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

Diplomová práce

Docházka a výkazy práce pro systém IMIS na platformě Android

Plzeň 2013 Martin Kadlec

Prohlášení

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

V Plzni dne 4. července 2013

Maxipes Fík

Abstract

Text of abstract.

Obsah

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

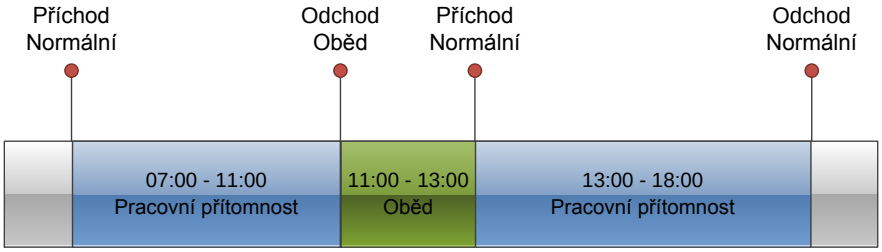
V této práci se zaměřuji na funkcionalitu z oblasti systému věnující se personalistice. Konktrétně se jedná o moduly pro zápis příchodů a odchodů na pracoviště a vykazování provedené práce. Jedná se o činnosti, které zaměstnanec provádí jako každodenní rutinu a zároveň jsou to činnosti s nejširší skupinou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

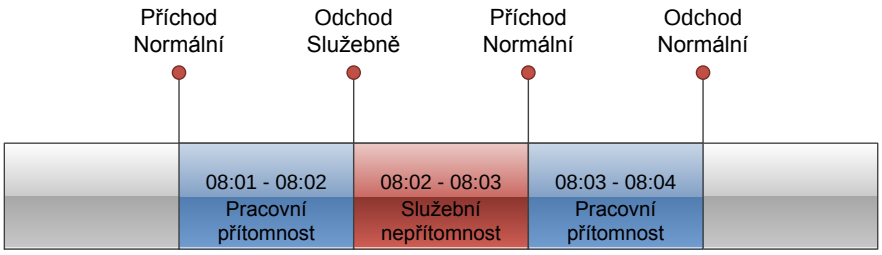
Eviduje se každý příchod i odchod z pracoviště společně s účelem události. Účel události je důležitý, protože délka pracovní přestávky či odchod z pracoviště k lékaři jsou legislativně ošetřené záležitosti.

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 umožnující 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.

[TODO kam? 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]

2.2 Použitá technologie

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

2.2.1 Oracle forms

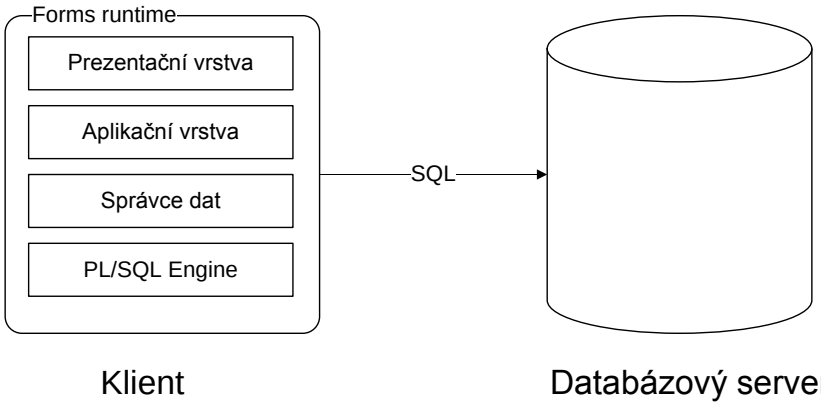
Oracle forms je softwarový produkt vyvinutý společností Oracle. Slouží k vytváření formulářů, které interagují s Oracle databází. Jako programovací jazyk využívá PL/SQL. Produkt byl původně používán jako terminálové rozhraní pro komuikaci se serverem. Později byl přepracován do architektury klient-server.

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. Serverm je myšlel databázový server. Architektura je znázorněna na obr. 3.1



Obrázek 2.3: Klient-server architektura Oracle Forms aplikace

Forms prostředí běhu

- **Prezentační vrstva**
  Zobrazuje informace pro uživatele formou grafického uživatelského rozhraní. Kontrolovat zadávané vstupy.
- **Aplikační logika**
  Stará se o provedení aplikační logiky.
- **Správce dat**
  Stará se o zpracování dat se kterými formulář pracuje. Řídí databázovou transakci.
- **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

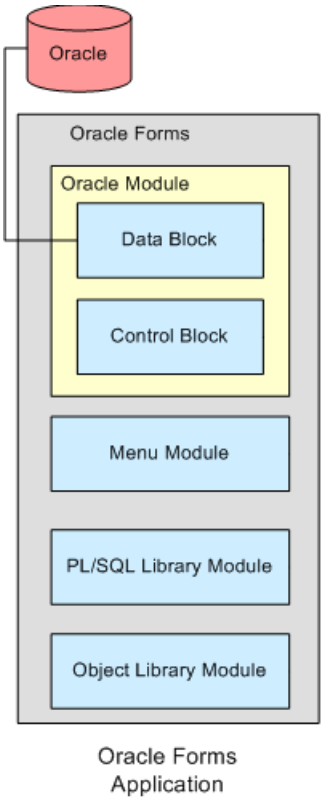
Modul formuláře Modul formuláře je hlavní komponenta aplikace. Poskytuje kód nezbytný pro interakci s úložištěm a uživatelským rozhraním. Data poskytovaná databází jsou reflektovaná v prvcích uživatelského rozhraní jako jsou textová pole, zaškrtávací políčka, přepínače, talčítka atd. Formulář je logicky 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 umožnujě 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 hiearchii 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 fuknce, 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.



