

Sem vložte zadání Vaší práce.



ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE  
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ  
KATEDRA SOFTWAREVÉHO INŽENÝRSTVÍ



Bakalářská práce

## **Systém osobního plánování na základě metodiky Getting Things Done**

*Michal Sláma*

Vedoucí práce: Ing. Jiří Mlejnek

13. května 2015



---

## Poděkování

Děkuji vedoucímu své práce za hodnotné rady a své rodině za podporu při vytváření této práce.



---

## Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. května 2015

.....

České vysoké učení technické v Praze  
Fakulta informačních technologií

© 2015 Michal Sláma. Všechna práva vyhrazena.

*Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.*

### **Odkaz na tuto práci**

Sláma, Michal. *Systém osobního plánování na základě metodiky Getting Things Done*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.



---

## Abstrakt

Bakalářská práce obsahuje analýzu osobního plánování na základě metodiky GTD[1]. Srovnává existující řešení, navrhuje vlastní systém a hledá pro něj vhodné technologie. Implementuje aplikaci pro centrální uložení dat vystavující REST API a mobilní aplikaci pro OS Android na principu tenkého klienta. Zároveň navrhuje a implementuje publikaci dat na Facebook a Google kalendář. Pro nasazení aplikace instaluje server na portálu DigitalOcean.

**Klíčová slova** webová aplikace, mobilní aplikace, osobní plánování, spring, android, REST API, Facebook, Google kalendář, DigitalOcean

---

## Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

**Keywords** web application, mobile application, personal planning, spring, android, REST API, Facebook, Google Calendar, DigitalOcean



---

# Obsah

<b>Todo list</b>	<b>1</b>
<b>Úvod</b>	<b>3</b>
<b>1 Cíl práce</b>	<b>5</b>
<b>2 Analýza metodiky <i>GTD</i></b>	<b>7</b>
2.1 Metodika GTD - Mít vše hotovo . . . . .	7
2.2 Současné implementace . . . . .	11
2.3 Shrnutí analýzy metodiky <i>GTD</i> . . . . .	15
<b>3 Návrh</b>	<b>17</b>
3.1 Rozdělení aplikace . . . . .	17
3.2 Facebook a Google kalendář . . . . .	18
3.3 Technologie . . . . .	21
3.4 Specifikace REST API . . . . .	26
3.5 Databázový model . . . . .	26
<b>4 Realizace</b>	<b>29</b>
4.1 Aplikace pro centrální uložení dat . . . . .	30
4.2 Mobilní aplikace pro Android . . . . .	41
4.3 DigitalOcean - instalace . . . . .	54
4.4 GitHub . . . . .	54
<b>5 Testování aplikace</b>	<b>55</b>
<b>6 Návrhy k rozvoji</b>	<b>57</b>
<b>Závěr</b>	<b>59</b>
<b>Literatura</b>	<b>61</b>

A Seznam použitých zkratek	69
B Obsah přiloženého CD	71

---

## Seznam obrázků

2.1	Digram zpracování schránky dle metodiky <i>GTD</i> [2] . . . . .	9
2.2	Podoba webové aplikace Toodledo.[3] . . . . .	12
2.3	Mobilní aplikace Doit.[?] . . . . .	13
2.4	Wunderlist [4] . . . . .	14
3.1	Model nasazení GTD (šedivě jsou zbarvena úvaha dalšího rozšíření)	18
3.2	Webové prostředí pro návrh REST API ve specifikaci RAML od firmy Mulesoft . . . . .	28
4.1	Databázový model . . . . .	36
4.2	Registrace aplikace na Facebook . . . . .	47
4.3	Registrace aplikace na Google . . . . .	50
5.1	Projekt SOAP UI pro testování RESP API . . . . .	55



---

## Seznam tabulek

2.1	Porovnání existujících implementací metodiky <i>GTD</i> - (* značí placenou verzi) . . . . .	15
3.1	Rozdíly mezi SOAP a REST . . . . .	21
3.2	Funkce HTTP metod . . . . .	26
3.3	Rozhraní REST API přístupné s tokenem . . . . .	27
3.4	Rozhraní REST API přístupné bez tokenu . . . . .	27
4.1	Současné nastavení hostingu na serveru DigitalOcean tokenu . . . .	54





---

## Todo list

citace zdroje . . . . .	23
verze knihoven . . . . .	29
co s ukážeme u android aplikace . . . . .	29
blbe sekce . . . . .	30
blbe reference . . . . .	41
na čem je nasazeno, kolik lidí to používá a další rozvoj . . . . .	60



---

# Úvod

*„Ze všeho nejvíce požírá čas a energii neustále neproduktivní přemýšlení o věcech, které musíme udělat.“*

Kerry Glesson

V současném informačním věku naše práce přestává být fyzickou. Práce už není ohraničena jasnými hranicemi a existuje velké množství informačních zdrojů použitelných pro její zpracování. Řešitel je tím vystaven tlaku nejasného zadání i rozsahu a musí si sám určit cíle své práce.

Naše mysl nejasné nepřijímá, jakékoliv takové úkoly nám neustále připomíná, vyžaduje se jejich řešení a znemožňuje nám tím koncentraci. Potřebujeme nový přístup. Přístup, abychom naši mysl dokázali přesvědčit, že máme vše jasné a právě teď se můžeme koncentrovat pouze na aktuální úkol.

Jedním z takových přístupů je metodika vytvořená Davidem Allenem *GTD* [1]. Jeho motto *„Jak zvládnout práci i život a cítit se při tom dobře“* je mým kýženým životním cílem.

Moje práce se zabývá rozšířením návrhu a implementace informačního systému pro osobní plánování založeném na metodice *GTD*, které jsem zpracovával v rámci týmového projektu SP2. Provedu analýzu samotné metodiky, navrhnu vhodnou infrastrukturu splňující potřebná kritéria *GTD* [1], implementuji aplikaci pro centrální uložení dat a mobilní aplikaci pro uživatele.



---

## Cíl práce

Prvním cílem práce je analyzovat metodiku *GTD* [1]. Porovnat existující implementace osobního plánování podporující *GTD* [1]. Navrhnout vlastní informační systém.

Vytvoření aplikaci poskytující WS pro centrální uložení dat navazující na práci v předmětu SP2. Vybrat nejvhodnější typ webových služeb. Pro tuto aplikaci najít potřebný hosting a na něm provést konfiguraci aplikačního a databázového serveru.

Implementace mobilní aplikace pro OS Android využívající WS služeb centrálního uložení dat a fungující na principu tenkého klienta.

Prozkoumat postupy pro práci s API *Facebooku* a *Google kalendáře* s cílem implementovat publikaci úkolu na zeď Facebooku a do Google kalendáře. S tím je spojena integrace přihlášení uživatele pomocí účtů těchto systémů.

Cílem práce není vytvoření odladěné aplikace, ale získání know-how s vývojem takového systému.



## Analýza metodiky *GTD*

V této kapitole se zabývám analýzou metodiky *GTD*, hledám její základní stavební prvky a její přínos pro osobní plánování. Definuji důležité pojmy a postupy pro využití v dalším návrhu aplikace. Dále srovnávám existující implementace osobního plánování a hledám jejich výhody, kterými podporují kvalitnější a přehlednější osobní plánování. Na závěr spojuji výsledky obou předchozích bodů a navrhuji základní strukturu a funkce budoucí aplikace.

### 2.1 Metodika *GTD* - Mít vše hotovo

Kniha *GTD* [1] vznikla v roce 2001 a jejím autorem je David Allen. Samotná metodika nepředstavuje jasné řešení, podle kterého se můžeme okamžitě začít řídit a změnit tím svůj život. Snaží se vysvětlit obecné zákonitosti lidské mysli a najít takových způsob plánování, který s ní bude v souladu.

Pojďme se na metodiku podívat podrobněji.

#### 2.1.1 Změna práce

Naše práce přestává být fyzikou a přesunuje se do roviny sběru informací, jejich zpracování a vytvoření závěrů. Zároveň nejsme schopni jednoznačně určit prioritu, protože ta se stává proměnnou závislou na mnoha vnějších i vnitřních faktorech. Naše práce přestává mít jasné hranice, které naše mysl potřebuje. Bez jasných hranic nám mysl bude takové věci neustále sama připomínat a požadovat jejich řešení. Paradoxem je, že naše mysl v tomto ohledu nejedná rozumně a připomíná nám naše nedořešené věci bez ohledu na vhodnost situace. Při takovém stavu si říkáme „nemohu se soustředit“, mysl nám ruší naši vlastní koncentraci. A koncentrace je nutnou podmínkou pro vysokou výkonnost a my potřebujeme aktuálnímu úkolu věnovat 100% pozornosti.

Jak toho ale dosáhnout? Potřebujeme naši mysl přesvědčit, že o všech úkolech víme, jsou zaznamenány mimo ní, víme přesně co máme dělat a není riziko zapomenutí.

### 2.1.2 Produktivita

*„Ve znalostní práci není úkol dán, je třeba jej určit. Jaký je očekávaný výsledek práce? Je klíčovou otázkou pro produktivitu znalostních pracovníků. Je to otázka, která si žádá riskantního rozhodnutí. Obvykle na ni neexistuje správná odpověď, místo ní jsou tu možnosti. A má-li být dosaženo produktivity, je třeba jasně specifikovat očekávané výsledky.“*

Petr Ducky[1]

Mysl potřebuje jasný cíl, aby mohla udržet potřebné soustředění, nepřemohla ji únava a dokázala rozlišit důležité informace pro zapamatování.

*Co udělat při zpracování úkolu:*

1. vyjasnit si požadovaný výsledek
2. mít jasno o dalším kroku
3. uložit si úkol do důvěryhodného systému

Začít je třeba od nezákladnější věci. To nám pomůže zvládnout a odpoutat se od každodenních závazků a poté se můžeme lépe soustředit na větší projekty a vize. Při zpracování potřebujeme využít kontroly ve dvou rovinách: horizontální a vertikální. V horizontální přemýšlíme, co všechno potřebujeme udělat. Ve vertikální řešíme konkrétní věc a co je potřeba vykonat k jejímu splnění.

Typický horizontální postup:

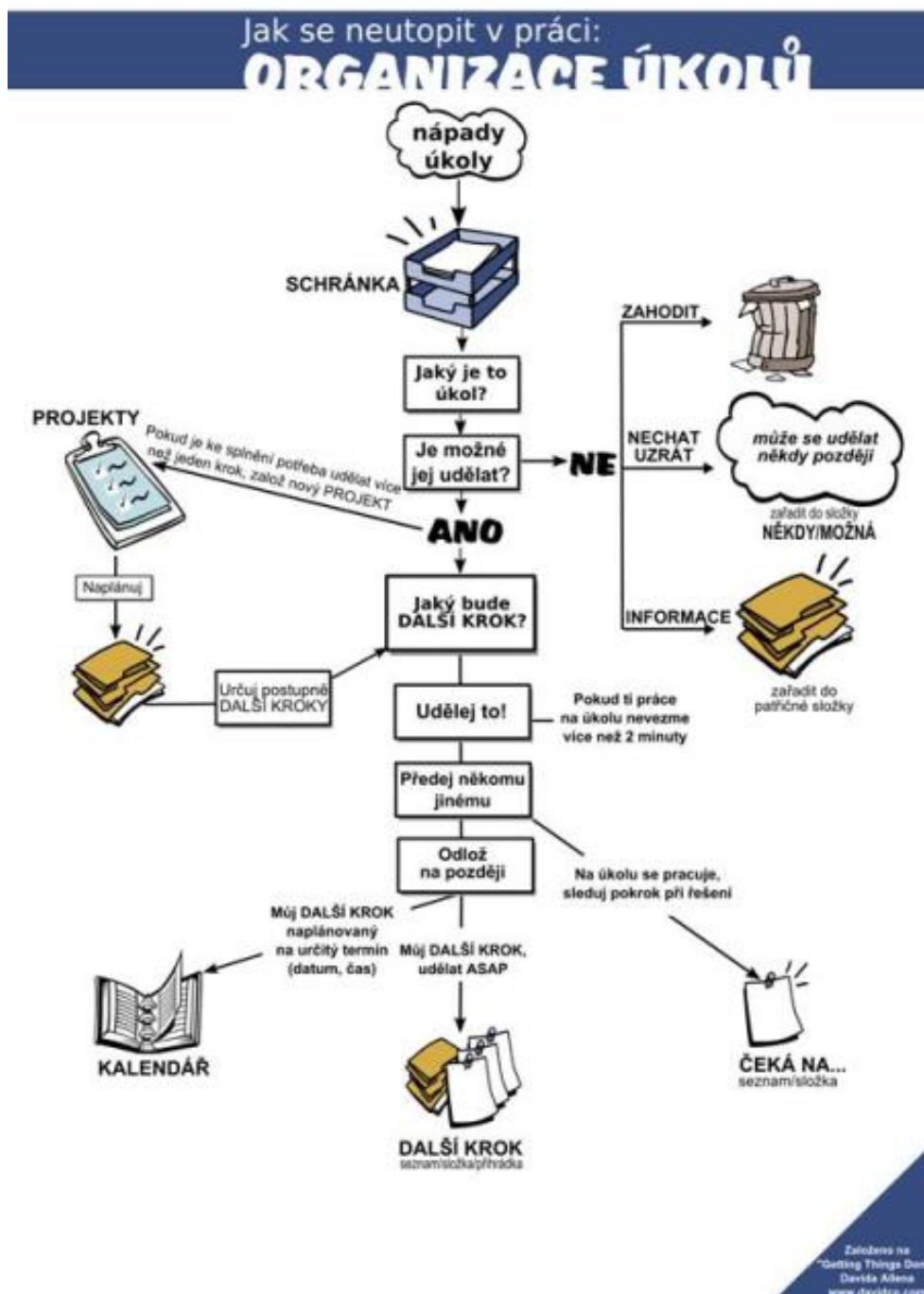
1. sběr věcí, které si získaly naši pozornost
2. hodnotíme jejich význam a kroky k jejich splnění
3. uspořádáme si výsledky
4. máme je v plánu a hledáme vhodný čas k provedení
5. provedeme

#### 2.1.2.1 Sběr věcí

Pozornost si získávají věci u kterých si řekneme „*měl bych*“, „*potřebuji*“, „*musím*“. Je potřeba sesbírat všechny věci a žádnou nelze vynechat. Pro naši mysl je důležitým slovem „*všechno*“ a nesmí vzniknout jakákoliv pochybnost. V tuto chvíli ještě nepotřebujeme pouze jedno místo uložení, protože spojujeme všechny naše zdroje např. emaily, fyzické dokumenty, ... Označme taková místa naší schránkou.



