://www.oracle.com/technetwork/developer-tools/forms/overview/index.html>
- [5] Kosek, J.: Využití webových služeb a protokolu SOAP při komunikaci. [online], [cit. 2013-7-6].  
URL <http://www.kosek.cz/diplomka/html/websluzby.html>
- [6] OASIS: UDDI. [online], [cit. 2013-7-6].  
URL <http://uddi.xml.org/uddi-101>
- [7] Oracle: Oracle Database Mobile Server. [online], [cit. 2013-7-7].  
URL <http://www.oracle.com/technetwork/products/database-mobile-server/overview>
- [8] W3C: Web Services Description Language (WSDL) 1.1. [online], [cit. 2013-7-6].  
URL <http://www.w3.org/TR/wsdl>

# A Uživatelská dokumentace









## B Manifest?