Triggery

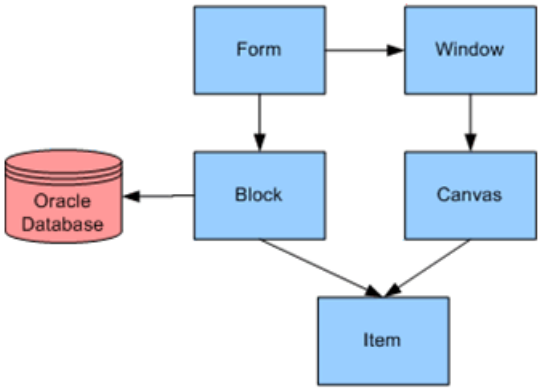
Aplikace v Oracle pracuje s následujícími typy triggerů:

- 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 správné položky v bloku potomka.

- Message-handling triggers - zpracovávájí 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 nestandartní akci při těchto úkonech, provede překrytí těchto triggerů s vlastní definovanou akcí.

2.3 Uživatelské rozhraní



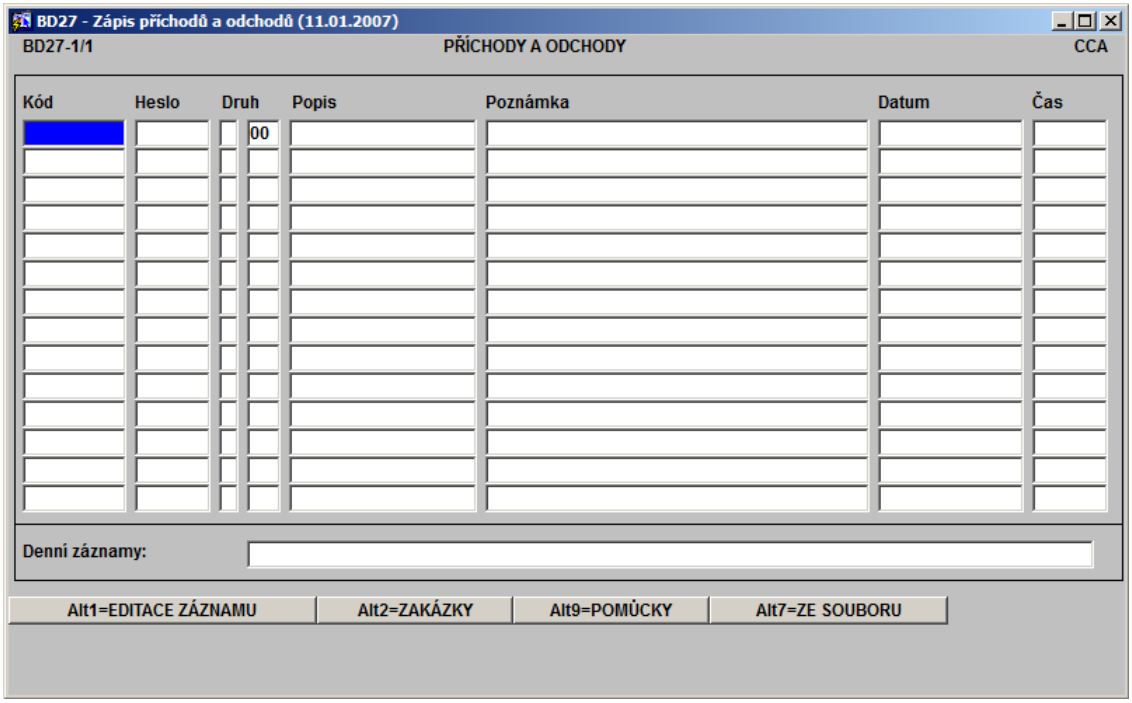
Plátno je objekt, na který je nakresleno celé GUI formuláře, tedy všechny viditelné objekty. Může mít prakticky jakoukoli velikost. Okno ohraničuje plochu plátna, která bude zobrazena. View řídí, jak bude plátno v určité době zobrazeno v okně.
TODO přepsat