### 2.1.2.2 Zpracování schránky



Obrázek 2.1: Digram zpracování schránky dle metodiky GTD [2]

## 2. ANALÝZA METODIKY GTD

---

Jak *GTD* graf interpretovat (popíši důležité body):

1. Co je to za úkol? Potřebujeme se nad úkolem zamyslet, čeho se týká a co bude potřeba k jeho splnění.
2. Realizovatelný/ nerealizovatelný. Zde se rozhodujeme, jestli úkol můžeme a chceme splnit. Nerealizovatelný úkol může být pouhá informace. Něco o čem zatím neuvažujeme a nebo pro nás není úkol relevantní a zahazujeme jej.
3. Pro realizovatelné úkoly zjistíme kolik kroků bude potřeba k jejich splnění. V případě, že jich je více zakládáme projekt a pod ním více úkolů. Hierarchie projektů a úkolů má stromovou strukturu, kde úkoly jsou vždy listy stromu.
4. Provedení úkolu
  - a) trvá-li splnění úkolu kratší dobu než 2 minuty - splňme ho
  - b) úkol musí vyřešit někdo jiný - deleguj ho
  - c) úkol řeším a rozhodnu se o dalším kroku nebo zařadím do kalendářePo vyhodnocení úkol končí vždy ve stavu, kdy je jasné jeho další zpracování.

Pravidelné vyprazdňování našich schránek představuje práci navíc a z počátku k němu bude existovat odpor. Musíme docílit návyku a plné důvěry, proč to děláme.

Zde už potřebujeme jednotný systém pro správu našich úkolů a projektů. Jeho navržením se budu zabývat v další kapitole.

Důležité pojmy z *GTD*

- *Činnost/ závazek/ věc/ úkol* – něco, co máme vykonat a k čemu jsme se zavázali
- *Úkol* – zde ve smyslu konkrétního kroku směřujícího ke splnění činnosti
- *Projekt* – může obsahovat další *projekty* a úkoly společné pro jednu činnost
- *Schránka* – místo/ místa na kterých se nacházejí naše činnosti. Zpravidla se jedná o email, šanony,...

### 2.1.2.3 Pravidelná kontrola úkolů

Kritickým klíčem k úspěchu jsou pravidelná zhodnocení všech aktuálních úkolů. Ideálně každý týden. Naším cílem je udržet si plný přehled o úkolech a zohlednit do nich současné priority.

Stejně jako u vyprazdňování schránky se snažíme o vytvoření návyku.

#### 2.1.2.4 Co mám dělat?

Otázkou, kterou si nyní musíme zodpovědět je, jak nám náš seznam úkolů pomůže v rozhodnutí, co nyní dělat.

K tomu nám budou sloužit *kontexty*.

Představme si, že máme chvíli času před schůzkou a po ruce mobilní telefon. Musíme mít možnost rychle zjistit úkoly splnitelné zavoláním z mobilního telefonu. U úkolů zavedeme kontext představující prostředek k jeho splnění. Např. mobilní telefon, notebook, internet, domov,...

Kontexty jsou pro každého z nás specifické a potřebujeme mít možnost si je spravovat.

Dalšími možnými filtry jsou *dostupný čas*, *dostupná energie* (jak moc „svěží“ musíme být jeho splnění) a *priorita*.

#### 2.1.3 Shrnutí metodiky

- *vše máme mimo naši mysl* – cokoliv potřebujeme udělat máme poznamenáno mimo naši hlavu
- *známe další krok*
- *umíme naše schránky zpracovat*
- *jednotný systém*. Máme jednotný systém, kam zaznamenáváme všechny úkoly/ projekty při zpracování schránky. Máme k němu 100% důvěru a umíme ho správně použít.

## 2.2 Současné implementace

Zde se seznámíme s existujícími implementacemi pro osobní plánování podporujících metodiku GTD. Řešení si vzájemně porovnáme z pohledu ovládání, funkcí, začlenění s Google kalendářem, sociálními sítěmi a podpory pro web, iOS a Android.

### 2.2.1 Toodledo

Odkaz: [www.toodledo.com](http://www.toodledo.com)[5]

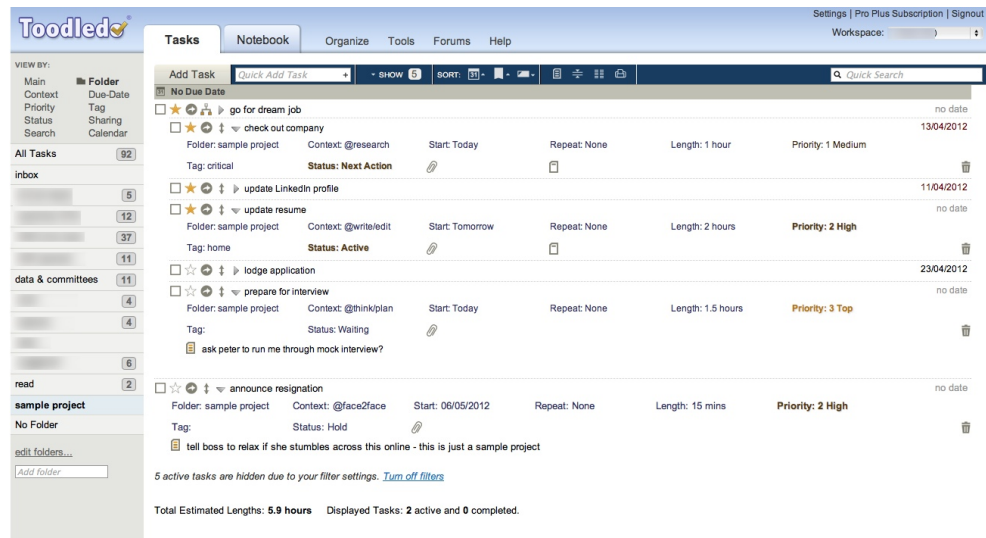
Podporovaná zařízení web: ano, iOS: ano, Android: ano

Výhody: Plně podporuje metodiku GTD. Zakládání a editace úkolů je zde velmi snadná a vše je ukládáno okamžitě při změně. U úkolů lze nastavit široké spektrum filtrů od základních kontextů až po definování lokace. Podporuje Google kalendář (widget).

Nevýhody: Neobsahuje integraci se sociálními sítěmi. Projekty jsou až od placené verze.

Tento systém v současnosti používám pro své osobní plánování.

## 2. ANALÝZA METODIKY *GTD*



Obrázek 2.2: Podoba webové aplikace ToodleDo.[3]

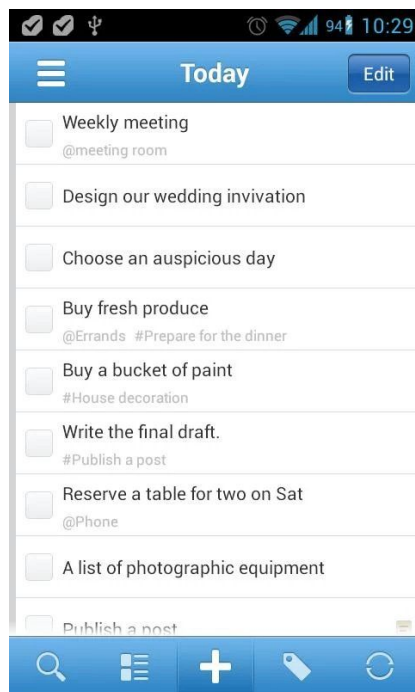
### 2.2.2 Doit.im

Odkaz: [doit.im](http://doit.im)[6]

Podporovaná zařízení web: ano, iOS: ano, Android: ano

Výhody: Plně podporuje metodiku GTD. Příjemná minimalistická aplikace, která je ihned po registraci připravena k použití.

Nevýhody: Mírně nepřehledná webová aplikace.



Obrázek 2.3: Mobilní aplikace Doit.[?]

### 2.2.3 Wunderlist

Odkaz: [www.wunderlist.com](http://www.wunderlist.com)[7]

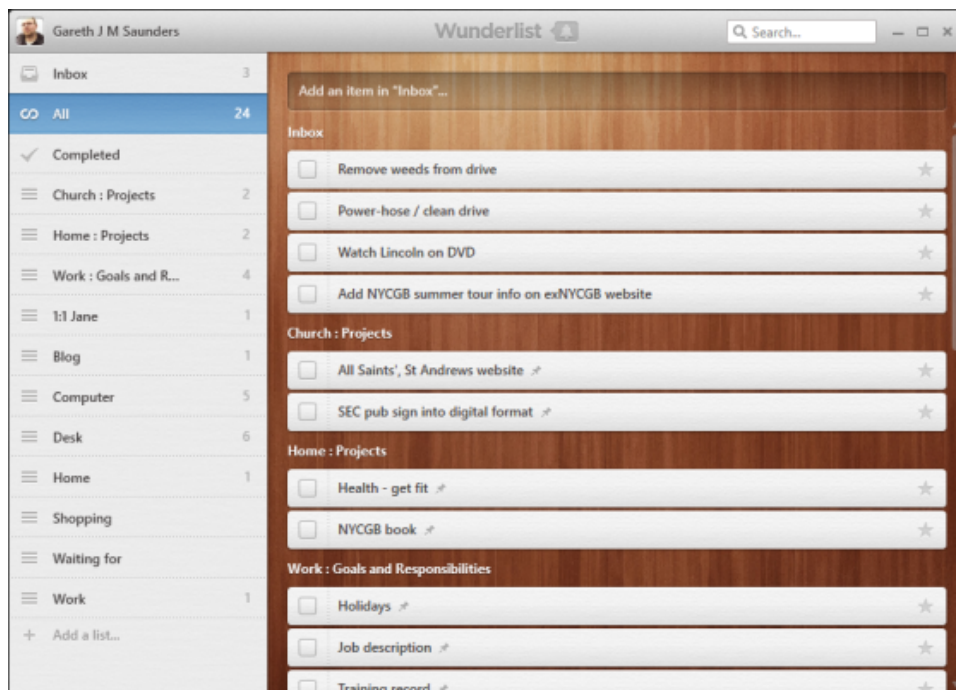
Podporovaná zařízení web: ano, iOS: ano, Android: ano

Výhody: Plná podpora *GTD*. Pěkné grafické zpracování s další možností přizpůsobení. Po registraci má vše potřebné k okamžitému použití. Jako u ToodleDo jsou změny okamžitě ukládány. Zakládání úkolů zasláním emailu. Sdílení úkolů.

Nevýhody: Bez sociálních sítí.

## 2. ANALÝZA METODIKY *GTD*

---



Obrázek 2.4: Wunderlist [4]

### 2.2.4 Todoist

Odkaz: [todoist.com](http://todoist.com)[8]

Podporovaná zařízení web: ano, iOS: ano, Android: ano

Výhody: Projekty jsou zde součástí základní verzí. Pokud chceme jednoduché ovládání bez další funkcí.

Nevýhody: Nesplňuje nároky *GTD*, ale umožňuje všechny potřebné základní funkce. Desktopová aplikace vychází z omezení mobilních zařízení a nevyužívá celou šířku obrazovky.

### 2.2.5 Srovnání existujících řešení

Výsledné srovnání je shrnuto v tabulce 2.1.

Tabulka 2.1: Porovnání existujících implementací metodiky GTD - (\* značí placenou verzi)

Funkce	Toodledo	Doit.im	Wunderlist	ToDoList
Web/Android/iOS	ano/ano/ano	ano/ano/ano	ano/ano/ano	ano/ano/ano
Automatické ukládání	ano	ne	ano	ne
Intuitivní ovládání	ano	mírně nepřehledné	ano	ano
Podpora Google kalendáře	ano*	ano	ano	ne
Facebook	ano*	ne	ne	ne

## 2.3 Shrnutí analýzy metodiky GTD

Z samotné metodiky GTD jsem odvodil tyto nároky na aplikaci:

- *Základní funkce.* Není třeba vytvářet složité funkce. Ale ty které existují, musí být spolehlivé a uživatel nesmí mít problém s jejich ovládáním.
- *Dostupnost.* Aplikace se nemůže soustředit pouze na jednu platformu a musí být dostupná z více platforem (web, mobilní zařízení,...).
- *Spolupráce s aplikacemi třetích stran.* U uživatele lze očekávat Facebook/Gmail účet a aplikace může toho využít.

Současné implementace můj závěr potvrzují. I u malých projektů je podpora pro všechny platformy téměř samozřejmostí. Minimalistické na funkčnost zaměřené uživatelské rozhraní. Synchronizace k emailovými účty je často také přítomna, ačkoliv je už součástí placené verze.

*Implementace bude obsahovat*

- *Aplikace pro centrální uložení dat vystavující WS pro jejich správu*
- *Aplikace pro OS Android*





---

# Návrh

Tato kapitola popisuje volbu technologií pro implementovanou aplikaci poskytující centrální uložení dat skrze WS a aplikaci pro OS Android. Nároky na technologie vycházejí ze závěrů analýzy předešlé kapitoly a dále je rozvíjejí. Protože jedna z aplikací potřebuje hosting, budeme hledat vhodný server splňující naše potřeby.

