Žilinská Univerzita v Žiline
Fakulta riadenia a informatiky

Bakalárska práca

Študijný odbor: **Informatika**

Adam Gavlák

Webová aplikácia pre podporu tvorby katedrových rozvrhov

Vedúci: **Mgr. Michal Kaukič, PhD.**

Reg.č. 23/2016 Máj 2017

Abstrakt

Gavlák Adam: *Webová aplikácia pre podporu tvorby katedrových rozvrhov* [Bakalárska práca]
Žilinská Univerzita v Žiline, Fakulta riadenia a informatiky, Katedra matematických metód a operačnej analýzy.
Vedúci: Mgr. Michal Kaukič, PhD.
FRI ŽU v Žiline, 2017 — ? s.

Obsahom práce je analýza, návrh a implementácia webovej aplikácie pre podporu tvorby katedrových rozvrhov s využitím moderných webových technológii, ktorá umožní urýchlenie a automatizáciu niektorých interných procesov Katedry matematických metód.

Abstract

GAVLÁK ADAM: *Web application for support of development of departmental schedules* [Bachelor thesis]
University of Žilina, Faculty of Management Science and Informatics, Department of mathematical methods and operational analysis.
Tutor: Mgr. Michal Kaukič, PhD.
FRI ŽU in Žilina, 2017 — ? p.

The content of bachelor thesis describes analysis, design and implementation of web application for support of development of departmental schedules using modern web technologies, which will aid some of the internal processes of Department of Mathematical Methods and Operations Research.

Prehlásenie

Prehlasujem, že som túto prácu napísal samostatne a že som uviedol všetky použité pramene a literatúru, z ktorých som čerpal.

V Žiline, dňa 21.4.2017 Adam Gavlák

Obsah

Úvod **5**

1 Analýza a prehľad technológii **6**

1.1 Požiadavky aplikácie 6

1.2 Back-end technológie 7

1.3 Databázové riešenia 7

2 Návrh aplikácie **8**

2.1 Prípady použitia 8

2.2 Jazyk – Ruby 8

2.2.1 Zaujímavé vlastnosti jazyka Ruby 8

2.2.2 RubyGems 10

2.3 Framework – Ruby on Rails 11

2.3.1 Model, view a controller 11

2.3.2 ActiveRecord 11

2.3.3 Migrácie 11

2.3.4 Generátory 11

2.4 Databáza a entitno-relačný diagram 11

2.5 Systém pre správu verzií – Git 11

3 Implementácia **13**

3.1 Vývoj s pomocou príkazového riadku 14

3.2 Smerovanie 14
3.3 Modely 14
3.4 Controllery 14
3.5 Views 14
3.5.1 HTML a ERB 14
3.5.2 CSS 14
3.5.3 JavaScript 14
3.6 Spracovanie úloh na pozadí 14
3.7 Generovanie PDF prehľadov 14
3.8 Odosielanie e-mailov 14
3.9 Užívateľské rozhranie 15
3.9.1 Prihlásenie a registrácia 15
3.10 Turbolinks 16
3.11 Problémy pri implementácii a ich riešenie 16
3.11.1 N+1 queries 16
3.11.2 Dlhé časy požiadaviek 17

Záver **18**

Prílohy **19**

Príručka administrátora 19

Literatúra **24**

Zoznam obrázkov

3.1 Rozhranie pre prihlásenie do aplikácie. 15

Úvod

Na počiatku bol Internet používaný hlavne pre akademické účely, ale za roky prevádzky sa rozvinul do dnešnej gigantickej podoby, keď už každý z nás môže mať prístup k Internetu aj vo vrecku, pričom sa stále rozrastá každým dňom.

Podobnou rýchlosťou sa vyvíjajú aj nástroje na vývoj webových aplikácii a v posledných rokoch sa začínajú deliť na oddelené disciplíny. Jednou z mojich úloh bolo porovnať a zvoliť si technológiu, na ktorej postavím internú webovú aplikáciu pre Katedru matematických metód a operačnej analýzy.