Seznam hodnot

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

2.4 Důležité formuláře

2.4.1 Zápis příchodů a odchodů



Obrázek 2.4: Formulář pro zápis příchodů a odchodů

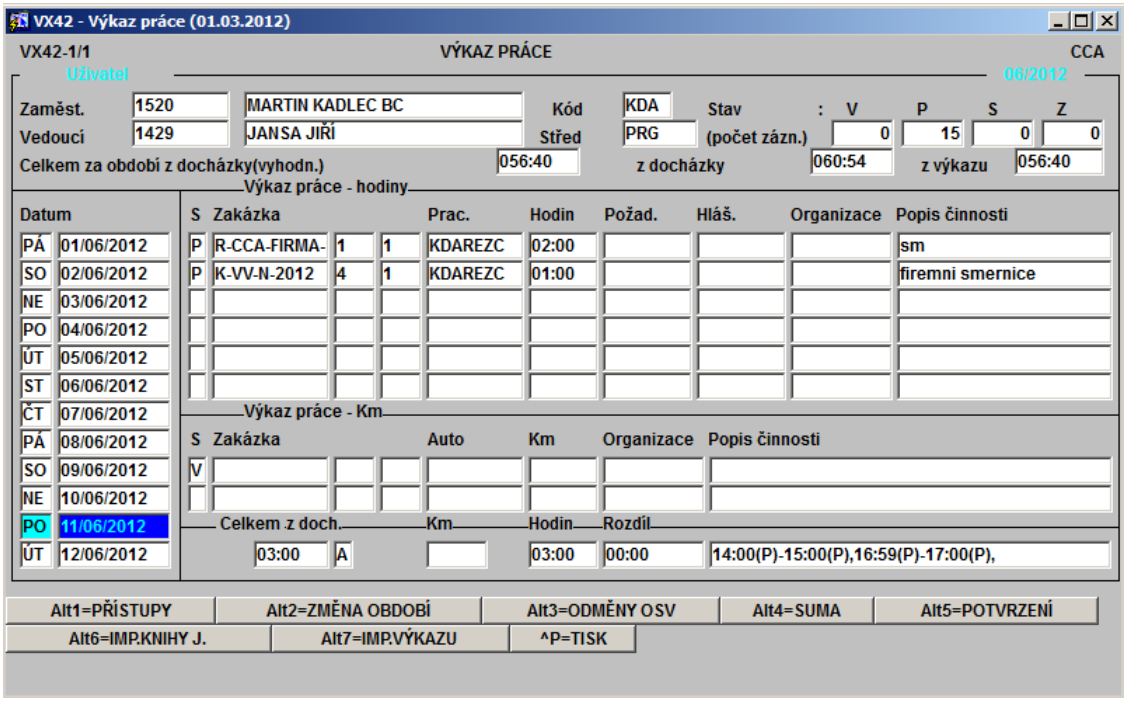
- Položky bloku pro zápis docházky

  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.

2.4.2 Výkazy práce



Obrázek 2.5: Formulář pro pracovní výkazy

[TODO popsat z pohledu uzivatele]

3 Analýza

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 využití webové služby, která by sloužila jako rozhraní mezi klientskou aplikací a databázovým serverem.

3.1.1 Přímé připojení k databázi

Přestože príme 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 funkčnosti 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 je server zajišťující synchronizaci dat mezi Oracle databází a mobilními zařízeními. Klíčovou vlastností tohoto produktu je synchronizační jádro, které je schopné zajistit synchronizaci velké počtu mobilních zařízení se vzdálenou 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 umožuje šifrování dat, jak pro přenos tak i pro jejich persistenci.

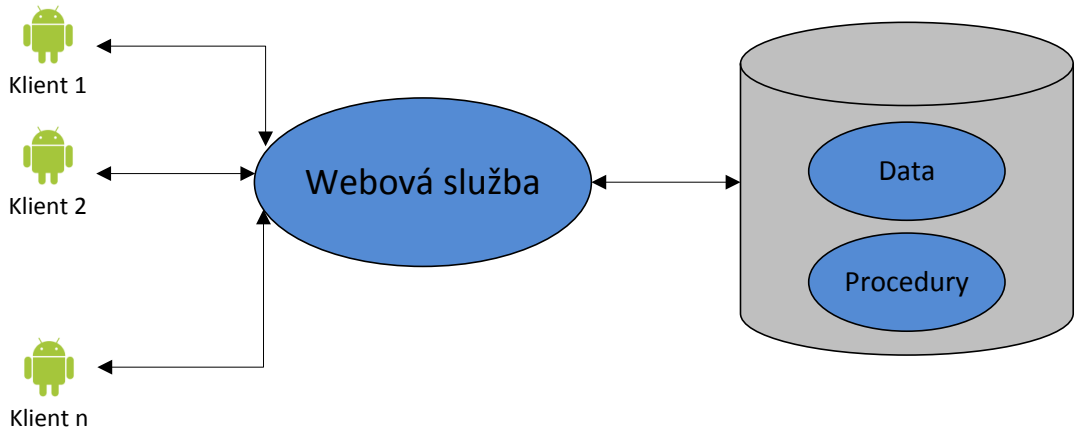
Tato varianta byla zamítnuta protože řeší pouze synchronizaci dat a neumožňuje zajistit provedení business logiky. Dalším důvodem je skutečnost, že její použití by vyžadovalo zakoupení licence pro tento server.

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 platfromy Java, Android, Blackberry, Windows a Linux. (http://www.oracle.com/technetwork/products/dat mobile-server/overview/index.html)

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 minimalizace odezev při zajištění business logiky systému. Mezi klientem a webovou službou se přenášejí pouzy 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 změne logiky bude potřeba úpravy v kódu pouze na straně webové služby. Cílová architektura na vyobrazena na obrázku 3.1



Obrázek 3.1: Zvolená architektura

3.2 Architektura

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 srozumitelneji TODO schema komunikace -HHTP, JDBC