Seznámíme se postupem pro propojení mobilní aplikace na Facebook a Google kalendář. Přidělíme každé aplikaci oddělené úkoly. Mobilní aplikace zajistí přihlášení uživatele a získání access tokenu. Tento token předáme aplikaci pro centrální uložení dat a ta zajistí samotnou práci s API protistrany.

Dále kapitola specifikuje REST API a ukazuje webového vývojového prostředí od společnosti Mulesoft, které je k návrhu použito.

Kapitola dále popisuje vytváření datového modelu.

## 3.1 Rozdělení aplikace

Ze závěrů analýzy vzešlo, že aplikace musí být pro uživatele dostupná více platformách zahrnující např. mobilní zařízení a web. Proto jsem se rozhodl rozdělit návrh aplikace na dvě části. První bude zajišťovat centrální uložení dat. Druhá pak zajistí pro uživatele samotné GUI pro osobní plánování a správu dat deleguje na první aplikaci. Díky tomuto návrhu se zbavím nutnosti implementovat uložení dat na každé uživatelské aplikaci.

### 3.1.1 Aplikace pro centrální uložení dat

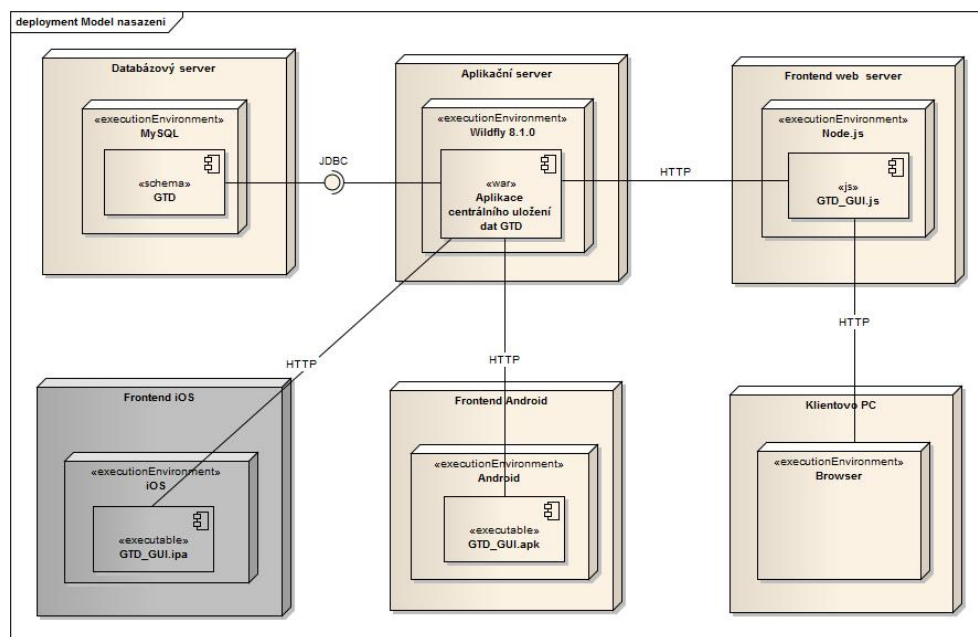
Skrze WS bude poskytovat centrální uložení dat do databáze. Služby budou dostupné přes internet a samotná aplikace bude nasazena na aplikační server. Současně umožní publikovat na Facebook a Google kalendář.

### 3. NÁVRH

#### 3.1.2 Mobilní aplikace pro Android

První implementace GUI cílená na mobilní zařízení se systémem Android. Fungovat bude na principu tenkého klienta[9]. Aplikace využije přihlašovací komponenty Facebooku a Googlu pomocí kterých získá uživatel prostředek pro autorizaci aplikace a přístup k publikaci dat do těchto sítí.

#### 3.1.3 Model nasazení



Obrázek 3.1: Model nasazení GTD (šedivě jsou zbarvena úvaha dalšího rozšíření)

Vycházení z návrhu rozdělení aplikace. GUI aplikace jsou unikátní pro každou platformu s centrální aplikací pro správu dat komunikují přes HTTP protokol. Část nasazení *Frontend web server* zobrazuje aplikaci vytvořenou v rámci týmového projektu SP2. Ta není v současnosti funkční a není součástí nasazení.

## 3.2 Facebook a Google kalendář

Aplikace pro osobní plánování nesmí být izolovanou. Zde si rozebereme návrh pro integraci do Facebooku a Google kalendáře.

### 3.2.1 Facebook

#### 3.2.1.1 Proč Facebook

Facebook[10] je v současnosti dominantní sociální sítí a jeho používání se pro mnohé stává denní samozřejmostí[11]. Poskytuje širokou možnost integrace např. [12].

Vývoj na *zdi* uživatele se pro většinu jedinců stává moderním elektronickým deníkem. Do něj chtějí zaznamenávat co nejvíce svých osobních a sociálních informací.

#### 3.2.1.2 Oddělení přihlášení od publikace

K oddělení funkce nastane pro přihlášení uživatele a publikace. Android aplikace umožní uživateli přihlášení k Facebooku s cílem získat access token. Tento token bude následně předán aplikaci pro centrální uložení dat současně s informací identifikující úkol. Ta zajistí jeho publikaci.

Více k access tokenu na [13].

#### 3.2.1.3 Přihlášení

Bude zajišťováno na straně Android aplikace.

Musíme řešit:

- *registrace aplikace na straně Facebooku*
- *nastavení naší aplikace*

*Registrace aplikace na straně Facebooku*

Provádí se přes stránky Facebook developer[14]. Příklad postupu lze najít na [15].

*Nastavení aplikace*

Pro přihlášení uživatele využijeme komponentu a funkce z SDK Facebooku pro Android viz. 3.3.2.4. Pro další nastavení budeme postupovat podle návodu v [16].

Komponenta zajistí správu nad procesem přihlašování uživatele. Výstupem procesu přihlášení je objekt obsahující access token.

#### 3.2.1.4 Publikace

Access token umožní autorizaci vůči Facebook API v rámci uživatelem schválených práv. V našem případě budeme mít přístup k editaci *zdi* uživatele.

Testy nad Facebook API lze skrze ??.

#### 3.2.2 Google kalendář

##### 3.2.2.1 Proč Google kalendář

Google kalendář je součástí rodiny Google aplikací[17]. Ze statistik pro Gmail[18] můžeme odvodit, že se jedná o rozšířenou součást osobního i firemního plánování.

Google kalendář dále umožňuje synchronizaci do dalších aplikací (např. MS Outlook<sup>1</sup>) a pro odlišné platformy existují další aplikace zobrazení jeho dat (např. aCalendar<sup>2</sup> pro Android).

##### 3.2.2.2 Oddělení přihlášení od publikace

Stejně jako u Facebooku 3.2.1.2 oddělíme funkci na přihlášení a publikaci.

Více o access tokenu na [19].

##### 3.2.2.3 Přihlášení

Bude zajišťováno na straně Android aplikace.

Musíme řešit:

- *registrace aplikace na straně Google*
- *nastavení naší aplikace*

*Registrace aplikace na straně Google*

Provádí se přes stránky Google developer console[20]. Příklad postupu lze najít na [21].

*Nastavení aplikace*

Pro přihlášení uživatele využijeme komponentu a funkce z Google play service viz. 3.3.2.5. Komponentou bude Google+ Sign-in [22]. Pro další nastavení budeme postupovat podle návodu v [16].

Samotný proces přihlášení bude mít dvě fáze. V první kroku vyzveme uživatele k přihlášení pomocí svého google účtu. Po tomto kroku ještě nemáme nutný přístup a musí následovat další krok, ve kterém uživatele požádáme o přidělení práv pro naši aplikaci. Tím získáme potřebný access token[23].

##### 3.2.2.4 Publikace

Access token umožní autorizaci vůči Google API v rámci uživatelem schválených práv. V našem případě budeme mít přístup k datům v jeho Google kalendáři s právem editace a vyvážení.

Testy nad Google API lze skrze [24].

---

<sup>1</sup>[tools.google.com/dlpage/gappssync](https://tools.google.com/dlpage/gappssync)

<sup>2</sup>[play.google.com/store/apps/details?id=org.withouthat.acalendar](https://play.google.com/store/apps/details?id=org.withouthat.acalendar)

### 3.3 Technologie

V této části si představíme technologie potřebné pro následující implementaci.

#### 3.3.1 Aplikace pro centrální uložení dat

##### 3.3.1.1 Webová služba – REST API

Při volbě webového API jsem se rozhodoval mezi dvěma typy webových služeb.

- *SOAP*
- *REST*

Nyní se soustředím na hlavní rozdíly mezi těmito službami.

Tabulka 3.1: Rozdíly mezi SOAP a REST

Typ	SOAP	REST
Formát dat[25]	XML	nezávislý (XML, JSON, plain)
Protokol[25]	více možných (HTTP, SMTP,...)	HTTP
Možnost cachování[25]	ne	ano
Formát specifikace[26][27]	WSDL[28]	existují formáty jako WADL[29], Swagger[30], RAML[31] a další, ale žádný nemůžeme považovat na hlavní [32]
Vystavuje[33]	operace	zdroje
Zabezpečení[33]	SSL + WS-Security	SSL
Rozšířenost[33]	REST získává na stále větší popularitě . Příkladem uvedu např. Google, který se nahradil své dřívější SOAP API právě RESTem.	
Rychlost zpracování[34]	XML parsing je pomalejší než JSON[35]	
Jednodušší implementace[25]	u RESTu na straně klienta i serveru	

Návrh nepočítá s využitím zabezpečení (přihlášení bude řešeno pomocí tokenu 4.1.4) a komunikace bude probíhat přes HTTP protokol (SOAP ztrácí výhodu). Myšlenkou RESTu je poskytnout CRUD[33] operace nad vystavenými zdroji[36] a to přesně odpovídá potřebám navrhovaného centrálního uložení dat. Mezi další výhody patří rychlá implementace a přenos dat ve formátu JSON.

### 3. NÁVRH

---

Volba pro WS je REST.

Pro návrh REST API bude použit jazyk RAML. Jedná se nový jazyk založený na formátu YAML, který v říjnu 2014 přešel do verze 1.0[31] a na jehož vývoji se podílí technologičtí odborníci ze známých IT firem[37]. Díky YAML se kód dobře píše a výsledná dokumentace je čitelná. Pro vývoj bude použito webového prostředí od firmy MuleSoft[38]. Jedná se o firmu, která je s RAML úzce propojena, protože na specifikaci RAML se podílí její CTO<sup>3</sup>.

#### 3.3.1.2 Programovací jazyk – Java

Zde je volba určena již existující aplikací z projektu SP2, která je napsána v jazyce Java. Zároveň aplikace pro zvolený OS Android 3.3.2.1 jsou psány také v Javě.

#### 3.3.1.3 Základní framework – Spring

Nyní hledám framework, který mi umožní rychle implementovat REST API. Ve světě Javy na podobné otázky existuje vždy více možných odpovědí.

Podívejme se na hlavní možnosti:

- *JAX-RS*<sup>4</sup> – REST API od Javy
- *Jersey*<sup>5</sup> – implementace REST API založená na JAX-RS
- *Spring MVC*<sup>6</sup> – vlastní implementace z rodiny Spring

Všechny řešení podobný postup. Pomocí anotací ovlivňují chování tříd a method pro příjem/ odeslání zpráv v kontextu REST API.

Volba padla na Spring. Hlavním důvodem byl rozsah celého Spring frameworku. Díky jeho univerzálnosti, snadné konfigurovatelnosti a dobré dokumentaci je použití Springu a Javy velmi častým jevem.

Méně častou volbou je mé rozhodnutí pro rozšíření Spring boot<sup>7</sup>. Slouží pro vytváření stand-alone Spring aplikací a zlehčuje základní konfiguraci aplikace. Pozitivním efektem je také odstranění XML konfigurací[39].

#### 3.3.1.4 Persistence dat – Hibernate

Hibernate[40] používá ORM, což je způsob, kdy je objektový model mapován na relační databázi. Programátor pomocí anotací (nejčastěji nad třídami modelu) definuje podobu databázových objektů, ale o samotnou správu na úrovni

---

<sup>3</sup>[www.linkedin.com/in/sarid](http://www.linkedin.com/in/sarid)

<sup>4</sup>[jax-rs-spec.java.net](http://jax-rs-spec.java.net)

<sup>5</sup>[jersey.java.net](http://jersey.java.net)

<sup>6</sup><http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html>

<sup>7</sup>[projects.spring.io/spring-boot/](http://projects.spring.io/spring-boot/)

databáze se už nestará. Hibernate také zajišťuje nezávislost na použité databázi. Nicméně např. pro Oracle databázi je potřeba použít speciálních anotací (databáze nemá auto inkrementální sloupce a pracuje přes definici sekvencí hodnot [41]).

Použití Hibernate je Springem přímo podporováno a jeho konfigurace je velice snadná. Jiné možnosti (jako např. myBatis<sup>8</sup>) nebyly zvažovány.

#### 3.3.1.5 Facebook – Restfb

Restfb je klient pro práci s Facebook API[42].

Neslouží k přihlášení uživatele, ale přijímá access token pro autorizaci požadavků. Poskytuje implementované rozhraní nad Facebook API, které zajišťuje odeslání a přijetí požadavku.

#### 3.3.1.6 Databáze – MySQL

Jedná se o relační databázi, která si v současnosti získává obrovskou oblibu. Je to díky její nenáročnosti, jednoduchosti, výkonnosti a stabilitě. Vlastníkem je firma Oracle, vydávající také databázi Oracle DB, která už vyžaduje specializovanou správu a není pro tento projekt vhodná[43]. Díky mým dřívějším zkušenostem jsme pro správu MySQL zvolil phpMyAdmin.

citace zdroje

#### 3.3.1.7 Aplikační server Wild Fly

Jedná se pokračování známého JBoss AS[44].

