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

Bakalářská práce

Finanční rozšíření aplikace Redmine

Plzeň, 2014 Michal Mayer

Prohlášení

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

V Plzni dne 26. června 2014, Michal Mayer

Abstract

Hlavním cílem této práce je vytvoření finančního rozšíření aplikace Redmine. Ta slouží pro řízení projektů. V úvodu práce je proto vysvětlena problematika řízení projektů, zejména potom problematika řízení nákladů. Dále se práce zaměřuje na programovací jazyk Ruby a framework Ruby on Rails, které aplikace využívá. Poté je k aplikaci navrženo a vytvořeno finanční rozšíření, které je následně otestováno.

Abstract

The main purpose of this Bachelor thesis is development of a finance plugin for Redmine Application. This application is made for project management. Project management is explained at the beginning of this thesis, especially issue of cost management is mentioned. The thesis is further focused on programming language Ruby and framework Ruby on Rails. The finance plugin for the application is designed and developed in the practical part. The plugin is then tested on suitable data at the end of the thesis.

Poděkování

Tímto bych chtěl poděkovat Ing. Jindřichu Skupovi za vedení mé bakalářské práce, cenné rady při konzultacích a hlavně za ochotu a trpělivost při její realizaci.

Obsah

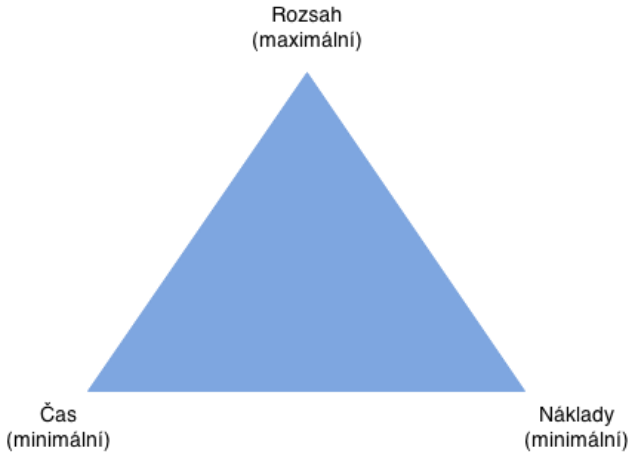
1. Úvod

V současné době je většina prací ve firmách a organizacích řešena pomocí projektů. Projekt může být v podstatě cokoliv - vývoj softwaru, zavedení nových podnikových procesů nebo např. výstavba budovy. Úspěšnost projektu závisí na jeho správném řízení. Řízení projektů rozlišuje tři základní oblasti řízení - rozsah, čas a náklady. Řízení projektů se stalo běžnou profesí a je vyučováno v mnoha kurzech po celém světě. Dnes existuje celá řada počítačových programů na podporu řízení projektů. Ovšem používání těchto programů není stejné jako řídit projekt, tyto programy pouze poskytují relevantní data, která nám umožňují lépe se rozhodovat. Jedním takovým programem je i velmi rozšířená aplikace Redmine.

Cílem této práce je seznámit čtenáře s problematikou řízení projektů, analyzovat samotnou aplikaci Redmine a možnosti jejího rozšíření. Z této analýzy budou poté vyhodnoceny možnosti rozšíření této aplikace o finanční řízení projektů. Práce dále bude obsahovat návrh a vypracování rozšíření. Rozšíření bude následně otestováno na vhodných datech.

2. Projekt

Základem projektového řízení je projekt. Projekt je jednorázová, časově omezená akce, jejímž účelem je vytvoření jedinečného produktu, služby nebo výsledku. Projekt musí mít jasně definovaný účel. Složitější projekty se rozdělují na menší části, podprojekty nebo etapy. Projekt má definované milníky, ty jsou nejčastěji mezi jednotlivými částmi projektu. Tyto milníky jsou vhodným místem pro kontrolu a zhodnocení projektu. Projekt je nejčastěji omezen rozsahem, časem a náklady. Toto omezení se často označuje jako projektový trojúhelník (Obr. 1). Každý projekt musí hledat rovnováhu mezi těmito protichůdnými omezeními. Pokud nejdůležitějším cílem je rozsah projektu, bude potřeba počítat s nárůstem nákladů a času. Naopak pokud je důležité dodržení nákladů, bude nutné zmenšit rozsah projektu. [1][2][3]



Obr. 1 – Projektový trojúhelník [4]

Subjekty podílející se na projektu jsou jednotlivci nebo organizace, které ovlivňují nebo jsou ovlivňováni realizací nebo výsledkem projektu. Mezi nejdůležitější subjekty patří:

- projektový manažer
  - zodpovídá za celkový projekt
  - vyhodnocuje stav projektu
  - sestavuje projektový tým
  - rozděluje úkoly mezi jednotlivé členy projektového týmu
  - plánuje termíny a rozsah jednotlivých prací
  - odpovídá za jednotlivé členy projektového týmu
  - podává hlášení o stavu projektu

- projektový tým
    - musí plně rozumět projektu
    - má jasně definované úkoly a termíny
    - jednotlivý členové mezi sebou musí komunikovat
- zákazník
    - určuje cíl projektu
    - stanovuje čas a rozpočet
    - musí být průběžně informován o výsledcích projektu pro případ nepochopení nebo změnu požadavků

3. Řízení projektů

"Řízení projektů je soubor modelů, metod, postupů, nástrojů a technik pro plánování a řízení realizace složitých projektů." [1]

Řízení projektů je možné definovat i jako použití znalostí, metod a nástrojů k úspěšnému splnění projektu. Projektový manažer využívá těchto technik k naplánování a řízení projektů. Řízení projektů se rozděluje do deseti tzv. oblastí řízení (Project Management Knowledge Areas). [1][2][3]

1. Řízení rozsahu
    - plánování a vedení veškeré práce nutné k dokončení projektu
2. Řízení času
    - odhadnutí časové náročnosti
    - naplánování časového harmonogramu
    - kontrola dodržování termínů
3. Řízení nákladů
    - plánování, odhadování, evidence a kontrola rozpočtu
4. Řízení kvality
    - kontrola plnění zadaných požadavků projektu
5. Řízení lidských zdrojů
    - výběr členů projektu
    - efektivní využití všech členů projektu
6. Řízení komunikace
    - získávání, sběr, výměna a uchovávání informací
7. Řízení rizika
    - hledání a vyhodnocování možných rizik projektu
8. Řízení nákupu
    - nákup zboží a služeb
9. "Řízení zainteresovaných stran" (Stakeholder management)

- hledání a vyhodnocování požadavků subjektů podílejících se na projektu nebo subjektů ovlivněných projektem

10. Řízení integrace

- propojení jednotlivých oblastí řízení

Tato práce se bude dále zabývat převážně řízením nákladů.

3.1. Řízení nákladů

Řízení nákladů zahrnuje procesy k úspěšnému splnění projektu ve schváleném rozpočtu. Projekty velmi často překračují svoje rozpočty. Studie publikovaná v *Harvard Business Review* v roce 2011 zkoumala přes 1500 IT projektů. Výsledkem bylo, že IT projekty průměrně překročí svůj rozpočet o 27 procent. [5]

Úspěšnost splnění rozpočtu nezávisí pouze na správném řízení nákladů, ale i na řízení rozsahu a času. Projekt musí mít velmi dobře naplánovaný rozsah prací a k nim vhodně odhadnutý čas k jejich splnění. Projekt by měl počítat i s rezervou pro případnou opravu chyb nebo změny v požadavcích. Čím lépe projektový manažer zvládne řízení rozsahu a času, tím lépe je možné následně naplánovat rozpočet projektu.[1][2][3]

Každý subjekt, který vede podvojné účetnictví, zpracovává tři finanční výkazy, ze kterých je možné získat informace o finanční situaci podniku.

1. Rozvaha

Rozvaha je finanční výkaz zpracovávaný k určitému dni, který zobrazuje přehled majetku subjektu (aktiva) a způsob jeho krytí (pasiva). Každá operace je tak zaznamenána na dvou účtech, pokaždé však na opačné straně (Má dáti nebo Dal). Tento způsob zobrazení se nazývá podvojné účetnictví. Aktiva se dělí na dlouhodobý a oběžný majetek a poté dále na hmotný, nehmotný a finanční, pasiva se dělí na vlastní a cizí zdroje. Rozvahu zobrazuje obr. 2. [6][7]

Rozvahu rozdělujeme na druhy:

- počáteční rozvaha - sestavuje se při založení podniku (ke dni zapsání do obchodního rejstříku)
- řádná rozvaha - sestavuje se vždy ke konci účetního období

- mimořádná - v případě potřeby se sestavuje mimořádná rozvaha (např. likvidace)

Aktiva	Pasiva
1. Stálá aktiva	1. Vlastní kapitál
Hmotný majetek	Základní kapitál
Nehmotný majetek	Fondy
Finanční majetek	Výsledek hospodaření
2. Oběžná aktiva	2. Cizí kapitál
Zásoby	Dlouhodobé závazky
Dlouhodobé pohledávky	Krátkodobé závazky
Krátkodobé pohledávky	Bankovní úvěry
Peněžní prostředky	

Obr. 2 – Rozvaha [4]

2. Výkaz zisku a ztrát

Výkaz zisku a ztrát porovnává náklady a výnosy a zobrazuje hospodářský výsledek za sledované období. Výkaz může být zkrácený nebo v plné verzi.

3. Cash-flow

Cash-flow porovnává příjmy a výdaje. Na rozdíl od výkazu zisku a ztrát cash-flow sleduje skutečný stav finančních prostředků, např. nákup zboží nijak nemění zisk nebo ztrátu, ale dojde k úbytku finančních prostředků.

Základem finanční analýzy projektu je cash-flow (finanční toky). Analýza cash-flow je sledování příjmů a výdajů v čase. Dobře naplánované cash-flow zabezpečuje dostatek finančních prostředků po celou dobu trvání projektu. Jednotlivé finanční toky můžeme rozdělit do tří kategorií - investiční, provozní a finanční. Výdaje můžeme dělit na fixní a variabilní.

Pro finanční řízení nepotřebujeme znát strukturu majetku, hospodářský výsledek, ani např. odpisy, ale pouze finanční toky. Proto finanční řízení nepracuje s rozvahou ani s výkazem zisku a ztrát, ale pouze s výkazem cash-flow. Finanční řízení rozšiřuje analýzu o porovnávání naplánovaného a skutečného cash-flow. [6][7][8]

4. Redmine

Redmine je webová aplikace pro správu projektů. Aplikace využívá framework Ruby on Rails. Jedná se o multiplatformní systém, Redmine podporuje většinu nejznámějších operačních systémů (Unix, Linux, Mac a Windows). Aplikace Redmine využívá modulární architekturu, každý projekt tak může obsahovat různé moduly. Mezi základní moduly patří správa úkolů, sledování času, novinky, dokumenty, soubory, wiki, úložiště, fórum, kalendář a Ganttův diagram. Aplikaci je možné dále rozšiřovat dalšími moduly (rozšíření, plugin), tvůrci aplikace spravují rozsáhlý adresář dostupných rozšířeni. Aplikace dále obsahuje správu uživatelů a uživatelských rolí. Jednotlivé uživatelské role mají definovaná různá práva.