3.3 Výběr typu webové služby

3.3.1 Representational State Transfer

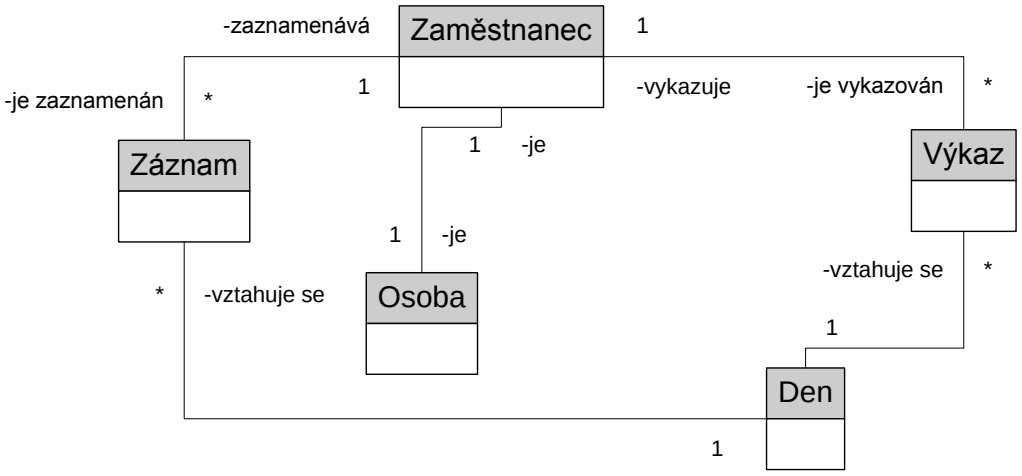
3.3.2 Simple Object Access Protocol

3.4 Datová vrstva

V této kapitole analyzuji datovou vrstvu systému.

3.4.1 Datový model

V následujícím diagramu (obr. 3.2) zobrazuji entity nacházející se v systému, které jsou relevantní pro zkoumanou část systému. Další tabulky a číselníky jsou pro jednoduchost vypuštěny.



Obrázek 3.2: 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 krátky popis události.
- Výkaz
  Pracovní výkaz. Obsahuje informaci o čase, datu, osobě zadavatele, osobě řešitele, vykazované době, vazbu na zakázku či chybové hlášení, stav výkazu, zaměsnancův 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
  Rozlišení pracovních dnů a svátků.

3.4.2 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 deserializace 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

3.4.3 Kritika datové vrstvy

co se mi nelibilo a co bych navrhl jinak a jak, navrh prichody/odchody - jeden radek, chybi primarni klic - ROWID jako unikatni identifikator, problemy ktere to prinasi, format casu - problemy s prevodem

3.5 Business logika

existuje někajá možnost převodu formsů do javy - oracle adf - co to je, co to resi, proc to neresi muj problem
prijde to do webove sluzby - duvody

3.5.1 Triggery

jen ty, jejichž funkčnost bude muset být implementována.

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

3.5.2 Databázové balíčky a uložené procedury

ukázky volání z javy

3.5.3 Forms knihovny

3.6 Synchronizace dat

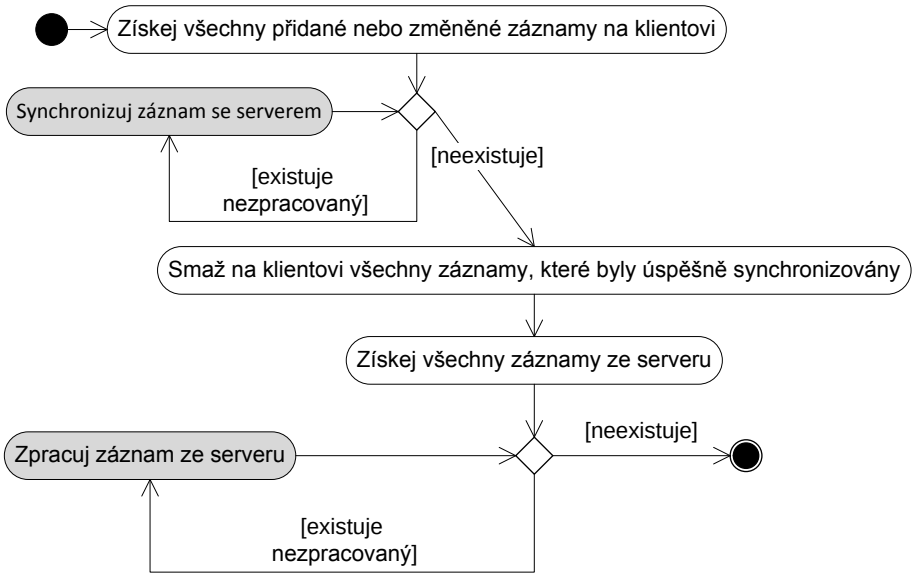
V současném systému uživatel zadává data prostřednictvím příslušného formuláře. Změny jsou aplikovány bezprostředně po uložení během databázové transakce. Mobilní klient přináší nový způsob použití - data lze zadat i v režimu offline, kdy mobilní klient není v dosahu webové služby. Tyto data jsou uložena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