Vlastnosti:

- *rychlá instalace*
- *funkce auto deploy* – zkopírování waru<sup>9</sup> aplikace do určité složky serveru dojde k jejímu automatickému nasazení
- *administrátorské rozhraní*
- *dokumentace* – díky jeho rozšíření existuje velké množství návodů
- *podpora Java EE*<sup>10</sup>

#### 3.3.1.8 Hosting – DigitalOcean

Pro hosting jsem zvolil služby DigitalOcean<sup>11</sup>.

Jako další možnost byl server OpenShift<sup>12</sup>. Ten poskytuje pevně dané aplikace, které je možné na virtuální server nainstalovat. Instalace serveru probíhá

<sup>8</sup>[mybatis.github.io/mybatis-3/](http://mybatis.github.io/mybatis-3/)

<sup>9</sup><http://docs.oracle.com/javase/6/tutorial/doc/bnaby.html>

<sup>10</sup><http://www.oracle.com/technetwork/java/javase/overview/index.html>

<sup>11</sup>[www.digitalocean.com](http://www.digitalocean.com)

<sup>12</sup>[www.openshift.com](http://www.openshift.com)

formou konfigurace a je díky tomu velmi snadná. Bohužel omezení systémových prostředků u neplacené verze vedlo k nespolehlivosti mého serveru a hledání jiné varianty.

DigitalOcean poskytuje tzv. droplet. Funguje jako virtuální server a správce si volí OS, který je něm automaticky nainstalován. Další instalace se už provádějí přímo ve vybraném OS. Mou volbou byl Ubuntu verze 14. Jde o jednoduchý a spolehlivý linuxový systém, se kterým mám jen kladné zkušenosti.

DigitalOcean je placený, ale v rámci GitHub Education packu<sup>13</sup> je pro studenty k dispozici 100 dolarový kapitál. To stačí na mnohaměsíční provoz i silnějších variant dropletů s více systémovými prostředky. Pro naši aplikaci jsem zvolil variantu 2 GB paměti, 2 CPU, 40 GB SSD disk za 20 dolarů na měsíc.

### 3.3.2 Mobilní aplikace pro Android

#### 3.3.2.1 OS Android

Rozhodl jsem se svou mobilní aplikaci implementovat pro OS Android.

Vybíral jsem mezi OS Android do firmy Google<sup>14</sup> a iOS od firmy Apple<sup>15</sup>.

Celkové množství aplikací je u obou OS téměř stejné a ani náročnost vývoje neznamena výrazné rozdíly.[45]. Rozdíl programovacích jazyků (Android:Java a iOS:Objective-C) pro mě znamená důležitý faktor. U předešlé aplikace jsem se rozhodl pro Javu a díky Androidu mohu zůstat ve stejném programovacím jazyce.

*Trh* Android má silnou převahu v zastoupení na světovém trhu. Zatímco iOS si drží 15% podíl, Android se pohybuje na 80% [46]. Silnější pozice dosahuje iOS na trhu v U.S., kde můžeme zastoupení obou OS považovat za vyrovnané [47]. Pro mě je důležité, že Android při stejném množství aplikace poskytuje širší uživatelskou základnu.

Závěrem analýzy existujících implementací osobního plánování je důraz na podporu co nejširšího spektra platforem. Z tohoto pohledu chápu mé rozhodnutí pro vývoj na OS Android jako volbu první mobilní platformy pro implementaci. A v rámci dalšího rozvoje projektu plánuji podporu pro další.

#### *Shrnutí*

- *Java* – mohu stavět na současných znalostech tohoto jazyka a využít stejné vývojové prostředí
- *zastoupení na trhu* – více uživatelů mi dává vyšší šance si najít své zákazníky
- *osobní preference* – vlastním mobil s OS Android

---

<sup>13</sup>[education.github.com/pack](https://education.github.com/pack)

<sup>14</sup>[www.android.com](http://www.android.com)

<sup>15</sup>[www.apple.com/cz/ios/](http://www.apple.com/cz/ios/)



### 3.3.2.2 Android-bootstrap

Pro implementaci jsem se rozhodl použít framework Android-bootstrap od Donna Felkera <sup>16</sup>.

Framework dodává implementaci základních funkcí aplikace pro OS Android. Používá se jako základ při vytváření aplikace a umožňuje rychlý vývoj. Zdrojové kódy jsou přístupné na GitHubu, a autor se společně s komunitou podílí na dalším rozvoji.

*Obsahuje*

- *napojení na REST API*
- *implementace přihlašování*
- *stránkování jako hlavní komponenta uživatelského rozhraní* – snadné rozšíření o další stránky
- *vlastní implementace zobrazení seznamu dat* – načtení dat v asynchronním vlákně, specifikace zobrazení pomocí vlastního layoutu<sup>17</sup>
- *jednotný grafický vzhled*

Nevýhodou je nárok na verzi Android SDK 15 a vyšší.

### 3.3.2.3 Verze SDK Android

Rozhodl jsem pro minimální verzi SDK 16 (Android verze 4.1 s označením Jelly bean) a cílovou verzi SDK 19 (Android verze 4.4 s označením Kitkat). Toto rozhodnutí vzešlo z nároků mnou zvolené technologie a současného stavu trhu.

Použitím frameworku Bootstrap (viz 3.3.2.2) jsem stanovil minimální verzi SDK na 15 a vyšší. V rámci vývoje jsem následně použil funkce vyžadující verzi SDK 16+.

Autoři Androidu doporučují volbou SDK pokrýt 90% aktivních zařízení[48]. V aktuálním zastoupení SDK vede verze 19 a suma od verze 4.1 výše je cca 88%[49].

### 3.3.2.4 Facebook SDK

Pro přístup k přihlášení na Facebook využijeme *Facebook SDK* [50].

Poskytuje širokou škálu funkcí pro práci s touto sociální sítí.

Verze SDK a příslušné verze aplikace Facebook lze stáhnout na href="https://developers.facebook.com/docs/android/sdk" href="https://developers.facebook.com/docs/android/sdk">https://developers.facebook.com/docs/android/sdk

### 3.3.2.5 Google play service

K přihlášení na Google použijeme Google play service[51].

<sup>16</sup><http://www.androidbootstrap.com/>

<sup>17</sup><http://developer.android.com/guide/topics/ui/declaring-layout.html>

### 3. NÁVRH

---

Google play service musí být současně nainstalována i na uživatelském mobilním zařízení[52].

#### 3.3.2.6 Klient pro REST API - Retrofit

Pro konzumaci REST API použijeme implementaci klienta Retrofit [53]. Konfigurace a použití jsou velmi snadné a k dispozici je kvalitní dokumentace. Nevýhodou jsou vyšší nároky na paměť[54].

### 3.4 Specifikace REST API

Budeme pracovat s těmito čtyřmi metodami HTTP 3.2:

Tabulka 3.2: Funkce HTTP metod

HTTP metoda	Popis
GET	získá informace daného zdroje
POST	vytvoří nový zdroj
PUT	aktualizuje daný zdroj
DELETE	smaže daný zdroj

Navržené REST API rozhraní je zobrazeno v tabulkách 3.3 a 3.3 (\* – představuje společnou URL adresu určenou nasazením a verzí API).

Na základě volby v 3.3.1.1 vznikl návrh REST API ve webovém vývojovém prostředí Mulesoft.

Přístupné přes: Mulesoft API Designer<sup>18</sup>

Přihlašovací jméno: drugnanov

Heslo: Qazwsxedc369

Práce s tímto prostředím je příjemná. Některé funkce potřebují ještě vylepšit. Chybí online nápověda v kontextu specifikace RAML a např. auto complete nepracuje s objekty vytvořenými uživatelem. Zároveň příkladů návrhu API v jazyku RAML na webu není mnoho. To bohužel prodlužuje čas učení a samotného návrhu. Přesto má tento nástroj vykročeno stát se součástí DDD[55].

### 3.5 Databázový model

V 3.3.1.4 jsem se rozhodl pro použití Hibernatu. To nám umožňuje specifikovat strukturu databáze přímo v datovém modelu aplikace. Specifikace se provádí pomocí anotací nad objekty tříd, metod a atributů[56].

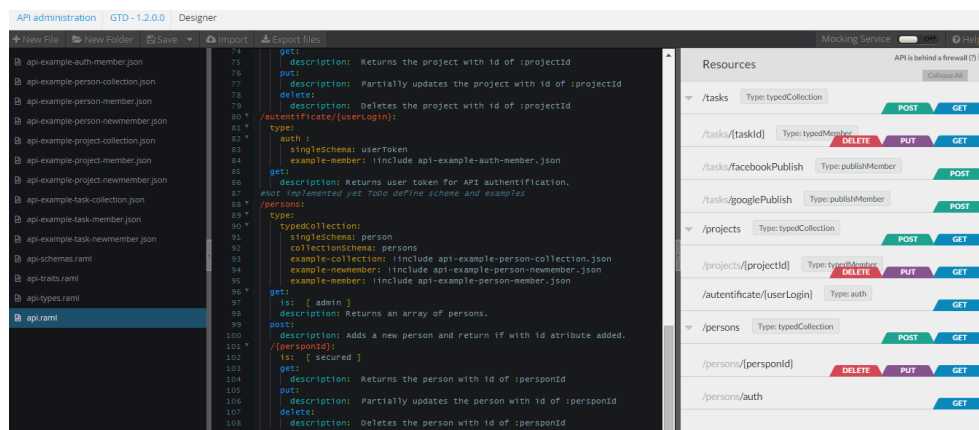
Tabulka 3.3: Rozhraní REST API přístupné s tokenem

<b>Zdroj URI</b>	<b>HTTP metoda</b>
1	*/projects/ POST
2	*/projects/ GET
3	*/projects/{id} GET
4	*/projects/{id} PUT
5	*/projects/{id} DELETE
6	*/tasks/ POST
7	*/tasks/ GET
8	*/tasks/{id} GET
9	*/tasks/{id} PUT
10	*/tasks/{id} DELETE
11	*/tasks/{id}/facebookPublish POST
12	*/tasks/{id}/googlePublish POST
13	*/persons/ GET
14	*/persons/{id} GET
15	*/persons/{id} PUT
16	*/persons/{id} DELETE
17	*/persons/auth GET

Tabulka 3.4: Rozhraní REST API přístupné bez tokenu

<b>Zdroj URI</b>	<b>HTTP metoda</b>
*/authenticate/{userLogin}	GET
*/persons	POST

### 3. NÁVRH



Obrázek 3.2: Webové prostředí pro návrh REST API ve specifikaci RAML od firmy Mulesoft

## Realizace

V této kapitole se seznámíme s praktickým použitím dříve zvolených technologií. U aplikace pro centrální uložení dat si ukážeme strukturu projektu, podobu práce s REST API, generování struktury databáze pomocí frameworku hibernate, identifikace uživatele s použitím tokenu a práci s API Facebooku či Google kalendáře. U android aplikace si uk

verze knihoven

co s ukážeme u  
android aplikace

## 4.1 Aplikace pro centrální uložení dat

Zde si ukážeme části implementace aplikace pro centrální uložení dat.

### 4.1.1 Struktura projektu

libs .....	externí knihovny používané aplikací
src .....	základní balík implementace projektu
main .....	balík aplikace
java .....	složka s kódy aplikace
GTD .....	základní balík GTD
BL .....	balík business vrstvy
DL .....	balík datové vrstvy
restapi .....	balík pro REST API
resources .....	složka obsahující konfigurace
databases .....	složka obsahující pomocné skripty pro správu databáze
messages .....	složka pro stringové konstanty aplikace
gtd.api.properties .....	property aplikace
hibernate.cfg.xml .....	konfigurace hibernatu
logback.xml .....	konfigurace logování pomocí logbacku
test .....	složka obsahující unit testy
build.gradle .....	build soubor pro Gradle
settings.gradle .....	nastavení Gradlu

### 4.1.2 REST API

Implementace je rozdělena na několika úrovních:

- *kontrolery*
- *servisy*
- *persistence dat*

*Kontrolery*

Jejich úkolem je přijetí požadavku a odeslání odpovědi. Samotné zpracování delegují na servisy. Příklad v 4.3.

Listing 4.1: Příklad třídy kontroleru

```
@RestController
@RequestMapping("/api/v1/tasks")
public class TaskRestController extends RestControllerAbs {

    protected Logger logger =
        LoggerFactory.getLogger(this.getClass());
```

```
private TaskAdmin taskAdmin;
private PersonAdmin personAdmin;

@Autowired
public void setTaskAdmin(TaskAdmin taskAdmin) {
    this.taskAdmin = taskAdmin;
}

@Autowired
public void setPersonAdmin(PersonAdmin personAdmin) {
    this.personAdmin = personAdmin;
}

@RequestMapping(value =("/{id}", method = RequestMethod.GET)
@ResponseBody
public Task get(@PathVariable int id, ServletWebRequest wr,
    Principal auth) {
    logRequest(wr);
    String userLogin = ((UsernamePasswordAuthenticationToken)
        auth).getPrincipal().toString();
    Person user = personAdmin.getOsoba(userLogin);
    return taskAdmin.getUkol(id, user);
}

@RequestMapping(value =("/{id}", method =
    RequestMethod.DELETE)
@ResponseStatus(HttpStatus.OK)
public void delete(@PathVariable int id, ServletWebRequest
    wr, Principal auth) {
    logRequest(wr);
    String userLogin = ((UsernamePasswordAuthenticationToken)
        auth).getPrincipal().toString();
    Person user = personAdmin.getOsoba(userLogin);
    Task t = taskAdmin.getUkol(id, user);
    taskAdmin.deleteUkol(t, user);
}

...

@RequestMapping(method = RequestMethod.POST)
@ResponseStatus(HttpStatus.CREATED)
public Task create(@RequestBody Task task, Principal auth,
    ServletWebRequest wr) {
    logRequest(wr);
    String userLogin = ((UsernamePasswordAuthenticationToken)
        auth).getPrincipal().toString();
    Person user = personAdmin.getOsoba(userLogin);
    taskAdmin.addUkol(task, user, null);
    return task;
}

...
}
```

### *Servisy*

Na této úrovni se odehrává business logika. Každá servisa obsluhuje funkce pro zpracování konkrétní entity (např. task/projekt). Využívá k tomu vlastní implementaci, další servisy a datovou vrstvu příslušnou entitě kterou obsluhuje. Příklad v 4.2.