Projekt obsahuje členy projektu a role, které v daném projektu mají. Dle těchto rolí mají uživatelé přístup k různým funkcím projektu. Jednotlivým členům projektu je možné přiřazovat úkoly. Úkoly mají kategorii, prioritu, status, datum začátku a konce, odhadovaný čas ke splnění a procenta splnění. Pokud je aktivní modul sledování času, uživatelé mohou přidávat informace, jak dlouho na daném úkolu pracovali. Tyto výkazy obsahují datum, počet hodin a aktivitu. Úkoly je možné také zobrazit pomocí kalendáře nebo Ganttova diagramu.

Každý projekt dále také obsahuje aktivitu a plán. Pomocí aktivity je možné sledovat vše, co se v daném projektu děje - změny úkolů, zprávy, úpravy wiki atd. Plán zobrazuje jednotlivé verze projektu, zbývající čas a míru jejich splnění v závislosti na odhadovaných časech jednotlivých úkolů. [9]

5. Ruby on Rails

Ruby on Rails (RoR) je framework, který umožňuje snadné a rychlé vytváření webových aplikací, využívající programovací jazyk Ruby. Ruby on Rails využívá třívrstvou architekturu Model-View-Controller. Ruby on Rails vytvořil David Heinemeier Hansson v roce 2003 a v současnosti ho vylepšují tisíce přispěvatelů a používají ho společnosti jako je Apple, Oakley, The New York Times, Twitter, ElectronicArts, Github nebo YellowPages.

RoR prosazuje tzv. "Ruby Way", jedná se o způsob programování, který je podle něj ten nejlepší. RoR upřednostňuje konvenci před konfigurací. Dodržování určitého způsobu programování velmi usnadňuje práci, protože RoR následně dokáže bez konfigurace odvodit vazby mezi jednotlivými objekty a databází. Mezi tyto způsoby patří např. CamelCase - jednotlivá slova s velkým písmenem bez mezer, snake_case – nahrazování mezer podtržítkem, adresářová struktura nebo struktura databáze. [10]

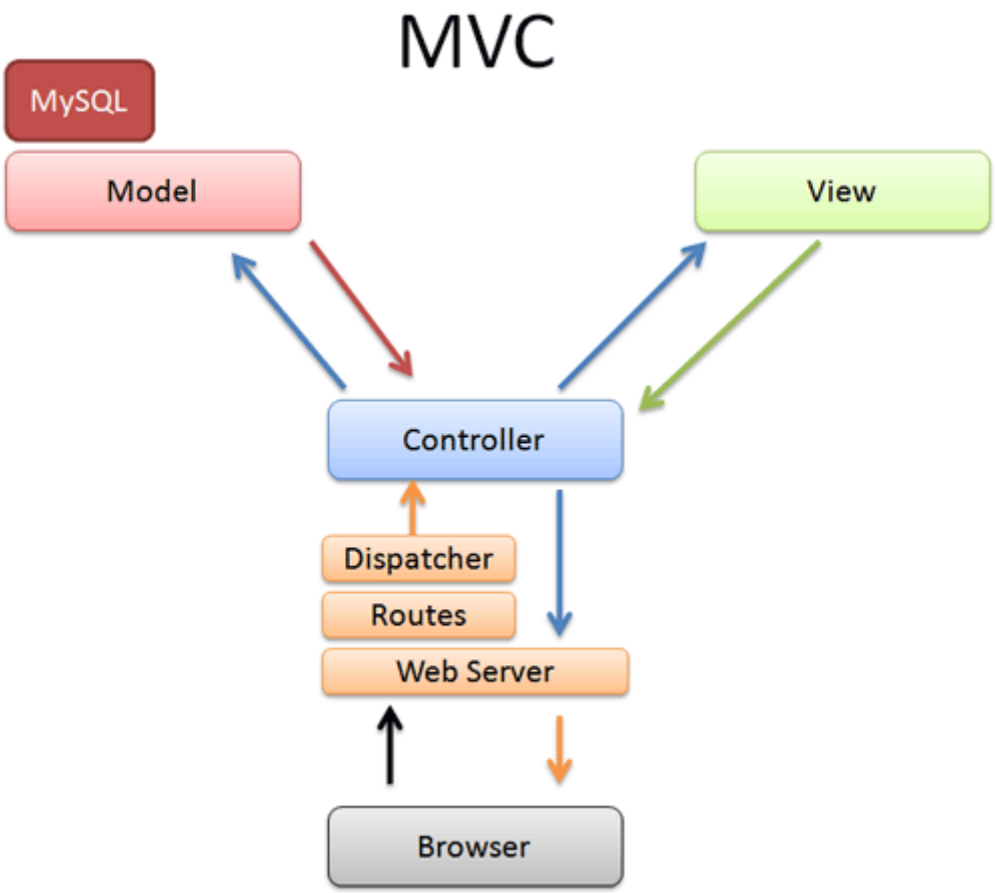
5.1. Model-View-Controller

Model-View-Controller (MVC) (Obr. 3) je nejjednodušší a nejpoužívanější návrhový vzor. Návrhový vzor MVC rozděluje program na tři části, které jsou od sebe navzájem odděleny, model - pro práci s daty a výpočtovou logiku, view – pro interpretaci dat a controller – pro vzájemnou komunikaci.

Model zajišťuje práci s daty. Jedná se o most mezi aplikací a datovým úložištěm. Model definuje strukturu dat a vazby na další modely. Model je správné místo pro finální validaci dat. Model před uložením dat kontroluje např. správný formát, jestli je číslo v daném intervalu apod. Úkolem modelu ovšem není jen načítat a ukládat data, ale zároveň se zde nachází i výpočetní logika např. výpočet průměru nebo sumy. Nikdy by se nemělo stát, že model vybere a předá controlleru všechna data a controller následně z těchto dat počítá průměr.

View zajišťuje zobrazení dat v určitém formátu, které dostane od controlleru. View může interpretovat data libovolnou formou na základě hlavičky HTTP (HTML, XML, JSON atd.). View nijak neupravuje daná data ani nevytváří nová. Např. součet dvou čísel neprobíhá ve view, ale v controlleru. View poté pouze zobrazí daný výsledek.

Controller reaguje na požadavky uživatele. Pokud uživatel provede nějaké změny, controller musí zajistit, aby model vypočítal nové hodnoty a došlo k vykreslení nového view. Když uživatel např. ve view zadá požadavek na řazení dat, view tento požadavek předá controlleru, ten následně požadavek vyhodnotí a požádá model o seřazená data, které následně předá do view. Controller obsahuje metody, které reprezentují dané požadavky. [11][12][13]

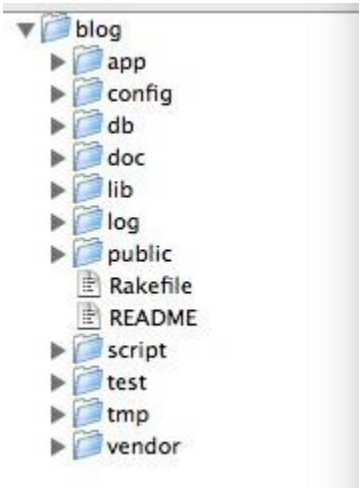


Obr. 3 – MVC [13]

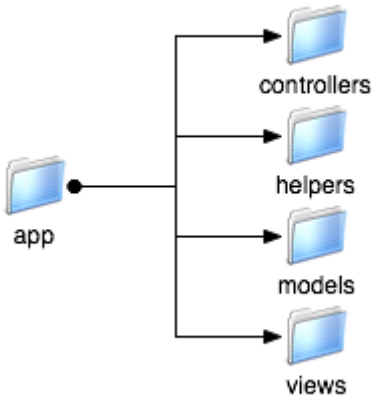
5.2. Adresářová struktura

Ruby on Rails používá přesně danou adresářovou strukturu (Obr. 4) a pojmenovávání souborů. Mezi nejdůležitější adresáře patří *app* (Obr. 5), který obsahuje modely, controllery a view, *config*, který obsahuje nastavení databáze a cest, *db*, který obsahuje databázové migrace, *lib*, který obsahuje knihovny a *test*, který obsahuje testy. Aplikace Redmine navíc obsahuje adresář *plugins*, kde se nachází adresáře s jednotlivými pluginy. Adresářová struktura pluginů je podobná samotné aplikaci. Nachází se zde adresáře *app, assets, config, db, lib* a *test*. Adresář *assets* slouží pro

uložení obrázků, stylů a javascriptu. Dále se zde nachází soubor *init.rb*, kde se nastavují základní informace o pluginu.[10]



Obr. 4 – Adresářová struktura [14]



Obr. 5- Složka app [15]

5.3. Model

V RoR každý model využívá knihovnu *ActiveRecord*. Model děděním třídy *ActiveRecord* získává všechny její metody a funkce. Pro název modelu se používá jednotné číslo s použitím *CamelCase*. Příslušná tabulka naopak používá snake_case, tedy pouze malá písmena a množné číslo, v případě mezery je použito podtržítko. Pokud budeme mít např. seznam produktů, model bude mít název *Product*, příslušná tabulka se bude jmenovat *products*. *ActiveRecord* přidává sloupečky *created_on* a *updated_on* a automaticky se stará o jejich aktualizaci. Zároveň přidává sloupeček *id* pro primární klíč.

Model definuje vazby na další modely a validace. Pro vazby se používá deklarace *has_many, has_on* nebo *belongs_to* v závislosti na druhu vazby a název modelu. Cizí klíč je definován jako *nazev_modelu_id*. Jelikož model je finální branou před databází je vhodné zde provádět validaci dat. Validace probíhá pomocí deklarace *validates* název atributu a podmínka. Validovat můžeme přítomnost dat, jestli se jedná o číslo, větší, menší atd.