3.6.1 Obousměrná synchronizace

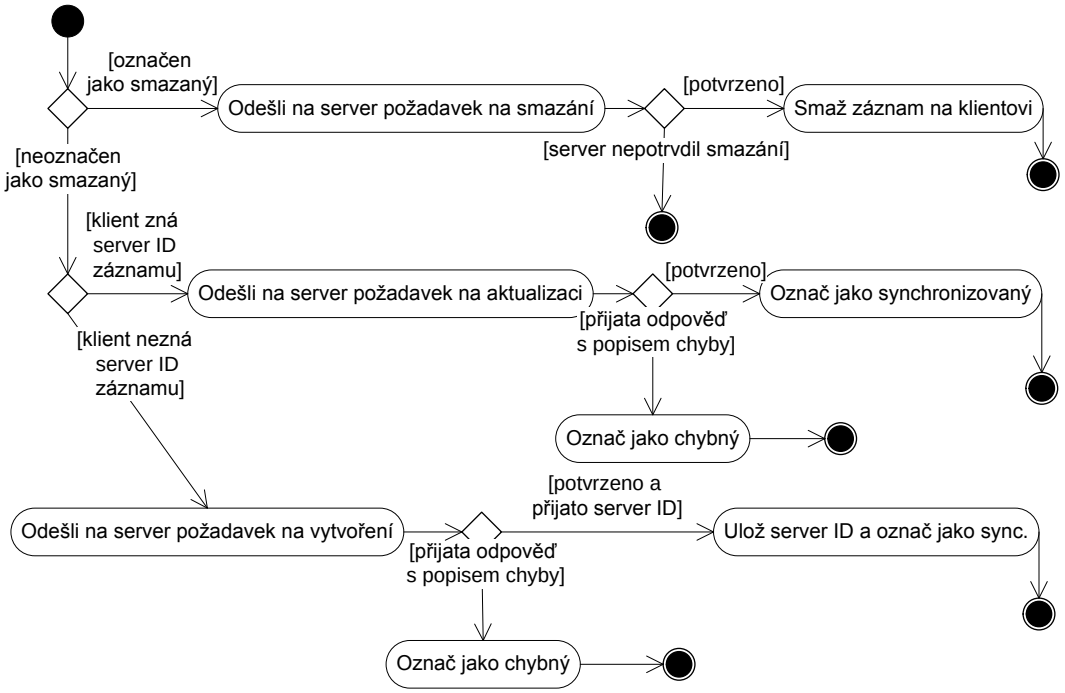
Při obousměrné synchronizaci se odesílají data ze strany klienta na server tak i opačným směrem ze serveru na klienta. Klienta lze navrhnout tak, aby si uchovával informaci o změnách na svojí straně. Při analýze databázového schématu pro docházku v současném systému jsem zjistil, že databáze neuchovává 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 3.3. 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 neúspěsného 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 zůstavají na klientovi, až do doby úspěšného pokusu o synchronizaci.



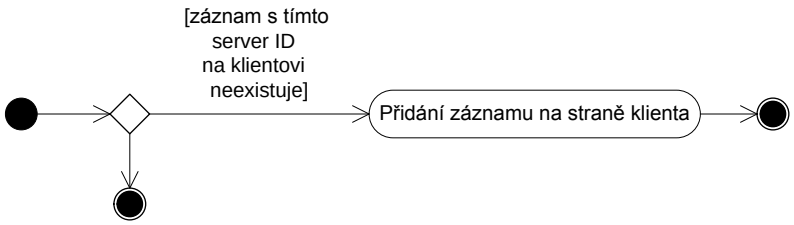
Obrázek 3.3: Diagram aktivit pro průběh synchronizace

Diagram 3.4 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 3.4: Diagram aktivit pro odeslání požadavku na server

Diagram 3.4 podrobněji rozepisuje průběh přijetí záznamu ze serveru. TODO popsat kolizi



Obrázek 3.5: Diagram aktivit pro přijetí záznamu ze serveru

3.6.2 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 přížnak 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.

3.6.3 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 změne dat, tak mobilní klient vždy přepíše záznam na serveru.

3.6.4 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.

3.7 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.