Listing 4.2: Příklad třídy servisy

```
@Component
public class PersonAdmin {

    @Autowired
    private IDAOPerson daoOsoba;
    @Autowired
    private IDAOSState daoStav;
    @Autowired
    private IDAOPersonToken daoPersonToken;
    @Autowired
    private TaskAdmin taskAdmin;
    @Autowired
    private ProjectAdmin projectAdmin;
    @Autowired
    private ContextAdmin contextAdmin;
    @Autowired
    private TokenUtils tokenUtils;

    ...

    public void addPersonToken(PersonToken personToken) {
        if (!personToken.checkValidate()) {
            throw new InvalidEntityException("Invalid token
                data:" + personToken.toString());
        }
        daoPersonToken.create(personToken);
    }

    public PersonToken createPersonToken(Person osoba) throws
        UnsupportedEncodingException,
        NoSuchAlgorithmException {
        PersonToken personToken =
            PersonToken.createPersonToken(osoba);
        addPersonToken(personToken);
        return personToken;
    }

    public Person addOsoba(Person person)
        throws UnsupportedEncodingException,
        NoSuchAlgorithmException,
        ResourceExistsException {
        if (!isValid(person)) {
```



```
        throw new InvalidEntityException("Invalid person
            data.");
    }
    if (daoOsoba.existPerson(person.getUsername())) {
        throw new ResourceExistsException("Cannot add
            person due to existing login:" +
            person.getUsername());
    }
    person.setPassword(HashConverter.md5(person.getPassword()));
    daoOsoba.create(person);
    person = daoOsoba.get(person.getId());
    tokenUtils.getToken(person);
    return daoOsoba.get(person.getId());
}

...

}
```

---

#### *Persistence dat*

Využívá Hibernatu 3.3.1.4 s implementací generiky[57] pro základní operace. Příklad v

#### Listing 4.3: Příklad třídy kontroleru

```
@Component
public class DAOPerson extends DAOGeneric<Person> implements
    IDAOPerson {

    ...

    @Override
    @SuppressWarnings("unchecked")
    public Person getOsoba(String login) {
        Session session = null;
        Transaction tx = null;
        List<Person> persons = null;
        try {
            session = this.openSession();
            tx = session.beginTransaction();
            Query query = session.createQuery(
                "from " + Person.class.getName() + " p "
                + "where p.username = :login"
            );
            query.setParameter("login", login);
            persons = (List<Person>) query.list();
            tx.commit();
        } catch (HibernateException e) {
            handleException(e, tx);
        } finally {
            if (session != null) {
                session.close();
            }
        }
    }
}
```

```
    }

    if (persons == null || persons.isEmpty()) {
        throw new ItemNotFoundException("User '" + login
            + "' not found");
    }
    return persons.get(0);
}

...

@Override
public Person get(int id) {
    return (Person) this.get(Person.class, id);
}
}
```

---

### 4.1.3 Databáze

Příklad použití anotací na základě 4.4.

Listing 4.4: Příklad použití Hiberante

```
@Entity
@Table(name = "person")
@JsonSerialize(include = JsonSerialize.Inclusion.NON_NULL)
@JsonIgnoreProperties(ignoreUnknown = true)
public class Person {

    public static final int MAX_LENGTH_FIRST_NAME = 20;
    public static final int MAX_LENGTH_LAST_NAME = 20;
    public static final int MAX_LENGTH_LOGIN = 20;
    public static final int MAX_LENGTH_PASSWORD = 50;

    @Id
    @GeneratedValue
    private int id;

    @Column(length = MAX_LENGTH_FIRST_NAME, nullable = false,
        name = "name")
    private String name;

    @Column(length = MAX_LENGTH_PASSWORD, nullable = false)
    private String password;

    @Transient
    private String passwordRaw;

    @Column(nullable = false, name = "right_generate_token")
    @JsonIgnore
    private Boolean rightGenerateToken;
```

```
@OneToMany(mappedBy = "person", cascade =
    CascadeType.ALL, fetch = FetchType.EAGER)
private List<PersonToken> tokens;

@OneToMany(mappedBy = "owner", cascade = CascadeType.ALL,
    fetch = FetchType.EAGER)
private List<Contact> contacts;

@Column(length = MAX_LENGTH_LOGIN, nullable = false,
    unique = true, name = "login")
private String username;

@Column(length = MAX_LENGTH_LAST_NAME, nullable = false,
    name = "surname")
private String surname;

@ManyToOne
@JoinColumn(nullable = false, name = "state_id")
@JsonProperty(value = ApiConstants.STATE)
private PersonState state;

@OneToMany(mappedBy = "owner", cascade = CascadeType.ALL)
@JsonIgnore
private List<Project> projects;

@OneToMany(mappedBy = "owner", cascade = CascadeType.ALL)
@JsonIgnore
private List<Context> contexts;

....
}
```

---

Význam anotací dle [56].

Doplněním anotací k prvkům datového modelu a konfigurací hibernatu byla v MySQL vygenerována následující databázová struktura 4.1.

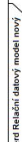
#### 4.1.4 Identifikace uživatele

Identifikace uživatele je realizována formou tokenu v hlavičce HTTP požadavku. Token lze získat pomocí dvou způsobů (viz. 3.4):

- *registrací* – po vytvoření účtu je ve vráceném objektu *osoby* obsažen také token
- *přihlášením* – autentifikace pomocí loginu a hesla uživatele, v takovém případě je zpět vrácen pouze token a login uživatele.

Při vytvoření uživatele je token vygenerován automaticky. V případě přihlášení je vrácen aktivní token uživatele. Pokud uživatel token nemá (např. token byl deaktivován) a má právo si token vytvořit, je mu automaticky vygenerován.

Pokud uživatel už aktivní token má, může použít dotaz číslo 17 z 3.3. API na základě tokenu rozpozná *osobu* a vrátí její objekt *osoby* obsahující všechny



tokeny.

Token je generován pomocí kódu v 4.5 (person je objekt typu Person).

Listing 4.5: Generování uživatelského tokenu

```
String stringToCrypt = person.toString() +
    PersonToken.tokenSalt + currentTimeMillis();
String hash = HashConverter.md5(stringToCrypt);
```

Samotná kontrola tokenu v příchozím požadavku je realizována pomocí implementace vlastního filtru[58] viz. 4.6.

Listing 4.6: Filter pro kontrolu tokenu

```
@Component
public class AuthenticationTokenProcessingFilter extends
    GenericFilterBean {

    ...

    @Override
    public void doFilter(ServletRequest request,
        ServletResponse response,
        FilterChain chain) throws IOException,
        ServletException, PermissionDeniedException {
        String token = ((HttpServletRequest)
            request).getHeader("token");
        if (token != null && !token.isEmpty()) {
            // validate the token
            try {
                if (tokenUtils.validate(token)) {
                    // determine the user based on the
                    (already validated) token
                    Person person =
                        tokenUtils.getUserFromToken(token);
                    // build an Authentication object with
                    the user's info
                    UsernamePasswordAuthenticationToken
                        authentication
                        = new
                            UsernamePasswordAuthenticationToken
                                (person.getUsername(),
                                    person.getPassword());
                    authentication.setDetails(new
                        WebAuthenticationDetailsSource()
                            .buildDetails(
                                (HttpServletRequest)
                                    request));
                    // set the authentication into the
                    SecurityContext
                    SecurityContextHolder.getContext()
                        .setAuthentication(
                            authManager
```

```
                .authenticate(authentication));
            }
        } catch (AuthenticationException e) {
            SecurityContextHolder.clearContext();
            customAuthEP.commence((HttpServletRequest)
                request, (HttpServletResponse) response,
                e);
        } catch (Throwable e) {
            SecurityContextHolder.clearContext();
            customAuthEP.commence((HttpServletRequest)
                request, (HttpServletResponse) response,
                new BadCredentialsException
                    ("BadCredentialsException:" +
                    e.getMessage()
                    , e));
        }
    }
    // continue thru the filter chain
    chain.doFilter(request, response);
}
}
```

---

#### 4.1.5 Facebook

Podoba adresy pro publikaci úkolu na Facebook je v tabulce 3.3 pod číslem 11. Z adresy je určen úkol, který se má publikovat a v těle požadavku je zaslán access token 3.2.1.2.

O publikování se stará třída FacebookPublisher využívající restfb 3.3.1.5. Její ukázka 4.7.

Kroky ke zpracování požadavku:

1. *přijetí požadavku*
2. *získání instancí uživatele a úkolu*
3. *kontrola, jestli byl požadavek už dříve publikován* – stejný úkol lze znovu odeslat po 24 hodinách od prvního odeslání
4. *vytvoření instance třídy facebookPublisher* – v konstruktoru je předán access token
5. *z úkolu je vytvořena textová zpráva*
6. *publikace zprávy pomocí restfb*
7. *kontrola vráceného id zprávy* – po vytvoření Facebook API vrátí identifikaci vzniklého objektu
8. *aktualizace úkolu o datu publikace a id zprávy*
9. *konec zpracování*

Listing 4.7: Ukázka ze třídy FacebookPublisher

```
public class FacebookPublisher {
```

```
private final FacebookClient facebookClient;

...

public FacebookPublisher(String accessToken) {
    this.facebookClient = new
        DefaultFacebookClient(accessToken,
            Version.VERSION_2_2);
}

...

public String publishSimpleMessage(String msgBody) {
    FacebookType publishMessageResponse = facebookClient
        .publish("me/feed", FacebookType.class,
            Parameter.with("message", msgBody));
    return publishMessageResponse.getId();
}

public String publishTask(Task task, String userMessage)
    throws GtdApiException {
    ...

    String id = null;
    try {
        id = publishSimpleMessage(msgText);
    }
    catch (Exception e) {
        ...
    }
    if (id == null) {
        throw new ...
    }
    return id;
}
...
}
```

---

#### 4.1.6 Google

Řešení je podobné verzi s Facebookem 4.1.5.

Implementace čerpá z ukázky na [59]. Využíváme Google knihoven. Ty umožňují autorizaci pomocí access tokenu a poskytují specializované rozhraní pro publikaci na API Google kalendáře.

O publikování se stará třída GooglePublisher. Její ukázka 4.8.

Kroky ke zpracování požadavku:

1. *přijetí požadavku*
2. *získání instancí uživatele a úkolu*

#### 4. REALIZACE

---

3. kontrola, jestli úkol nebyl dříve publikován a jedná se o úkol ve stavu v kalendáři
4. vytvoření instance třídy *googlePublisher* – v konstruktoru je předán access token
5. vytvořena instance *event* z dat úkolu
6. publikace zprávy pomocí *google api*
7. kontrola vráceného *id* eventu – po vytvoření Google API vrátí identifikaci vzniklého objektu
8. aktualizace úkolu o datu publikace a *id* eventu
9. konec zpracování

Listing 4.8: Ukázka ze třídy *GooglePublisher*

```
public class GooglePublisher {

    ...

    public GooglePublisher(String accessToken) {
        credential = new
            GoogleCredential().setAccessToken(accessToken);
    }

    public Event publishEvent(Event event) throws IOException,
        GeneralSecurityException {
        com.google.api.services.calendar.Calendar service =
            getCalendarService(credential);
        return service.events().insert("primary",
            event).execute();
    }

    public String publishTask(Task task, String userMessage)
        throws GtdApiException,
            GtdGoogleTaskAlreadyReportedException,
            GtdPublishInvalidTokenException {
        ...

        //object from google api
        Event event = new Event();
        ...

        String id = null;
        try {
            // Insert the new event
            Event createdEvent = publishEvent(event);
            id = createdEvent.getId();
        }
        catch (Exception e) {
            ...
        }
        ...
        return id;
    }
}
```



```

...

public static com.google.api.services.calendar.Calendar
getCalendarService(GoogleCredential credential) throws
    IOException, GeneralSecurityException {
    //Credential credential = authorize();
    HTTP_TRANSPORT =
        GoogleNetHttpTransport.newTrustedTransport();
    return new
        com.google.api.services.calendar.Calendar.Builder(
            HTTP_TRANSPORT, JSON_FACTORY, credential)
            .setApplicationName(APPLICATION_NAME)
            .build();
}
}

```

## 4.2 Mobilní aplikace pro Android

Dále si ukážeme části implementace aplikace pro centrální uložení dat.

### 4.2.1 Struktura projektu

Projekt respektuje konvenci pro adresářovou strukturu [60].

```

| libs ..... externí knihovny používané aplikací
|_ src ..... základní balík implementace projektu
|   |_ main ..... balík aplikace
|       |_ assets ..... soubory k použití
|       |_ java ..... složka s kódy aplikace
|           |_ cz.slama.android.gtd ..... balík GTD
|       |_ res ..... resourcy aplikace
|       |_ AndroidManifest.xml ..... manifest pro android aplikaci
|_ build.gradle ..... build soubor pro Gradle
|_ default.properties ..... property aplikace

```

Za zmínku stojí soubor `AndroidManifest.xml`. Android systém ho používá jako zdroj základních informací o aplikaci[61].