Active Record využívá pro práci s databází objektově relační mapování (ORM). ORM zajišťuje automatický převod dat mezi objektově orientovaným programovacím jazykem a relační databází. Díky využití ORM není nutné psát databázové dotazy pomocí jazyka zvolené databáze. Tím se aplikace stává nezávislou na zvoleném

databázovém systému. Syntaxe pro databázové dotazy se skládá z názvu vybraného modelu a metoda pro přístup k datům. Máme-li model *Product,* výběr všech položek provedeme pomocí *Product.all.* Výběr pouze určitých produktů můžeme deklarovat pomocí *Product.where(:atribut => true).* Pokud má model definované vazby na jiné modely, výběr s těmito modely probíhá pomocí deklarace *includes: Product.includes(:model).* Pro uložení nového záznamu je nejprve vytvořena jeho instance *Product.new(:atributy)* a pomocí deklarace *save* dojde k jeho uložení do databáze. Pro aktualizaci záznamu v databázi se používá deklarace *update_attributes(:parametry).* [10]

5.4. View

View jsou šablony pro zobrazení dat z controlleru. Pro každý controller je určen stejnojmenný adresář ve složce */app/views/,* který obsahuje šablony pro jednotlivé akce controlleru. Každá metoda controlleru tudíž může mít vlastní view. Jednotlivé šablony mohou být různých typů např. *erb, builder, json, haml* a další. Šablony *erb* jsou psány s pomocí Ruby a HTML kódu a mají koncovku *.erb*. Šablony *builder* generují XML kód a mají koncovku *.builder*. V souborech *erb* je Ruby kód rozlišen znaky <% %> a <%= %>. První znak provede Ruby kód, ale nevrací žádné hodnoty např. podmínky nebo cykly. U druhého znaku navíc dochází k výpisu. Znak <%# %> slouží pro vkládání komentářů.

Ukázka erb souboru

```
<h2>Contact list</h2>
<% @contacts.each do |contact| %>
  <%= contact.name %><br>
<% end %>
```

Jednotlivá view se můžou skládat z menších částí, ty se nazývají *partials*. Použitím *partials* se view stávají přehlednější a snáze upravitelná - změna se provede pouze jednou. Pro rozlišení od view začínají *partials* podtržítkem, které se ovšem v kódu nevyskytuje.

Ukázka užití partials

```
<%# Zobrazeni souboru _menu.html.erb %>
<%= render "menu" %>
```

U webových stánek se často setkáváme s tím, že rozvržení prvků je většinou stejné, pouze se mění obsah. Nejčastěji zůstává stejná hlavička, patička a menu,

popřípadě boční sloupec. RoR toto řeší pomocí funkce *layout. Layout* je klasické *view*, které se nachází v adresáři */app/views/layouts/* Příkaz *yield* určuje kam se v *layout* má zobrazit aktuální *view*. Funkce *layout* se nastavuje v *controlleru*.[10]

Ukázka layout souboru

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<htmlxmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
  <div id="container">
    <div id="header">
      <%= render “header“ %>
    </div>
    <div id="content">
      <%= yield -%>
    </div>
    <div id="sidebar">
      <%= render “sidebar“ %>
    </div>
  </div>
</body>
</html>
```

Ukázka použití layout v controlleru

```
classCategoryController < ApplicationController

   layout 'basic'
...
```

5.5. Helper

V RoR existují pomocné třídy (helper), které usnadňují práci s řetězci, daty a formuláři. *FormHelper* slouží k vytváření formulářů propojených s modelem. Metoda *form_for* generuje HTML znaky pro jednotlivé prvky formuláře. Máme-li model *Category* a proměnou *@category* vytvořenou v controlleru, bude kód pro vytvoření nové kategorie vypadat následovně:

Ukázka použití form_for

```
<%= form_for @category, url: {action: "create"} do |f| %>
  <%= f.text_field : name %>
  <%= submit_tag 'Create' %>
<% end %>
```

Po odeslání formuláře je vytvořen hash *params[]*, ze kterého v controlleru vytvoříme nový objekt a pomocí funkce *save* vytvoříme nový záznam v databázi.

Ukázka vygenerovaného hashe

```
{"action" => "create", "controller" => "category", "category" => {" name" => "New category"}}
```

Ukázka vytvoření nové kategorie s pomocí form_for

```
@category = Category.new(params[:category])
@category.save
```

Mezi další pomocné třídy patří metody pro generování HTML znaků a metody na převod čísel. [10]

Ukázka pomocných tříd

```
stylesheet_link_tag "style"
# =><link href="/assets/style.css" media="screen" rel="stylesheet" />
javascript_include_tag "script"
# =><scriptsrc="/assets/script.js"></script>
image_tag("logo.png")
 # =><imgsrc="/assets/logo.png" alt="Logo" />
time_ago_in_words(3.minutes.from_now)
# => 3 minutes
link_to_function "Greeting", "alert('Hello world!')"
# =><a onclick="alert('Hello world!'); return false;" href="#">Greeting</a>
number_to_currency(1234567890.50)
# => $1,234,567,890.50
number_with_precision(111.2345, 2)
 # => 111.23
```

5.6. Controller

V RoR každý controller využívá knihovnu *ApplicationController*. V RoR existují dva druhy proměnných – globální a lokální. Lokální proměnné jsou přístupné pouze v dané metodě, globální proměnné jsou navíc přístupné z view. Globální proměnné na rozdíl od lokálních začínají znakem zavináče @. Pro přístup k předávaným parametrům, bez ohledu na HTTP metodu, slouží hash *params[:nazev_parametru]*. Hash je asociativní pole, tedy místo číselných indexů je možné používat text.

Bez jakékoli konfigurace, controller očekává model ve složce *app/models* s názvem *název_model.rb* a view ve složce *app/views/název_controlleru/* s názvem *název_metody.html.erb*. Pokud tedy chceme zobrazit všechny produkty, do globální proměnné je vložíme následovně: *@product = Product.all*.

5.7. REST

REST (Representational State Transfer) je architektura rozhraní, pro jednotný přístup ke zdrojům (*resources*), u RoR se používá pro směrování na jednotlivé akce controlleru. Zdroje mají identifikátor a REST definuje základní přístupy k nim (GET, POST, PUT, DELETE). RoR identifikuje data pomocí URL a přístup k nim pomocí http metody. V následující tabulce (Tab. 1) jsou data transakce a identifikátor *transaction*.[10]

Metoda http	URL	Akce
GET	/transaction/	Výpis všech transakcí
GET	/transaction/1	Výpis transakce s id 1
POST	/transaction/	Vložení nové transakce
PUT	/transaction/1	Aktualizace transakce s id 1
DELETE	/transaction/1	Smazání transakce s id 1

Tab. 1 – REST

URL pro aplikace, resp. přesměrování z URL na akci controlleru, se nastavuje v souboru */config/routes.rb*, pro každou URL je možné vytvořit identifikátor. Použitím deklarace *identifikátor_path*v kódu se bude automaticky generovat URL. RoR dále definuje dvě URL - */transaction/new* a */transaction/1/edit* k zobrazení formuláře pro vložení resp. úpravu transakce. Výsledný soubor může vypadat jako na Obr. 6. Pokud chceme použít tyto URL, můžeme použít deklaraci *resources* a identifikátor. [10]

Ukázka použití REST – soubor routes.rb

```
get ‘/transaction /‘, =>:to ‘transaction#index‘
get ‘/transaction /new/‘, =>:to ‘transaction#new‘
post ‘/transaction /‘, =>:to ‘transaction#create‘
get ‘/transaction /:id/‘, =>:to ‘transaction#edit‘
put ‘/transaction /:id/‘, =>:to ‘transaction#update‘
delete ‘/transaction /:id/‘, =>:to ‘transaction#delete‘
```

5.8. Migrace

Ruby on Rails podporuje migrace. Migrace obsahují postup změn dat. Může se jednat např. o úpravu obrázků nebo o změnu struktury tabulky databáze. Při úpravě databáze vytvoříme migraci, která obsahuje metody *up* a *down*. Up definuje změny, které provádíme, *down* definuje odebrání těchto změn. Změnou může být např. přidání sloupečku do tabulky, v metodě *down* poté bude odebrání sloupečku. Tímto lze snadno vrátit provedené změny v případě potřeby nebo problému. [10]

5.9. Generátor

Generátor usnadňuje vytváření aplikací. S jeho pomocí můžeme snadno vytvořit model, controller nebo view. Pro vytvoření modelu stačí pouze generátoru zadat parametry, které má model obsahovat a ten následně vytvoří soubor s daným modelem a migraci. Poté je nutné spustit migraci, která vytvoří tabulku v databázi. Pro vytvoření controlleru zadáme generátoru požadované metody. Při generování controlleru jsou vytvořeny i view pro každou metodu. [10]

6. Tvorba rozšíření

Rozšíření, která nejsou standardně součástí aplikace, se umisťují do adresáře *plugins*. Každé nové rozšíření obsahuje inicializační soubor *init.rb*. Tento soubor obsahuje základní informace a nastavení. Mezi základní informace patří jméno a popis rozšíření, jméno autora, číslo verze a odkaz na stránky rozšíření a autora. Dále se zde nastavují podmínky pro rozšíření. Podmínkou může být požadavek např. na určitou verzi aplikace Redmine nebo existenci jiného rozšíření. V tomto souboru se dále nachází deklarace oprávnění na jednotlivé akce controlleru. Jedno oprávnění můžu sloužit pro jednu nebo více akcí controlleru.

Dále tento soubor slouží pro přidání odkazů do menu. Na výběr je horní menu, které se nachází v levém horním rohu, uživatelské menu, které se zobrazuje, když se uživatel nenachází v projektu, projektové menu, které se naopak zobrazuje, když se uživatel v projektu nachází, uživatelské menu, které se nachází v pravém horním rohu a administrační menu, které se nachází na stránce administrace. Jednotlivé položky menu můžou mít různé možnosti. Odkazům můžeme nastavit pozici na začátek nebo konec menu, příp. před nebo po již existujícím odkazu. Odkaz dále můžeme zobrazit pouze za určitých podmínek, nejčastěji pokud je uživatel přihlášen nebo pokud má oprávnění na danou akci controlleru.

Takto nastavené rozšíření bude aktivní pro všechny projekty. Pokud deklaraci pro oprávnění vložíme do funkce *project_module*, vytvoří se modul, který umožní používat rozšíření pouze na určené projekty. Mezi další možnosti inicializačního souboru patří možnost navázání *hooku* a *patche*.

Každé z těchto nově přidaných rozšíření se chová jako samostatná aplikace, obsahuje tedy vlastní funkce a metody. Při vytváření rozšíření pro aplikaci Redmine se může stát, že původní kód aplikace nevyhovuje nebo neobsahuje např. potřebné vazby nebo metody. Úprava přímo zdrojového kódu aplikace není vhodná, ať už z pohledu složitějšího zprovoznění rozšíření nebo např. provádění této změny s každou změnou verze Redmine. Pro nezasahování přímo do zdrojového kódu slouží *hook* a *patch*.

Hook a *Patch* se ukládají do adresáře */app/lib/*. Funkce pro volání *hooku* se nachází na různých místech v aplikaci a umožňují na sebe navázat další funkce, které jsou vykonány, když je v průběhu zpracování kódu tento *hook* nalezen. Mezi základní *hook* patří zobrazení daného HTML kódu např. na stránce přehledu projektu. Pomocí

patche vkládáme metody přímo do daného modelu. V inicializačním souboru je volána funkce *require_dependency* na vložení požadovaného *hooku* nebo *patche*. V samotném *hooku* nebo *patchi* se poté nachází daná metoda a požadavek na její vložení. [9]

Ukázka použití patche