Keďže celý vývoj musím zvládnuť sám rozhodol zvoliť z obrovského množstva dnes dostupných technológii jazyk Ruby a použiť framework Ruby on Rails, pretože už s ním mám skúsenosti, poskytne mi všetky potrebné nástroje na splnenie daných cieľov a verím že mi uľahčí efektívny vývoj webovej aplikácie, kde sa môžem lepšie sústrediť na spracovanie požiadaviek koncových používateľov.

V tejto práci analyzujem vytvorenie kompletnej webovej aplikácie pre podporu tvorby katedrových rozvrhov, od analýzy a návrhu až po samotné nasadenie aplikácie na produkčný server. Konečným cieľom tejto práce je webová aplikácia, ktorá dokáže vypočítať úväzky pre jednotlivých vyučujúcich katedry, generovať prehľady do PDF súborov a následne ich aj odoslať danému vyučujúcemu cez e-mail.

Kapitola 1

Analýza a prehľad technológii

1.1 Požiadavky aplikácie

Na základe zadania a nasledovnými konzultáciami s koncovými používateľmi aplikácie bol vytvorený nasledujúci prehľad požiadaviek aplikácie:

- Vytvánie, úprava a zmazanie nasledujúcich dát:
  - Predmety katedry
  - Študijné skupiny
  - Vyučujúci
- Vzťahy medzi jednotlivými údajmi
- Prehľady predmetov, študijných skupín a vyučujúcich
- Prepočet úväzkov vyučujúcich podľa stanovených pravidiel
- Generovanie prehľadových dokumentov pre vyučujúcich
- Odosielanie emailov vyučujúcim s vygenerovanými prehľadmi

1.2 Back-end technológie

Na vývoj aplikácie je dôležité vybrať správny nástroj. V dnešnej dobe, je ich na trhu obrovské množstvo, preto je dobré zvoliť taký nástroj, ktorý nám umožní vytvoriť aplikáciu prehľadne, v rozumnom časovom rozsahu, následne ju s prehľadom spravovať a možno aj rozširovať.

PHP

Python

Ruby

1.3 Databázové riešenia

MySQL

PostgreSQL

SQLite

Kapitola 2

Návrh aplikácie

2.1 Prípady použitia

2.2 Jazyk – Ruby

Ruby je dynamický, reflektívny objektovo orientovaný programovací jazyk, ktorý kombinuje syntax inšpirovanú jazykmi Perl, Smalltalk, Eiffel, Lisp a má dynamickú správu pamäti. Autor – Yukihiro Matsumoto – pri návrhu vychádzal z filozofie, že programátor by sa pri práci s jazykom mal baviť ale zároveň byť produktívny. Taktiež zdôrazňoval, že dizajn systémov by sa mal zameriavať viac na potreby človeka ako na potreby počítača. [3]

2.2.1 Zaujímavé vlastnosti jazyka Ruby

Ruby je veľmi zaujímavý jazyk, avšak má mnoho vlastností, ktoré by mohli samostatne pokryť ďalšiu prácu. Preto som sa rozhodol vybrať také, ktoré sú netradičné a priblížiť ich.

Symbol

Symbol na prvý pohľad vyzerá ako premenná, ktorá má pred názvom dvojbodku, napríklad *:user*. Výhodou symbolov je, že ich nie je potrebné deklarovať a v celom programe

budú mať rovnakú hodnotu. Zoberme nasledujúci príklad:

```
puts "string".object_id
puts "string".object_id
puts :symbol.object_id
puts :symbol.object_id
```

Po spustení tohto programu sa zobrazil nasledujúci výstup a naozaj dokázal, že symboly majú počas behu programu rovnakú hodnotu:

```
$ ruby symbol.rb
90520360
90520080
801628
801628
```

Blok

Bloky sú v iných jazykoch nazývané aj *closure*. V Ruby sú veľmi užitočnou vlastnosťou pretože nám umožňujú odoslať blok kódu priamo do funkcie. Definujú sa:

```
call_block(arg) { |a| puts a }
# alebo
call_block(arg) do |a|
  puts a
end
```

Pre využitie bloku vo funkcii musíme použiť funkciu *yield*. Avšak, ak funkcia používa *yield* ale, žiadny block nebol prijatý tak Ruby vyvolá výnimku. Blok môžeme vo funkcii zavolať nasledujúco:

```
def call_block(arg)
    yield(arg)
end
```