### 4.2.2 Klient pro konzumaci REST API

Retrofit 3.3.2.6 je potřeba nejdříve nakonfigurovat viz ??. V naší konfiguraci definujeme základní část URL, interceptor pro přidání tokenu 4.1.4 do HTTP hlavičky,

blbe reference

- *společná část URL pro všechny požadavky 3.4*

## 4. REALIZACE

---

- *errorhandler* – zavolá se při chybě požadavku (chyba spojení, chybový HTTP status [62], ...)
- *interceptor pro přidání tokenu* 4.1.4
- *úroveň logování*
- *parser pro data* – data jsou ve formátu JSON a pro parsování je použita knihovna GSON [63].

Listing 4.9: Konfigurace Retrofit

```
/**
 * Dagger module for setting up provides statements.
 * Register all of your entry points below.
 */
@Module(
    complete = false,
    injects = {
        ...
    }
)
public class BootstrapModule {

    ...

    @Provides
    RestErrorHandler provideRestErrorHandler(Bus bus) {
        return new RestErrorHandler(bus);
    }

    @Provides
    RestAdapterRequestInterceptor
    provideRestAdapterRequestInterceptor(UserAgentProvider
        userAgentProvider) {
        return new
            RestAdapterRequestInterceptor(userAgentProvider);
    }

    @Provides
    RestAdapter provideRestAdapter(RestErrorHandler
        restErrorHandler,

                                RestAdapterRequestInterceptor
                                restRequestInterceptor,
                                Gson gson) {
        return new RestAdapter.Builder()
            .setEndpoint(Constants.Http.URL_BASE)
            .setErrorHandler(restErrorHandler)
            .setRequestInterceptor(restRequestInterceptor)
            .setLogLevel(RestAdapter.LogLevel.FULL)
            .setConverter(new GsonConverter(gson))
            .build();
    }

    ...
}
```

Dalším krokem je specifikace rozhraní<sup>19</sup> pro volání konkrétního API (příklad je v 4.10). Rozhraní definuje adresu volaného API (k hodnotě je přidána společná URL) a parametry vstupu/ výstupu.

Listing 4.10: Definice rozhraní pro Retrofit

```
/**
 * User service for connecting the the REST API and
 * getting the users.
 */
public interface UserService {

    @GET(Constants.Http.URL_AUTH_FRAG)
    PersonAuth
        authenticate(@Path(Constants.Http.PARAM_USERNAME)
            String username,
            @Header(Constants.Http.PARAM_PASSWORD) String
                password);

    @POST(Constants.Http.URL_PERSONS_FRAG)
    Person createPerson(@Body PersonCreate personCreate);

    @GET(Constants.Http.URL_PERSON_BY_TOKEN_FRAG)
    Person getUsersByToken();

    @PUT(Constants.Http.URL_PERSON_FRAG)
    Person updatePerson(@Path(Constants.Http.PARAM_PERSON_ID)
        Integer personId, @Body Person person);
}
```

---

A konečně propojíme předešlé dva body 4.11. Příklad volání 4.12.

Listing 4.11: Třída pro práci s Retrofit

```
/**
 * Bootstrap API service
 */
public class BootstrapService {

    private RestAdapter restAdapter;

    ...

    //used

    private UserService getUserService() {
        return getRestAdapter().create(UserService.class);
    }
}
```

---

<sup>19</sup><https://docs.oracle.com/javase/tutorial/java/concepts/interface.html>

```
...

//*****
//*****      PERSON
//*****
public Person getPersonByToken() {
    return getUserService().getUsersByToken();
}

public Person updatePerson(int personId, Person person) {
    return getUserService().updatePerson(personId, person);
}

public PersonAuth authenticate(String email, String
    password) {
    return getUserService().authenticate(email, password);
}

public Person createPerson(PersonCreate personCreate) {
    return getUserService().createPerson(personCreate);
}
}
```

---

Listing 4.12: Příklad volání Retrofit

```
//serviceProvider.getService(getActivity()) - return object
//of type BootstrapService
serviceProvider.getService(getActivity()).getPersonByToken()
```

---

### 4.2.3 Asynchronní načítání dat

Práce s daty přes REST API je operace vyžadující čas. Takovou operaci nesmíme provádět v hlavním vlákne aplikace[64]. Protože spuštění náročné operace v hlavním vlákne vede z pohledu uživatele k „zamrznutí“ aplikace. Co hůře, pokud je vlákno vytíženo déle jak 5 s dojde k zobrazení hlášení „application not responding“. Řešením je asynchronní zpracování, které neblokuje hlavní vlákno, ale vytváří si vlastní.

Implementace asynchronního zpracování je součástí 3.3.2.2. Objekt se jmenuje *SafeAsyncTask*.

Příklad takového zpracování 4.13. Funkce *handleAction* je volána při stisku tlačítka pro akci u úkolu.

- *uživatel klikl na tlačítko pro akci u úkolu* – pro řešení této události je u tlačítka registrována funkce *handleAction*
- *zkontrolujeme, že asynchronní task neexistuje* – ochrana, abychom task nespustili vícekrát
- *vytvoříme task typu SafeAsyncTask* – překryjeme funkce podle zamýšleného chování

- *task spustíme*

Za zmínku stojí dvě cesty pro zpracování výjimek. Jedna je součástí samotného asynchronní části *onException*. V případě chyby při volání API ?? jsou volány zaregistrování posluchači pro konkrétní události. Požita je knihovna Otto[65]. V ukázce jsou posluchači funkce s anotací *@Subscribe*.

Listing 4.13: Příklad asynchronního volání

```
public class TaskActivity extends BootstrapActivity
    implements AdapterView.OnItemClickListener {

    ...

    public void handleAction(final View view) {
        //cant run more than once at a time
        if (actionTask != null) {
            return;
        }
        final Task task = getTaskFromForms();
        if (task == null) {
            return;
        }
        showProgress();
        actionTask = new SafeAsyncTask<Boolean>() {
            public Boolean call() throws Exception {
                cz.slama.android.gtd.model.Task taskResponse;
                switch (actionType) {
                    case TASK_CREATE:
                        taskResponse = bootstrapService.createTask(task);
                        break;
                    case TASK_UPDATE:
                        taskResponse =
                            bootstrapService.updateTask(task.getId(),
                                task);
                        break;
                    case TASK_DELETE:
                        bootstrapService.deleteTask(task.getId());
                        break;
                }
                return true;
            }
        };

        @Override
        protected void onException(final Exception e) throws
            RuntimeException {
            // Retrofit Errors are handled inside of the {
            if (!(e instanceof RetrofitError)) {
                final Throwable cause = e.getCause() != null ?
                    e.getCause() : e;
                if (cause != null) {
                    Toaster.showLong(TaskActivity.this,
                        getString(R.string.label_something_wrong));
                }
            }
        }
    }
}
```

#### 4. REALIZACE

---

```
        }
        actionTask = null;
    }

    @Override
    public void onSuccess(final Boolean authSuccess) {
        ReloadStatus.setTaskToReload(true);
        actionTask = null;
        goHome();
    }

    @Override
    protected void onFinally() throws RuntimeException {
        hideProgress();
        actionTask = null;
    }
};
actionTask.execute();
}

...

@Subscribe
public void onUnauthorizedErrorEvent(UnAuthorizedErrorEvent
    unauthorizedErrorEvent) {
    Toaster.showLong(TaskActivity.this,
        R.string.message_bad_credentials);
}

@Subscribe
public void onNetworkErrorEvent(NetworkErrorEvent
    networkErrorEvent) {
    Toaster.showLong(TaskActivity.this,
        R.string.message_bad_network);
}

@Subscribe
public void onRestAdapterErrorEvent(RestAdapterErrorEvent
    restAdapterErrorEvent) {
    Toaster.showLong(TaskActivity.this,
        R.string.message_bad_restRequest);
}

@Subscribe
public void onBadRequestErrorEvent(BadRequestErrorEvent
    badRequestErrorEvent) {
    Toaster.showLong(TaskActivity.this,
        R.string.message_bad_restRequest);
}

@Subscribe
public void
    onAlreadyReportedErrorEvent(AlreadyReportedErrorEvent
```

```

        alreadyReportedErrorEvent) {
            Toaster.showLong(TaskActivity.this,
                R.string.task_post_already_reported);
        }
    }
}

```

#### 4.2.4 Facebook

Podle postupu v 3.3.2.4 byla aplikace zaregistrována na Facebooku. Kromě názvu balíku a hlavní aktivity [66] je pro registraci potřeba získat hash klíče, kterým je aplikace podepsána [67]. Vzor pro získání hashe je v 4.14.

Listing 4.14: Získání hashe vývojového klíče Android prostředí

```

keytool -exportcert -alias androiddebugkey -keystore
%HOMEPATH%\android\debug.keystore | openssl sha1 -binary
| openssl base64

```

The screenshot shows the Facebook Developers console interface. On the left is a sidebar with navigation links: GTD, Dashboard, Settings, Status & Review, App Details, Roles, Open Graph, Alerts, Localize, and Canvas Payments. The main content area is titled 'Basic' and contains fields for App ID (1597950240491583), App Secret (0b614f1d96a7a2ce1b935e3733bfd7af), Display Name (GTD), Namespace, App Domains, and Contact Email (drugnanov84@gmail.com). Below this is an 'Android' section with fields for Google Play Package Name (cz.slama.android.gtd), Class Name (cz.slama.android.gtd.MainActivity), and Key Hashes (JEDH5BFqEAO/DZuO5FuZxNTsU=).

Obrázek 4.2: Registrace aplikace na Facebook

Po registraci je aplikaci přiděleno *App id* a *App secret*. Ty potřebujeme v následující konfiguraci podle [68]. Příklad našeho manifestu 4.15.

Listing 4.15: Konfigurace Facebooku v manifestu Android aplikace

```

<?xml version="1.0" encoding="utf-8"?>

<manifest ...>

```

#### 4. REALIZACE

---

```
...

<uses-permission
    android:name="android.permission.INTERNET"/>
...

<!--android:icon="@drawable/icon"-->
<application
    ...>

<meta-data android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version"/>

...

<!--Facebook data-->
<meta-data android:name="com.facebook.sdk.ApplicationId"
    android:value="@string/facebook_app_id"/>
<activity android:name="com.facebook.FacebookActivity"
    android:configChanges=
"keyboard|keyboardHidden|screenLayout|screenSize|orientation"
    android:theme="@android:style/Theme.Translucent.NoTitleBar"
    android:label="@string/app_name"/>
<provider
    android:authorities=
"com.facebook.app.FacebookContentProvider1597950240491583"
    android:name="com.facebook.FacebookContentProvider"
    android:exported="true"/>

</application>

</manifest>
```

---

Facebook SDK je třeba před použitím inicializovat 4.16.

##### Listing 4.16: Inicializace Facebook SDK

```
FacebookSdk.sdkInitialize(getApplicationContext())
```

---

Komponenta tlačítka od Facebooku, které spravuje přihlášení/ odhlášení uživatele, je *com.facebook.login.widget.LoginButton*.

Zbývajícím krokem je implementace pro získání access tokenu 4.20. Postup je následující.

- *specifikujeme požadovaná práva* – pomocí *setPublishPermissions* nastavíme požadovaná oprávnění, která budou uživateli předložena ke schválení
- *registrujeme callbacky*
- *získání access tokenu* – po úspěšném přihlášení dostáváme objekt typu *LoginResult* a z něj access token. Úspěšný login ale nemusí znamenat při-



dělení všech práv, o která jsme žádali. Proto kontrolujeme, jestli nedošlo k jejich zamítnutí.

- *uložení access tokenu* – access token ukládáme do tzv. shared preferences[69].

Listing 4.17: Získání access tokenu s použitím Facebook SDK

```
package cz.slama.android.gtd.ui;

import android.content.Intent;
import android.os.Bundle;
public class FacebookActivity extends BootstrapActivity {

    ...

    @Override
    protected void onCreate(final Bundle savedInstanceState) {
        ...
        //facebookLoginButton is instance of
        com.facebook.login.widget.LoginButton
        facebookLoginButton
        .setPublishPermissions(Arrays.asList("publish_actions"));
        LoginManager.getInstance().registerCallback(callbackManager,
            new FacebookCallback<LoginResult>() {
                @Override
                public void onSuccess(LoginResult loginResult) {
                    successLogin(loginResult);
                }

                @Override
                public void onCancel() {
                    cancelLogin();
                }

                @Override
                public void onError(FacebookException exception) {
                    errorLogin(exception);
                }
            });
    }

    ...

    private void successLogin(LoginResult loginResult) {
        ...
        if (loginResult.getRecentlyDeniedPermissions().size() >
            0) {
            ...
            return;
        }
        ...

        accessToken = AccessToken.getCurrentAccessToken();
    }
}
```

## 4. REALIZACE

---

```
...

    ShrPrefUtils.saveFacebookToken(accessToken.getToken());

    ...
}
}
```

---

### 4.2.5 Google

Postup přihlášení a získání access tokenu je podobný tomu u Facebooku 4.2.4. Dále si popíšeme důležité odlišnosti.

Budeme postupovat podle návodu [70].

Výsledkem registrace naší aplikace na google konzoli je 4.3. Pro získání fingerprintu byl použit postup uvedený na [71].

Obrázek 4.3: Registrace aplikace na Google

Do manifestu potřebujeme zanést verzi použitých Google play services[?] a práva aplikace pro správu účtů ??.

Listing 4.18: Konfigurace Android manifestu pro Google play services

```
<meta-data android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version"/>
```

---

Listing 4.19: Konfigurace Android manifestu pro získání práv ke správě účtů

```

<uses-permission
    android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission
    android:name="android.permission.MANAGE_ACCOUNTS"/>
<uses-permission
    android:name="android.permission.AUTHENTICATE_ACCOUNTS"/>
<uses-permission
    android:name="android.permission.USE_CREDENTIALS"/>

```

Stejně jako Facebooku budeme pro správu přihlášení používat komponentu od Googlu *com.google.android.gms.common.SignInButton*.

