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

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

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

Zoznam obrázkov

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 Technológie strany klienta

Ako pri každej webovej aplikácii, na strane klienta budú použité základné technológie ako značkovací jazyk HTML, štýlovací jazyk CSS a JavaScript, ktorý slúži pridávanie rôznej funkcionality od práce s DOM-om, cez použitie AJAX-u až po spracovanie dát na strane klienta.

HTML

HTML (HyperText Markup Language) je najzákladnejší stavebný prvok web stránky, pretože opisuje a definuje obsah web stránky. *HyperText* v názve popisuje odkazy, ktoré spájajú jednotlivé web stránky v rámci jedného webového portálu alebo spájajú viaceré webové portály. Odkazy sú podstatným aspektom webu, pretože užívateľ nahrávajúci obsah na internet sa stáva aktívnym prispievateľom World Wide Web-u.

CSS

CSS (Cascading Style Sheets) je jazyk štýlovania používaný na popísanie dokumentu napísaného v HTML alebo XML (zahŕňa aj dialekty XML ako SVG alebo XHTML). CSS popisuje ako majú byť jednotlivé elementy zobrazené na obrazovke, papieri, v reči alebo na iných médiách.

JavaScript

JavaScript je dynamický interpretovaný programovací jazyk s možnosťami objektovo orientovaného programovania, založeného na prototypoch. Najrozšírenejší je ako programovací jazyk na strane klienta webov, kde poskytuje API k tomu ako by sa mala web stránka správať, keď sa vyskytne nejaká udalosť, ale dokáže spravovať aj rôzne iné aspekty a správanie web stránok.

AJAX

AJAX (Asynchronous JavaScript + XML) nie je samostatná technológia, ale pojem vytvorený v roku 2005 Jesse Jamesom Garretom, ktorý opisuje nový spôsob ako pristupovať k spojeniu rôznych existujúcich technológii ako HTML, XML, CSS, JavaScript, DOM a objekt XMLHttpRequest ktoré vytvárajú model AJAX-u. Použitím AJAX-u majú webové aplikácie možnosť vykonávať rýchle a inkrementálne úpravy užívateľského rozhrania bez potreby znovu načítania celej webovej stránky. To robí webovú aplikáciu rýchlejšiou a responzívnejšiou z pohľadu užívateľov.

1.3 Technológie strany servera

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. Technológie na strane servera sú potrebné, keď technológie na strane užívateľa už nedokážu požadovanú funkcionalitu dosiahnuť. Každá technológia má svoje výhody a nevýhody a preto priblížim tie, ktoré poznám najviac.

PHP

PHP (rekurzívna skratka pre PHP: Hypertext Preprocesor) je skriptovací jazyk na strane servera, ktorý je používaný hlavne na tvorbu webových aplikácii. Za roky vývoja od roku 1994 prešiel rôznymi zmenami a dnes sa môže porovnávať aj s inými jazykmi, ale iba v rámci vybavenia a podporou komunity.

Bolo v ňom vytvorených veľké množstvo úspešných projektov ako napríklad *Wikipedia*, *Facebook* (ktorý už v dnešnej dobe používa vlastnú verziu PHP nazvanú HHVM), *Wordpress* alebo *Yahoo*. Výhodou jazyka PHP je jednoduchosť nasadenia na server, keďže PHP tvorí na trhu väčšinu bežiacich webových aplikácii a drvivá väčšina poskytovateľov hostingu podporuje iba PHP.

V posledných rokoch vzniklo aj niekoľko populárnych frameworkov, ktoré uľahčujú

prácu pre vývojárov, keďže nemusia všetku funkcionalitu programovať od začiatku. Najviac známe z týchto frameworkov sú *Laravel*, *Symphony*, alebo *Nette*.

Composer

Composer je správca závislostí vytvorený pre PHP. Umožňuje deklaráciu použitých softvérových knižníc použitých v projekte od ktorých je program závislý a spravuje ich za užívateľa (inštalácia/aktualizovanie).

Všetky závislosti sú definované v súbore *composer.json* a operácie sa dajú vykonávať cez aplikáciu v príkazovom riadku pomocou programu *composer*.

