AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI
I ELEKTRONIKI

KATEDRA TELEKOMUNIKACJI

Praca dyplomowa

magisterska

Imię i nazwisko **Igor Rzegocki**

Kierunek studiów Elektronika i Telekomunikacja

Temat pracy dyplomowej **„Migracja serwisu WWW z techniki PHP na Ruby on Rails”**

Opiekun pracy dr inż. Krzysztof Juszkiewicz

Kraków, rok 2008

Pracę swą dedykuję Grzegorzowi, którego nagła i niespodziewana śmierć pozostawiła ogromną pustkę w sercach jego przyjaciół

Spis treści

Wstęp

Dynamiczny rozwój Internetu stawia przed programistami coraz to nowe wyzwania. Jeszcze kilka lat temu, statyczna strona z podstawowymi informacjami była produktem odpowiadającym wymaganiom większości klientów, a sam rynek internetowy nie był szczególnie atrakcyjny. Dziś jest inaczej. Szacuje się, że co piąty człowiek na Ziemii[1] korzysta z sieci. To ogromny rynek, obok którego producenci i reklamodawcy nie mogą przejść obojętnie. Powstaje coraz więcej firm ukierunkowanych tylko na biznes internetowy, a strony WWW dawno już przestały być prostymi wizytówkami – obecnie są to już całe aplikacje, które niewiele ustępują programom do instalacji w tradycyjny sposób. Przeglądarka internetowa stała się podstawowym programem, bez którego wielu ludzi nie wyobraża sobie korzystania z komputera.

Niniejszą pracę można podzielić na dwie części – implementacyjną i opisową.

Część implementacyjną stanowi gotowy do wdrożenia kod źródłowy. Jest to aplikacja będąca internetowym serwisem Kierunku Elektroniki i Telekomunikacji AGH. Źródła zawierają cały serwis przepisany na nowo z nieco zmienioną funkcjonalnością – usunięto katalog odnośników i forum. Największą zmianą było przeprojektowanie systemu według zupełnie nowej koncepcji programistycznej. Obecnie stosowane skrypty PHP zastąpił język Ruby wraz ze środowiskiem programistycznym Ruby on Rails. Oznaczało to przebudowanie całej koncepcji algorytmicznej aplikacji na nowo. Należało zaprojektować nową bazę danych zgodną z konwecją stosowaną w Ruby on Rails. Następnie potrzebny był konwerter przenoszący bezstratnie dane z obecnej bazy do nowej. Nowej logiki wymagał też system deklaracji studenckich.

Ponieważ serwis zostanie wdrożony i będzie utrzymywany przez pracowników Katedry Telekomunikacji, konieczne jest dostarczenie im odpowiedniej dokumentacji projektowej. Niniejsza praca stanowi kompromis pomiędzy taką dokumentacją, a ścisłymi wymaganiami stawianym pracom magisterskim.

W rozdziale 1 opisane zostały obecnie stosowane rozwiązania serwerowe w aplikacjach internetowych. Skupiono się na najpopularniejszych językach programowania i dostępnych dla nich środowiskach programistycznych.

[1] *Światowe statystyki dostępu do Internetu*, zob. http://www.internetworldstats.com/stats.htm

Kolejna część nr 2 to zapoznanie się z rozwiązaniami stosowanymi po stronie klienta. Koncentruje się ona głównie na obecnie najpopularniejszych rozwiązaniach obsługiwanych przez przeglądarki internetowe i na związanych z nimi problemach.

Następnie (3) przedstawione zostały wszystkie koncepcje programistyczne, które wymusza na programistach platforma Ruby on Rails. Krótki opis każdej koncepcji zawiera myśl przewodnią, informacje o rozwiązaniach oraz wskazuje możliwości wykorzystania w aplikacjach internetowych.

Rozdział nr 4 to dokładny opis aplikacji zarazem stanowiący jej dokumentację. Opisane zostały wszystkie tabele w bazie danych, relacje między nimi, mapa aplikacji oraz wszelkie niezbędne komponenty. Przedstawione zostały również problemy, na jakie można się natknąć przy tworzeniu serwisu o podobnym stopniu skomplikowania.

Nadrzędnym celem tej pracy było stworzenie kodu, który będzie mógł być wykorzystywany przez następne lata, będzie elastyczny i czytelny, a każdy nowy programista w projekcie będzie mógł nanosić swoje poprawki nie zmniejszając czytelności źródeł.