```
require_dependency 'model'

module ModelPatch
 …….
end

Model.send(:include, ModelPatch)
```

Rozšíření je možné lokalizovat do libovolného jazyka. Soubory s překlady se nachází v adresáři *config/locales/* a mají příponu *.yml*. Název souboru je shodný se zkratkou jazyka, tedy *cs* pro češtinu nebo *en* pro angličtinu. Překlad je definován jako *nazev_prekladu: "Překlad"*, výpis překladu se poté provádí pomocí funkce *l(:nazev_perekladu).*

7. Návrh řešení

Z předchozích kapitol víme, že mezi základní oblasti řízení patří řízení rozsahu, času a nákladů. Aplikace Redmine již obsahuje řízení rozsahu pomocí systému úkolů a řízení času pomocí přiřazení odhadovaných časů k jednotlivým úkolům a správou skutečně odpracovaných hodin. Aplikace ovšem nijak neřeší řízení nákladů. Přidání této funkce je úkolem finančního rozšíření. Toto rozšíření bude sledovat cash-flow daného projektu, včetně jeho verzí a podprojektů. Dále bude sledovat rozdíl mezi plánovaným a skutečným cash-flow a tím dávat přehled o finanční stránce jednotlivých projektů.

Aby bylo možné sledovat cash-flow je potřeba evidovat jednotlivé finanční toky. Pro porovnání plánu a reality budou muset být jednotlivé finanční toky rozděleny na plánované a reálné. Finanční toky tvoří buďto příjem nebo výdaj daného projektu. Pro evidování finančních toků budou použity transakce.

Transakce budou tvořit dva základní druhy příjmů a výdajů - platby za zboží nebo služby (externí práce, energie, platby za projekty apod.) a platby zaměstnancům. Aby transakce měly smysl, musí probíhat mezi dvěma subjekty. Prvním subjektem bude vždy daný projekt, druhým subjektem může být uživatel pro výplatu mzdy, bonusu apod. Dále pak jiný projekt např. pro půjčku apod. a nakonec firma nebo člověk mimo naši aplikaci. V prvních dvou případech už správa uživatelů a projektů v aplikaci existuje, pro třetí bude potřeba vytvořit seznam firem. U jednotlivých transakcí je dále potřeba rozlišit, jestli se jedná o příjem nebo výdaj a o plánovanou nebo skutečnou transakci. Pro rozlišení účelu dané platby bude možné transakce rozdělovat do volitelných kategorií.

Seznam firem bude zároveň sloužit jako seznam kontaktů. Jednotlivé firmy budou obsahovat kontaktní údaje, adresu, číslo účtu atd. Často je vhodnější kontaktovat ve firmě přímo konkrétní osobu, proto bude možné ke kontaktu přidávat kontaktní osoby, jednu kontaktní osobu bude možné nastavit jako výchozí, a ta se pak bude zobrazovat ve výpisu firem. Firmy bude možné na sebe hierarchicky vázat. K jednotlivým projektům bude možné přiřadit určité kontakty, které se u něj budou následně zobrazovat. V případě nějakého jednání s firmou nebo požadavků od firmy je vhodné vědět, jak je daná firma pro nás významná. Aby projektový manažer měl přehled o počtu a výši transakcí mezi projektem a danou firmou, bude u každého kontaktu zároveň i výpis všech uskutečněných transakcí.

Odměňovat zaměstnance lze několika způsoby. Částka může být předem určená za daný úkol nebo projekt nebo počítána z hodinové mzdy a z výkazu odpracovaných hodin. Odhadnutí odměny za celý projekt je velmi náročné a vytváření transakce za každý úkol je administrativní zátěž, proto budou platby zaměstnancům automaticky generované na základě hodinové mzdy a výkazu odpracovaných hodin. K tomu bude nutné vytvořit správu hodinových mezd. Každý zaměstnanec bude mít vlastní hodinou mzdu. Tato hodinová mzda bude mít platnost pro určité období, případně pouze platnost od určitého data bez omezení konce.

7.1. KPI

KPI je sada ukazatelů, které hodnotí určitá data nebo procesy. Vytvoření KPI má dvě fáze. Nejprve je nutné určit jaká data nebo procesy je vhodné hodnotit, poté stanovujeme způsob hodnocení. Hodnotit můžeme pomocí procentuálního vyjádření nebo pomocí intervalů. [16]

Tři základní ukazatele KPI pochází už ze samotného projektového trojúhelníku. Z důvodu omezení času a rozsahu musíme sledovat, jestli jsme danou práci schopni splnit v daném čase. Z důvodu omezení nákladů musíme sledovat, jestli je daný projekt v zisku a jestli je projekt plněn tak, jak bylo předem naplánováno. Proto bude rozšíření používat následující tři ukazatele:

- Poměr reálných příjmů a výdajů
- Poměr dnů zbývajících do konce verze a počet zbývající práce (na základě odhadovaných časů jednotlivých úkolů)
- Poměr plánovaného a reálného zůstatku

Hodnotit budeme pomocí tří intervalů, které budou reprezentovány pomocí semaforů. Pokud bude hodnota KPI menší než stanovené minimum, bude zobrazováno červené kolečko. V případě, že bude větší než stanovené maximum, bude kolečko zelené. Bude-li hodnota mezi, zobrazí se žluté kolečko.

V případě, že uživatel nenastaví vlastní hodnoty pro jednotlivé KPI bude použito výchozí nastavení. U poměru reálných příjmů a výdajů budou mezní hodnoty nastaveny na 1 a 1,2. Tedy pokud budou příjmy menší než výdaje, bude KPI červené. V případě, že příjmy budou o 20 procent vyšší, bude KPI zelené, pro poměr mezi hodnotami bude KPI žluté. U poměru zbývající dní a práce budou mezní hodnoty nastaveny na 0 a 10.

Pokud tedy procentuální vyjádření zbývající dnů bude menší než procentuální vyjádření zbývající práce, KPI bude červené. Jestli tento rozdíl bude větší než 10 procent, KPI bude zelené. V případě, že hodnoty budou mezi, KPI bude žluté. U poměru plánovaného a reálného zůstatku budou hodnoty nastaveny stejně jako u prvního KPI, tedy 1 a 1,2. Pokud tedy bude plánovaný zůstatek menší než reálný, KPI bude červené. V případě, že bude o 20 procent vyšší, KPI bude zelené. U poměru mezi hodnotami bude KPI žluté.

7.2. Oprávnění

Redmine umožňuje přiřazovat uživatelům různé role pro různé projekty. Role reprezentuje sadu práv k jednotlivým metodám controlleru. Redmine bohužel nijak neřeší globální práva. Správa kontaktů nebo hodinových mezd bude jednotná pro všechny projekty. Proto bude vytvořena vlastní metoda kontroly práv. Uživatel k nim bude mít přístup, pokud bude mít roli, která tento přístup umožňuje alespoň u jednoho projektu. V IT projektech lze identifikovat následující role a potřebná oprávnění:

- Projektový manažer - všechna práva
- Obchodní manažer - seznam kontaktů, seznam hodinových sazeb => pro určení nabídky
- Asistentka, účetní - práce s transakcemi, seznam kontaktů
- Grafik, programátor - kontakty v projektu, zobrazení vlastní hodinové mzdy

Na základě rozboru potřeb těchto rolí byla navržena následují struktura práv (Tab. 2).

Controller	Metoda
Seznam kontaktů	zobrazení seznamu, detail
	kontakt - přidání, změna, smazání
Seznam kontaktů v projektu	zobrazeni
	přidání a odebrání z projektu
Hodinová sazba	zobrazeni všech / pouze vlastní
	sazba - přidání, změna, smazání
Transakce	zobrazení seznamu, detail

	transakce - přidání, změna, smazání
Rozpočet	zobrazení

Tab. 2 - Práva

7.3. Rozložení

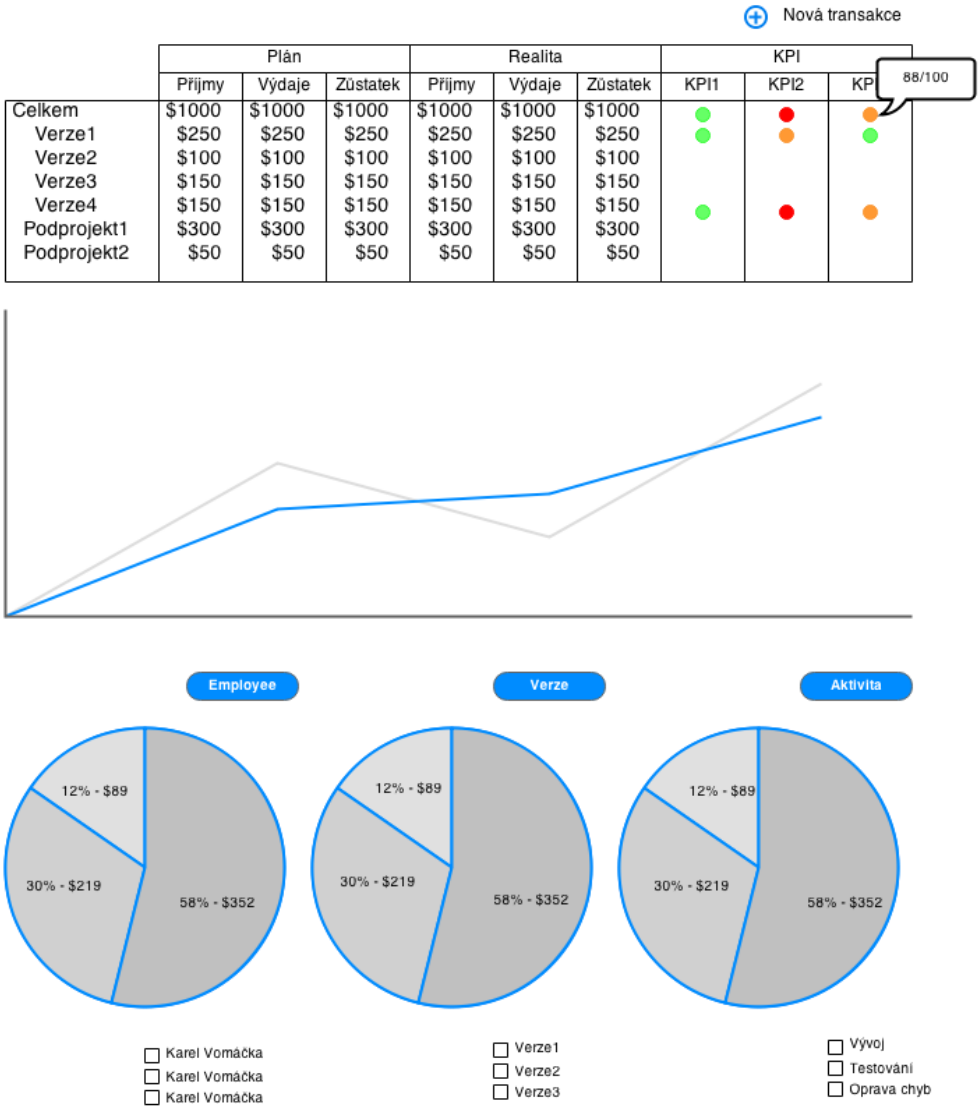
Pro správné řízení nákladů nestačí pouze zobrazení transakcí. Projektový manažer potřebuje zobrazovat různé pohledy na stejná data. Proto budou transakce zobrazovány nejen jako seznam, ale také pomocí tabulky a několika grafy, jejichž zobrazení bude volitelné v nastavení rozšíření.

Stránka rozpočtu bude obsahovat tři záložky. Dvě budou zobrazovat seznam transakcí a seznam výkazu hodin. Třetí záložka (Obr. 6) bude zobrazovat tabulku s přehledem součtů plánovaných a skutečných transakcí projektu včetně jeho verzí a podprojektů. Pod tabulkou bude spojnicový graf zobrazovat všechny příjmy a výdaje. Ten bude mít celkem šest datových sad- plánované příjmy, plánované výdaje, plánovaný zůstatek, reálné příjmy, reálné výdaje a reálný zůstatek. Projektového manažera nejčastěji zajímá reálné cash-flow, proto první tři datové sady budou standardně skryté. Pod spojnicovým grafem se bude dále nacházet několik základních koláčových grafů zobrazující poměry dle různých kritérií:

- reálné příjmy, výdaje zůstatek podle podprojektů a verzí daného projektu – Tento graf zobrazuje porovnání poměru jednotlivých verzí a podprojektů na příjmech, výdajích a zůstatku. Projektový manažer z tohoto grafu velmi snadno pozná, která verze nebo podprojekt generuje největší zisk, příp. ztrátu.
- reálné výdaje podle kategorií – Z tohoto grafu získáme přehled o výdajích dle kategorií. Následně můžeme vidět, jestli např. neplatíme příliš velkou část na režijní náklady oproti skutečné práci nebo jestli by nebylo vhodnější zajistit více prací externě nebo naopak.
- výdaje na jednotlivé zaměstnance (s možností přepnutí podle stráveného času) – Umožňuje ověřit správné rozdělení práce mezi jednotlivé členy projektu.
- výdaje podle jednotlivých aktiv všech zaměstnanců (s možností přepnutí podle stráveného času) – Graf zobrazuje porovnání jednotlivých aktivit členů projektu. Z tohoto grafu můžeme vidět, kolik práce bylo věnováno na jakou aktivitu. Projektový manažer může sledovat, jestli není příliš mnoho práce věnováno

např. opravám chyb. V tomto případě by bylo vhodné zamyslet se nad příčinou a popř. změnit přiřazení úkolů nebo nahradit některé členy projektu.

Z důvodu předchozích zkušeností byla pro zobrazování grafů použita JavaScriptová knihovna Highcharts, která umožňuje zobrazení interaktivních grafů různých druhů (spojnicový, sloupcový, koláčový atd.). Highcharts využívá JavaScriptovou knihovnu JQuery a je distribuován zdarma pro nekomerční využití.



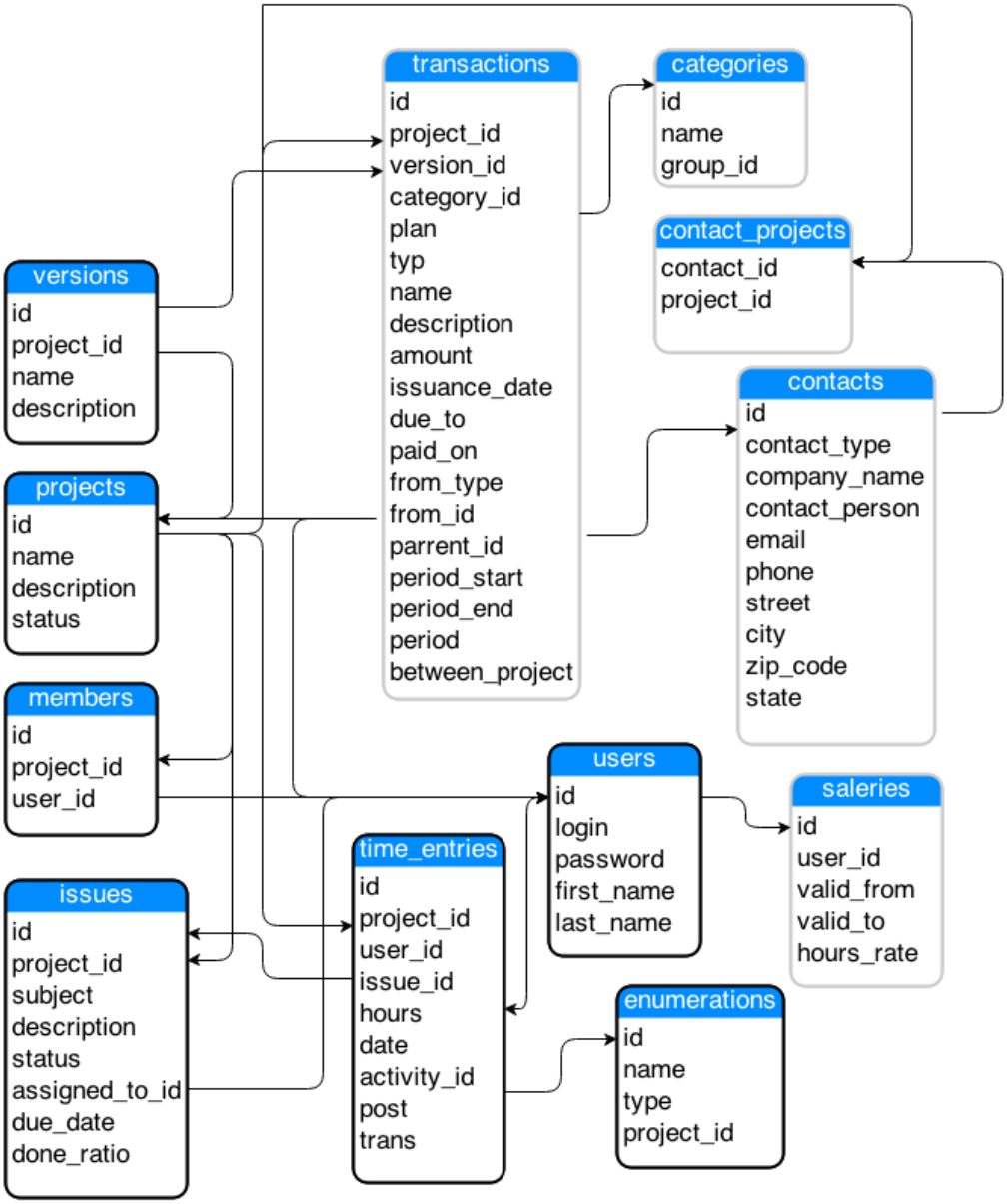
Obr. 6 – Návrh stránky rozpočtu [4]

8. Finanční rozšíření

8.1. ER diagram

Stávající aplikace bude rozšířena o pět modelů. Vazby mezi jednotlivými modely jsou zobrazeny na ER diagramu (Obr. 7) – černě orámované entity jsou součástí aplikace Redmine.

- Contact
  - seznam firem a kontaktních osob
  - contact_typ - rozlišení firmy a osoby
  - parent_id - navázání firmy nebo osoby na firmu
  - def - výchozí osoba pro zobrazení v seznamu firem
  - picture - adresa obrázku
- ContactProjekt
  - navázání kontaktu na projekt
- Salary
  - hodinová sazba zaměstnanců
  - valid_from - platná od data
  - valid_to - platná do data, případně NULL pro neomezenou platnost
  - hours_rate - hodinová sazba
- Category
  - kategorie transakcí
- Transaction
  - obsahuje vazbu na projekt, verzi a kategorii
  - typ - 0 příjem, 1 výdaj
  - plan - 0 plánovaná, 1 reálná
  - from_type - transakce mezi projektem a 0 - firmou, 1 - uživatelem, 2 - projektem



Obr. 7 – ER diagram [4]

8.2. Seznam kontaktů

Odkaz na seznam úkolů se nachází v levém horním menu. Úvodní obrazovka seznamu kontaktů (Obr. 8) zobrazuje seznam jednotlivých firem. U každého kontaktu se také zobrazuje kontaktní osoba, pokud nějaká existuje. Kontaktní osoba se zobrazuje náhodně, pokud není nastavena výchozí osoba. Zobrazení všech kontaktních osob je možné po kliknutí na šipku nacházející se na začátku řádku. V kontaktech je možné vyhledávat pomocí připraveného formuláře.

Obr. 8 – Seznam kontaktů [4]

Po kliknutí na název kontaktu se zobrazí detail daného kontaktu (Obr. 9) a všech jeho kontaktních osob. Pod informacemi o kontaktu se nachází výpis transakcí, které patří k danému kontaktu. Na pravé straně se dále zobrazuje obrázek resp. logo, pokud bylo ke kontaktu nahráno. Pod logem se nachází seznam navázaných kontaktů.

Obr. 9 – Detail kontaktu [4]

Pro snadnější práci je možné kontakty přiřazovat přímo k projektům. Zobrazení je stejné jako u výpisu všech kontaktů s tím rozdílem, že se tu nachází rozbalovací menu pro přepínání mezi všemi kontakty a kontakty přiřazenými k projektu a u každého kontaktu se nachází odkaz na přidání resp. odebrání daného kontaktu z projektu.

8.3. Hodinová sazba

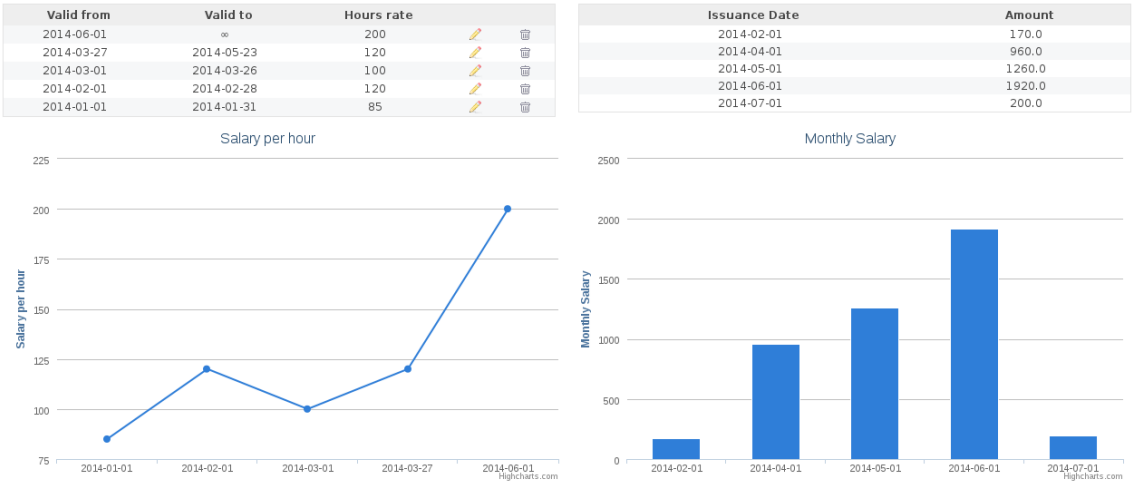
Hodinová sazba rozšiřuje model *User*. Každý uživatel může mít více hodinových sazeb, proto je model *User* rozšířen pomocí *UserPatch* o vazbu *has_many :salary*.

Ukázka použití UserPatch