Python

Python je vysoko úrovňový programovací jazyk navrhnutý pre vývoj softvéru, ktorý môže byť použitý v širokej škále aplikačných domén. Vývoj webových aplikácii je v jazyku Python dosť náročný, pretože používateľ by musel všetkú funkcionalitu naprogramovať sám. Tu ale prichádzajú framework-y, ktoré tento proces výrazne uľahčia a tým robia vývoj webových aplikácii jednoduchší pre širšie publikum vývojárov.

Full-stack frameworky kombinujú rôzne komponenty ako HTTP server, možnosť perzistencie dát do databázy, šablóny... Najznámejšími z full-stack frameworkov sú *Django*, *TurboGears* alebo *web2py*. Ďalšími z populárnych frameworkov, ktoré už nie sú tak vybavené ako full-stack frameworky a užívateľ si môže zvoliť a dodefinovať tieto ďalšie komponenty sú napríklad *Flash* alebo *CherryPy*.

Pip

Od verzie 3.4 je Python dodávaný so správcom balíčkov Pip. Balíčky, ktoré sa môžu inštalovať nájdeme na webe Python Package Index (PyPI) na adrese `https://pypi.python.org/pypi`. Pip umožňuje inštaláciu spravovanie balíčkov cez konzolovú aplikáciu *pip*

Ruby

Ruby, podobne ako Python je programovací jazyk, ktorý môže byť použitý v širokej škále aplikačných domén. A rovnako ako Python, vývoj webových aplikácii v Ruby bez použitia frameworku by bol príliš zložitý a náročný. Najznámejšie z frameworkov sú: *Ruby on Rails*, *Sinatra* a *Padrino*. Ruby on Rails je full-stack framework zatiaľ čo Sinatra a Padrino sú menšie, neobsahujú toľko funkcionality, ale vývojár si môže funkcionalitu jednoduch doplniť pomocou balíčkou nazývaných *Gems*.

Ruby

RubyGems je správca balíčkov pre jazyk Ruby. Pri vývoji väčších alipkácii je často doplňovaný aplikáciou *bundler*, ktorá umožnuje zapísanie všetkých závislostí do súboru nazývaného *Gemfile*

1.4 Databázové riešenia

Dáta z webovej aplikácie musia byť niekde uložené Databázové riešenia používané pri vývoji webových aplikácii môžeme rozdeliť na dve hlavné kategórie – relačné databázy a NoSQL (Not only SQL) databázy. Hlavný rozdiel medzi týmito kategóriami je, kedy je užitočné uprednostniť jednu kategóriu pred druhou. NoSQL má niekoľko výhod hlavne v rýchlosti a objeme spracovaných dát, ale kým relačná databáza nespôsobuje spomalenie celej aplikácie tak je dostačujúca a v niektorých prípadoch, výhodnejšia a jednoduchšia. Najznámejšie NoSQL databázy sú *Redis*, *MongoDB* a *CouchDB*.

Relačné databázy

Najznámejšími relačnými databázami sú:

- *MySQL* – najpopulárnejšia a najviac využívaná relačná databáza
- *PostgreSQL* – najpokročilejšia a open-source relačná databáza

- *SQLite* – relačná databáza používaná hlavne pri vstavaných riešeniach a malých webových aplikáciach

Kapitola 2

Návrh aplikácie

Pri vývoji aplikácie budem používať všetky technológie spomenuté v *Technológie strany klienta*. Keďže tieto technológie na vývoj webovej aplikácie nestačia musím si zvoliť technológiu, ktorú použijem na strane servera. Rozhodol som sa zvoliť si jazyk Ruby, pretože už mám s ním niekoľko rokov skúseností, a dokážem v ňom tvoriť webové aplikácie rýchlo a efektívne.

2.1 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]

2.1.1 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 pre iné programovacie jazyky a priblížiť ich.

Premenné

Ruby často používa veľmi obmedzenú interpunkciu, ale niektoré znaky sú použité na dekoráciu. Ďalšia vlastnosť je, že Ruby nepotrebuje deklarovať premenné. Zároveň sa používajú jednoduché konvencie na odlíšenie rozsahu premenných:

- *var* je lokálna premenná
- *@var* je premenná inštancie triedy
- *@@var* je premenná triedy
- *$var* je globálna premenná

Tieto znaky uľahčujú čitateľnosť výsledného kódu a umožňujú programátorovi jednoducho rozoznať roly každej premennej. [4]

Symboly

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 rovnaké. 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 sú počas behu programu rovnaké:

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

Bloky

Bloky v Ruby sú inšpirované funkcionálnym programovaním, prinášajú do kódu mnoho flexibility a 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*. V skratke, Module je kolekcia metód a konštánt. Môžeme ho 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
```

2.1.2 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`.

2.2 Framework – Ruby on Rails

Ruby on Rails je framework pre tvorbu webových aplikácii na strane servera pomocou jazyka Ruby. Bol vytvorený v roku 2006 a jeho autorom je David Heinemeier Hansson. Framework je vytvorený na zaujímavej ideológii - konvencia má prednosť pred konfiguráciou. To znamená, že ak vývojár vytvára webovú aplikáciu s pomocou Ruby on Rails tak nastavuje len to čo je dôležité pre chod aplikácie. Ak vývojár dodržuje tieto konvencie tak spraví menej rozhodnutí, čoho výsledkom je rýchlejší čas vývoja, vylepšená spolupráca na projekte a jednoduchšia údržba aplikácie.

2.2.1 Model, view a controller

Ruby on Rails je založený na návrhovom vzore Model-view-controller (v skratke MVC). Ako názov naznačuje tento vzor sa skladá z troch dôležitých komponentov, ktoré riadia správanie sa aplikácie.

- **Model** reprezentuje iba dáta. Nie je závislý na controller-och alebo view-och
- **View** zobrazuje dáta modelov a odosiela interakciu používateľa cotroller-om
- **Controller** poskytuje dáta z modelu do view-u a spracováva interakciu užívateľa, pričom ontroller je závislý na modeloch a view-och

2.2.2 Active Record

Active Record je Object-relational mapping (ORM) knižnica, ktorú Ruby on Rails využíva pre všetky svoje modely. Pri načítaní modelu z databázy sú všetky dáta uložené v pamäti, kde ich môžeme modifikovať a keď sa neskôr rozhodneme ich uložiť, tak Active Record odošle databáze SQL dotaz na zmenu týchto údajov.

2.2.3 Migrácie

Migrácie sú dôležitá súčasť knižnice Active Record, ktoré umožňujú aby sa schéma databázy menila priebežne. Obrovskou výhodou je, že zmeny v schéme sa nemenia pomocou čistého SQL, ale pomocou čistého Ruby DSL na popísanie zmien v tabuľkách. Každá migrácia je uložená vo svojom súbore. Môže vyzerať napríklad takto:

```
class CreateProducts < ActiveRecord::Migration[5.0]
  def change
    create_table :products do |t|
      t.string :name
      t.text :description

      t.timestamps
    end
  end
end
```

Môžeme uvažovať, že každá migrácia je nová verzia schémy pre databázu. Schému môžeme napísať ručne alebo vygenerovať a potom doplniť operácie ktoré chceme vykonať ako pridanie/vymazanie tabuliek, stĺpcov, indexov alebo iných obmedzení.

Generátory súborov

Ruby on Rails je nainštalovaný aj spolu s jednoduchou aplikáciou, ktorá nám okrem iných nástrojov pri vývoji umožňuje aj generovať súbory, čo nám značne uľahčuje a

urýchľuje vývoj aplikácie. Generovanie z príkazového riadku prebieha naslednovne:

```
$ rails generate GENERATOR [args] [options]
```

Slovo *GENERATOR* zameníme za generátor, podľa toho aký súbor/y chceme vygenerovať. Ak do príkazového riadku zadáme iba *rails generate* zobrazí sa nám krátka nápoveda ako generátory používať ale aj zoznam dostupných generátorov. Najdoležitejšie generátory pri vývoji aplikácie sú tieto:

- controller
- mailer
- migration
- model
- task

Ďalej môžeme zadať argumenty, ktoré slúžia na vygenerovanie funkcií, stĺpcov alebo atribútov v závislosti od toho aký generátor máme zvolený.

2.3 Prípady použitia

2.4 Databáza a entitno-relačný diagram

Na perzistentné uloženie dát aplikácie som 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. [5]

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

Existujú aj mnohé služby ktoré poskytujú hosting Git repozitárov či už platené, ako GitHub alebo open-source ako GitLab, ktoré majú aj funkcionalitu zameranú na kolaboráciu komunity ako *Pull request* alebo sledovanie a nahlasovanie problémov, ktoré sa vyskytli pri používaní softvéru.

Kapitola 3

Implementácia

3.1 Vývoj s pomocou príkazového riadku

Ruby on Rails má zabudovanú konzolovú aplikáciu, ktorá uľahčuje vývoj pomocou príkazov. Ak zadáme *$ rails help* tak dostaneme nasledovný výstup