Moduly

Ruby nepodporuje viacnásobnú dedičnosť. Keď chceme aby niekoľko tried implementovalo určitú funkcionalitu musíme využiť *module*. Module je kolekcia metód a konštánta môžeme modul definovať nasledujúco:

```
module Mod
  def call_func(arg)
    puts arg
  end
end
```

V triede kde požadujeme funkcionalitu modulu potom jednoducho zahrnieme modul pomocou funkcie *include*. Napríklad:

```
class MyClass
    include Mod
end
```

2.2.2 RubyGems

RubyGems je správca balíčkov používaný spolu s jazykom Ruby, ktorý ponúka štandardný formát pre distribúciu Ruby programov a knižníc vo formáte nazývanom *gem* a uľahčuje ich inštaláciu. Všetky balíčky sa dajú nájisť, prehľadávať alebo pridávať na stránke `https://rubygems.org`. V čase písania práce bolo publikovaných 131 tisíc balíčkov a počet stiahnutí presiahol 13,3 miliárd.

2.3 Framework – Ruby on Rails

2.3.1 Model, view a controller

2.3.2 ActiveRecord

2.3.3 Migrácie

2.3.4 Generátory

2.4 Databáza a entitno-relačný diagram

Na perzistentné uloženie dát aplikácie som po porade zvolil databázu SQLite, ktorá má podľa mňa dostatočnú kapacitu a spĺňa všetky požiadavky pre škálu budovanej aplikácie. Má dokonca aj niekoľko výhod:

- Jednoduchá inštalácia
- Nie je potrebná skoro žiadna konfigurácia
- Dáta sú ľahko zálohovateľné, pretože databáza je uložená v jedinom súbore
- Šetrí zdroje systému

2.5 Systém pre správu verzií – Git

Pri tvorbe softvéru vývojári vytvárajú zdrojový kód. Menia, rozširujú ho, potrebujú vrátiť zmeny alebo sa vrátiť späť k starším verziám. Taktiež je potrebné zabezpečiť situáciu, keď na jednom projekte pracuje viac vývojárov naraz. [4]

Git je jedným z mnohých programov na správu verzií zdrojového kódu vytvorený Linusom Torvaldom v roku 2005. Používať ho budem počas celého vývoja aplikácie aj napriek tomu, že nie som súčasťou žiadneho týmu, pretože mi stále poskytuje niekoľko obrovských výhod, a to:

- Jednoduché a prehľadné zdieľanie zdrojového kódu s vedúcim práce

- História všetkých zmien zdrojového kódu
- Uľahčenie nasadenia na server
- Záloha aplikácie v prípade straty na lokálnom systéme

