

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 17. července 2013

Maxipes Fík

Abstract

Text of abstract.

Obsah

1	Úvod	1
2	Současný systém	2
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
3	Analýza	12
3.1	Architektura	12
3.1.1	Přímé připojení k databázi	12
3.1.2	Oracle Database Mobile Server	13
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	Datový model v mobilní aplikaci	19
3.3.3	Práce s datumem a časem	20
3.3.4	Výhrady k datové vrstvě	21

3.4	Business logika	22
3.4.1	Triggery formuláře	23
3.4.2	Databázové triggery	23
3.4.3	Databázové uložené funkce a procedury	23
3.4.4	Forms knihovny	24
3.4.5	Shrnutí	24
3.5	Synchronizace dat	25
3.5.1	Obousměrná synchronizace	25
3.5.2	Obousměrná synchronizace s úpravou databáze	27
3.5.3	Řešení kolizí	28
3.5.4	Srovnání	28
3.6	Uživatelské rozhraní	28
4	Zabezpečení	29
4.1	Autentizace a autorizace	29
4.2	VPN pro vzdálený přístup	30
4.3	Autentizace proti databázi	31
4.4	Autentizace proti LDAP	32
4.4.1	LDAP	32
4.5	Shrnutí	34
5	Webová služba	35
5.1	REST	35
5.2	Filtry	38
5.3	web.xml	38
5.4	REST	38
5.5	Klienti jiných platforem?	38
6	Android aplikace	39
6.1	Funkcionalita	39
6.1.1	Nastavení a konfigurovatelnost	40
6.1.2	Uživatelská přívětivost	41
6.1.3	Návrhy na vylepšení	41
6.2	Výběr podporovaných verzí Android API	41
6.3	Komponenty Android aplikace	42
6.3.1	Activity	42
6.3.2	Fragment	44
6.3.3	Async task	46
6.3.4	Service	47
6.3.5	Intent	48
6.3.6	Content provider	48

6.3.7	Broadcast receiver	49
6.3.8	Loader	50
6.3.9	Alarm manager	50
6.4	Komponenty pro synchronizaci	51
6.5	Uživatelský účet	51
6.6	Ukládání dat	53
6.7	Widgety	55
6.8	Získávání geografické polohy zařízení	56
6.9	Google Maps Android API	56
6.10	Notifikace	57
6.11	Vytváření grafů	57
6.12	Oprávnění	58
6.13	Distribuce aplikace	60
7	Závěr	62
	Seznam zkratk	63
	Literatura	65
A	Uživatelská dokumentace	66
A.1	Docházka	66
A.2	Menu	68
A.3	Ovládací panel	69
A.4	Grafy s statistiky	69
A.5	Výkazy	71
A.6	Přítomnost zaměstnanců	72
A.7	Vytvoření účtu	73
A.8	Notifikace	76
A.9	Widgety	76
A.10	Nastavení	78
A.10.1	Sít'ové připojení	79
A.10.2	Poloha pracoviště	80
A.10.3	Nastavení barev typů událostí a zakázek	81
B	Manifest?	82

1 Úvod

2 Současný systém

Integrovaný manažerský informační systém (IMIS) je součástí informačního systému Ramses ERP vyvinutého společností CCA Group a.s. Jedná se o 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

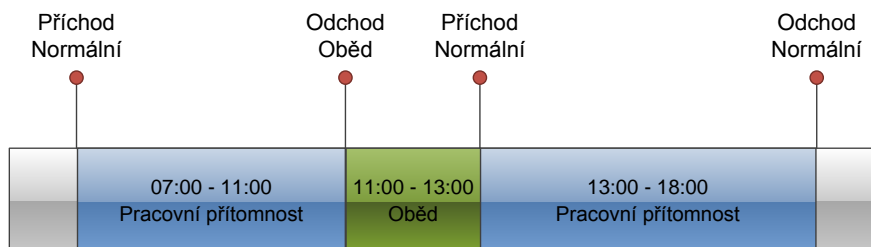
V této práci se zaměřuji na funkcionalitu z oblasti systému věnující se personalistice. Konkrétně se jedná o moduly pro zápis příchodů a odchodů na pracoviště a vykazování provedené práce. Jedná se o činnosti, které zaměstnanec provádí jako každodenní rutinu a zároveň jsou to činnosti s nejširší skupinou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

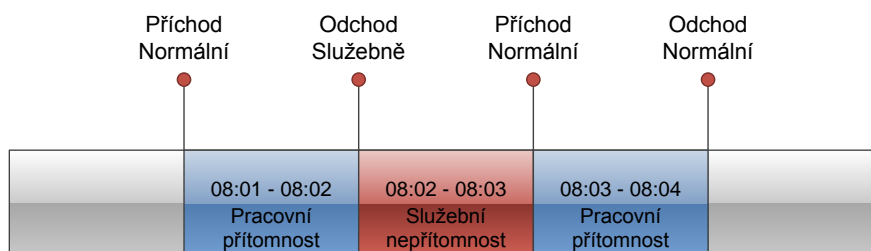
Eviduje se každý příchod i odchod z pracoviště společně s účelem události. Účel události je důležitý, protože délka pracovní přestávky či odchod z pracoviště k lékaři jsou legislativně ošetřené záležitosti.

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[8] 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.

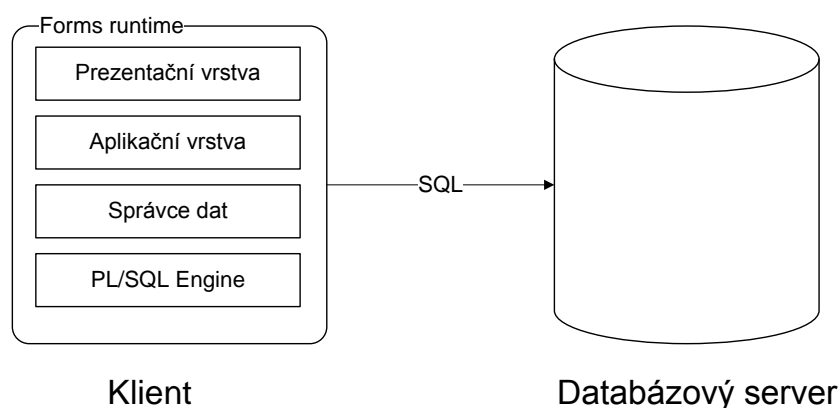
Prostředí běhu zajišťuje defaultní správu transakcí. Díky tomu je Oracle Forms silný nástroj pro efektivní vývoj aplikací, jejichž primárním cílem je přístup k datům uložených v databázi.

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 bussines logiku aplikace. Serverem je myšlen databázový server. Architektura je znázorněna na obr. 2.3



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, zaškrtačací políčka, přepínače, tlačítka atd. Formulář je logicky organizován do bloků. Existují dva typy bloků:

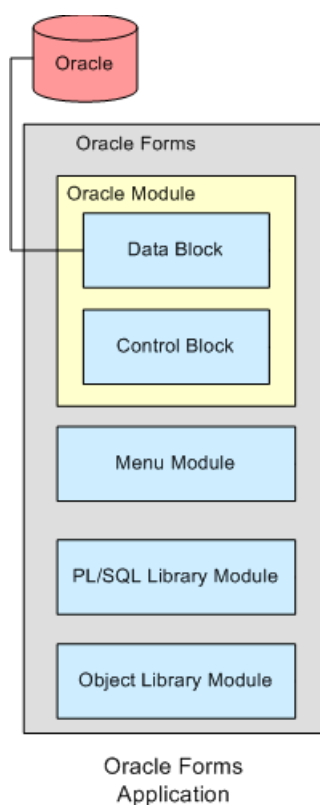
- **Datový blok**
Datový blok zobrazuje zdrojová data a poskytuje abstrakci pro způsob jakým jsou tato data získávána. Blok může být asociován s databázovou tabulkou, databázovým pohledem, uloženou procedurou, dotazem do databáze nebo transakčním triggerem. Asociace datového bloku a databázových dat standartně umožňuje přístup k těmto datům a jejich modifikaci.
Datové bloky mohou být navzájem svázany 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[7]

Triggery

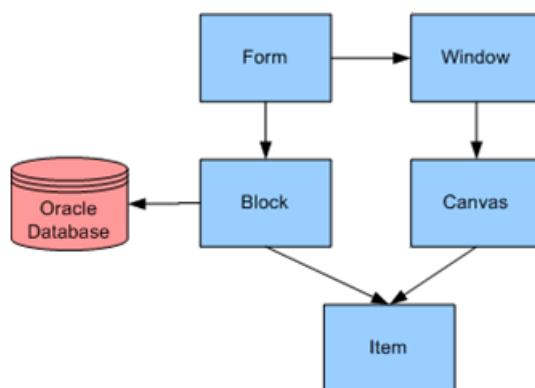
Aplikace v Oracle pracuje s následujícími typy triggerů:

- 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[7]

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

BD27 - Zápis příchodů a odchodů (11.01.2007)

Kód	Heslo	Druh	Popis	Poznámka	Datum	Čas
		00				

Denní záznamy:

Ait1=EDITACE ZÁZNAMU	Ait2=ZAKÁZKY	Ait9=POMŮCKY	Ait7=ZE SOUBORU
----------------------	--------------	--------------	-----------------

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)

VÝKAZ PRÁCE CCA

Uživatel

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

Vedoucí 1429 JANSÁ JIŘÍ Střed PRG (počet zázn.) 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

S	Zakázka	Auto	Km	Organizace	Popis činnosti
V					

Celkem z doch. 03:00 Km A Hodin 03:00 Rozdíl 00:00 14:00(P)-15:00(P),16:59(P)-17:00(P),

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

Alt6=IMPKNÍHY J. Alt7=IMPVÝKAZU ^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

V této kapitole se zabývám analýzou současného systému a návrhem cílové aplikace. Nejprve popíšu zvolenou architekturu a s tím související výběr typu webové služby. Dále popisuji datovou vrstvu jak současného systému tak i cílové aplikace a popisuju některé výhrady k datové vrstvě současného systému. Popisuji kde všude se nachází bussines logika relevantní pro funkčnost cílové aplikace a možnosti využití současného kódu. Popisuji způsob synchronizace dat mezi klientskou aplikací a podnikovou databází. Nakonec zmiňuji některé aspekty týkající se uživatelského rozhraní cílové aplikace.

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[13] 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 přerušování. 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

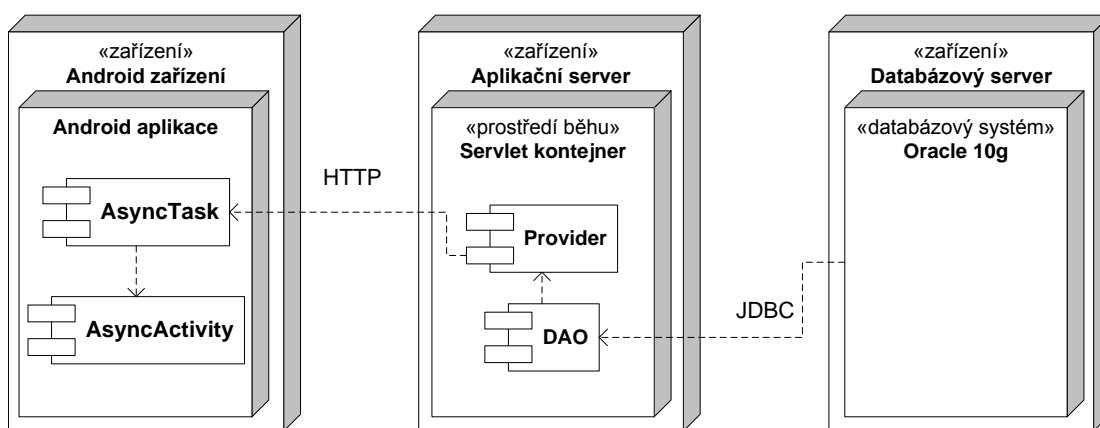
Jako použitou architekturu jsem zvolil použití webové služby, která bude fungovat jako rozhraní mezi klientskou aplikací a databázovým serverem. Android klient v této architektuře funguje jako tenký klient spravující jen část funkčnosti z původního tlustého klienta. Business logika je umístěna na straně 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

Android aplikace funguje jako tenký klient, který se připojuje k webové službě. Webová služba přistupuje k samotné databázi. Jednotlivé komponenty jsou zobrazeny na obr. 3.1 a popsány níže:

- Android - aplikace je podporovaná od API 10. Obsahuje persistentní úložiště. Pro komunikace s webovou službou je určena *AsyncTask* komponenta, která ukládá záznamy do databáze. Záznamy z databáze jsou zobrazeny v uživatelském rozhraní *AsyncActivity*. Kromě tohoto scénáře se navíc bude úložiště automaticky synchronizovat s databázovým serverem prostřednictvím webové služby (není zobrazeno na obr. 3.1).
- Webová služba - je implementovaná v Java EE 6. Dodržuje RESTful principy s využitím Java API pro RESTful webové služby (JAX-RS) 1.1. Webová služba zpřístupňuje data pro klienta prostřednictvím *Provider* komponent zpracovávající HTTP požadavek. Data jsou získávány z databáze pomocí *DAO* (Data Access Object) objektů. Jako produkční aplikační server bude použit GlassFish 4.
- Databázový server - hostuje databázi Oracle 10g, která obsahuje data ke kterým Android aplikace přistupuje. Navíc obsahuje databázové procedury a funkce, které jsou součástí bussines logiky používané v současném systému.