```
require_dependency 'user'

module UserPatch
 defself.included(base)

   base.class_eval do
     unloadable
     has_many :salary
   end
```

```
 end
end

User.send(:include, UserPatch)
```

Odkaz na hodinovou sazbu se nachází stejně jako odkaz na seznam kontaktů v horním levém menu. Úvodní obrazovka seznamu hodinový sazeb zobrazuje seznam všech aktivních uživatelů včetně jejich aktuální hodinové mzdy. Na stránce detailu hodinové mzdy daného uživatele může uživatel přidávat hodinové mzdy pomocí zobrazeného formuláře. Pod formulářem se nachází dvě tabulky - hodinová mzda a měsíční platby. Zároveň zde dochází k zobrazení dvou grafů - vývoj hodinové mzdy a měsíční mzda (Obr. 10).



Obr. 10 – Hodinová mzda [4]

Před uložením nové sazby pro daného uživatele dochází ke kontrole, jestli se nová sazba nepřekrývá s některou již existující sazbou. Kontrola se mírně liší podle toho, jestli nová sazba má omezenou platnost nebo ne. U omezené platnosti je použitý vzorec *(StartA<= EndB) AND (EndA>= StartB)*. U neomezené platnosti je kontrolováno, že začátek nové je větší než konec již existujících: *valid_to>params[:salary][:valid_from]*. Po vytvoření sazby je volána metoda*create_timelogs.*

Při změně sazby probíhá kontrola překrývání s jinými sazbami stejně jako u metody *create*. Před uložením je volána metoda *delete_timelogs*, po uložení je volána metoda *create_timelogs*. Před smazáním sazby je volána metoda *delete_timelogs*.

Metoda *create_timelogs* vyhledá *time_entry*, které nejsou započítány do transakcí a spadají do dané hodinové sazby. Poté jsou do dané transakce započteny, příp. ji vytvoří. Metoda *delete_timelogs*vyhledá *time_entry*, které jsou započítány do transakce a spadají do dané hodinové sazby. Poté jsou od dané transakce odečteny, pokud je výsledná částka rovna nule, transakce je smazána.

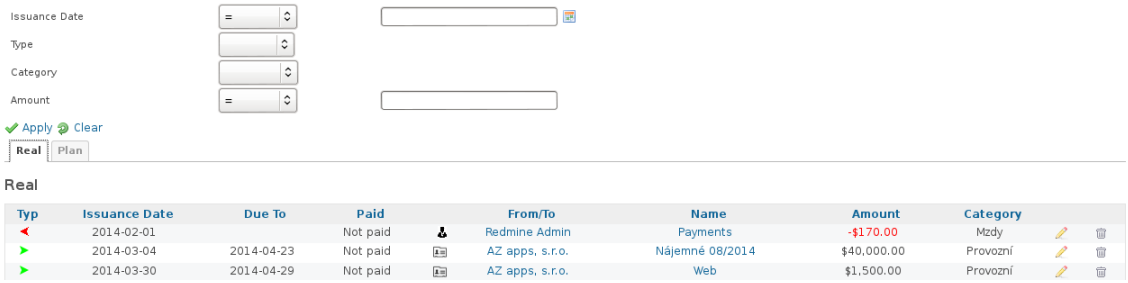
Hodinová sazba není vázaná na projekty, proto je autorizace prováděna vlastní metodou. Pomocí funkce *before_filter* zajistíme, že se daná metoda provede před metodou, která je požadována. Metoda pomocí funkce *User.current.allowed_to?* s parametrem :*global =>true* testuje, jestli má uživatel právo na danou metodu. Jestliže uživatel má právo na danou metodu alespoň v jednom projektu, je vrácena hodnota *true*, pokud ne pomocí *deny_acess* se zobrazí chybové hlášení. V případě, že uživatel přistupuje k metodě *index* a nemá právo zobrazení sazeb všech uživatelů, chceme mu umožnit zobrazení alespoň vlastní. Proto metoda *global_authorize*testuje, zda uživatel chce zobrazit svoji hodinovou mzdu, pokud ne je na ni přesměrován. Zároveň dochází k testování na právo pro přidání nové hodinové mzdy pro zobrazení formuláře.

Ukázka vytvořené autorizační funkce

```
before_filter :global_authorize

defglobal_authorize
  ifUser.current.allowed_to?({:controller => :salary, :action => :action_name}, nil,
{:global =>true})
   @right_to_new_salary = User.current.allowed_to?({:controller => :salary, :action =>
:new}, nil, {:global =>true})
   true
  elsifaction_name.to_s == "index" || action_name.to_s == "show"
   ifparams[:id].to_i == User.current.id &&action_name.to_s == "show"
     true
   else
     redirect_tosalary_path(:id => User.current.id)
   end
  else
    render_403
  end
end
```

8.4. Transakce

Seznam transakcí (Obr. 11) nachází na stránce rozpočtu a je rozdělen do dvou záložek podle typu transakce - reálné a plánované. U každé transakce jsou zobrazeny její základní informace - ikonka příjmu nebo výdaje, datum vystavení, splatnosti a zaplacení, původ (jestli je transakce vedena mezi projektem a kontaktem, uživatelem nebo jiným projektem), název a částka. U plánovaných transakcí je navíc odkaz na vytvoření reálné transakce, pokud ještě nebyla vytvořena. Po kliknutí na tento odkaz je zobrazen formulář na vytvoření nové transakce, který je předvyplněn podle dané plánované transakce. V transakcích je možné vyhledávat pomocí připraveného formuláře. Zároveň můžou být transakce řazeny podle jednotlivých polí.



Obr. 11 – Seznam transakcí [4]

8.5. Sledování času

Mzdy jsou vypočítávány z hodinové mzdy a ze záznamu o sledování času. Pro všechny *time_entries* je vytvářena souhrnná transakce za daný měsíc pro daného uživatele a to vždy k prvnímu dni následujícího měsíce. Proto při vytvoření, změně nebo smazání *time_entries* a vytvoření, změně nebo smazání hodinové mzdy, je nutné danou transakci vytvořit, upravit nebo smazat. Daný uživatel ovšem nemusí mít pro dané datum nastavenu hodinovou mzdu, proto pro rozlišení již započítaných *time_entries* je model *time_entries* rozšířen o hodnotu *post*, která obsahuje *true* nebo *false* a atribut *trans*, který obsahuje *id* transakce.

Aby pro započítání *time_entries* nedošlo k zásahu do kódu samotné aplikace Redmine, je použit *TimeEntryPatch*, který modelu *TimeEntry* přidá metody *increase_transaction* a *decrease_transaction*, které přidávají nebo ubírají z transakce částku z *time_entries*. Před uložením upraveného *time_entry* je potřeba ho odebrat z dané transakce, proto pomocí funkce *before_save* je volána metoda *decrease_transaction*. Po uložení, ať už nového nebo upraveného *time_entry* je pomocí

funkce *after_save* volána metoda *increase_transaction*. Před smazáním je pomocí metody *before_destroy* volána metoda *decrease_transaction*.

Transakce je přiřazena k verzi podle úkolu, ke kterému je vytvářeno *time_entry*. Proto při úpravě úkolu musí být zajištěna i případná změna verze transakce. K tomu je využit *IssuePatch*, který pomocí funkce *after_save* kontroluje po uložení úkolu, jestli došlo k změně verze a případně aktualizuje transakci.

Ukázka IssuePatch

```
require_dependency 'issue'

module IssuePatch
 defself.included(base)
   base.extend(ClassMethods)

   base.send(:include, InstanceMethods)

   base.class_eval do
    unloadable
    after_save :check_version
   end

 end
defcheck_version
  ifself.id.present?
     iss = Issue.find(self.id)
     entry = TimeEntry.where(:issue_id =>iss,:post =>true).first()
     trans = Transaction.find(entry.trans)
     unlessiss.fixed_version_id == trans.version_id
        entries = TimeEntry.where(:issue_id =>iss,:post =>true).uniq{|p| p.trans}
        entries.each do |e|
           ife.trans.present?
              trans = Transaction.find(e.trans)
              trans.update_attribute(:version_id, iss.fixed_version_id)
           end
        end
     end
  end
end

Issue.send(:include, IssuePatch)
```

Zobrazení *time_entries* je rozděleno na dvě části. V rozpočtu daného projektu jsou v záložce *Timelogs* zobrazeny součty z výkazů o sledování času pro jednotlivé uživatele za celý projekt a za aktuální měsíc. Po kliknutí na určitého uživatele se zobrazí

detailní výpis všech výkazů (Obr. 12) daného uživatele včetně hodinové mzdy a celkové částky pro daný výkaz a seznam celkové měsíční částky za výkazy. V případě, že uživatel nemá pro daný výkaz nastavenou hodinovou sazbu je vypsána chybové hlášení s odkazem na přidání nové sazby.

Date	Issue	Activity	Hours	Hour rate	Sum
2014-06-01	-	Design	1.0	200	200.0
2014-05-25	-	Design	1.0	Salary missing	
2014-05-04	issue2	Design	1.0	120	120.0
2014-05-04	issue2	Design	1.0	120	120.0
2014-01-28	issue2	Design	3.0	85	255.0
2014-01-28	issue3	Design	15.0	85	1275.0
2014-01-28	issue2	Development	2.0	85	170.0

Issuance Date	Amount
2014-07-01	200.0
2014-06-01	240.0
2014-02-01	1700.0

Obr. 12 – Sledování času [4]

8.6. Nastavení

Každý plugin je možné konfigurovat. Hodnoty nastavení pluginu jsou uloženy v hashi, které jsou přístupné pomocí *Setting.plugin_budget[:nazev_hodnoty]*. Inicializace nastavení probíhá v souboru *init.rb*. Úprava hodnot se provádí v administracina záložce doplňky v sekci konfigurace. V nastavení pluginu jsou uchovávány mezní hodnoty pro KPI, kategorie transakcí a zobrazované grafy.

Ukázka použití nastavení – soubor init.rb