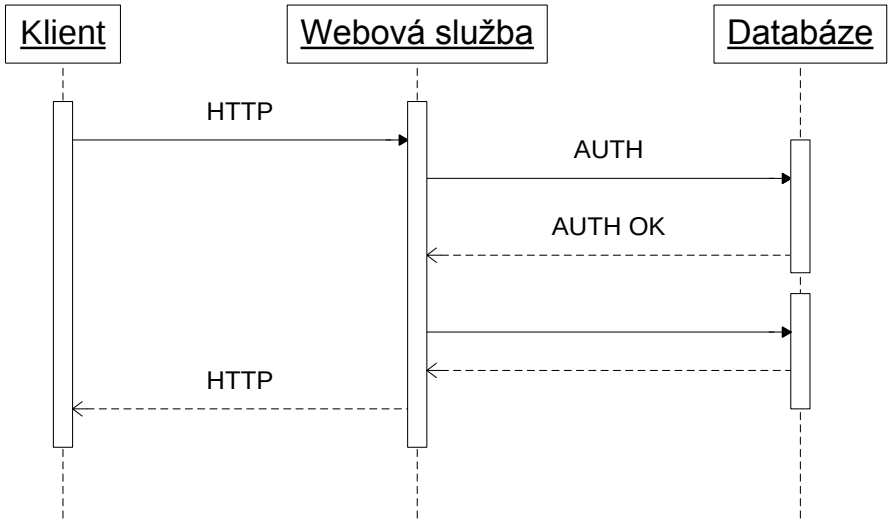
Vlastnosti připojení VPN

- 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

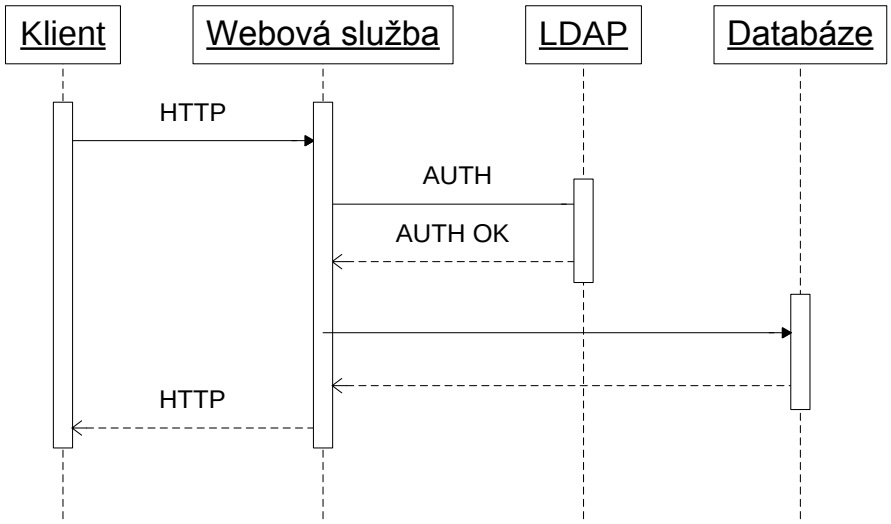
Klient i server jsou součástí jedné VPN sítě, která zajišťuje důvěryhodný komunikační kanál. Klient posílá na server HTTP požadavek jehož součástí je autentizační hlavička nesoucí uživatelské jméno a heslo. Webová služba ověří jméno a heslo pomocí databázové procedury. Jméno a heslo uživatele je uloženo 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 aktualizačních 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 vyhedávání
- attributes - seznam atributů
- scope - specifikuje vyhledávácí 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. Stávají 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

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 zaměstnaců 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: • icp - identifikátor zaměstnance • from - datum začátku období • to - datum konce období
DELETE	events/{rowid}
	Smaže danou událost Parametry: • rowid - identifikátor události
POST	events
	Vytvoří událost, používá se bez parametrů protože identifikátor pro událost vytváří server
PUT	events/{rowid}
	Aktualizuje danou událost Parametry: • rowid - identifikátor události
GET	events/sum/{icp}?from={from}&to={to}
	Získá součet přítomnosti zaměstnance za dané období Parametry: • 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 zaměstnaci identifikovém pomocí parametru Parametry: • icp - identifikátor zaměstnance
GET	employees/all/{icp}
	Získá seznam všech zaměstnanců, kteří jsou aktuálně v zaměstnaneckém poměru, obsahuje informaci zda jsou tito zaměstnanci podřízení, vzhledem k zaměstnanci identifikovém pomocí parametru Parametry: • 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 identifikovém 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: • 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: • 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

6 Android aplikace

6.1 Funkcionalita

[TODO ještě upřesnit] 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 synchronzaci 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 kompoment, 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

6.2 Komponenty Android aplikace

6.2.1 Aktivita

Aktivita představuje jednu obrazovku s uživatelským rozhraním. Každá akvitita 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 uživatelovi 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 provedením *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. modifilkace 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.

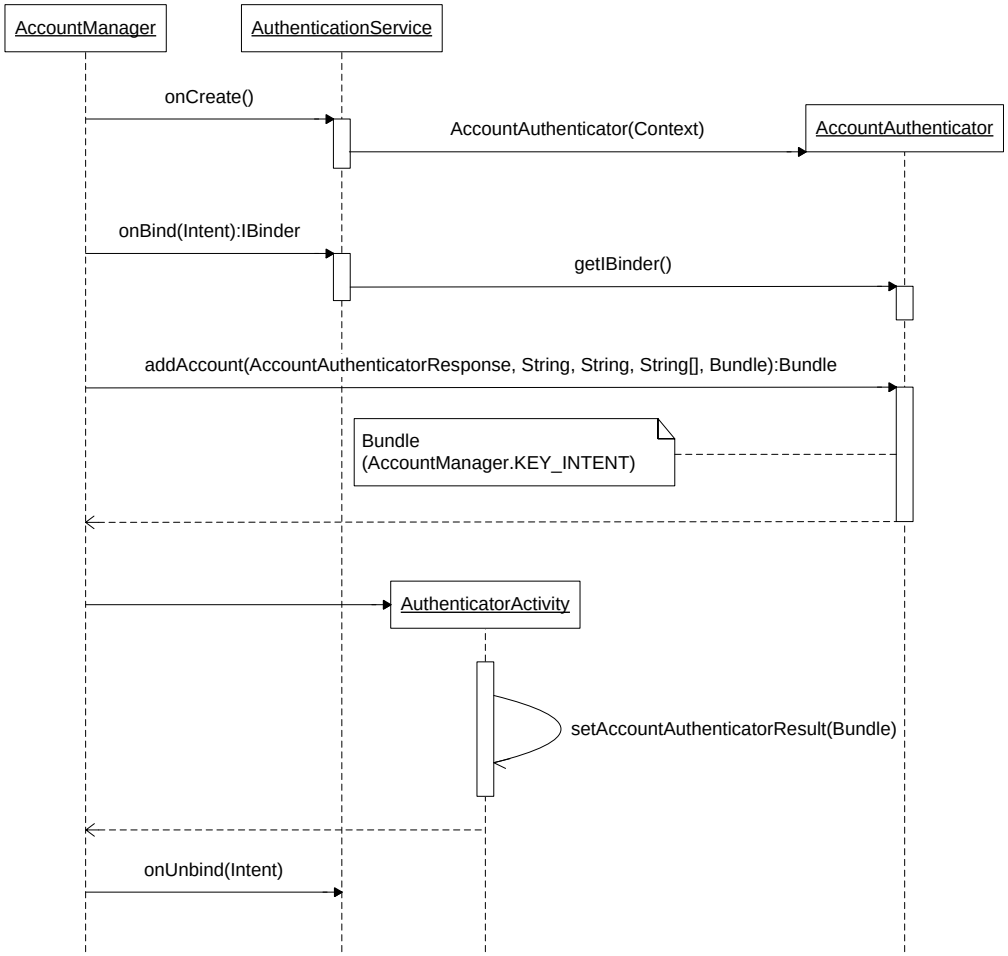
Různé online služby mohou využívat odlišný způsob pro autentizaci. Android manažer účtů využívá kompomentu *authenticator* [1], která je obvykle poskytována třetí stranou - poskytovatelem dané služby. Příkladem služby, která poskytuje vlastní *authenticator* je např. Google, Facebook a Microsoft