Obrázek 3.1: Diagram nasazení

3.2 Výběr typu webové služby

Po zvolení architektury využívající webovou službu následuje výběr vhodného typu webové služby. V úvahu přichází varianty řešení s využitím SOAP protokolu či REST architektury.

3.2.1 Representational State Transfer

Representational State Transfer[5] (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:

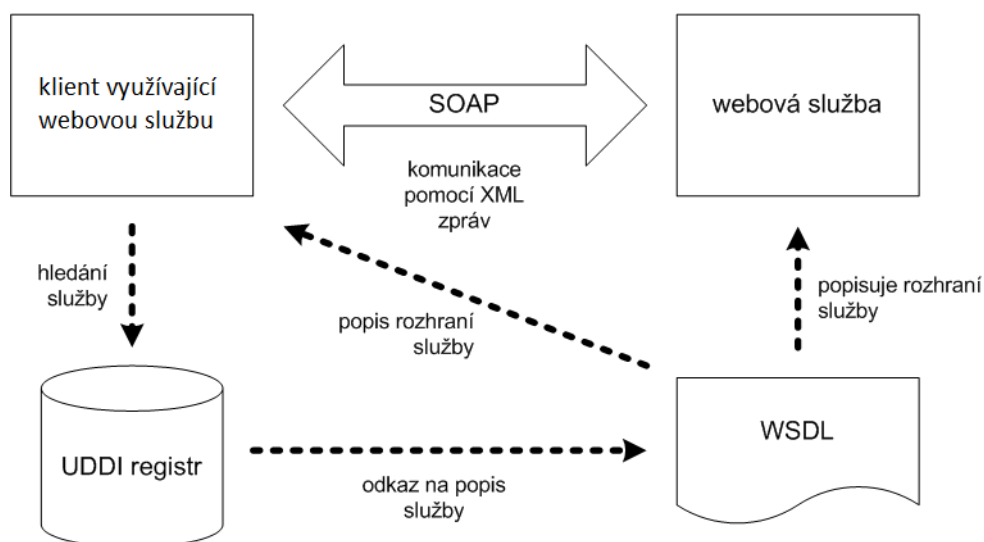
- základní URI pro webové API
- typ internetového média dat poskytovaných API (JSON, XML nebo jiný splňující hypertextové standardy)
- množina operací podporovaná webovým API pomocí HTTP method
 - k získání zdroje se používá GET metoda
 - k aktualizaci zdroje se používá PUT metoda
 - k vytvoření zdroje 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[9]

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

WSDL[14] (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.

3.2.3 Odůvodnění výběru

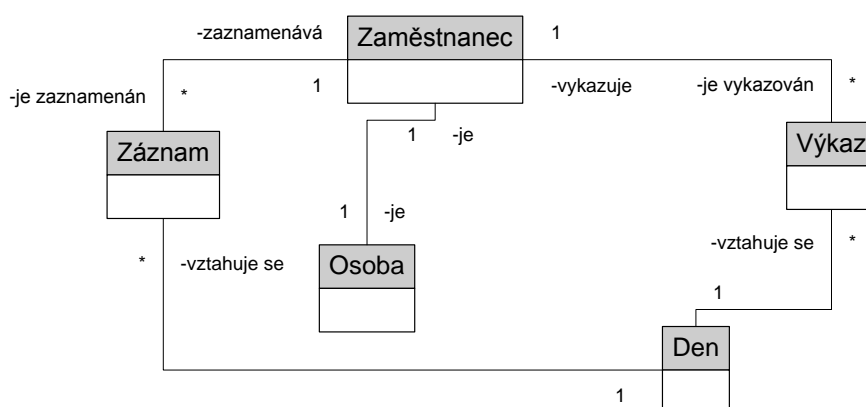
Vzhledem k tomu, že webová služba má sloužit ke správě vzdálené databáze, jeví se REST se svojí CRUD maticí operací jako jasná volba. Při použití REST společně s přenosem dat ve formátu JSON bude objem přenesených dat malý, což je důležité, protože klientskou aplikací má být mobilní zařízení. Webovou 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. Identifikuji datové entity, které jsou relevantní pro zkoumanou část systému a popisují některé nedostatky používaného datového schématu.

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 Datový model v mobilní aplikaci

Na straně Android aplikace budou persistentně ukládány pouze tyto entity:

Záznam Příchod nebo odchod na pracoviště pro uživatele, který rovněž vytváří záznamy prostřednictvím mobilní aplikace. Dále záznamy ostatních uživatelů, které si uživatel stáhne k prohlížení.

Výkaz Výkazy práce které si uživatel stáhne k prohlížení.

Zaměstnanec Lokální databáze zaměstnanců, kde se ukládají základní údaje společně s informací o poslední docházkové události. Databáze zaměstnanců bude sloužit pro naplnění rozbalovací nabídky pro výběr zaměstnance, která bude součástí uživatelského rozhraní. Další využití je pro zobrazení seznamu přítomných zaměstnanců.

Data budou ukládané ve třech tabulkách SQLite databáze. Datový model nebude řešit referenční integritu těchto tabulek. Zajištění referenční integrity je záležitostí databáze systému a její zajištění na straně klienta je zbytečné.

3.3.3 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	s
SQLite INTEGER	2.12.292269055 BC	17.8.292278994 AD	ms
Java long	2.12.292269055 BC	17.8.292278994 AD	ms

Tabulka 3.1: Datové typy reprezentující časový údaj

3.3.4 Výhrady k datové vrstvě

Při práci s datovou vrstvou aplikace jsem vyhodnotil některé vlastnosti jako důsledek špatného návrhu datového schématu.

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 tabulkovému prostoru.

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

- záznam je fyzicky přemístěn do jiného tabulkového prostoru či jiné databáze, v tom případě bude vygenerován nový ROWID a záznam uložený na klientovi bude odkazovat na neexistující umístění či v horším případě na existující ale jiný záznam
- pokud uživatel smaže záznam prostřednictvím jiného klienta zatímco má záznam uložený na svém mobilním zařízení, může rovněž dojít k situaci, kdy záznam uložený na klientovi bude odkazovat na neexistující umístění nebo na existující ale jiný záznam

Vhodným řešením by byla změna databázového schématu tabulky pro záznam docházkových událostí a to přidáním primárního klíče. Vzhledem k tomu že se se záznamy pro docházku po zpracování aktuální měsíce již dále nepracuje, tak případná změna starších dat nemůže napáchat reálné škody.

Schéma pro docházku

Záznam o příchodu či odchodu reprezentuje vždy jeden záznam pro každou z těchto událostí. Při práci s těmito daty na aplikační vrstvě je nutné vždy spárovat příchody a odchody týkající se daného zaměstnance. Je na zvážení zda nemohlo být datové schéma navržené lépe např. příchod či odchod by představoval jeden záznam v tabulce nebo pokud by každý odchod odkazoval na příslušný příchod. Kromě jednoduššího zpracování na aplikační vrstvě by byla rovněž snazší kontrola chybného vstupu, kdy zaměstnanec omylem zadá dva příchody či odchody následující po sobě. Taková kontrola se v současnosti v systému neprovádí.

Datum a čas odděleně

Datum a čas docházkové události se uchovává odděleně, kdy každá z těchto hodnot má svůj příslušný sloupec. Je na zvážení, zda tyto údaje neměly být ukládány jako jedna hodnota.

Čas jako reálné číslo

Jako špatné rozhodnutí hodnotím skutečnost, že údaj o čase je reprezentován jako reálné číslo nabývající hodnot $<0.0, 24.0>$. Tento fakt přináší komplikace při převodu čísla na hodiny a minuty. Údaj o čase by mohl být reprezentován jako celé číslo představující minuty.

3.4 Business logika

Při analýze zdrojového kódu zkoumané části současného systému jsem identifikoval kód v těchto umístěních:

- trigger, které jsou součástí vybraných formulářů
- knihovny využívané vybranými formuláři
- databázové trigger, funkce a procedury

Veškerý kód je napsán v jazyce PL/SQL.

3.4.1 Triggery formuláře

Z množiny triggerů (zmíněných v 2.2.3) použitých ve formulářích bude nutné implementovat triggery pracující na datovém bloku a validační triggery. Ostatní typy triggerů nemají v kontextu implementované aplikace smysl. Konkrétně se jedná o tyto:

Block-processing trigger: On-Delete, On-Insert, On-Update, Pre-Delete, Pre-Insert, Pre-Update

Validation trigger: When-Validate-Item

3.4.2 Databázové triggery

Databázové triggery se provádějí při definovaných událostech jako vložení, změna či smazání záznamu do konkrétní tabulky. Jejich činnost provádí databáze a to bez ohledu na to, který klient k databázi přistupuje. Jejich funkčnost tedy zůstává nezměněna při použití společně s webovou službou.

3.4.3 Databázové uložené funkce a procedury

Funkce a procedury uložené v Oracle databázi lze volat z aplikace prostředky jazyka Java. Slouží k tomu rozhraní *java.sql.CallableStatement*. Ukázka volání uložené funkce z Javy:

```
CallableStatement callableStatement = conn.prepareCall(
    "{? = call IMISOID_HESLO_WRAPPER(?, ?)}");
callableStatement.setString(2, icp);
callableStatement.setString(3, heslo);
callableStatement.registerOutParameter(1, OracleTypes.NUMBER);
callableStatement.executeUpdate();
BigDecimal bool = callableStatement.getBigDecimal(1);
```

Funkce vracející typ BOOLEAN

Problém nastává pokud uložená funkce vrací typ BOOLEAN. Ovladač databáze Oracle JDBC nepodporuje typ BOOLEAN jako návratový datový

typ [12]. Řešením je použití obalující procedury, která zpracuje výsledek volané procedury vracející BOOLEAN, ale sama vrací podporovaný datový typ (např. NUMBER). Ukázka takové obalující procedury:

```
FUNCTION IMISOID_HESLO_WRAPPER(PIcp in Varchar2, PHeslo in
    Varchar2) RETURN NUMBER AS
    myvar boolean;
BEGIN
    myvar := ccap_heslo_pkg.SpravneHeslo(PIcp, PHeslo);

    IF myvar THEN
        RETURN 1;
    ELSE
        RETURN 0;
    END IF;
END;
```

3.4.4 Forms knihovny

Další zdrojový kód používaný formuláři se nachází v knihovnách. Ačkoli se jedná o nezávislé jednotky, není způsob jak použít procedury v nich obsažené z prostředí jazyka Java.

3.4.5 Shrnutí

Část současného kódu zajišťující business logiku formulářů bude muset být přepsána do jazyka Java. Tento přepsaný kód bude umístěn do webové služby. Konkrétně se jedná o zdrojový kód umístěný v současných formulářích a knihovnách, které jsou jím používány. Kód umístěný na straně databáze lze využít a nebude muset být přepsán.