```
$ rails help
Usage: rails COMMAND [ARGS]

The most common rails commands are:
 generate    Generate new code (short-cut alias: "g")
 console     Start the Rails console (short-cut alias: "c")
 server      Start the Rails server (short-cut alias: "s")
 test        Run tests (short-cut alias: "t")
 dbconsole   Start a console for the database specified in config/database.yml
             (short-cut alias: "db")
 new         Create a new Rails application. "rails new my_app" creates a
             new application called MyApp in "./my_app"

 ...
```

Tieto príkazy dokážu urýchliť vývoj, keďže nemusíme konfigurovať štruktúru adresárov, sťahovať repozitáre Ruby on Rails a vytvárať súbory ručne alebo poskytuje iné nástroje. Najdôležitejšie pri vývoji sú príkazy *rails server*, *rails console* a *rails new*.

$ rails server – ako meno napovedá spustí server, ktorý je určený v súbore balíčkov – *Gemfile*, pričom môžeme flagmi špecifikovať na akom porte a adrese sa má aplikácia spustiť.

$ rails console – spúšťa interaktívnu konzolu, kde vývojár môže otestovať správanie jednotlivých komponentov webovej apliácie, bez toho aby musel spúšťať server a testovať cez užívateľské rozhranie v prehliadači.

$ rails new my_ app – sa nepoužíva príliš často, keďže tento príkaz vytvára novú aplikáciu aj s celou jej štruktúrou. Zároveň nastaví a stiahne všetky závislosti. Pri vytváraní sa môžu určité závislosti, ako použitá databáza alebo použitý JavaScript-ový framework pomocou flagov (napríklad *-d=postgresql* ak chceme použiť pri všetkých štádiach vývoja databázu PostgreSQL.

Ďalšia časť príkazov sú generátory súborov ako bolo spomenuté v kapitole 2.2. Generátory vytvárajú iba kostru súboru spolu aj s názvom, kde potom vývojar môže dopísať svoj kód. Všetky typy generátorov súborov môžeme zobraziť pomocou príkazu *$ rails generate*.

Dôležité príkazy sú pre prácu s migráciami a teda databázou. Keď má vývojár migrácie napísané tak ich potrebuje aby sa podľa nich v databáze vytvorili tabuľky, stĺpce, indexy... Príkazov pre prácu s databázou je mnoho ale pri vývoji som používal najčastejšie nasledujúce:

```
$ rails db:[COMMAND]
$ rails db:create   # Vytvorí databázu
$ rails db:migrate  # Prevedie všetky migrácie z DSL do databázy
$ rails db:seed     # Vytvorí prichystané objekty do databázy
                    # zo súboru ./db/seeds.rb
$ rails db:rollback # Vráti všetky zmeny z predchádzajúcej migrácie
$ rails db:drop     # Odstráni databázy
```

3.2 Smerovanie

Ruby on Rails využíva pri mapovaní URL adries router. Tieto mapovania môže vývojár ľubovoľne meniť a teda priradiť určitú cestu danému controller-u. Všetky tieto cesty sídlia v súbore *./config/routes.rb*. Cesty môžeme priebežne sledovať aj pomocou príkazu *$ rails routes*. Tu je ukážka definovaných ciest z aplikácie pre podporu rozvrhov:

```
Rails.application.routes.draw do
  root "pages#index"
  devise_for :users

  namespace :api do
    post "/teacher/group/course", to: "teacher_group_courses#create"
    delete "/teacher/group/course", to: "teacher_group_courses#destroy"
  end

  resources :courses
  resources :groups
  resources :teachers do
    resources :reports, controller: "teacher_reports" do
      get "/email", to: "teacher_reports#email"
      post "/email", to: "teacher_reports#email_send"
    end
  end
  ...
end
```

Existuje niekoľko príkazov ktoré tieto cesty umožňujú vytvárať pomocou a sú nazvané podľa typu HTTP requestu (GET, POST). Keď sú cesty definované pomocou týchto funkcií majú tvar:

```
get "/stranka", to: "controller#akcia"
```

Kde hash *to: STRING* mapuje cestu na špecifický controller a akciu v ňom. Sú však aj ďalšie špecifické prípady, napríklad domovská stránka aplikácie sa definuje pomocou funkcie *`root "controller#akcia"`*

Príkaz *`resources :symbol`* je v skratkou pre vytváranie REST-ful ciest a cesty vygenerované týmto príkazom vyzerajú nasledujúco:

```
       NAZOV VERB   CESTA                CONTROLLER#AKCIA
symbol_index GET    /symbol              symbol#index
             POST   /symbol              symbol#create
  new_symbol GET    /symbol/new          symbol#new
 edit_symbol GET    /symbol/:id/edit     symbol#edit
      symbol GET    /symbol/:id          symbol#show
             PATCH  /symbol/:id          symbol#update
             PUT    /symbol/:id          symbol#update
             DELETE /symbol/:id          symbol#destroy
```

V príklade môžeme vidieť aj preddefinovaný príkaz *devise_for :users*, ktorý je súčasťou balíčka autentifikácie užívateľov nazvaného Devise a vytvára cesty potrebné pre fungovanie tohto balíčka.

Ďalej je použitá aj funkcia *namespace :api ...*, ktorá dokáže zoskupovať podobné príkazy pod spoločnú url, v príklade je to api, čiže všetky cesty budú začínať *`api/cesta`*.

3.3 Modely

3.4 Controllery

3.5 Views

3.6 Spracovanie úloh na pozadí

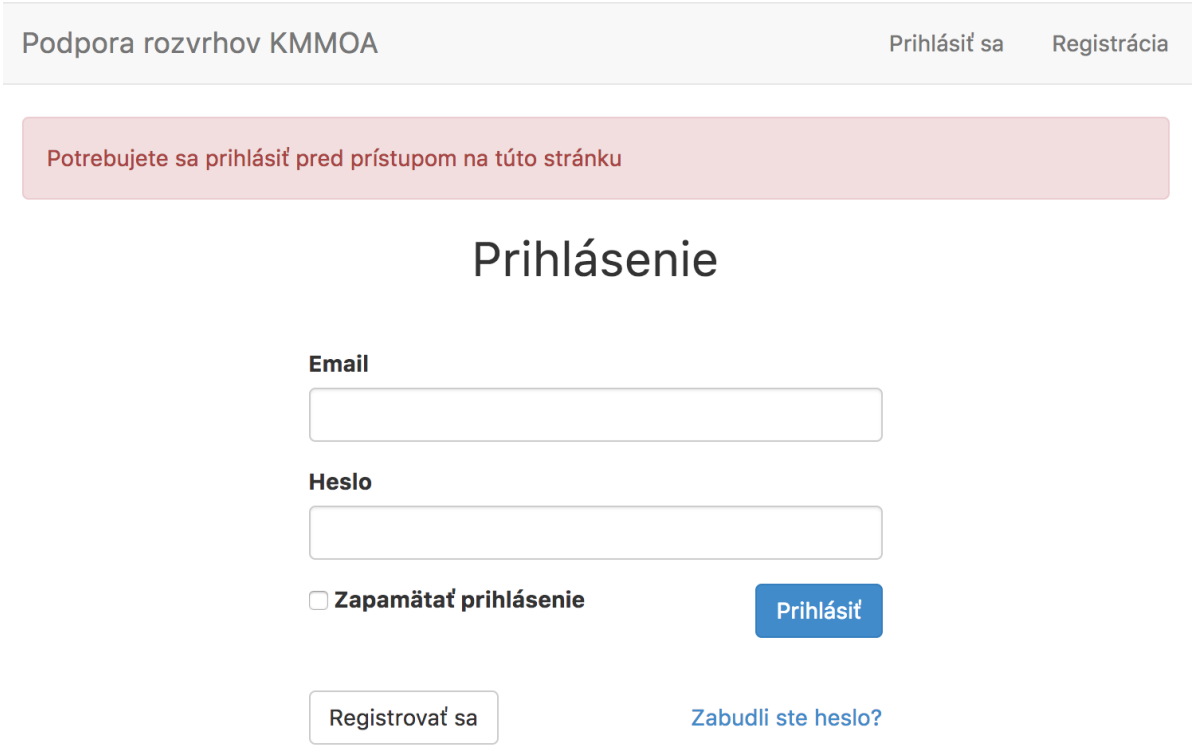
3.7 Generovanie PDF prehľadov

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] Members of the Ruby community. About ruby. `https://www.ruby-lang.org/en/about/`. [Online, 20.2.2017].

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