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.

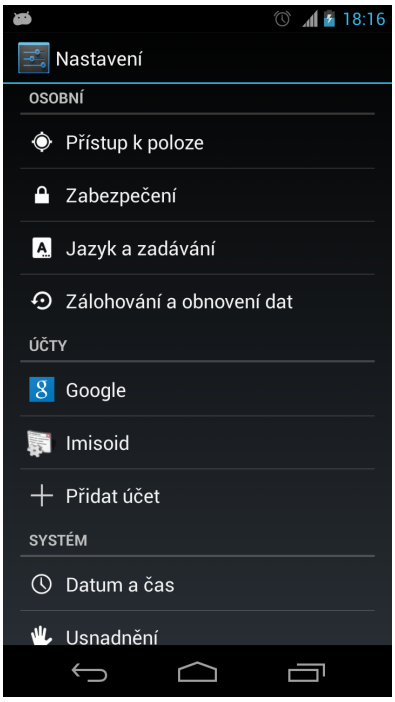
Na diagramu 6.1 je zobrazen průběh vytvoření účtu, který je použit v implementované aplikaci. Jakmile uživatel v sekci *Nastavení->Účty* (obr. 6.2) zvolí přidání nového účtu a následně zvolí typ účtu (obr. 6.3), manažer účtů spustí službu *AutheticationService*. Služba vytvoří instanci třídy *AccountAuthenticator - authenticator* komponenty. *AccountManager* následně volá metodu *addAccount()*, která zkontroluje zda již účet daného typu existuje. Pokud ne, vrací 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).



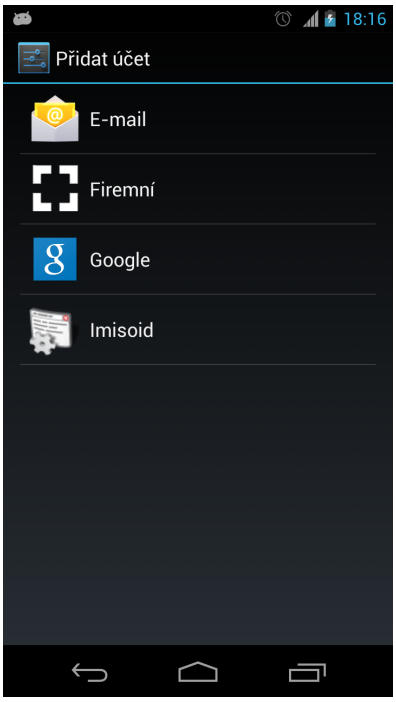
Obrázek 6.1: Sekvenční diagram zobrazující průběh vytvoření účtu

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.



Obrázek 6.2: Nastavení



Obrázek 6.3: Přidání účtu

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 transkační 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 mohou být editována i mimo aplikaci.

SQLite databáze

Data lze ukládat persistentně pomocí SQLite databáze. Vytvořená databáze je dostupná jakkékoli třídě v aplikaci, ale není přístupná mimo aplikaci, která jí vytořila. SQLite databázi detailněji popisuji 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

RestaTemplates - springframework

1. REST operace - davkove vs jednotlive

2. REST, tabulka URI,

6.8 Struktura projektu

(jen ty pouzite)

6.9 Manifest + oprávnění

persmission v manifestu, vypsat 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 testovaci scenare synchronizace]

7.1 O čem psát...

1. pripraveno webove sluzby na dalsi mobilni platformy
2. cinnost apliakce 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