3.5 Synchronizace dat

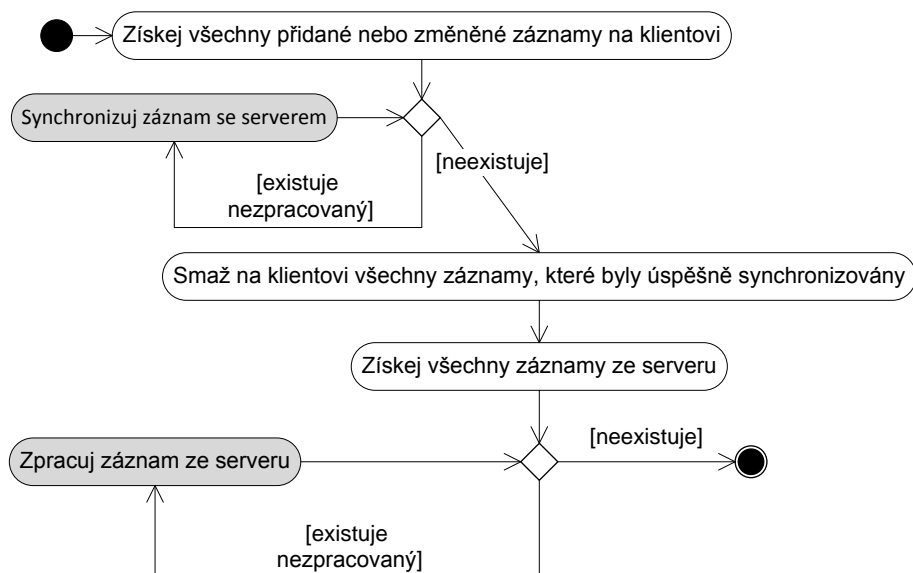
V současném systému uživatel zadává data prostřednictvím příslušného formuláře. Změny jsou aplikovány bezprostředně po uložení během databázové transakce. Mobilní klient přináší nový způsob použití - data lze zadat i v režimu offline, kdy mobilní klient není v dosahu webové služby. Tyto data jsou uložena 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.

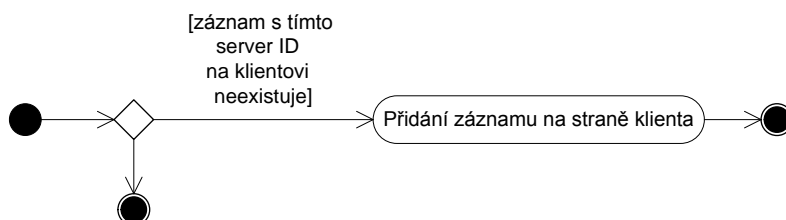
Celkový průběh synchronizace je znázorněn v diagramu 3.4. Klient nejprve odešle všechny svoje změny na server. Poté smaže všechny úspěšně odeslaná data. Bez smazání by nebylo možné zjistit, že na serveru došlo ke změně či smazání dat jiným klientem. Poté už zbývá pouze stažení aktuálních dat ze serveru. 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.



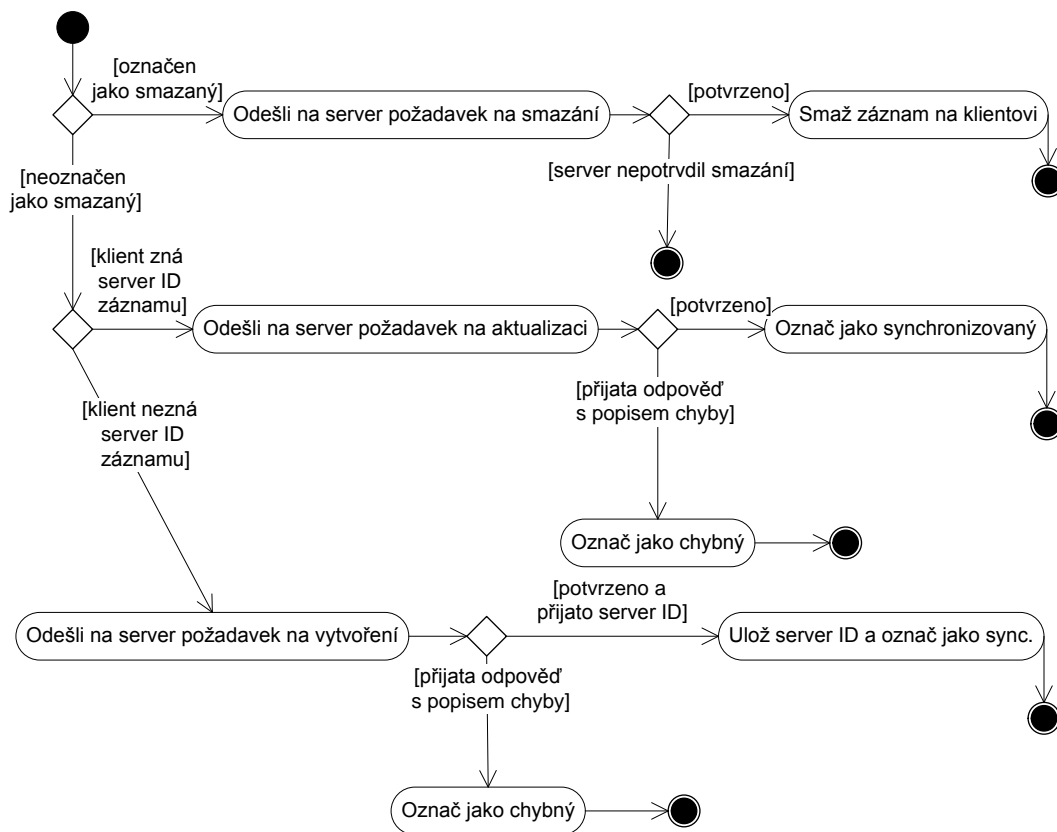
Obrázek 3.4: Diagram aktivit pro průběh synchronizace

Diagram ?? 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.

Diagram 3.5 podrobněji rozepisuje průběh přijetí záznamu ze serveru.



Obrázek 3.5: Diagram aktivit pro přijetí záznamu ze serveru



Obrázek 3.6: Diagram aktivit pro odeslání požadavku na server

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 změní 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

Uživatelské rozhraní současného systému je desktopové rozhraní. Při implementaci mobilního klienta budou muset být respektovány vlastnosti takového zařízení jako je velikost displeje či nepohodlnost používání softwarové klávesnice.

Uživatel mobilního zařízení nebude muset zadávat veškeré vstupy jako na formuláři 2.6, protože zařízení si bude pamatovat jeho identitu. Dále bude vhodné využít některé komponenty poskytované Android SDK jako je např. widget pro kalendář.

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.

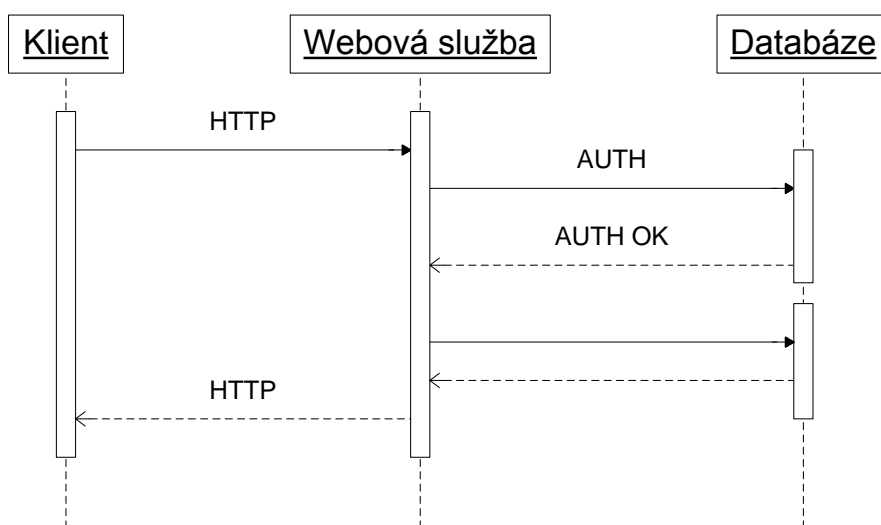
Vlastnosti připojení VPN

- 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

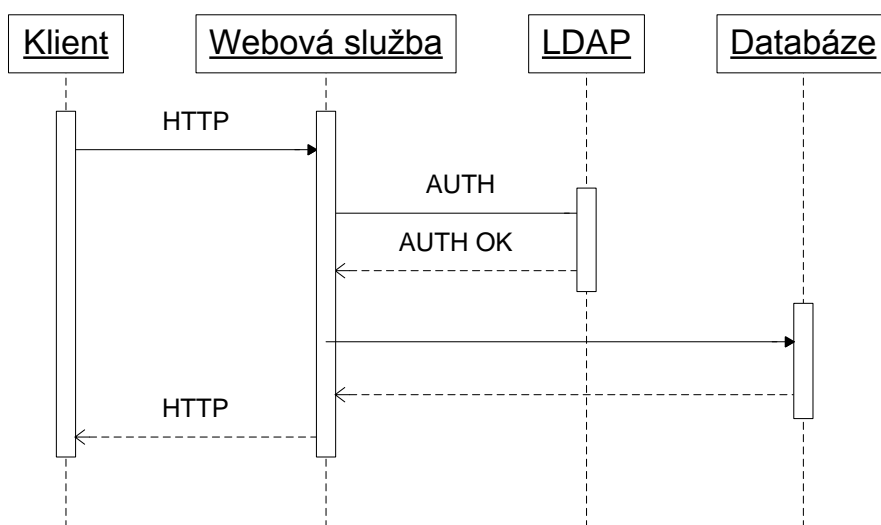
Klient i server jsou součástí jedné VPN sítě, která zajišťuje důvěryhodný komunikační kanál. Klient posílá na server HTTP požadavek jehož součástí je autentizační hlavička nesoucí uživatelské jméno a heslo. Webová služba ověří jméno a heslo pomocí databázové procedury. Jméno a heslo uživatele je uloženo 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 jejích 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"> • icp - identifikátor zaměstnance • from - datum začátku období • to - datum konce období
DELETE	events/{rowid}
	Smaže danou událost Parametry: <ul style="list-style-type: none"> • rowid - identifikátor události
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"> • rowid - identifikátor události
GET	events/sum/{icp}?from={from}&to={to}
	Získá součet přítomnosti zaměstnance za dané období Parametry: <ul style="list-style-type: none"> • icp - identifikátor zaměstnance • from - datum začátku období • to - datum konce období

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

Tabulka 5.3: Služby pro výkazy práce

5.2 Filtry

5.3 web.xml

5.4 REST

RestaTemplates - springframework= jak to vola klient...

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

5.5 Klienti jiných platforem?

6 Android aplikace

6.1 Funkcionalita

[TODO co uz bylo ve starem, co v novem] Na základě současného systému a potřeb zaměstnanců byla vybrána k implementaci následující funkčnost:

Docházka

- Zobrazení událostí docházky daného zaměstnance pomocí denní časové osy
- Zobrazení událostí docházky daného zaměstnance pomocí seznamu
- Uživatel má možnost přidávat, editovat a mazat svoje události
- Aplikace zajišťuje automatickou synchronizaci těchto údajů s firemní databází
- Zobrazení poměru typů docházkových událostí za dané období pro vybraného zaměstnance
- Zobrazení vývoje daného typu události v daném období pro vybraného zaměstnance
- Zobrazení souhrnných statistik událostí za dané období pro vybraného zaměstnance

Výkazy práce

- Zobrazení seznamu výkazů za dané období pro vybraného zaměstnance
- Zobrazení poměru typů zakázek za dané období pro vybraného zaměstnance
- Zobrazení vývoje daného typu zakázky v daném období pro vybraného zaměstnance
- Zobrazení souhrnných statistik výkazů za dané období pro vybraného zaměstnance

Aktuální přítomnost na pracovišti

- Zobrazení seznamu všech zaměstnanců a jejich poslední docházkové události (aktuální stav přítomnosti či nepřítomnosti na pracovišti)
- Uživatel má možnost spravovat seznam svých oblíbených zaměstnanců a tento seznam zobrazovat přednostně

Widgety

- Widget pro rychlé zadání docházkové události
- Widget pro zobrazení poslední docházkové události vybraného zaměstnance (aktuální stav přítomnosti či nepřítomnosti na pracovišti)

Notifikace

- Notifikace při opomenutí zadání příchodu či odchodu

Online vs offline

Veškeré výše zmíněné funkce jsou dostupné jak v režimu online tak i offline. Rozdíl je v tom, že v režimu offline nemusí aplikace pracovat s aktuálními daty.