Kapitola 3

Implementácia

3.1 Vývoj s pomocou príkazového riadku

3.2 Smerovanie

3.3 Modely

3.4 Controllery

3.5 Views

3.5.1 HTML a ERB

3.5.2 CSS

3.5.3 JavaScript

3.6 Spracovanie úloh na pozadí

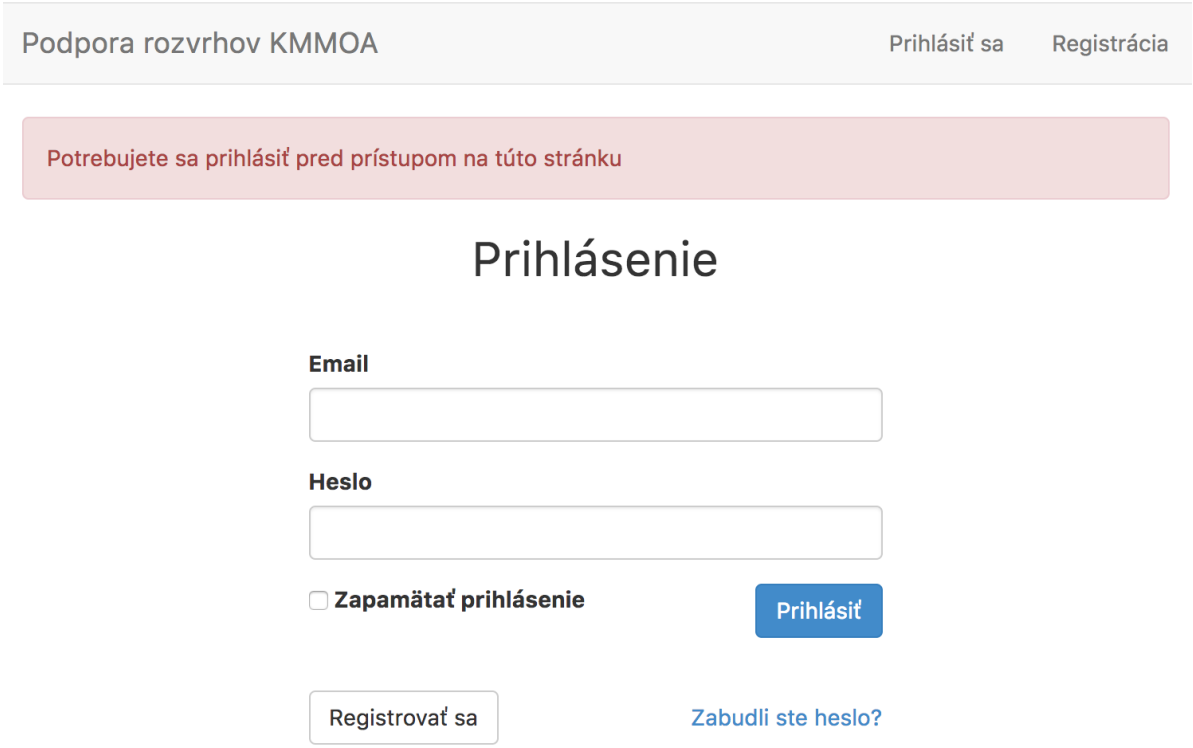
3.7 Generovanie PDF prehľadov

3.8 Odosielanie e-mailov

3.9 Užívateľské rozhranie

3.9.1 Prihlásenie a registrácia

Pred začatím používania aplikácie sa musí užívateľ najprv prihlásiť alebo registrovať. Všetky podstránky aplikácie sú zabezpečené a užívateľ je presmerovaný pokiaľ nie je prihlásený.



Obr. 3.1: Rozhranie pre prihlásenie do aplikácie.

Užívatelia si taktiež môžu prenastaviť heslo pomocou emailu ak ho náhodou zabudli. Na zadaný email sa zašle link na stránku, kde užívateľ môže heslo zmeniť. Registrácia sa navyše dá kedykoľvek vypnúť administrátorom použitím premennej prostredia.

3.10 Turbolinks

Turbolinks je JavaScript knižnica, ktorá robí navigáciu po webovej stránke rýchlejšou. Poskytuje rýchlostné benefity single-page aplikácie, bez pridanej komplexnosti iných JavaScript frameworkov. Vykreslenie celej HTML stránky môže bez obáv prebiehať na strane servera. Keď chce užívateľ prejisť na inú stránku pomocou kliknutia na link Turbolinks pomocou AJAX-u vyzdvihne danú stránku a vymení obsah tagu *<body>* a zlúči obsah v tagoch *<head>*, to všetko bez potreby znovu načítať celú stránku. [1]

3.11 Problémy pri implementácii a ich riešenie

3.11.1 N+1 queries

Pri načítaní niektorých prehľadov, ako napríklad prehľad vyučujúcich sa začali v logoch servera objavovať dlhé zoznamy SQL dotazov. Ako prvé som si všimol, že je to spôsobené iteráciou cez vzťahy kolekcie modelov. Ako napríklad:

```
@techers = Teacher.all()

@teachers.courses.each do |course|
  puts course.title
end
```

To spôsobí, že pre každý model vo vzťahu je načítaný samostatným SQL selectom, čo ale nechceme, pretože to spôsobuje nadmerné zaťaženie databázy. Po zamyslení nad týmto problémom je jasné, že musíme načítať modely aj všetky ich vzťahy cez ktoré chceme iterovať naraz s použitím JOIN-u. Našťastie ale vývojári ActiveRecord-u na toto správanie mysleli a nezabudli ho implementovať. Riešenie je veľmi jednoduché:

```
@techers = Teacher.includes(:courses).all()

@teachers.courses.each do |course|
```

```
  puts course.title
end
```

Pri načítaní modelov použijeme funkciu *includes()* do ktorej môžeme napísať symboly, ktoré reprezentujú jednotlivé vzťahy. Po skontrolovaní logov teraz vidíme, že pri načítaní náhľadu sa teraz spustí iba 1 SQL query.