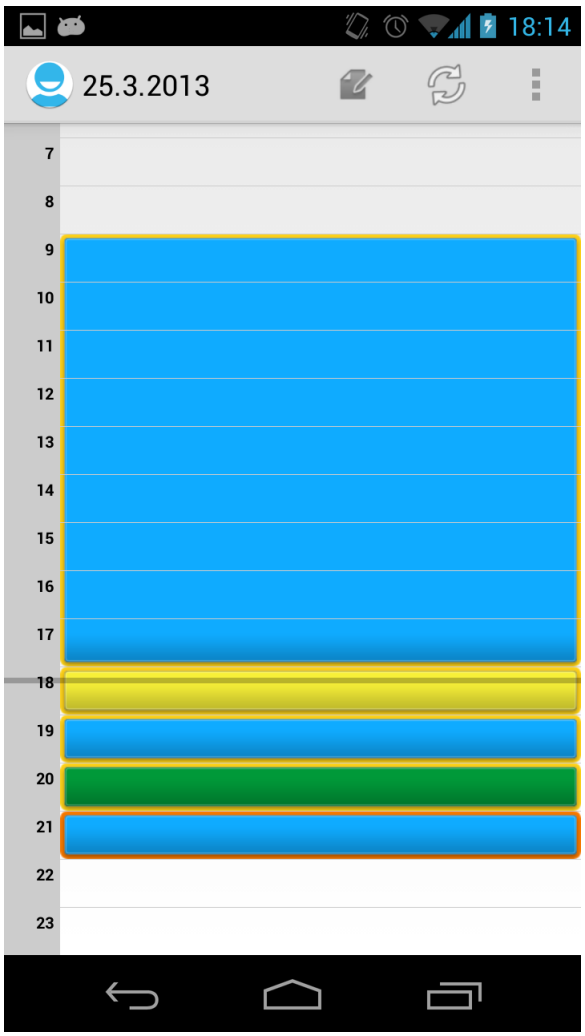
Seznam zkratek

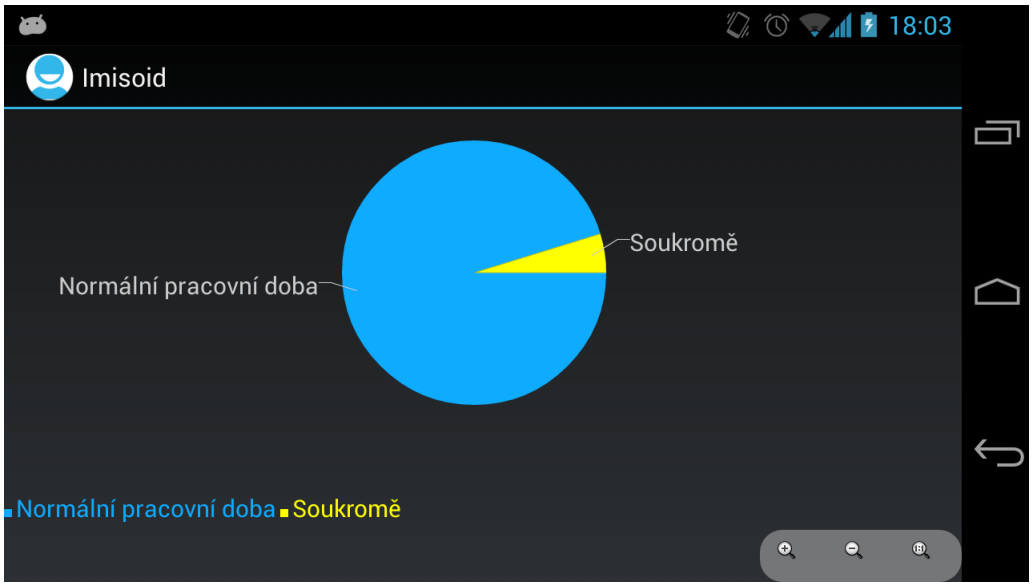
IMIS	Integrovaný manažerský informační systém
ERP	Enterprise Resource Planning
PL/SQL	Procedural Language/Structured Query Language

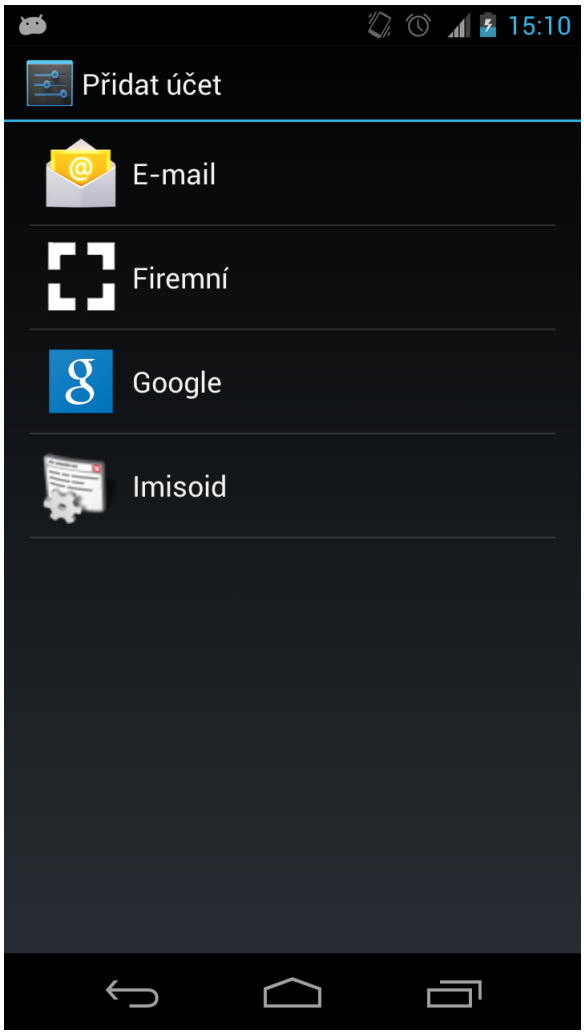
Literatura

[1] http://developer.android.com/: The Developer's Guide. [online], [cit. 2013-7-2].
URL http://developer.android.com/reference/android/accounts/AccountManager.ht

A Uživatelská dokumentace




18:03
Imisoid
Soukromě
Normální pracovní doba
Normální pracovní doba
Soukromě


15:10
Přidat účet
E-mail
Firemní
Google
Imisoid


18:15
Imisoid
Síť
Host
10.0.0.1
Port
8081
Dostupnost
WS
DB
Test spojení

B Manifest?