```
settings :default => {'empty' =>true}, :partial => 'settings/budget_settings'
```

8.7. Rozpočet

Rozpočet je rozdělen na tři části - přehled, transakce a výkazy o sledování času. Přehled zobrazuje transakce několika různými pohledy. Hlavní část přehledu tvoří tabulka se součty za jednotlivé verze, podprojekty a za celý projekt. Tabulka je dále doplněna zobrazením indikátorů KPI. Pod tabulkou se nachází lineární graf se šesti datovými řadami. Pokud je v dané datové řadě a v daný den nějaká transakce, je tento bod v grafu zvýrazněn. Po kliknutí na tento bod jsou zobrazeny transakce pro daný den. Přehled je doplněn o koláčové grafy. Rozpočet je dále možné zobrazit pouze pro danou verzi nebo projekt bez podprojektů. V případě, že uživatel nemá pro daný výkaz nastavenou hodinovou sazbu je vypsána chybové hlášení s odkazem na přidání nové sazby.

9. Instalace a konfigurace rozšíření

9.1. Instalace rozšíření

Rozšíření je instalováno na operační systém Debian GNU/Linux s použitím Ruby verze 1.9.3 a Redmine verze 2.5. Konfigurace slouží pro připravení rozšíření na testování.

Instalace rozšíření je velmi rychlá a snadná. Složku s rozšířením uložíme do adresáře */instalace_redmine/plugins/*. Poté v terminálu spustíme pomocí příkazu *rake redmine:plugin:migrate* migraci pluginu, která provede potřebné změny v databázi. Tímto je rozšíření nainstalováno.

9.2. Konfigurace rozšíření

Po nainstalování provedeme v administraci konfiguraci rozšíření. Konfigurace závisí na potřebách daného projektu, popř. projektového manažera. Tato konfigurace bude sloužit pro potřeby testování. Na záložce doplňky vidíme seznam rozšíření. Na stránce konfigurovat nastavíme hodnoty pro KPI a vytvoříme tři kategorie, pro rozlišení účelu plateb:

- Mzdy - pro odlišení nákladů na pracovníky
- Provozní - nájem apod.
- Projektové - náklady spojené s projektem - potřebné licence, hosting apod.

Na záložce *Uživatelé* vytvoříme potřebné uživatele, v tomto projektu budou tři členové - grafik, programátor a programátor manažer. Na záložce *Role a práva* upravíme jednotlivé role, manažer bude mít všechna práva, grafik a programátor pouze právo pro zobrazení kontaktů v projektu. Na záložce *Seznamy* můžeme upravit aktivity pro výkazy odpracovaných hodin. Pro tento projekt budeme používat celkem tři aktivity - Design, Vývoj a Oprava chyb. Na stránce *Salaries* nastavíme hodinové mzdy pro jednotlivé zaměstnance a na stránce *Contacts* vytvoříme kontakty.

10. Testování

Finanční rozšíření bylo testováno na reálném projektu vytvoření webové stránky. Účelem bylo ověřit nejen správné fungování aplikace, ale zároveň i relevantnost zobrazovaných informací.

Cílem tohoto projektu je vytvoření webové stránky pro taneční skupinu Inspiral. Částka za vytvoření stránky byla stanovena na 20 000,- korun, stránka musí být hotova do jednoho měsíce. Projekt bude rozdělen na tři etapy vždy po deseti dnech - design, vývoj a rezerva. Celkový zisk je stanoven na 5000,- korun. Náklady byly stanoveny takto:

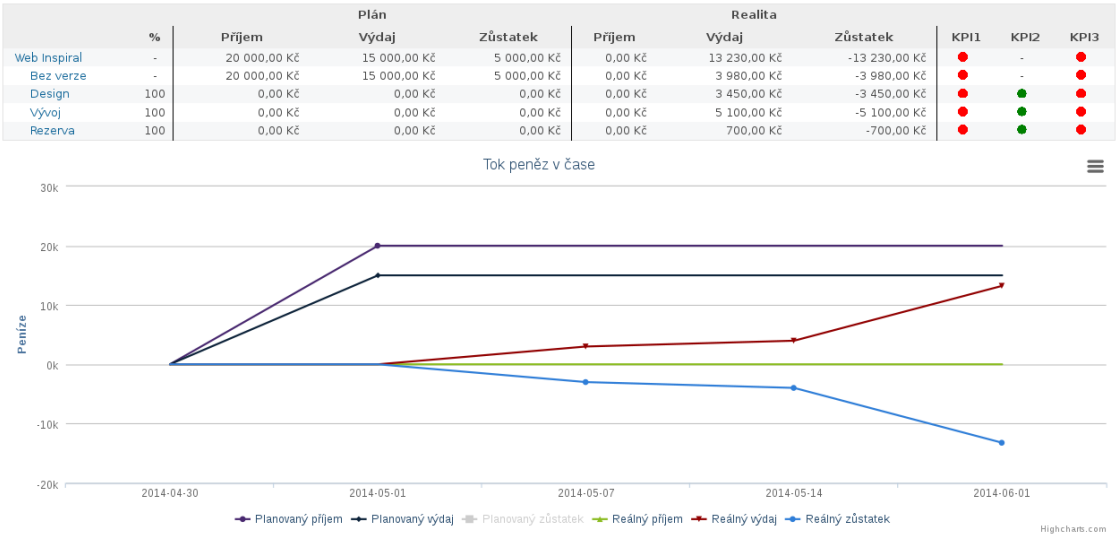
- 3000,- - poměrná část na provozní náklady (nájem, energie, internet)
- 1500,- - webhosting a doména
- 10 500,- - náklady na vytvoření - 70 hodin x 150 Kč/hod

Nyní vytvoříme samotný projekt, jeho verze a úkoly včetně odhadovaných hodin na vytvoření. Při vytváření nového projektu musíme pro zapnutí rozšíření vybrat modul *Budget*. Po zadání plánovaných transakcí vypadá rozpočet jako na Obr. 13.

Obr. 13 – Rozpočet po zadání plánovaný transakcí [4]

V průběhu projektu byla zaúčtována platba za webhosting a platba za provozní náklady. V reálném čase bylo možné sledovat růst mezd. Avšak náklady byly po celou dobu nižší než plánované. První a třetí KPI byly celou dobu červené, protože jediná

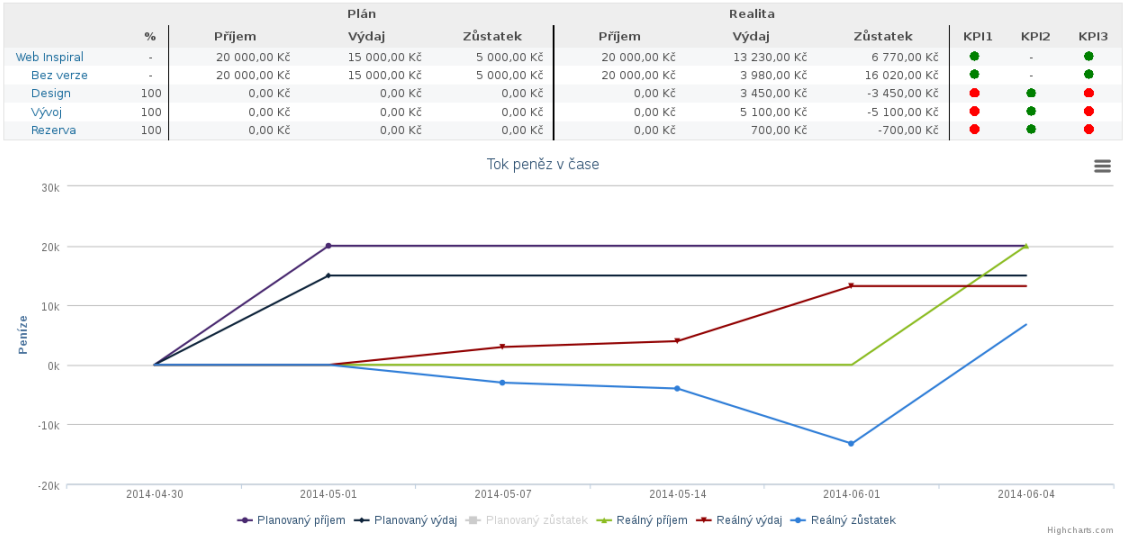
příjmová platba byla provedena až po dodání stránek (Obr. 14). Na konci projektu je patrné, že reálné výdaje byly o necelé dva tisíce menší.



		Plán			Realita					
	%	Příjem	Výdaj	Zůstatek	Příjem	Výdaj	Zůstatek	KPI1	KPI2	KPI3
Web Inspiral	-	20 000,00 Kč	15 000,00 Kč	5 000,00 Kč	0,00 Kč	13 230,00 Kč	-13 230,00 Kč	●	-	●
Bez verze	-	20 000,00 Kč	15 000,00 Kč	5 000,00 Kč	0,00 Kč	3 980,00 Kč	-3 980,00 Kč	●	-	●
Design	100	0,00 Kč	0,00 Kč	0,00 Kč	0,00 Kč	3 450,00 Kč	-3 450,00 Kč	●	●	●
Vývoj	100	0,00 Kč	0,00 Kč	0,00 Kč	0,00 Kč	5 100,00 Kč	-5 100,00 Kč	●	●	●
Rezerva	100	0,00 Kč	0,00 Kč	0,00 Kč	0,00 Kč	700,00 Kč	-700,00 Kč	●	●	●

Obr. 14 – Rozpočet před příjmovou platbou [4]

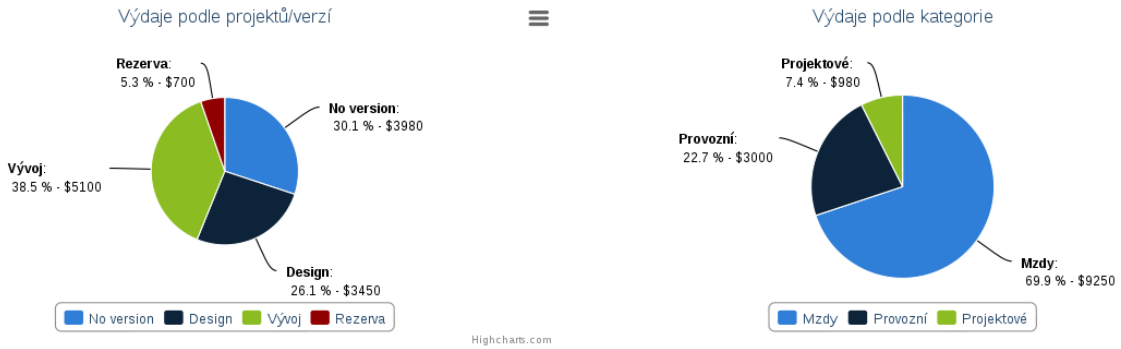
Po zaúčtování příjmové platby jsou již první a třetí KPI zelené, plánované a reálné příjmy se vyrovnaly. Z grafu je zároveň patrný o trochu větší reálný zůstatek oproti plánovanému - světle a tmavě modrá datová řada (Obr. 15).



		Plán			Realita					
	%	Příjem	Výdaj	Zůstatek	Příjem	Výdaj	Zůstatek	KPI1	KPI2	KPI3
Web Inspiral	-	20 000,00 Kč	15 000,00 Kč	5 000,00 Kč	20 000,00 Kč	13 230,00 Kč	6 770,00 Kč	●	-	●
Bez verze	-	20 000,00 Kč	15 000,00 Kč	5 000,00 Kč	20 000,00 Kč	3 980,00 Kč	16 020,00 Kč	●	-	●
Design	100	0,00 Kč	0,00 Kč	0,00 Kč	0,00 Kč	3 450,00 Kč	-3 450,00 Kč	●	●	●
Vývoj	100	0,00 Kč	0,00 Kč	0,00 Kč	0,00 Kč	5 100,00 Kč	-5 100,00 Kč	●	●	●
Rezerva	100	0,00 Kč	0,00 Kč	0,00 Kč	0,00 Kč	700,00 Kč	-700,00 Kč	●	●	●

Obr. 15 – Konečný rozpočet [4]

První koláčový graf zobrazuje, že na rezervu bylo použito pouze pět procent z celkových nákladů, tudíž je patrné, že projekt byl dobře naplánován a nenastala žádná komplikace. Provozní náklady a webhosting nebyly přiřazeny k žádné verzi. Druhý koláčový graf nám říká, že přes 70 procent nákladů bylo použito na mzdy (Obr. 16).



Obr. 16 – Výdajové grafy [4]

Z třetího grafu víme, že náklady na všechny tři zaměstnance jsou přibližně stejné, tudíž byla mezi ně vhodně rozvržena práce. Ze čtvrtého grafu jsou patrné o trochu větší náklady na vývoj než na grafický design (Obr. 17).

Obr. 17–Graf výdajů na zaměstnance [4]

Poslední dva koláčové grafy nejsou z důvodu pouze jedné příjmové platby příliš užitečné (Obr. 18). Své uplatnění by nalezly až u rozsáhlejších projektů.

Obr. 18 – Grafy příjmů a zůstatků [4]

Během testování byla ověřena správná funkčnost rozšíření. Práce se zaměřovala pouze na samostatný projekt, ale vhodnou strukturou projektů je možno získat celkový pohled na všechny projekty, ať už převedením zisku nebo ztráty do jednoho centrálního projektu nebo vytvářet všechny projekty jako podprojekty jedno centrálního projektu. Následně role s právy na mzdy a kontakty by byly přidělovány pouze k tomuto centrálnímu projektu, čímž by se usnadnila jejich správa.

11. Závěr

Tato práce se nejprve zaměřuje na problematiku projektového řízení. Poté je detailně rozebrána samotná aplikace Redmine a framework RubyonRails, který využívá. Pomocí analýzy jsou navrženy potřeby pro finanční rozšíření, které je následně vyvinuto a otestováno.

Výsledkem práce je finanční rozšíření aplikace Redmine, které je schopno podávat podrobné informace o stavu IT projektů. Aplikace sleduje cash-flow jednotlivých projektů pomocí manuálně zadaných transakcí a automaticky generovaných měsíčních výplat zaměstnancům, vypočítaných na základě výkazu odpracovaných hodin a hodinové mzdy. Tyto informace jsou následně prezentovány pomocí tabulek a nejrůznějších grafů, které mají za účel nejen informovat o aktuálním stavu projektu, ale zároveň i pomoci projektovému manažerovi správně se rozhodnout. Rozšíření dále funguje jako propracovaný seznam kontaktů.

Testování by bylo vhodnější na rozsáhlejším projektu, případně i na více projektech. Během testování nebyla nalezena žádná chyba a rozšíření podávalo relevantní informace. Všechny body zadání byly splněny. Práci by bylo vhodné dále rozšířit např. o centrální pohled na všechny projekty nebo o detailnější propracování systému plateb.

K práci je přiloženo CD, na kterém se nachází zdrojové kódy aplikace Redmine a finančního rozšíření, soubor s obsahem MySQL databáze použité pro testování a tento text v elektronické podobě.

Přehled zkratek

HTTP – *HyperText Transfer Protocol* – protokol určený pro přenos HTML souborů

JSON - *JavaScriptObjectNotation* – datový formát

KPI – *Key Performance Indicators* – klíčové ukazatele výkonnosti – sada ukazatelů hodnotící určité hodnoty nebo procesy

MVC – *Model ViewController* – návrhový vzor

REST - *RepresentationalState Transfer* - architektura rozhraní, pro jednotný přístup ke zdrojům

RoR – *Ruby on Rails* – framework využívající programovací jazyk Ruby

XML - *ExtensibleMarkupLanguage* - obecný značkovací jazyk

ORM - *Object-relational mapping* - zajišťuje automatický převod dat mezi objektově orientovaným programovacím jazykem a relační databází.

Literatura

[1] FIALA, Petr. *Projektové řízení: modely, metody, řízení*. 1. vyd. Praha: Professional Publishing, 2004, 276 s. ISBN 80-864-1924-X.

[2] SCHWALBE, Kathy. *Information Technology Project Management*. 7th ed. Mason, Ohio: South-Western, 2012. ISBN 978-1-133-62-722-7.

[3] ROSENAU, Milton D. *Řízení projektů: příprava a plánování, zahájení, výběr lidí a jejich řízení, kontrola a změny, vyhodnocení a ukončení*. Vyd. 2. Brno: ComputerPress, 2003, xii, 344 s. ISBN 80-722-6218-1.

[4] Vlastní archiv.

[5] BUDZIER, Alexander a Bent FLYVBJERG. WhyYour IT Project May BeRiskierThanYouThink. *Harvard Business Review* [online]. [cit. 2014-05-02]. Dostupné z: http://hbr.org/2011/09/why-your-it-project-may-be-riskier-than-you-think/

[6] SKÁLOVÁ, Petra. *Podniková ekonomika 1*. 2. vyd. V Plzni: Západočeská univerzita, 2008, 80 s. ISBN 978-80-7043-726-1.

[7] SYNEK, Miloslav. *Podniková ekonomika.* 4. přeprac. a dopl. vyd. Praha: C. H. Beck, 2006, 475 s. ISBN 80-717-9892-4.

[8] HÁLEK, Vítězslav. *EKONOMICKÉ MINIMUM*. [online]. [cit. 2014-05-04]. Dostupné z: http://halek.sk/www/prezentace/ekonomicke-minimum/eminimum-print.php?projection&l=01

[9]LANG, Jean-Philippe. *Redmine* [online]. [cit. 2014-05-04]. Dostupné z: http://www.redmine.org/

[10]HANSSON, David Heinemeier. *Ruby on Rails* [online]. [cit. 2014-05-04]. Dostupné z: http://rubyonrails.org/

[11] HORDĚJČUK, Vojtěch. Model-View-Controller. *Voho* [online]. [cit. 2014-05-04]. Dostupné z: http://voho.cz/wiki/informatika/oop/navrhovy-vzor/model-view-controller/

[12] BERNARD, Borek. Úvod do architektury MVC. *Zdroják* [online]. [cit. 2014-05-04]. Dostupné z: http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[13] MADANE, Milind. ApplicationDevelopmentUsing MVC Architecture. *MRCC Blog* [online]. [cit. 2014-05-04]. Dostupné z: http://blog.mrccsolutions.com/application-development-using-mvc-architecture/

[14] COOMANS, Alex. Ruby on RailsforDesigners. *Tuts+ Premium* [online]. [cit. 2014-05-04]. Dostupné z: http://code.tutsplus.com/articles/ruby-on-rails-for-designers--net-4455

[15] ЛЕНЗ, Патрик. Представляем Ruby on Rails. *ТехнологияКлиент-Сервер* [online]. [cit. 2014-05-04]. Dostupné z: http://www.k-press.ru/cs/2007/3/rails/rails_all.asp

[16] PARMENTER, David.*Key performance indicators: developing, implementing, and usingwinningKPIs*. Hoboken, N.J.: John Wiley, c2007, xv, 236 p. ISBN 978-047-0095-881.

A Instalace Redmine

Tento návod slouží k instalaci aplikace Redmine na operačním systému Debian GNU/Linux s použitím MySQL databáze.

Pro instalaci Ruby a RoR je možné použít program, který umožňuje spravovat a používat několik verzí Ruby, čímž usnadňuje testování funkčnosti aplikací na více verzích. Zvolil jsem program RVM (Ruby VersionManager). Instalaci RVM spustíme příkazem v terminálu.