3.11.2 Dlhé časy požiadaviek

Toto správanie je spôsobené tým, že funkcia v controlleri trvá dlhšie ako je obvyklé a tým zdržiava navrátatenie odpovede užívateľovi. Problém sa vyskytol najmä pri odosielaní e-mailov a generovaní PDF prehľadov. Ide o jednoduchý problém, ktorý sa dá vyriešiť niekoľkými spôsobmi.

V aplikácii je tento problém vyriešený použitím balíčka (*gem 'delayed_job'*), ktorý vykonáva dlho-trvajúce funkcie na pozadí. Balíček má v databáze svoju tabuľku, do ktorej ukladá všetky funkcie, ktoré má vykonať.

Avšak, tento balíček sa musí spustiť externe cez príkazový riadok, preto je múdre pridať aj balíček (*gem 'daemons'*) aby sme mohli tento proces daemonizovať. Spustíme ho zo zložky aplikácie príkazom:

```
$ bin/delayed_job start
```

Neskôr treba na produkčnom serveri tento skript nalinkovať aby sa automaticky spúšťal pri štarte/reštarte servera.

Záver

Prílohy

Príručka administrátora

Pri nasadení bude nutné nainštalovať niekoľko nástrojov, pre správny chod aplikácie a jej závislostí. Príručka je napísaná pre distribúciu Debian 8 (Jessie), ale mala by fungovať pre väčšinu Unix-like operačných systémov.

Začneme nainštalovaním všetkých potrebných balíčkov:

```
$ apt update
$ apt install git-core curl zlib1g-dev build-essential libssl-dev
  libreadline-dev libyaml-dev libsqlite3-dev sqlite3 libxml2-dev
  libxslt1-dev libcurl4-openssl-dev python-software-properties
  libffi-dev
```

Inštalácia manažéra prostredia Ruby – rbenv

Rbenv je veľmi užitočný nástroj, ktorý umožní jednoducho inštalovať a meniť verziu Ruby. Nainštalovaný je len lokálne pre užívateľa, ktorý vykonal inštaláciu. Nainštalujeme ho nasledovne:

```
$ cd
$ git clone https://github.com/rbenv/rbenv.git ~/.rbenv
$ echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc
$ echo 'eval "$(rbenv init -)"' >> ~/.bashrc

$ git clone https://github.com/rbenv/ruby-build.git ~/.rbenv/plugins/ruby-build
$ echo 'export PATH="$HOME/.rbenv/plugins/ruby-build/bin:$PATH"' >> ~/.bashrc
$ source ~/.bashrc

$ rbenv install 2.4.0
$ rbenv global 2.4.0
```

Teraz môžeme overiť, či sa Ruby správne nainštaloval pomocou príkazu

```
$ ruby -v
```

Po úspešnom nainštalovaní rbenv a Ruby nainštalujeme balíček Bundler, ktorý nám bude manažovať závislosti v aplikáciach pomocou súboru *Gemfile*.

```
$ gem install bundler
$ rbenv rehash
```

Príprava aplikácie na serveri

Teraz je čas stiahnuť našu aplikáciu na server (odporúčam naklonovať Git repozitár). Aplikáciu umiestnime jednoducho do */home/UZIVATEL/APLIKACIA*. Potom prejdeme do adresára kde sme aplikáciu umiestnili a spustíme príkaz:

```
$ bundle install
```

Ďalší krok je vytvorenie databázy. SQLite vytvorí súbor databázy a následne tabuľky v adresári *db* pomocou príkazov:

```
$ rake db:create
$ rake db:migrate
```

Príkaz (*rake db:create*) nám databázu vytvorí v zložke db v adresári aplikácie a *rake db:migrate* vytvorí tabuľky podľa migrácii.