Rozdział 1

Techniki serwerowe wykorzystywane w aplikacjach internetowych

Ponieważ rynek aplikacji webowych jest dzisiaj duży, wielu producentów oprogramowania chce na nim zaistnieć. Dlatego prawie każdy język proponuje rozwiązania wspomagające budowę aplikacji internetowych. Najpopularniejszymi technikami są C#.NET, Java, Python, Ruby oraz PHP. Jeszcze 10 lat temu dużą uwagę zwracano na wydajność. Sprzęt komputerowy był stosunkowo drogi, więc sięgano do języków niskopoziomowych, które mogły zapewnić szybkie działanie przy małym zużyciu zasobów. Jednak dzisiaj sytuacja diametralnie się zmieniła. Oczywiście wydajność jest nadal ważna, ale zmiana serwera na szybszy lub dokupienie kolejnego aby przyspieszyć działanie aplikacji, nie jest już tak relatywnie drogie. Na pierwszy plan wychodzą rozwiązania, które są wygodne dla programistów i pozwalają w krótkim czasie dodać nową funkcjonalność aplikacji lub zmodyfikować istniejącą. Obecnie niewielu programistów używa tylko standardowych bibliotek do budowy aplikacji. Każdy język oferuje dodatkowe biblioteki lub całe rozwiązania, które ułatwiają tworzenie i rozwój systemu.

Poniżej zostały scharakteryzowane wyżej wymienione języki, oraz dostępne dla nich rozwiązania.

1.1 Interfejs CGI

Na początku wszystkie strony internetowe były statyczne. To znaczy ich autorzy tworzyli osobny plik z kodem HTML dla każdej strony. Zmiana czegokolwiek wymagała każdorazowej edycji kodu. Oczywiście takich stron nie można nazwać aplikacją, bardziej pasujące byłoby tu określenie publikacja.

Przełomem w tworzeniu stron i początkiem aplikacji internetowych było stworzenie w 1993 specyfikacji interfejsu CGI.

„CGI (ang. Common Gateway Interface) to znormalizowany interfejs, umożliwiający komunikację pomiędzy oprogramowaniem serwera WWW a innymi programami znajdującymi się na serwerze. Zazwyczaj program serwera WWW wysyła do przeglądarki statyczne dokumenty HTML. Za pomocą programów CGI można dynamicznie (na żądanie klienta) generować dokumenty HTML uzupełniając je np. treścią pobieraną z bazy danych."[1]

Interfejs CGI jest ogniwem, które łączy serwer WWW ze skryptem CGI. Zasada działania jest bardzo prosta. Serwer WWW dostarcza dane do programu poprzez standardowe wejście, lub pod postacią zmiennych środowiskowych. Program dostarcza swój wynik na standardowe wyjście, który następnie jest wysyłany przez serwer do przeglądarki.

Dzięki temu, że po drugiej stronie interfejsu można umieścić program, twórca serwisu jest w stanie dynamicznie generować dokumenty przed wysłaniem ich do klienta. Jako źródło danych może posłużyć baza danych. Następnie te dane można sformatować aby były prezentowane w sposób czytelny w oknie przeglądarki. Program może otrzymać dane z formularza jako parametry, można w ten sposób generować, przetwarzać i zapisywać dane z ankiet i kwestionariuszy. Można też dynamicznie generować obrazy takie jak wykresy czy schematy.

Co najważniejsze implementacja programu CGI nie jest zależna od żadnej platformy. Każdy serwer WWW może zaimplementować (i przeważnie implementuje) mechanizm CGI. Programy CGI można pisać w dowolnym języku programowania, wiele języków posiada biblioteki ułatwiające obsługę interfejsu. Standard CGI[2] jest dostępny za darmo i nie zmienił się od 1995 roku.

Początkowo językami najczęściej wykorzystywanymi do współpracy z CGI były Perl oraz C. Taki wybór był w pełni uzasadniony. C był i jest bardzo popularnym językiem programowania, a jego największą zaletą jest niewątpliwie wydajność. Natomiast Perl jest językiem stworzonym do przetwarzania danych tekstowych, więc znakomicie nadaje się do generowania kodu HTML.