6.1.1 Nastavení a konfigurovatelnost

Zařízení si musí pamatovat údaje nutné pro snadnou obsluhu tzn. uživatelské jméno a heslo. V následujícím seznamu je výčet všech konfigurovatelných vlastností aplikace:

- Parametry připojení k webové službě (doména a port)
- Povolení automatické synchronizace pouze při WIFI připojení
- Perioda synchronizace vlastních docházkových událostí
- Perioda synchronizace widget pro zobrazení vybraných zaměstnanců
- Perioda synchronizace seznamu zaměstnanců
- Upozornění na chybějící příchod (vypnout či zapnout)
- Upozornění na chybějící odchod (vypnout či zapnout)
- Časová prodleva pro upozornění na chybějící událost docházky

- Barva typu docházkové události
- Barva typu zakázky

6.1.2 Uživatelská přívětivost

Uživatelské rozhraní aplikace klade důraz na přehlednost, ergonomii a časově efektivní obsluhu.[TODO jeste neco vymyslet]

6.1.3 Návrhy na vylepšení

Původní ambicí byla možnost zadávání výkazů práce. Vzhledem k tomu, že taková funkčnost by vyžadovala změny v současném systému, bylo od tohoto záměru upuštěno. Kromě zadávání výkazů práce by bylo možné rovněž vykazovat do knihy jízd.

Velikým přínosem by byla možnost vytváření šablon pro často opakované výkazy, která by uživateli ušetřila nezanedbatelné množství času.

Posledním vylepšením by byla možnost rychlého zavolání přímo z profilu vybraného zaměstnance.

6.2 Výběr podporovaných verzí Android API

Aplikace podporuje operační systémy od verze 2.3.3. Jak je vidět z tabulky 6.1 aplikace podporuje více než 95% trhu. Jako hlavní vývojové zařízení byl použit smartphone Google Nexus s operačním systémem ve verzi 4.2.2.

Verze	Kódové označení	API	Podíl uživatelů[%]
1.6	Donut	4	0.1
2.1	Eclair	7	1.4
2.2	Froyo	8	3.1
2.3.3 -2.3.7	Gingerbread	10	34.1
3.2	Honeycomb	13	0.1
4.0.3 -4.0.4	Ice Cream Sandwich	15	23.3
4.1.x	Jelly Bean	16	32.3
4.2.x	Jelly Bean	17	5.6

Tabulka 6.1: Poměr verzí zařízení na trhu k 8.7.2013[4]

6.3 Komponenty Android aplikace

6.3.1 Activity

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.

[TODO UML class diagram]

6.3.2 Fragment

Fragment představuje komponentu uživatelského rozhraní, která je součástí aktivity. Aktivita může obsahovat více fragmentů a fragment může být využíván ve více aktivitách. Jedná se tedy o modul umožňující nějakou funkčnost, který má vlastní životní cyklus, přijímá události z uživatelského rozhraní a

může být přidán či odstraněn z aktivity během její činnosti. Fragment musí být vždy součástí nějaké aktivity a je přímo ovlivňován životním cyklem hostující aktivity.

K vytvoření fragmentu je nutné oddědit od třídy *android.app.Fragment*.

Metody životního cyklu fragmentu

Tato třída je podobná třídě aktivity s tím rozdílem, že obsahuje několik metod životního cyklu navíc:

- **onAttach()**
Metoda je volána jakmile je fragment přiřazen k aktivitě.
- **onCreateView()**
Volána poté co je fragment vytvořen. Fragment který má uživatelské rozhraní musí vrátit objekt *View*, který obsahuje uživatelské rozhraní pro tento fragment.
- **onActivityCreated()**
Volána ve chvíli, kdy je vytvořena hostující aktivita.
- **onDestroyView()**
Volána ve chvíli, kdy je uživatelské rozhraní fragment odstraňováno.
- **onDetach()**
Volána ve chvíli, kdy je fragment odstraňován z aktivity.

[TODO příklady fragmentu z projektu]

Provádění transakcí

Fragment je možné přidávat, mazat či nahrazovat jinou v rámci rodičovské aktivity. Ty akce se nazývají transakce a provádějí se pomocí třídy *FragmentTransaction*. V následující ukázce kódu je současný fragment v aktivitě nahrazen fragmentem *PieChartFragment* zobrazující koláčový graf. Ukázka je ze třídy *ChartActivity*:

```
FragmentTransaction ft = getSupportFragmentManager().  
    beginTransaction();  
PieChartFragment pieFragment = new PieChartFragment();
```

```
ft.replace(R.id.displayChart, pieFragment, FRAG_PIE);  
ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_CLOSE);  
ft.commit();
```

Změna konfigurace s využitím fragmentu

Další uplatnění fragmentů se nachází v ošetření situace, kdy se v důsledku rotace zařízení změni poloha displeje. V případě kdy aktivita spouští asynchronní úkol (*AsyncTask* 6.3.3) a uživatel otočí displej, aktivita bude restartována. V momentě kdy *AsyncTask* dokončí svůj úkol bude jeho výsledek zasílat staré instanci aktivity, která v té chvíli už neexistuje. Kromě toho nově vytvořená aktivita spustí znovu ten samý *AsyncTask*. Obojí je samozřejmě nežádoucí. Řešením tohoto problému je využít fragment jako hostitele pro *AsyncTask* a nastavit metodou fragmentu *setRetainInstance (true)* vlastnost, že fragment nebude restartován při restartu jeho hostitelské aktivity[10]. [TODO odkazat se na tridy v projektu nebo dat ukazku do prilohy]

6.3.3 Async task

AsyncTask je komponenta sloužící k provedení operace paralelně k UI (User Interface) vláknu. Třída umožňuje provedení operace na pozadí a publikování jejích výsledků na UI vlákne bez manipulace s vlákny.

Generické parametry pro *AsyncTask*

1. **Params** typ parametrů předaných úkolu před provedením.
2. **Progress** typ jednotek pro průběh operace publikovaných během výpočtu na pozadí.
3. **Result** typ pro výsledek výpočtu.

Metody *AsyncTask*

Třída *AsyncTask* využívá těchto metod:

- **onPreExecute()**
volána na UI (User Interface) vlákne před tím, než je úkol spuštěn.

Metoda louží k přípravě asynchronní akce např. zobrazení dialogu s průběhem akce.

- **doInBackground(Params...)**
provádí se ve vlastním vlákne na pozadí. Vrací výsledek svojí akce. Může oznamovat průběh akce pomocí *publishProgress(Progress...)*.
- **onProgressUpdate(Progress...)**
volána na UI vlákne. Slouží k zobrazení průběhu akce zatímco úkol na pozadí je stále prováděn.
- **onPostExecute(Result)**
volána na UI vlákne po tom, co je úkol na pozadí dokončen. Výsledek akce na pozadí je předán jako parametr.

AsyncTask komponenty jsou vhodné pro krátké operace (max jednotky sekund), které přímo ovlivňují chování uživatelského rozhraní. Příkladem takové operace může být stažení dat, která mají být zobrazena okamžitě jakmile jsou dostupná. Taková akce musí být provedena asynchronně jinak by docházelo k efektu zamrzání uživatelského rozhraní. [TODO příklady AsyncTask z projektu+diagram tříd]

6.3.4 Service

Service neboli služba je komponenta aplikace, která provádí dlouhotrvající operaci na pozadí a neposkytuje uživatelské rozhraní. Služba může být spuštěna vlastní aplikací, z jiné aplikace či systémovou komponentou. Služba běží v hlavním vlákne hostujícího procesu. Služby lze klasifikovat na dva typy:

Started

Spuštěná služba, je taková služba, která byla spuštěna voláním metody *startService()* kontextu komponenty. Služba běží na pozadí dokud neskončí svojí činností, a to i tehdy pokud byla spouštějící komponenta zničena. Služba typická provádí nějakou operaci, bez interakce se spouštějící komponentou a po dokončení se sama zničí. Metody životního cyklu služby jsou:

- **onCreate()**
Volána ve chvíli kdy je služba vytvořena. Je určena k počáteční inicializaci.

- **onStartCommand()**
Volána ve chvíli kdy je služba spuštěna. Metodou *stopSelf()* se může služba sama ukončit.
- **onDestroy()**
Volána systémem ve chvíli, kdy je služba ničena. Určena pro uvolnění zdrojů.

Bound

Připoutaná (angl. bound) služba, je taková služba, ke které se komponenta připoutala voláním *bindService()*. Připoutaná služba poskytuje klient-server rozhraní, které umožňuje komunikaci s touto službou pomocí meziprocesové komunikace. Služba běží pouze pokud je k ní nějaká komponenta svázaná. Ke službě může být připoutáno více komponent, jakmile se všechny odpoutají, je služba zničena. Metody životního cyklu služby jsou kromě *onCreate()*, *onDestroy()* tyto:

- **onBind()**
Volána systémem ve chvíli, kdy se jiná komponenta chce ke službě přivázat voláním *bindService()*. Vrací objekt *IBinder*, který představuje rozhraní pro komunikaci.
- **onUnbind()**
Komponenta klienta se odpoutává od služby.

[TODO příklady v projektu]

6.3.5 Intent

6.3.6 Content provider

Content provider je jedna ze základních komponent pro tvorbu Android aplikace. Komponenta řídí přístup k centrálnímu úložišti dat představující SQLite databázi. *Content provider* zapouzdřuje data a poskytuje je ostatním aplikacím pomocí *android.content.ContentResolver* rozhraní. Pokud data aplikace nemají být přístupná i jiným aplikacím, není této komponenty potřeba a je možné přistupovat k datům přímo pomocí třídy *android.database.sqlite.SQLiteDatabase*. Důležité metody této komponenty jsou::

- **query()**
Provádí dotaz do úložiště a vrací výsledek dotazu.

- **insert()**
Vkládá data do úložiště.
- **update()**
Aktualizuje existující data v úložišti.
- **delete()**
Maže data z úložiště.
- **getType()**
Vrací MIME typ dat obsažených v úložišti.

Přístup pomocí URI

Umístění dat je specifikováno pomocí URI. Ukázka URI použitého v aplikaci:

```
content://user_dictionary/words
```

- **content://** představuje schéma
- **dictionary** symbolické jméno úložiště tzv. *authority*
- **words** název tabulky

6.3.7 Broadcast receiver

Komponenta *Broadcast receiver* slouží jako posluchač systémové či aplikační události. Všichni registrovaní posluchači jsou informováni jakmile událost nastane. Objekt musí být registrován pomocí *android.content.Context.registerReceiver()* metody kontextu k poslechu události specifikované pomocí filtru událostí *android.content.IntentFilter*. Událost může být vyvolána systémem (např. připojení nabíječky, změna stavu připojení k síti) nebo může být vytvořena aplikací. Nejdůležitější metoda této třídy:

- **onReceive(Context context, Intent intent)**
Metoda je volána ve chvíli kdy je přijata očekávaná událost. Životnost objektu po dokončení této metody končí.

[TODO příklad z aplikace]

6.3.8 Loader

Třída *android.content.Loader<D>* slouží k asynchronnímu načtení dat pro aktivitu či fragment. Monitoruje zdroj svých dat a automaticky se aktualizuje v momentě změny těchto dat. Loader spolupracuje s třídami *android.app.LoaderManager*, která řídí činnost všech Loader objektů patřících dané aktivitě či fragmentu. Aktivita či fragment musí implementovat rozhraní *android.app.LoaderManager.LoaderCallbacks<D>*, které obsahuje tyto callback metody:

- **onCreateLoader (int id, Bundle args)**
Určená pro inicializaci Loader objektu s daným *id*.
- **onLoadFinished (Loader<D> loader, D data)**
Volána po načtení dat. Poskytuje objekt s načtenými daty.
- **onLoaderReset (Loader<D> loader)**
Volána v případě kdy Loader byl resetován. Určena pro zrušení všech referencí na zdrojový objekt s daty.

6.3.9 Alarm manager

Třída *android.app.AlarmManager* je určena pro případy kdy má nějaká akce proběhnout v určený čas a to i v případě kdy aplikace není spuštěna. Ve chvíli kdy je akce prováděna *AlarmManager* drží zámek CPU do té doby než akce skončí. Volitelně lze nastavit, zda se má kvůli provedení akce vzbudit zařízení z režimu spánku.

- **setRepeating(int type, long triggerAtMillis, long intervalMillis, PendingIntent operation)**
Naplňuje pravidelně se opakující akci.
- **set(int type, long triggerAtMillis, PendingIntent operation)**
Naplňuje akci na určitý čas.

[TODO ukázka kódu + popis použití]

6.4 Komponenty pro synchronizaci

Pro synchronizaci je klíčovou komponentou třída *android.content*.

AbstractThreadedSyncAdapter. Třída provádí akci na pozadí ve vlastním vlákně.