Kroky pro získání access tokenu podle 4.20 jsou následující.

- *přihlášení uživatele k jeho google účtu* – zajišťuje komponenta *com.google.android.gms.common.SignInButton*.
- *specifikujeme požadovaná práva* – v předešlém bodě nemůžeme ještě žádat o rozšířená práva pro naši aplikaci[72]. Práva si připravíme (proměnná *SCOPE\_TO\_USE*) a v dalším bodě o ně zažádáme.
- *žádost o rozšířená práva* – uživatel je přihlášen ke svému google účtu a máme připravená práva pro aplikace. Přes třídu *GoogleAuthUtil* o ně požádáme. Uživateli je zobrazeno dialogové okno s žádostí o jejich potvrzení.
- *získání access tokenu* – výstupem předešlého bodu je access token, který ukládáme do shared preferences[69].

Listing 4.20: Získání access tokenu přes Google play services

```

public class GoogleActivity extends BootstrapActivity
    implements
        GoogleApiClient.OnConnectionFailedListener,
        GoogleApiClient.ConnectionCallbacks
    , View.OnClickListener {

    ...

    private final static String CALENDAR_API_SCOPE =
        "https://www.googleapis.com/auth/plus.login
        https://www.googleapis.com/auth/calendar";
        //https://www.googleapis.com/auth/calendar

    private final static String GOOGLE_CALENDAR_API_SCOPE =
        "oauth2:" + CALENDAR_API_SCOPE;

    private static String SCOPE_TO_USE =
        GOOGLE_CALENDAR_API_SCOPE;

    ...

    /* Request code used to invoke sign in user interactions. */
    private static final int RC_SIGN_IN = 0;
    private static final int REQUEST_CODE_TOKEN_AUTH = 2;

```

#### 4. REALIZACE

---

```
/* Client used to interact with Google APIs. */
private GoogleApiClient mGoogleApiClient;

/**
 * True if the sign-in button was clicked. When true, we
 * know to resolve all
 * issues preventing sign-in without waiting.
 */
private boolean mSignInClicked;

/**
 * True if we are in the process of resolving a
 * ConnectionResult
 */
private boolean mIntentInProgress;

...

@Override
protected void onCreate(final Bundle savedInstanceState) {
    ...
    mGoogleApiClient = new GoogleApiClient.Builder(this)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .addApi(Plus.API)
        .addScope(Plus.SCOPE_PLUS_PROFILE)
        .build();

    // googleLoginButton is instance of
    com.google.android.gms.common.SignInButton
    googleLoginButton.setOnClickListener(this);
    ...
}

...

public void obtainToken() {
    AsyncTask<Void, Void, String> task = new AsyncTask<Void,
        Void, String>() {
        @Override
        protected String doInBackground(Void... params) {
            ...
            String accountName =
                Plus.AccountApi.getAccountName(mGoogleApiClient);
            token = GoogleAuthUtil.getToken(
                GoogleActivity.this,
                accountName,
                SCOPE_TO_USE);
            ...
        }
        return token;
    }

    @Override
```

```
        protected void onPostExecute(String s) {
            super.onPostExecute(s);
            processGoogleToken(s);
        }
    };
    task.execute();
}

private void processGoogleToken(String s) {
    ...
    ShrPrefUtils.saveGoogleToken(s);
    ...
}

...

@Override
public void onConnected(Bundle connectionHint) {
    mSignInClicked = false;
    Toast.makeText(this, "User is connected!",
        Toast.LENGTH_LONG).show();
    obtainToken();
}

@Override
protected void onActivityResult(int requestCode, int
    responseCode, Intent intent) {
    if (requestCode == RC_SIGN_IN) {
        if (responseCode != RESULT_OK) {
            mSignInClicked = false;
        }
        mIntentInProgress = false;

        if (!mGoogleApiClient.isConnected()) {
            mGoogleApiClient.reconnect();
        }
    }
    if (requestCode == REQUEST_CODE_TOKEN_AUTH &&
        responseCode == RESULT_OK) {
        Bundle extra = intent.getExtras();
        String oneTimeToken = extra.getString("authtoken");
    }
}

public void onConnectionSuspended(int cause) {
    mGoogleApiClient.connect();
}

...
}
```

---

### 4.3 DigitalOcean - instalace

V 3.3.1.8 jsme pro hostování zvolil server DigitalOcean. Nastavení hostingu je v tabulce 4.1.

Tabulka 4.1: Současné nastavení hostingu na serveru DigitalOcean tokenu

Operační systém:	Ubuntu 14.04 <sup>20</sup>
Databáze:	MySQL verze 5.5.41 <sup>21</sup>
Admin databáze:	phpMyAdmin verze 4.0.10deb1 <sup>22</sup>
Aplikační server:	WildFly verze 8.2.0.Final <sup>23</sup>

Nyní si projdeme postup pro jeho instalaci.

1. *vytvoření dropletu* – nejdříve je třeba vytvořit droplet. Postupujeme podle návodu [73]. Současně získáme přihlašovací údaje pro vzdálené připojení.
2. *vzdálené připojení náš server* – využijeme např. `mobaxterm`<sup>24</sup>
3. *instalace Javy* – návod na [74]
4. *instalace WildFly* – návod na [75]
5. *instalace MySQL* – návod na [75]
6. *instalace phpMyAdmin* – návod na [76]

### 4.4 GitHub

Pro zálohování zdrojových kódů a samotné bakalářské práce je použit Git Hub.

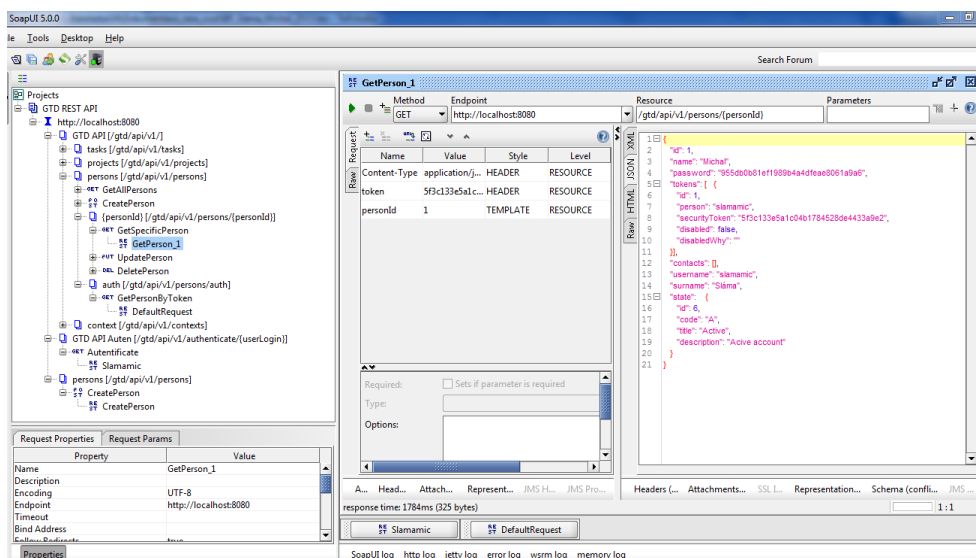
Repozitáře jsou na Git Hubu umístěny na adrese *github.com/Drugnanov*. Návod pro používání např. [77].

---

<sup>24</sup>`mobaxterm.mobatek.net`

## Testování aplikace

Pro testování REST API vznikl SOAP UI[78] projekt (viz. 5.1). Obsahuje seznam požadavků zasílaných na REST API. Parametry požadavku jako token nebo content-type jsou součástí nastavení zdrojů pro celou URL a podřízené požadavky je automaticky dědí.



Obrázek 5.1: Projekt SOAP UI pro testování REST API





---

## Návrhy k rozvoji

Zde uvádím další návrh pro rozvoj aplikace. Návrhy jsou seřazeny od neprioritnějšího.

Přejít do fáze testování s uživateli. Ověřit chování a fungování na reálných zařízeních. Získat zpětnou vazbu od uživatelů. Tu vyhodnotit a navrhnout další úpravy funkcí a vzhledu.

Využít HTML 5 [79] a s pomocí existujících frameworků (např. Sencha [80]) vytvořit aplikaci podporující více platforem.

Napojení na emailové schránky. Téměř každý člověk, který přijde do styku s internetem disponuje také emailovým účtem [81]. Současně právě přijetí nebo napsání emailu vytváří pro uživatele úkol. Proto dalším rozvojem je napojení na emailové servery (např. Gmail<sup>25</sup>, MS Outlook<sup>26</sup>, ...). Kteří to budou a co vše by takové napojení mělo znamenat, bude součástí další analýzy.

Opravit webovou aplikaci vzniklou v rámci předmětu SP2. Ta přestala fungovat na základě změn v aplikaci centrálního uložení dat souvisejících s touto bakalářskou prací.

---

<sup>25</sup>[mail.google.com](mailto:mail.google.com)

<sup>26</sup>[www.microsoft.com/cs-cz/outlook-com/](http://www.microsoft.com/cs-cz/outlook-com/)



---

## Závěr

Metodika *GTD* do mého života přinesla nový směr a podle něj se snažím řídit. Mysl potřebujeme použít k řešení problému a ne jako diář, který ještě funguje velmi podivně a události nám připomíná velmi nahodile. Diář si musíme vytvořit mimo ni. Na internetu nám k tomu pomůže celá řada produktů. Je jen na nás, který si vybereme. A volba je důležitá. Protože metodika nás nabádá, že musíme mít naprostou důvěru ke spolehlivosti svého systému a používat ho rádi. Plánování a revize úkolů se musí stát pravidelnou činností, ke které se nemusíme přemlouvat.

Z pohledu systému metodika definuje základní objekty typu projekt a úkol, jejich stavy a postup zpracování. Jedná se o jednoduchý proces, přičemž u systému je kladen důraz na jednoduchost ovládání a spolehlivost.

Existující řešení disponují širokým záběrem podporovaných platforem a přehledným ovládáním. Hlavní slabinu vidím v převážně chybějící podpoře pro integraci s dalšími programy (emaily, sociální sítě, ...).

Proto můj návrh neobsahuje speciální funkce pro osobní plánování, ale předpokládá integraci na Facebook a Google kalendář.

Aplikace pro centrální uložení dat je v současnosti nasazena na hostingu serveru DigitalOcean. Díky zvolené kombinaci technologií Spring/ Hibernate/ REST API byl její vývoj bez výrazných komplikací.

Aplikace pro OS Android je nasazena a testována na lokálním emulátoru. Testování probíhá ve spolupráci s mou rodinnou, kdy sbírám podněty k dalším úpravám. V blízké době dojde k nasazení na reálná zařízení a rozšíření testovacích uživatelů. První verzi aplikace vystavěnou na prázdném projektu jsem pro nepřehlednost kódu zahodil a s využitím Android Bootstrap3.3.2.2 projektu postavil novou. Byl to důsledek, že se jedná o mou první Android aplikaci a s tím je spojení učení za pochodu.

Integrace s Facebook a Google kalendářem je implementována a funguje. Uživatel může v Android aplikaci použít přihlášení ke svému Facebook/ Google účtů a publikovat vlastní úkol na zeď Facebooku nebo do kalendáře Googlu. Naplnil jsem navrhl k oddělené funkci publikování na dvě části a to získání

přístupových práv a samotné publikování. To představuje zjednodušení pro další vývoj uživatelských aplikací.

Získaných zkušeností si velice cením. Instalace a správa hostingu, REST API, technologie a principy Android aplikace společně s nastavením vývojového prostředí, napojení na Facebook a Google, . . . . To jsou zkušenosti, které určitě využiji ve svém dalším profesním životě a další práci na tomto projektu.

na čem je nasa-  
zeno, kolik lidí to  
používá a další  
rozvoj

---

## Literatura

- [1] Allen, D.: *Mít vše hotovo - Jak zvládnout práci i život a cítit se při tom dobře*. Brno: Jan Melvil Publishing, s.r.o., 2008, ISBN 978-80-903912-8-4.
- [2] Gregor, L.: Stručný průvodce po metodě GTD. [online], 2008, [cit. 2015-04-29]. Dostupné z: <http://www.mitvsehotovo.cz/2008/08/strucny-pruvodce-po-metode-gtd/>
- [3] toodledo: weird name, solid app. [online], 2012, [cit. 2015-04-29]. Dostupné z: <http://purplezengoa.com/2012/04/06/toodledo-weird-name-solid-app/>
- [4] Saunders, G. J. M.: Wunderlist — UI peculiarities. [online], 2013, [cit. 2015-04-29]. Dostupné z: <http://blog.garethjmsaunders.co.uk/tag/wunderlist/>
- [5] Toodledo. [online], 2015, [cit. 2015-05-11]. Dostupné z: [www.toodledo.com](http://www.toodledo.com)
- [6] Doit.im. [online], 2015, [cit. 2015-05-11]. Dostupné z: [doit.im](http://doit.im)
- [7] Wunderlist. [online], 2015, [cit. 2015-05-11]. Dostupné z: [www.wunderlist.com](http://www.wunderlist.com)
- [8] ToDoList. [online], 2015, [cit. 2015-05-11]. Dostupné z: [todoist.com](http://todoist.com)
- [9] Thin client. [online], 2015, [cit. 2015-04-29]. Dostupné z: [http://en.wikipedia.org/wiki/Thin\\_client](http://en.wikipedia.org/wiki/Thin_client)
- [10] Facebook. [online], 2015, [cit. 2015-05-05]. Dostupné z: <https://www.facebook.com/>
- [11] Social Media Update 2014. [online], 2015, [cit. 2015-05-05]. Dostupné z: <http://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users>