Posledná časť prípravy aplikácie je spustenie balíčka *delayed_job*, aby sa mohli spúšťať úlohy na pozadí. To spravíme spustením príkazu z adresára aplikácie:

```
$ bin/delayed_job start
```

Po týchto krokoch je aplikácia pripravená a môžeme prejisť na inštaláciu webového servera.

Inštalácia webového servera Nginx

Naša aplikácia bude využívať webový server Nginx s plugin-om *Passenger* od spoločnosti *Phusion*. V prvom kroku nainštalujeme (ako *sudo*) PGP kľúč a pridáme HTTPS podporu pre APT repozitára:

```
$ apt-key adv --keyserver hkp://keyserver.ubuntu.com:80
  --recv-keys 561F9B9CAC40B2F7
$ apt install apt-transport-https ca-certificates
```

V ďalšom kroku pridáme samotný repozitár a aktualizujeme list repozitárov

```
$ echo 'deb https://oss-binaries.phusionpassenger.com/apt/passenger
  jessie main' > /etc/apt/sources.list.d/passenger.list
$ apt update
```

V poslednom kroku nainštalujeme samotný web server Nginx aj spolu s Passengerom

```
$ apt install nginx-extras passenger
$ service nginx start
```

Keď otvoríme IP stroja vo webovom prehliadači, mala by sa zobraziť štandardná stránka nainštalovaného serveru Nginx. [2]

Konfigurácia webového servera Nginx

Súbory možno editovať v akomkoľvek textovom editore. V príklade je použitý editor *Joe*.

Nájdeme nasledujúci riadok v súbore */etc/nginx/nginx.conf* a odkomentujeme ho:

```
include /etc/nginx/passenger.conf;
```

Je potrebné zmeniť užívateľa pod ktorým bude Nginx bežať. V rovnakom súbore (úplne navrchu) zmeníme *user www-data;* na užívateľa, ktorý ma nainštalované Ruby s pomocou rbenv nasledovne:

```
user UZIVATEL;
```

Po úprave súbor uložíme a zavrieme. Otvoríme súbor */etc/nginx/passenger.conf* a zmeníme riadok ktorý začína *passenger_ruby* na pričom zameníme UZIVATEL za užívateľa ktorý aplikáciu inštaluje na:

```
passenger_ruby /home/UZIVATEL/.rbenv/shims/ruby;
```

Nginx je teraz nastavený, aby pri Ruby aplikáciach využíval aktuálne nainštalovanú verziu Ruby s pomocou rbenv. Posledný krok je konfigurácia prepojenie Nginx-u na aplikáciu. Po otvorení */etc/nginx/sites-enabled/default* obsah zmeníme na:

```
server {
        listen 80;

        server_name domena.sk;
        passenger_enabled on;
        rails_env    production;
        root         /home/UZIVATEL/APLIKACIA/public;

        # presmerovanie errorov na statickú stránku /50x.html
        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }
}
```

Samozrejme, je potrebné zameniť UZIVATEL za meno užívateľa, pod ktorým aplikácia beží a APLIKACIA za meno aplikácie, prípadne cestu kde je aplikácia uložená. Reštartujeme Nginx, aby prejavili zmeny v konfigurácii:

```
$ service nginx restart
```

A teraz aplikácia beží na zadanej doméne, prípadne IP servera.

Literatúra

[1] LLC Basecamp. Turbolinks makes navigating your web application faster. `https://github.com/turbolinks/turbolinks`. [Online, 22.2.2017].

[2] Phusion Holding B.V. Installing passenger + nginx on debian 8 (with apt) - passenger library. `https://www.phusionpassenger.com/library/install/nginx/install/oss/jessie/`. [Online, 20.3.2017].

[3] Yukio Matsumoto and K Ishituka. Ruby programming language, 2002.

[4] Stefan Otte. Version control systems. *Computer Systems and Telematics, Institute of Computer Science, Freie Universität, Berlin, Germany*, 2009.