Synchronizace je vyvolána systémovou komponentou *android.content.SyncManager*, v závislosti konkrétním nastavení pro konkrétní adaptér. Odpadá tak starost s prací s vlákny a k využití této komponenty stačí implementovat metodu *onPerformSync()*:

- **onPerformSync(Account account, Bundle extras, String authority, ContentProviderClient provider, SyncResult syncResult)**
 - **account** účet, který je synchronizován
 - **extras** parametry specifické pro tento adaptér
 - **authority** *authority*, pro kterou je požadována synchronizace
 - **provider** instance *ContentProviderClient*, která odkazuje na *ContentProvider* pro tuto *authority*
 - **syncResult** určeno pro výsledek synchronizace, který bude předán komponentě *SyncManager*

[TODO příklad z aplikace] [TODO diagram komponent]

6.5 Uživatelský účet

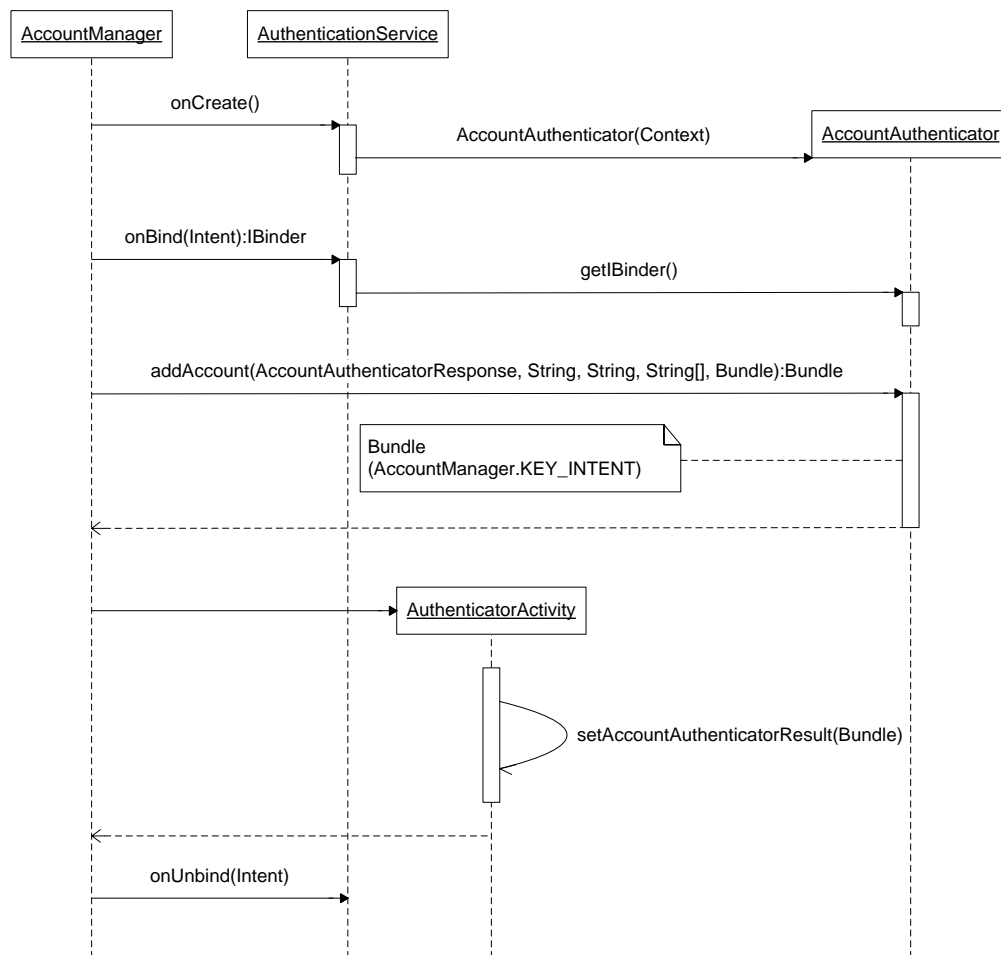
Android poskytuje správu účtů pro online služby. Uživatel zadá svoje přihlašovací údaje při vytvoření účtu a dává tak aplikaci svolení k využívání tohoto účtu.

Různé online služby mohou využívat odlišný způsob pro autentizaci. Android manažer účtů využívá komponentu *authenticator* [6], 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. A.14) zvolí přidání nového účtu a následně zvolí typ účtu (obr. A.15), 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. A.14).



Obrázek 6.1: Sekvenční diagram zobrazující průběh vytvoření účtu

6.6 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 ?? [TODO citace]

[TODO]je třeba resit delku dat napríklad stringu?, dynamic typing [TODO rozepsat podrobněji]

6.7 Widgety

Widget je malá aplikace, která může být zobrazena na hostiteli widget, typicky na domácí obrazovce (angl. homescreen) Android zařízení. Widget běží v rámci procesu svého hostitele.

K vytvoření widgety jsou zapotřebí *android.appwidget.AppWidgetProviderInfo* objekt obsahující metadata, implementace třídy *android.appwidget.AppWidgetProvider* definující chování prvku, layout definující vzhled prvku a volitelně i konfigurační aktivita, která bude spuštěna při vytvoření nové widget prvku. Zde je výčet nejdůležitějších metod třídy *AppWidgetProvider*:

- **onDeleted()**
Volána ve chvíli, kdy je widget smazán ze svého hostitele.
- **onDisabled()**
Volána ve chvíli, kdy je poslední instance widgety smazána ze svého hostitele.
- **onEnabled()**
Volána ve chvíli, kdy je vytvořena první instance widgetu.
- **onUpdate()**
Volána pokaždé kdy mám být widget aktualizován, což nastane v případě kdy uplyne perioda (definována v metadatach) nebo při přijetí *Intent* objektu vytvořeného za účelem aktualizace widgetu.

V aplikaci jsou k dispozici dva typy widget komponent. Widget pro rychlé zadání docházkové události je zobrazen na obr. A.20. Widget pro zobrazení přítomnosti (poslední docházkové události) vybraného uživatele je zobrazen

na obr. A.21 a jeho konfigurační aktivita pro výběr zaměstnance je na obr. A.22.

6.8 Získávání geografické polohy zařízení

V Androidu existují dvě možnosti zjišťování polohy zařízení. První možností je zjištění pozice pomocí GPS, která má vysokou přesnost, ale funguje pouze v otevřených prostorech, má pomalou odezvu a rychle spotřebovává baterii. Druhou možností je získávání pozice pomocí sítě (sítě operátorů a WI-FI), která funguje jak uvnitř tak v otevřených prostorech, má rychlou odezvu, spotřebovává méně baterie, ale není tak přesná. Je možné použít v aplikaci ke zjišťování polohy pouze jeden z těchto zdrojů nebo kombinaci obou.

Kvalita získané polohy

Získávání přesné polohy zařízení je komplikovaná záležitost. Do kvality získané polohy se podílejí tyto faktory:

- **Velké množství zdrojů polohy**
Informaci o poloze lze získávat z GPS, vysílačů operátora, či Wi-Fi. Vyhodnocení informací o poloze z těchto zdrojů je vždy kompromisem mezi přesností, rychlostí a spotřebovanou baterií.
- **Pohyb uživatele**
Kvůli pohybu živatele nemusí být získaná poloha v daný okamžik aktuální.
- **Měnění se přesnost**
Z důvodu různé přesnosti získané polohy nemusí být nejnovější údaj zdaleka ten nejlepší.

[TODO +příklad použití, alg detekce prit/nepřit], android dulezite tridy

6.9 Google Maps Android API

- `READ_GSERVICES`

- MAPS_RECEIVE

6.10 Notifikace

Notifikace je zpráva zobrazená uživateli mimo uživatelské rozhraní aplikace. Notifikace je po vytvoření zobrazena v notifikační oblasti obrazovky zařízení. Po rozbalení notifikační lišty lze vidět detailní popis notifikace a také na ní lze kliknout.

V aplikaci je pomocí notifikace oznámeno uživateli, že zapoměl zadat příchod či odchod na pracoviště. To je kontrolováno na pozadí a proto je uživatel informován tímto způsobem. Při kliknutí je puštěna aktivita umožňující přidání docházkové události. [TODO obr]

6.11 Vytváření grafů

Pro potřebu přehledného zobrazení statistik jsem se rozhodl využít grafy. Konkrétně koláčový graf pro zobrazení poměrů typů docházkových událostí či zakázek a sloupcový graf pro zobrazení jejich vývoje v čase.

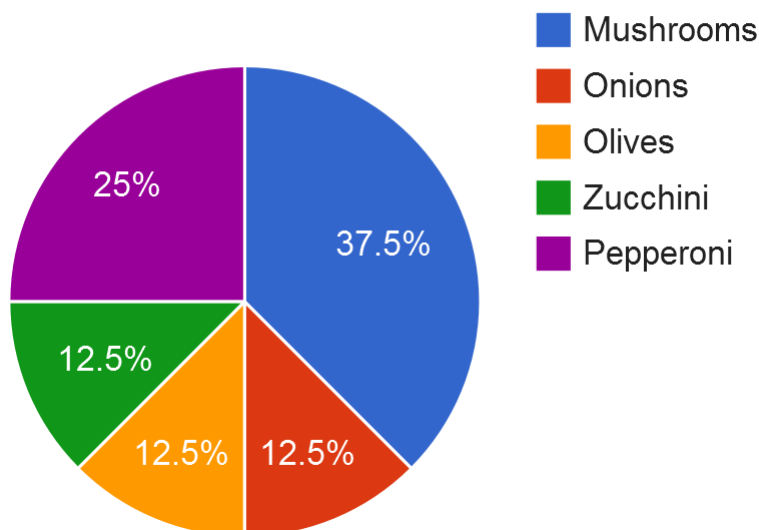
Implementace vlastní komponenty

Implementace vlastních komponent pro grafy by bylo velice časově náročné a proto jsem tuto variantu zamítl.

Využití cloudového řešení

Jako cloudové řešení se nabízí Google Charts API[3]. Google Charts API poskytuje Javascriptové knihovny, pro vygenerování grafu. Aplikace stáhne knihovny, které potřebuje pro požadový graf, vytvoří objekt grafu, naplní ho daty, upraví jeho parametry a nakonec ho umístí do *android.webkit.WebView* komponenty, která slouží k zobrazení webového obsahu. Nevýhoda použití Google Charts API spočívá v nedostupnosti jakýchkoli grafů při nedostupnosti síťového připojení. Na obr. 6.2 je ukázka grafu.

How Much Pizza I Ate Last Night



Obrázek 6.2: Ukázka grafu vytvořeného Google Charts API

Použití knihovny třetí strany

Jako nejlepší řešení jsem vyhodnotil použití knihovny třetí strany konkrétně AChartEngine [1] ve verzi 1.1.0. Výhoda tohoto řešení spočívá v tom, že s grafy lze pracovat i pokud je zařízení mimo dosah sítě. [TODO obr z aplikace]

6.12 Oprávnění

V operačním systému Android běží každá aplikace ve svém *sandboxu*, což je bezpečnostní mechanismus, který slouží pro oddělování běžících procesů. Pokud chce aplikace přístup k systémovým zdrojům, datům jiné aplikace či soukromým datům uživatele musí o to explicitně požádat. To se děje prostřednictvím oprávnění (angl. permissions) staticky deklarovaných v pomocí tagu `<uses-permission>` v souboru manifestu (`AndroidManifest.xml`). Při instalaci aplikace je uživateli zobrazen seznam všech těchto oprávnění, které aplikace požaduje a bez jeho souhlasu nemůže být aplikace nainstalována. Níže je seznam všech oprávnění, které aplikace vyžaduje: [TODO jeste ty pro mapy]

- **ACCESS_FINE_LOCATION**
Povoluje aplikaci přístup k přesné pozici ze zdrojů GPS, vysílačů operátora či Wi-Fi.
- **ACCESS_NETWORK_STATE**
Povoluje aplikaci přístup k informaci o sítích.
- **AUTHENTICATE_ACCOUNTS**
Povoluje aplikaci autentizovat vlastní účty.
- **GET_ACCOUNTS**
Povoluje aplikaci přístup k seznamu účtů v zařízení.
- **INTERNET**
Povoluje aplikaci přístup k síťovým službám.
- **MANAGE_ACCOUNTS**
Povoluje aplikaci přístup ke správě účtu uživatele.
- **READ_SYNC_SETTINGS**
Povoluje aplikaci přístup k číst nastavení pro synchronizaci.
- **READ_SYNC_STATS**
Povoluje aplikaci přístup ke čtení statistik synchronizace.
- **RECEIVE_BOOT_COMPLETED**
Povoluje aplikaci získat oznáení při restartu zařízení.
- **USE_CREDENTIALS**
Povoluje aplikaci přístup ke přihlašovacím údajům uživatele.
- **VIBRATE**
Povoluje aplikaci požadovat vibraci zařízení.
- **WRITE_EXTERNAL_STORAGE**
Povoluje aplikaci přístup ke sdílenému úložišti.
- **WRITE_SYNC_SETTINGS**
Povoluje aplikaci měnit nastavení pro synchronizaci.