```
\curl -sSL https://get.rvm.io | bash -s stable
```

Nyní máme nainstalováno RVM a můžeme přejít k Ruby, zvolil jsem verzi 1.9.3

```
rvminstall 1.9.3
```

V předchozím kroku jsme mohli nainstalovat více verzí Ruby, proto musíme nastavit, jakou verzi Ruby chceme používat. Pokud nechceme používat více verzí, může pomocí *--default*nastavit konkrétní verzi jako defaultní a nemusíme při každém spuštění definovat, kterou verzi chceme použít.

```
rvm use 1.9.3 --default
```

Správnost instalace můžeme ověřit.

```
ruby -v
```

Nyní už můžeme instalovat Rails.

```
gem installrails
```

Z oficiálních stránek si stáhneme Redmine a rozbalíme, terminál nastavíme na tento adresář.

Redmine podporuje databáze MySQL, pokud nemáme databázový server, nainstalujeme ho.

```
sudoapt-getinstallmysql-clientmysql-server libmysql-ruby libmysqlclient-dev
```

Nyní vytvoříme databázi pro Redmine a uživatele redmine, kterému přiřadíme všechna práva.

```
CREATEDATABASEredmine CHARACTER SET utf8;
CREATEUSER'redmine'@'localhost' IDENTIFIED BY'my_password';
GRANTALL PRIVILEGES ON redmine.* TO'redmine'@'localhost';
```

Soubor config/database.yml.example přejmenujeme na config/database.yml a nastavíme přístup do databáze.

```
production:
adapter: mysql2
database: redmine
host: localhost
username: redmine
password: my_password
```

Redmine používá Bundler pro správu knihoven (gems).

```
gem installbundler
```

Pomocí Bundleru nainstalujeme všechny potřebné knihovny

```
bundleinstall  --withoutdevelopmenttest
```

Nyní vygenerujeme nový klíč pro kódování cookies.

```
rakegenerate_secret_token
```

Vytvoříme strukturu databáze a naplníme ji základními daty.

```
RAILS_ENV=productionrakedb:migrate
RAILS_ENV=productionrakeredmine:load_default_data
```

Nyní ještě musíme vytvořit některé adresáře a nastavit jim práva.

```
mkdir -p tmptmp/pdf public/plugin_assets
sudochown -R redmine:redminefiles log tmp public/plugin_assets
sudochmod -R 755 files log tmp public/plugin_assets
```

Nyní spustíme server s podporou Ruby on Rails.

```
ruby script/rails s –e production
```

Aplikace nyní běží na adrese *localhost:3000.*