- [12] 10 Ways to Add Facebook Functionality to Your Website. [online], 2011, [cit. 2015-05-05]. Dostupné z: <http://www.socialmediaexaminer.com/10-ways-to-add-facebook-functionality-to-your-website>
- [13] Access Tokens. [online], 2015, [cit. 2015-05-10]. Dostupné z: <https://developers.facebook.com/docs/facebook-login/access-tokens>
- [14] Facebook developer. [online], 2015, [cit. 2015-05-05]. Dostupné z: <https://developers.facebook.com/>
- [15] Bais, T.: Facebook developer. [online], 2014, [cit. 2015-05-05]. Dostupné z: <http://examples.javacodegeeks.com/core-java/android-facebook-login-example>
- [16] Facebook Login for Android. [online], 2015, [cit. 2015-05-10]. Dostupné z: <https://developers.facebook.com/docs/facebook-login/android/v2.3>
- [17] Google Apps for Work. [online], 2015, [cit. 2015-05-10]. Dostupné z: [http://en.wikipedia.org/wiki/Google\\_Apps\\_for\\_Work](http://en.wikipedia.org/wiki/Google_Apps_for_Work)
- [18] By the Numbers: 10 Amazing Gmail Statistics. [online], 2015, [cit. 2015-05-10]. Dostupné z: <http://expandedramblings.com/index.php/gmail-statistics>
- [19] Android: using Google sign in to get access token. [online], 2014, [cit. 2015-05-10]. Dostupné z: <http://stackoverflow.com/questions/15098834/android-using-google-sign-in-to-get-access-token>
- [20] Google developers console. [online], 2015, [cit. 2015-05-10]. Dostupné z: <https://console.developers.google.com>
- [21] Start integrating Google Sign-In into your Android app. [online], 2015, [cit. 2015-05-10]. Dostupné z: <https://developers.google.com/identity/sign-in/android/getting-started>
- [22] Google+ Sign-in for Android. [online], 2015, [cit. 2015-05-10]. Dostupné z: <https://developers.google.com/+mobile/android/sign-in>
- [23] Getting Started with the Tasks API and OAuth 2.0 on Android. [online], 2013, [cit. 2015-05-10]. Dostupné z: <https://developers.google.com/google-apps/tasks/oauth-and-tasks-on-android>
- [24] OAuth 2.0 Playground. [online], 2015, [cit. 2015-05-10]. Dostupné z: <https://developers.google.com/oauthplayground>

- 
- [25] Dhingra, S.: REST vs. SOAP: How to choose the best Web service. [online], 2013, [cit. 2015-05-05]. Dostupné z: <http://searchsoa.techtarget.com/tip/REST-vs-SOAP-How-to-choose-the-best-Web-service>
- [26] Motamarri, J.: Compare RESTful vs SOAP Web Services. [online], 2014, [cit. 2015-05-05]. Dostupné z: <http://java.dzone.com/articles/j2ee-compare-restful-vs-soap>
- [27] SOAP vs. REST Challenges. [online], 2014, [cit. 2015-05-05]. Dostupné z: <http://www.soapui.org/testing-dojoworld-of-api-testing/soap-vs--rest-challenges.html>
- [28] Web Services Description Language (WSDL) 1.1. [online], 2001, [cit. 2015-05-05]. Dostupné z: <http://www.w3.org/TR/wsdl>
- [29] Web Application Description Language. [online], 2015, [cit. 2015-05-05]. Dostupné z: [http://en.wikipedia.org/wiki/Web\\_Application\\_Description\\_Language](http://en.wikipedia.org/wiki/Web_Application_Description_Language)
- [30] Swagger. [online], 2015, [cit. 2015-05-05]. Dostupné z: <http://swagger.io>
- [31] RESTful API Modeling Language. [online], 2015, [cit. 2015-05-05]. Dostupné z: <http://raml.org>
- [32] API Spec Comparison Tool. [online], 2014, [cit. 2015-05-05]. Dostupné z: <http://www.mikestowe.com/2014/12/api-spec-comparison-tool.php>
- [33] Francia, S.: REST Vs SOAP, The Difference Between Soap And Rest. [online], 2001, [cit. 2015-05-05]. Dostupné z: <http://spf13.com/post/soap-vs-rest>
- [34] Nene, A.: Web Services Architecture – When to Use SOAP vs REST. [online], 2014, [cit. 2015-05-05]. Dostupné z: <http://java.dzone.com/articles/web-services-architecture>
- [35] Daněk, B. A.: *Vyhodnocení a srovnání technologií pro servisně orientovanou architekturu*. Diplomová práce, České vysoké učení technické v Praze, 2014.
- [36] Create, read, update and delete. [online], 2015, [cit. 2015-05-05]. Dostupné z: [http://en.wikipedia.org/wiki/Create,\\_read,\\_update\\_and\\_delete](http://en.wikipedia.org/wiki/Create,_read,_update_and_delete)
- [37] RAML (software). [online], 2014, [cit. 2015-05-05]. Dostupné z: [http://en.wikipedia.org/wiki/RAML\\_\(software\)](http://en.wikipedia.org/wiki/RAML_(software))
- [38] Mulesoft. [online], 2014, [cit. 2015-05-05]. Dostupné z: [www.mulesoft.com](http://www.mulesoft.com)

- [39] Should You Use Spring Boot in Your Next Project? [online], 2015, [cit. 2015-05-11]. Dostupné z: <http://steveperkins.com/use-spring-boot-next-project>
- [40] Hibernate. [online], 2015, [cit. 2015-05-11]. Dostupné z: [hibernate.org](http://hibernate.org)
- [41] How to use existing Oracle sequence to generate id in hibernate? [online], 2010, [cit. 2015-05-11]. Dostupné z: <http://stackoverflow.com/questions/2155376/how-to-use-existing-oracle-sequence-to-generate-id-in-hibernate>
- [42] Allen, M.: restfb. [online], 2015, [cit. 2015-05-10]. Dostupné z: <http://restfb.com>
- [43] MySQL versus Oracle Features/Functionality. [online], 2008, [cit. 2015-05-11]. Dostupné z: [https://blogs.oracle.com/GeorgeTrujillo/entry/mysql\\_versus\\_oracle\\_features\\_functionality](https://blogs.oracle.com/GeorgeTrujillo/entry/mysql_versus_oracle_features_functionality)
- [44] What is WildFly? [online], 2015, [cit. 2015-04-29]. Dostupné z: <http://wildfly.org/about>
- [45] Hamlyn, M.: Should You Develop for Android, iOS or both? [online], 2014, [cit. 2015-04-29]. Dostupné z: <http://www.appmakr.com/blog/develop-android-ios/>
- [46] Comparison of mobile operating systems. [online], 2015, [cit. 2015-04-29]. Dostupné z: [http://en.wikipedia.org/wiki/Comparison\\_of\\_mobile\\_operating\\_systems](http://en.wikipedia.org/wiki/Comparison_of_mobile_operating_systems)
- [47] McCracken, H.: Who's Winning, iOS or Android? All the Numbers, All in One Place. [online], 2013, [cit. 2015-04-29]. Dostupné z: <http://techland.time.com/2013/04/16/ios-vs-android/>
- [48] Supporting Different Platform Versions. [online], 2015, [cit. 2015-04-29]. Dostupné z: <http://developer.android.com/training/basics/supporting-devices/platforms.html>
- [49] Dashboards. [online], 2015, [cit. 2015-04-29]. Dostupné z: <http://developer.android.com/about/dashboards/index.html>
- [50] Facebook SDK for Android. [online], 2015, [cit. 2015-05-10]. Dostupné z: <https://developers.facebook.com/docs/android>
- [51] Google Play Services. [online], 2015, [cit. 2015-05-10]. Dostupné z: <https://developer.android.com/google/play-services/index.html>
- [52] Setting Up Google Play Services. [online], 2015, [cit. 2015-05-10]. Dostupné z: <https://developer.android.com/google/play-services/setup.html>



- 
- [53] Retrofit. [online], 2015, [cit. 2015-05-10]. Dostupné z: <http://square.github.io/retrofit>
- [54] TAG, L.: Retrofit. [online], 2013, [cit. 2015-05-10]. Dostupné z: <http://stackoverflow.com/questions/16902716/comparison-of-android-networking-libraries-okhttp-retrofit-volley>
- [55] RAML: DDD (Delight-Driven Development) For APIs, Uri Sarid 20140716. [online], 2014, [cit. 2015-05-05]. Dostupné z: [https://www.youtube.com/watch?v=\\_RI5iZg0tis](https://www.youtube.com/watch?v=_RI5iZg0tis)
- [56] Hibernate Annotations. [online], 2010, [cit. 2015-05-05]. Dostupné z: [https://docs.jboss.org/hibernate/annotations/3.5/reference/en/html\\_single](https://docs.jboss.org/hibernate/annotations/3.5/reference/en/html_single)
- [57] Generic Dao Find All : DAO Generic DAO « Hibernate « Java. [online], [cit. 2015-05-05]. Dostupné z: <http://www.java2s.com/Code/Java/Hibernate/GenericDaoFindAll.htm>
- [58] Package org.springframework.web.filter. [online], 2015, [cit. 2015-05-05]. Dostupné z: <http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/filter/package-summary.html>
- [59] Java Quickstart. [online], 2015, [cit. 2015-05-10]. Dostupné z: <https://developers.google.com/google-apps/calendar/quickstart/java>
- [60] Directory Structure of an Android Project. [online], 2015, [cit. 2015-05-10]. Dostupné z: <http://www.compiletimeerror.com/2013/01/directorystructure-of-android-project.html>
- [61] App Manifest. [online], 2015, [cit. 2015-05-10]. Dostupné z: <http://developer.android.com/guide/topics/manifest/manifest-intro.html>
- [62] List of HTTP status codes. [online], 2015, [cit. 2015-05-10]. Dostupné z: [http://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_status\\_codes](http://en.wikipedia.org/wiki/List_of_HTTP_status_codes)
- [63] Singh, I.: Gson User Guide. [online], 2015, [cit. 2015-05-11]. Dostupné z: <https://sites.google.com/site/gson/gson-user-guide>
- [64] Processes and Threads. [online], 2015, [cit. 2015-05-11]. Dostupné z: <http://developer.android.com/guide/components/processes-and-threads.html>
- [65] Otto. [online], 2015, [cit. 2015-05-11]. Dostupné z: <http://square.github.io/otto>

- [66] Starting an Activity. [online], 2015, [cit. 2015-05-11]. Dostupné z: <http://developer.android.com/training/basics/activity-lifecycle/starting.html>
- [67] Signing Your Applications. [online], 2015, [cit. 2015-05-11]. Dostupné z: <http://developer.android.com/tools/publishing/app-signing.html>
- [68] Getting Started Android SDK. [online], 2015, [cit. 2015-05-11]. Dostupné z: [developers.facebook.com/docs/android/getting-started](http://developers.facebook.com/docs/android/getting-started)
- [69] SharedPreferences. [online], 2015, [cit. 2015-05-11]. Dostupné z: <http://developer.android.com/reference/android/content/SharedPreferences.html>
- [70] Android Quickstart. [online], 2015, [cit. 2015-05-11]. Dostupné z: <https://developers.google.com/google-apps/calendar/quickstart/android>
- [71] matreshkin: Get certificate fingerprint from android app. [online], 2014, [cit. 2015-05-11]. Dostupné z: <http://stackoverflow.com/questions/9293019/get-certificate-fingerprint-from-android-app>
- [72] Authorizing API requests. [online], 2015, [cit. 2015-05-11]. Dostupné z: <https://developers.google.com/+/api/oauth>
- [73] Sverdlov, E.: How To Create Your First DigitalOcean Droplet Virtual Server. [online], 2012, [cit. 2015-05-11]. Dostupné z: <https://www.digitalocean.com/community/tutorials/how-to-create-your-first-digitalocean-droplet-virtual-server>
- [74] Vlaswinkel, K.: How To Install Java on Ubuntu with Apt-Get. [online], 2014, [cit. 2015-05-11]. Dostupné z: <https://www.digitalocean.com/community/tutorials/how-to-install-java-on-ubuntu-with-apt-get>
- [75] gesker: WildFly 8.2.0 on Ubuntu 14.04. [online], 2015, [cit. 2015-05-11]. Dostupné z: <https://gesker.wordpress.com/2015/02/17/quick-install-wildfly-8-2-0-on-ubuntu-14-04>
- [76] Sverdlov, E.: How To Install and Secure phpMyAdmin on Ubuntu 12.04. [online], 2012, [cit. 2015-05-11]. Dostupné z: <https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-phpmyadmin-on-ubuntu-12-04>
- [77] Adding an existing project to GitHub using the command line. [online], 2015, [cit. 2015-05-11]. Dostupné z: <https://help.github.com/>

articles/adding-an-existing-project-to-github-using-the-command-line

- [78] SOAP UI. [online], 2015, [cit. 2015-05-11]. Dostupné z: [www.soapui.org](http://www.soapui.org)
- [79] HTML5. [online], 2015, [cit. 2015-05-11]. Dostupné z: <http://en.wikipedia.org/wiki/HTML5>
- [80] Sencha. [online], 2015, [cit. 2015-05-11]. Dostupné z: <http://www.sencha.com/>
- [81] van Rijn, J.: The ultimate mobile email statistics overview. [online], 2015, [cit. 2015-05-11]. Dostupné z: <http://www.emailmonday.com/mobile-email-usage-statistics>



## Seznam použitých zkratk



## Obsah přiloženého CD

	readme.txt.....	stručný popis obsahu CD
	exe .....	adresář se spustitelnou formou implementace
	src	
	impl.....	zdrojové kódy implementace
	thesis .....	zdrojová forma práce ve formátu L <sup>A</sup> T <sub>E</sub> X
	text .....	text práce
	thesis.pdf .....	text práce ve formátu PDF
	thesis.ps .....	text práce ve formátu PS