6.13 Distribuce aplikace

Vzhledem k tomu, že aplikace je určena pro použití v soukromé společnosti, je základním požadavkem její neveřejná distribuce. Standartní cesta publikováním aplikace na Google play tedy není vhodná a namísto ní jsem zvolil alternativní službu Appkilt Market[2]. Služba Appkilt umožňuje soukromou distribuci aplikace pomocí www odkazu ke stažení aplikace. Mimo to ještě umožňuje automatickou aktualizaci aplikace po vydání nové verze a reportování chyb na účet vývojáře.

Zde je výčet parametrů bezplatné verze služby:

- 500 stažení za měsíc
- 100 chybových reportů za měsíc
- 1 účet administrátor

Verze aplikace

Informace o verzi aplikace je nutná aby bylo možné provádět aktualizace. Verze aplikace se definuje pomocí atributů v souboru manifestu:

- **android:versionCode** Celočíselná hodnota reprezentující verzi aplikačního kódu, relativní k ostatním verzím. Hodnota by měla být zvýšena při každém vydání nové aplikace a díky tomu mohla být provedena aktualizace.
- **android:versionName** Řetězcová reprezentace verze určená pro uživatele.

Aktualizace aplikace

K tomu aby bylo možné využívat funkci automatické aktualizace je nutné přidat následující kód do každé aktivity aplikace, kde má být prováděna kontrola dostupnosti nové verze. Předpokladem je přítomnost potřebné knihovny *AppKiltLib.jar* v projektu.

```
@Override
protected void onPause() {
    super.onPause();
```

```
AppKilt.onUpdateableActivityPause();  
}  
  
@Override  
protected void onResume() {  
    super.onResume();  
  
    AppKilt.onUpdateableActivityResume(this);  
}
```

Podepsání aplikace

Android systém vyžaduje aby všechny instalované aplikace byly digitálně podepsány certifikátem, který patří vývojáři aplikace. Soukromý klíč si vygeneruje sám vývojář pomocí dostupných prostředků. Před vydáním aplikace je nutné podepsat distribuovaný balík tímto soukromým klíčem.

Reportování chyb

K tomu aby bylo možné využívat funkci reportování chyb je nutné přidat do projektu vlastní implementaci třídy *android.app.Application* a v ní volat funkci *AppKilt.init(this, "ID")*, kde řetězc *ID* bude nahrazen skutečným identifikátorem přiděleným aplikaci službou AppKlit. Předpokladem je přítomnost potřebné knihovny *AppKiltLib.jar* v projektu. Nevýhodou je skutečnost, že chybový report je odeslán pouze pokud je zařízení připojeno k síti v momentě kdy dojde k chybě v aplikaci, jinak se tato informace ztratí.

```
public class MyApplication extends Application {  
  
    @Override  
    public void onCreate() {  
        super.onCreate();  
  
        AppKilt.init(this, "ID");  
    }  
}
```

7 Závěr

Seznam zkratek

CRUD	Create, read, update and delete
DML	Data manipulation language
ERP	Enterprise Resource Planning
HTTP	Hypertext Transfer Protocol
IMIS	Integrovaný manažerský informační systém
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
PL/SQL	Procedural Language/Structured Query Language
REST	Representational State Transfer
RPC	Remote procedure call
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery, and Integration
URI	Uniform Resource Identifier
WSDL	Web Services Description Language

Literatura

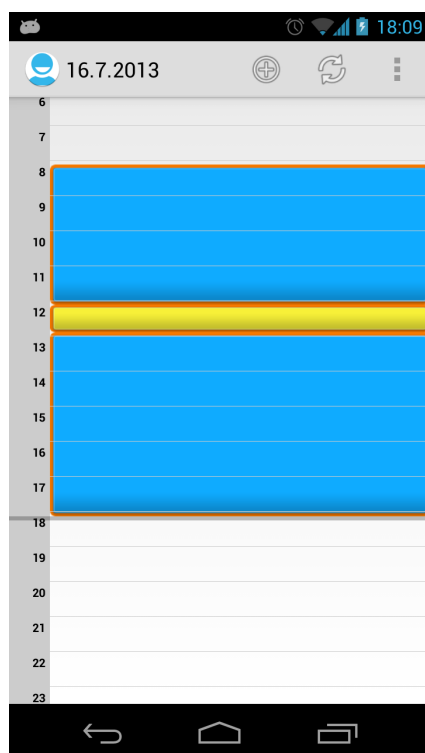
- [1] AChartEngine. [online], [cit. 2013-7-12].
URL <http://www.achartengine.org/>
- [2] AppKilt Market. [online], [cit. 2013-7-12].
URL <http://www.appkilt.com/>
- [3] Google Charts. [online], [cit. 2013-7-12].
URL <https://developers.google.com/chart/>
- [4] Platform Versions. [online], [cit. 2013-7-15].
URL <http://developer.android.com/about/dashboards/index.html>
- [5] 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>
- [6] <http://developer.android.com/>: The Developer's Guide. [online], [cit. 2013-7-2].
URL <http://developer.android.com/reference/android/accounts/AccountManager.html>
- [7] <http://technet.microsoft.com/>: Migrating Oracle Forms. [online], [cit. 2013-7-5].
URL <http://www.oracle.com/technetwork/developer-tools/forms/overview/index.html>
- [8] <http://www.oracle.com/>: Oracle Forms. [online], [cit. 2013-7-5].
URL <http://www.oracle.com/technetwork/developer-tools/forms/overview/index.html>
- [9] Kosek, J.: Využití webových služeb a protokolu SOAP při komunikaci. [online], [cit. 2013-7-6].
URL <http://www.kosek.cz/diplomka/html/websluzby.html>
- [10] Lockwood, A.: ANDROID DESIGN PATTERNS. [online], [cit. 2013-7-10].
URL <http://www.androiddesignpatterns.com/2013/04/retaining-objects-across-co>

-
- [11] OASIS: UDDI. [online], [cit. 2013-7-6].
URL <http://uddi.xml.org/uddi-101>
 - [12] Oracle: Oracle® Database JDBC Developer's Guide. [online], [cit. 2013-7-9].
URL http://docs.oracle.com/cd/E11882_01/java.112/e16548/apxref.htm
 - [13] Oracle: Oracle Database Mobile Server. [online], [cit. 2013-7-7].
URL <http://www.oracle.com/technetwork/products/database-mobile-server/overview>
 - [14] W3C: Web Services Description Language (WSDL) 1.1. [online], [cit. 2013-7-6].
URL <http://www.w3.org/TR/wsdl>

A Uživatelská dokumentace

A.1 Docházka

[TODO daytimeline 2x na stejných datech, casova osa, neukonceno]



Obrázek A.1: Grafický přehled bloků přítomnosti a některé nepřítomnosti

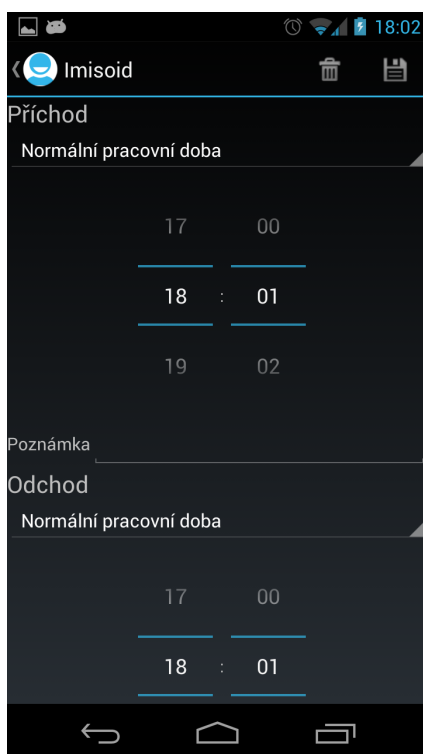


Obrázek A.2: Seznam docházkových událostí pro zvolený den řazených dle času

Hlavní obrazovka aplikace zobrazuje grafický přehled docházkových událostí pro daný den (obr. A.1). Zobrazuje se blok jakékoli přítomnosti a blok vybrané nepřítomnosti (oběd, služebně, lékař). Na obrázku je zobrazena přítomnost v práci od 8-12 hod následována odchodem na oběd ve 12 hod, příchodem z oběda ve 13 hod a následně dosud nukončená přítomnost v práci. Alternativně lze události docházky zobrazit jako seznam jednotlivých událostí (obr. A.2). Každá položka je ohraničena rámečkem jehož barva re-

prezentuje stav synchronizace. Barvy rámečku:

- oranžová - událost dosud nebyla synchronizována se serverem
- zelená - událost byla úspěšně synchronizována se serverem
- červená - při pokusu o synchronizaci se serverem došlo k chybě, chybu lze zobrazit kliknutím na blok přítomnosti (obr. A.1) či událost (obr. A.2)



Obrázek A.3: Obrazovka pro zadání či editaci událostí

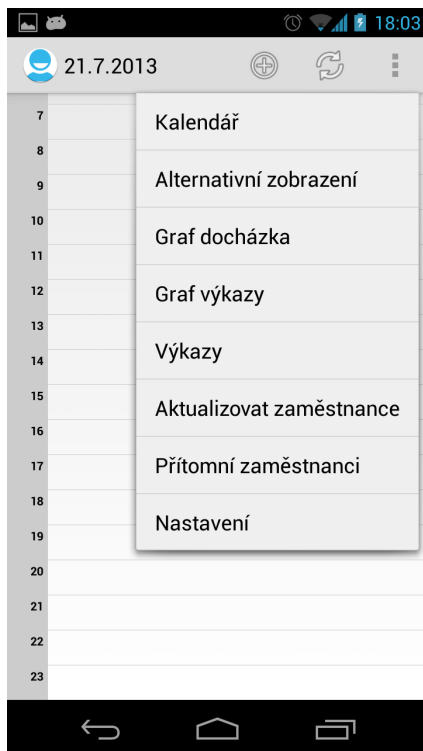


Obrázek A.4: Dialog s volbou smazání vybraných událostí

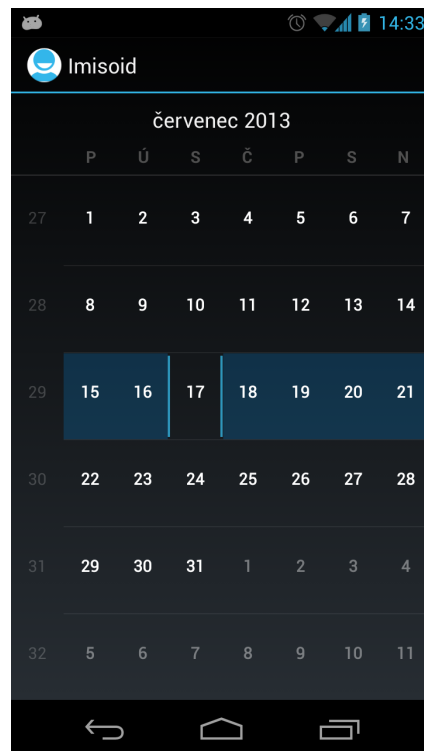
Tlačítkem se znakem '+' (A.1 či A.2) lze zadat novou událost. Na obrazovce A.3 lze zvolit parametry události (čas, typ, poznámka) a rovněž lze události smazat pomocí tlačítka '*popelnice*'. Mazání se potvrzuje pomocí dialogu (A.4). Zadané údaje lze uložit pomocí tlačítka '*disketa*'

A.2 Menu

Pomocí tlačítka v horním pravém rohu ('trojice čtverců' lze zobrazit menu aplikace (obr. A.5).



Obrázek A.5: Nabídka menu



Obrázek A.6: Kalendář

- **Přidat**
přidání události (obr. A.3)
- **Synchronizace**
okamžitá synchronizace událostí docházky se serverem
- **Kalendář**
zobrazí kalendář pro výběr dne pro který se zobrazí docházka (obr. A.6)
- **Alternativní zobrazení**
přepínání mezi obrazovkami A.1 a A.2
- **Graf docházka**
zobrazení grafů a statistik docházky (viz. sekce A.4)
- **Graf výkazy**
podobně jako *Graf docházka*
- **Výkazy**

zobrazení výkazů práce (viz. sekce A.5)

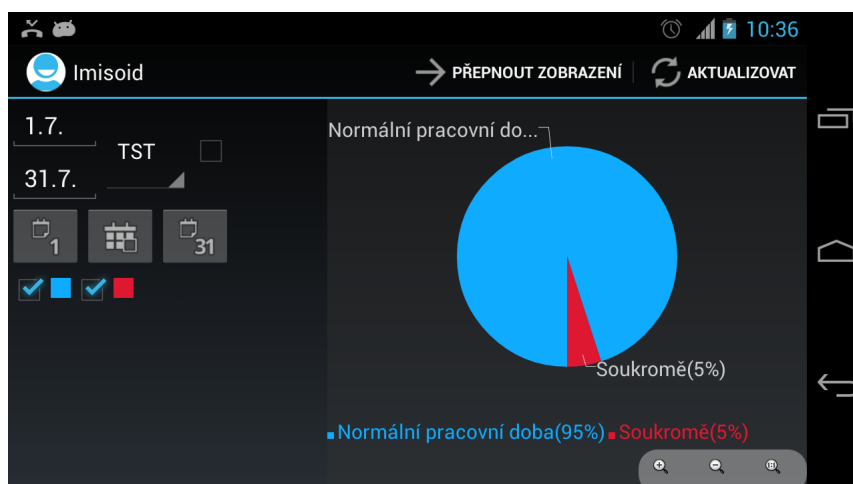
- **Aktualizovat zaměstnance**
aktualizuje seznam zaměstnanců, který se používá pro rozbalovací nabídku s volbou zaměstnance
- **Přítomní zaměstnanci**
zobrazí seznamu zaměstnanců a jejich stav přítomnosti (viz. sekce A.6)
- **Nastavení**
zobrazí možnosti nastavení (viz. sekce A.10)

A.3 Ovládací panel

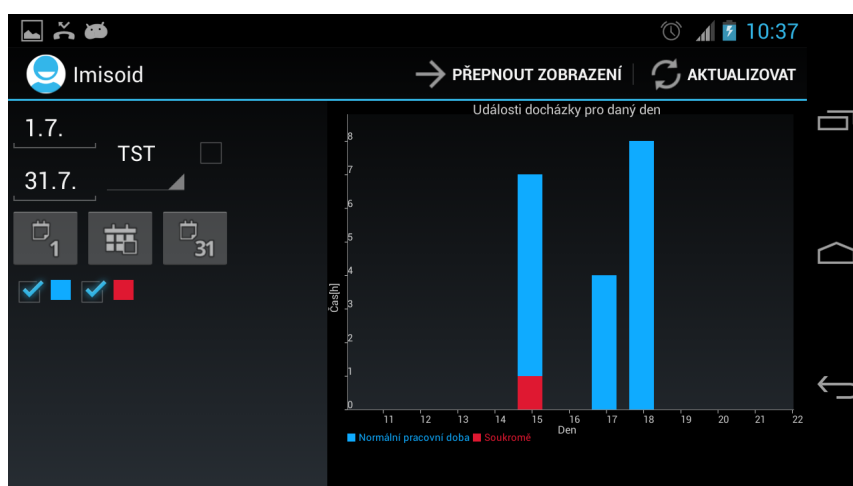
A.4 Grafy s statistiky

Modul pro grafy a statistiky je dostupný jak pro docházku tak pro výkazy. Zobrazuje data pro vybraného uživatele za daného období. Lze prohlížet data uložená offline (od předchozího dotazu) nebo aktuální data pomocí tlačítka 'Aktualizovat'. Mezi jednotlivými obrazovkami lze přepínat pomocí tlačítka 'Přepnout zobrazení'. Ovládací panel pro volbu zaměstnance a období je popsán v sekci A.3

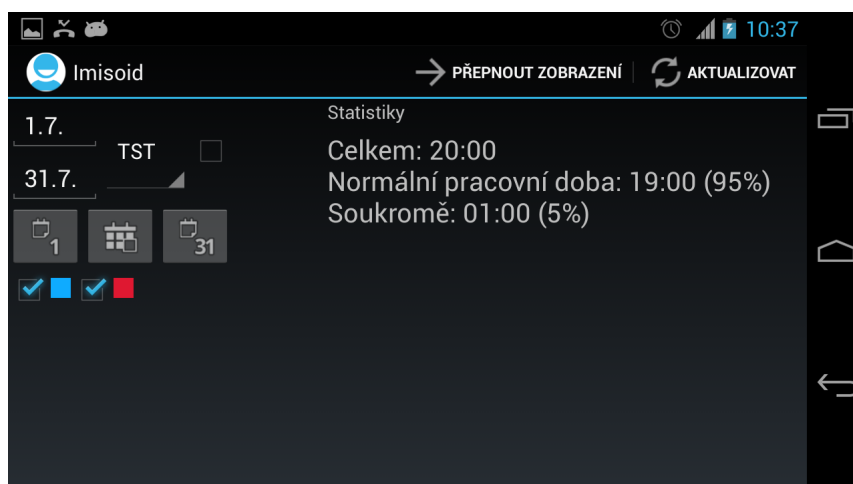
- **Koláčový graf**
zobrazuje poměr jednotlivých typů událostí docházky či typů zakázek (obr. A.7)
- **Sloupcový graf**
zobrazuje vývoj jednotlivých typů událostí docházky či typů zakázek po jednotlivých dnech (obr. A.8)
- **Statistiky**
zobrazuje poměr jednotlivých typů událostí docházky či typů zakázek a součet jejich času (obr. A.9)



Obrázek A.7: Koláčový graf



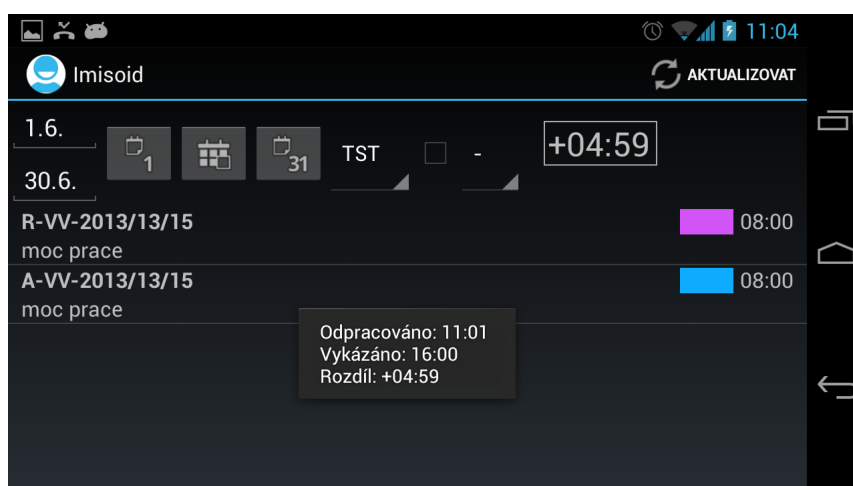
Obrázek A.8: Sloupcový graf



Obrázek A.9: Statistiky

A.5 Výkazy

Obrazovka zobrazuje seznam pracovních výkazů (obr. A.10). Zobrazuje data pro vybraného uživatele za daného období. Lze prohlížet data uložená offline (od předchozího dotazu) nebo aktuální data pomocí tlačítka 'Aktualizovat'. Po kliknutí na záznam lze zobrazit detail výkazu (obr. A.11). V obdélníkovém rámečku v pravo nahoře je zobrazen rozdíl mezi odpracovaným a vykázaným časem. Po kliknutí lze zobrazit detailní informace. Tyto údaje jsou stahovány ze serveru při každém požadavku, neukládají se persistentně na straně Android aplikace.



Obrázek A.10: Seznam pracovních výkazů



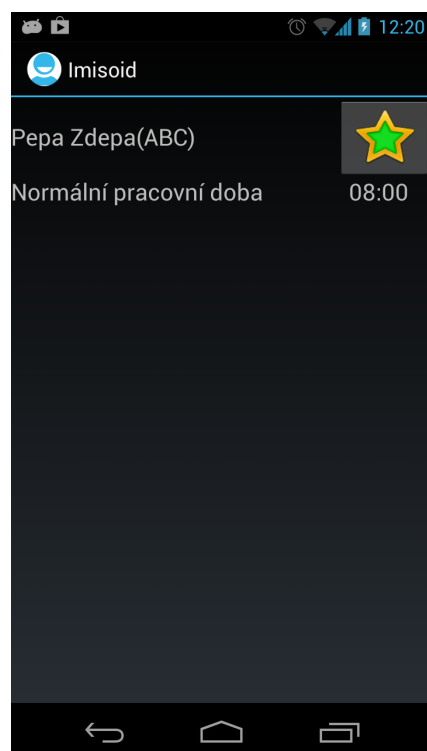
Obrázek A.11: Detail výkazu

A.6 Přítomnost zaměstnanců

Zobrazuje seznam zaměstnanců a jejich stav přítomnosti (poslední docházkovou událost) (obr. A.12). Při řazení záznamů mají přednost přítomní před nepřítomnými a oblíbení před těmi, které si uživatel neoznačil jako oblíbené, dále se řadí se podle abecedních názvů kódů pracovníků. Lze prohlížet data uložená offline (od předchozího dotazu) nebo aktuální data pomocí tlačítka 'Aktualizovat'. Po kliknutí na záznam zaměstnance lze zobrazit jeho detailní profil (obr. A.13). Na profilu zaměstnance lze kliknutí na ikonu 'hvězda' měnit status oblíbenosti.



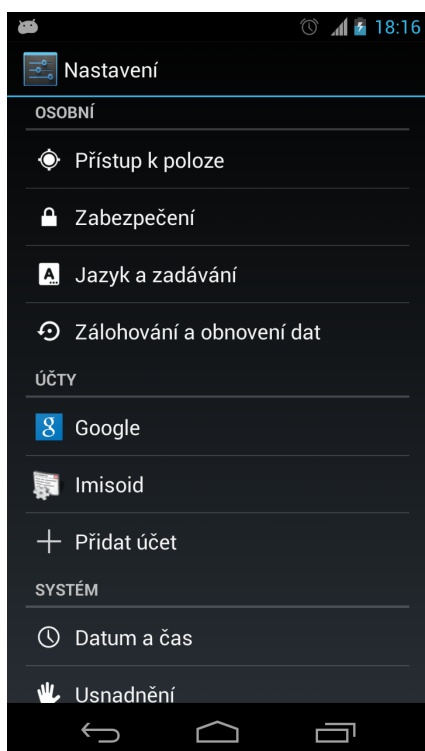
Obrázek A.12: Přítomnost zaměstnanců



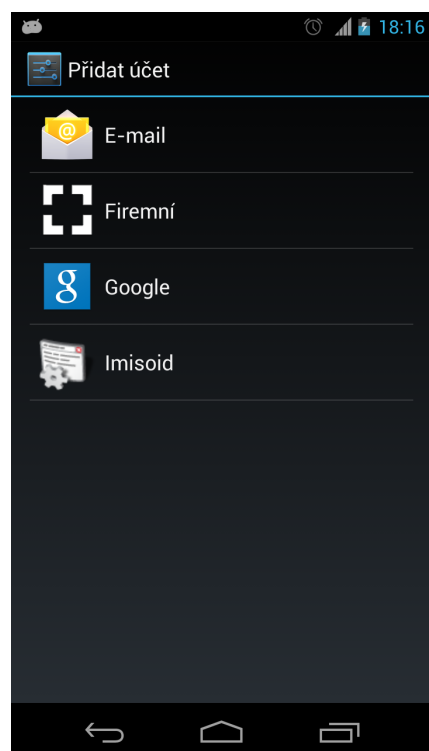
Obrázek A.13: Zobrazení profilu zaměstnance

A.7 Vytvoření účtu

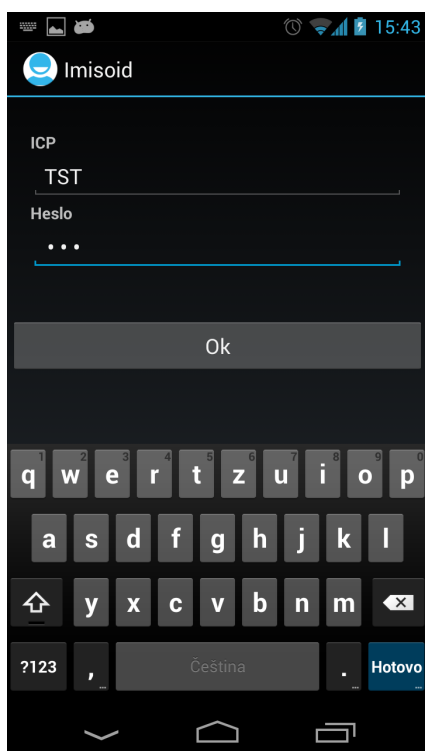
Uživatelský účet lze vytvořit v nastavení Android zařízení v sekci '*Nastavení*' a dále tlačítko '*Přidat účet*' nebo potvrzením dialogu s výzvou k přidání účtu v aplikaci. Po výběru účtu '*Imisoid*' je zobrazena obrazovka (obr. A.16) pro přidání přihlašovacích údajů. Předpokladem je fungující připojení k serveru pro aplikaci A.26). Po úspěšném vytvoření účtu se zobrazí dialog A.17)



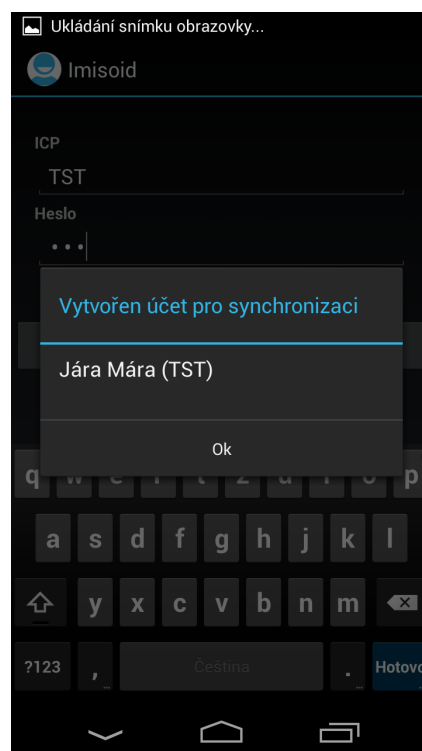
Obrázek A.14: Nastavení



Obrázek A.15: Přidání účtu

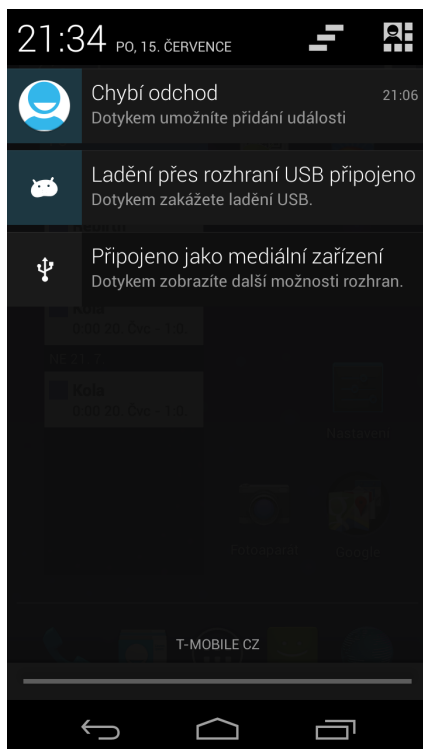


Obrázek A.16: Zadání přihlašovacího údaje

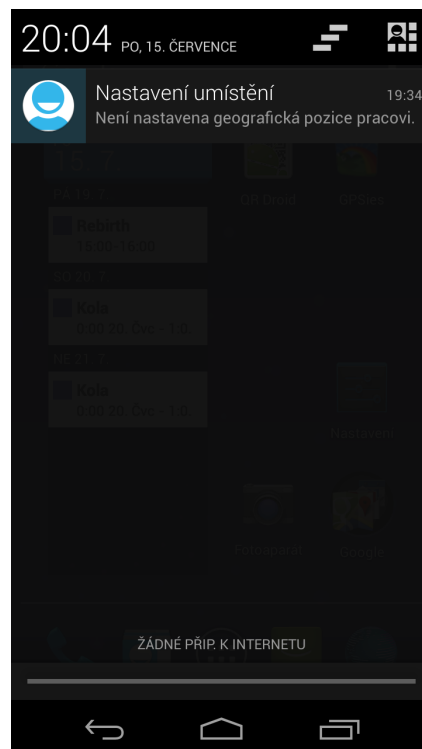


Obrázek A.17: Potvrzení vytvoření účtu

A.8 Notifikace

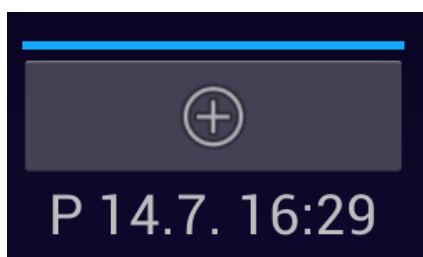


Obrázek A.18

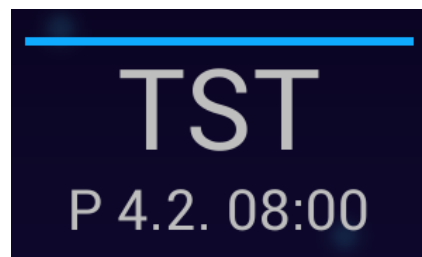


Obrázek A.19

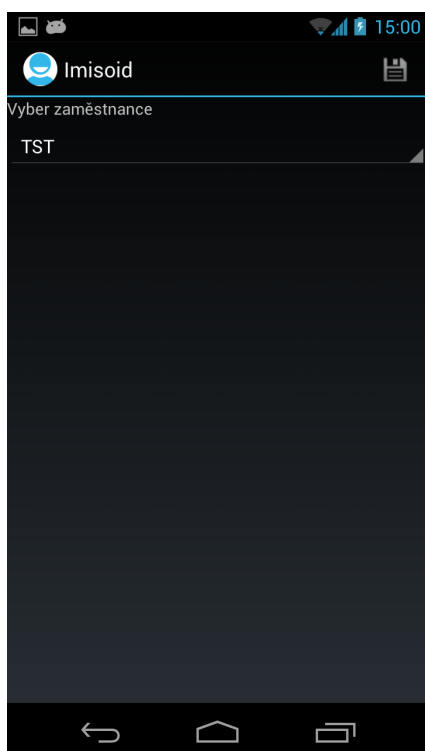
A.9 Widgety



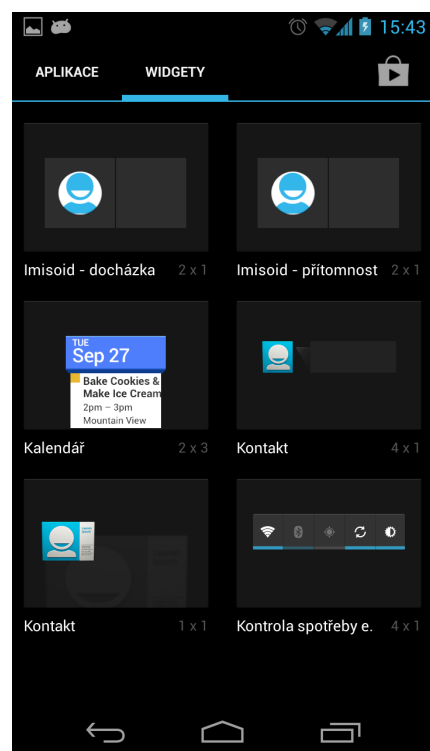
Obrázek A.20: Widget pro zadání docházky



Obrázek A.21: Widget s přítomností zaměstnance

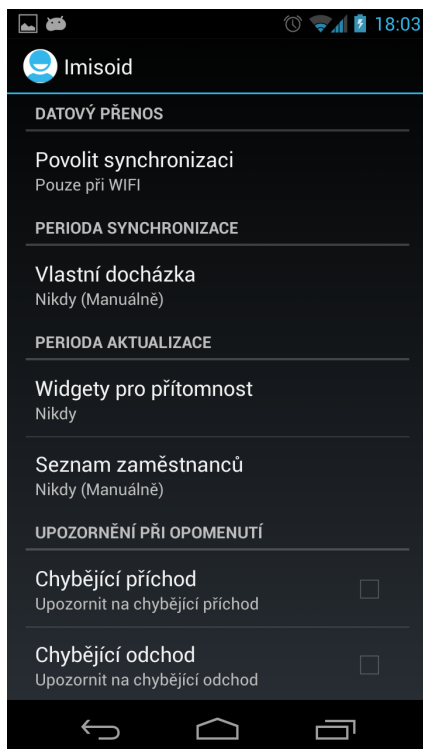


Obrázek A.22: Widget pro zadání docházky

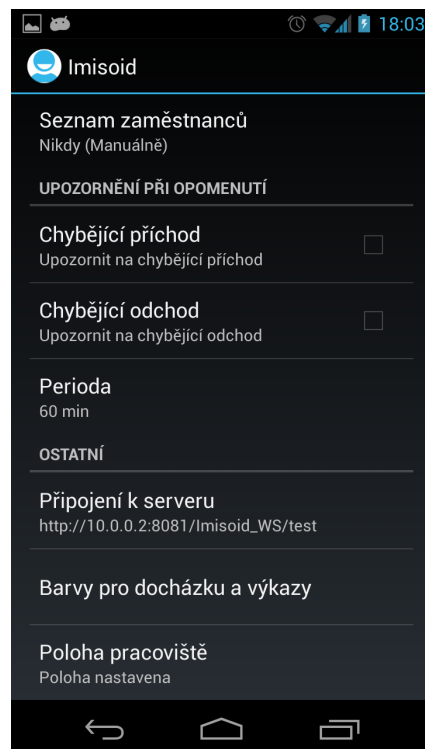


Obrázek A.23: Widget s přítomností zaměstnance

A.10 Nastavení



Obrázek A.24



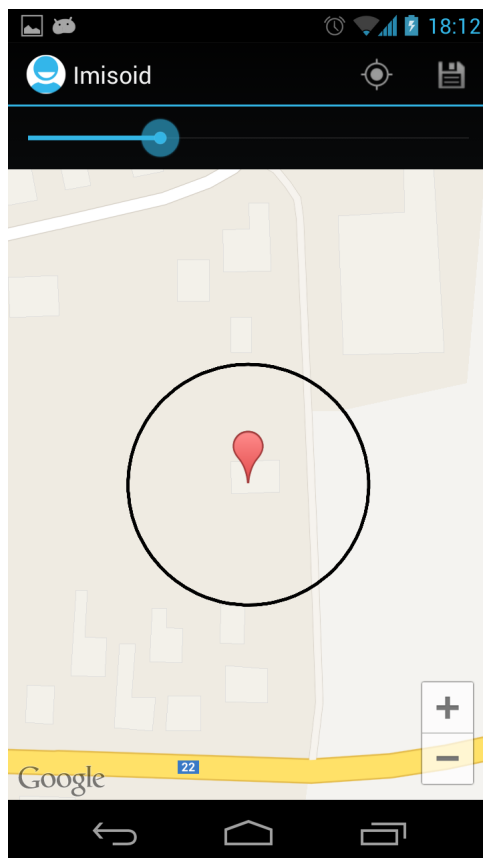
Obrázek A.25

A.10.1 Sít'ové připojení



Obrázek A.26

A.10.2 Poloha pracoviště

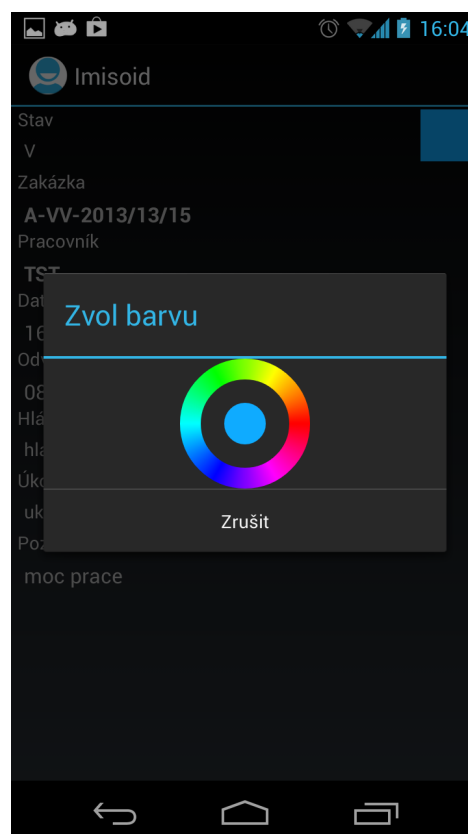


Obrázek A.27

A.10.3 Nastavení barev typů událostí a zakázek



Obrázek A.28



Obrázek A.29

B Manifest?