Mechanizm interfejsu CGI jest wykorzystywany do dzisiaj, jednak nie jest to najszybsza metoda komunikacji pomiędzy serwerem WWW, a aplikacją obsługującą żądania HTTP. Programy CGI zostały zastąpione przez rozszerzenia dla serwerów dedykowane dla danego języka programowania, lub nawet całe serwery dedykowane.

[1] *Definicja CGI*, zob. http://pl.wikipedia.org/wiki/CGI

[2] *Pełna specyfikacja standardu CGI*, zob. http://www.w3.org/CGI/ i http://hoohoo.ncsa.uiuc.edu/cgi/

1.2 Java

Java jest darmową techniką rozwijaną pod kierunkiem firmy Sun Microsystems. Kod języka jest kompilowany do tzw. bajtkodu, który następnie jest wykonywany przez maszynę wirtualną. Głównymi cechami języka są: pełna obiektowość, niezależność od architektury (raz stworzony bajtkod może być wykonywany na wirtualnej maszynie uruchomionej w dowolnym systemie operacyjnym), sieciowość i obsługa programowania rozproszonego, niezawodność i bezpieczeństwo.

Wczesne wersje języka były krytykowane za powolne działanie, jednak dzisiejsze wersje są o wiele szybsze i bardzo stabilne. Do budowy aplikacji webowych Sun proponuje serwlety wykonywane po stronie serwera a także aplety które mogą być osadzane na stronie WWW i wykonywane przez przeglądarkę na komputerze użytkownika. Jedną z alternatyw jest bezpłatny zestaw narzędzi Struts[3], wspomagany przez bibliotekę Hibernate[4] ułatwiającą obsługę bazy danych. Popularny jest także projekt Spring[5]. Technika Java posiada także swój serwer WWW – Tomcat[6]. Budowę aplikacji ułatwiają bardzo dobre zintegrowane środowiska programistyczne Eclipse lub NetBeans dedykowane dla tego języka.

Dużą zaletą języka jest to, że technika ta ma bardzo dużo zastosowań. Można w niej tworzyć takie rozwiązania, jak aplikacje okienkowe czy nawet aplikacje dedykowane dla telefonów komórkowych. Posiada bardzo rozbudowane biblioteki obsługi sieci i oprogramowania rozproszonego. Sprawdza się wszędzie tam, gdzie jest jeden duży system, składający się z mniejszych aplikacji, który ma za zadanie pełnić wiele ról przy jednoczesnej integracji. Taki przykładem może być system ERP (ang. Enterprise Resource Planning, planowanie zasobów przedsiębiorstwa) IFS[7], który jest w całości napisany w języku Java.

- Zalety języka:
  - stabilność i wydajność,
  - cena (za darmo lub otwarty kod),
  - międzyplatformowość,
  - mnogość możliwych rozwiązań.
- Wady:
  - mnogość możliwych rozwiązań (czasami jest ich taka ilość, że ciężko jest wybrać coś dobrego),

[3] *Strona domowa projektu Struts*, zob. http://struts.apache.org/
[4] *Strona domowa projektu Hibernate*, zob. http://www.hibernate.org/
[5] *Strona domowa projektu Spring*, zob. http://www.springframework.org/
[6] *Strona domowa projektu Tomcat*, http://tomcat.apache.org/
[7] *Polski oddział IFS*, zob. http://www.ifsworld.com/pl/

  - potrzeba przekompilowania kodu po każdej zmianie,
  - rozwój aplikacji w języku Java jest stosunkowo pracochłonny.

1.3 C#.NET

Język C# jest odpowiedzią firmy Microsoft na technikę Java. Podobnie jak ona jest obiektowym językiem wysokopoziomowym. Kompiluje się do języka Common Intermediate Language (CIL) (podobnie jak bajtkod Javy), który następnie wykonywany jest w odpowiednim środowisku uruchomieniowym. Do budowy aplikacji webowych wykorzystuje się platformę .NET[8]. Może ona współpracować również z innymi językami programowania takimi jak: Visual Basic, J#, Jscript.Net, COBOL, Fortran, Lisp, Pyton, Perl i kilka innych. Środowisko .NET ma o wiele większe możliwości niż tylko aplikacje webowe. Z powodzeniem może posłużyć do budowania aplikacji okienkowych dla systemu Windows. Dedykowany serwer WWW dla aplikacji .NET to IIS[9], oczywiście firmy Microsoft. Firma Microsoft dostarcza także środowisko programistyczne Visual Studio.

Platforma .NET z językiem C# ma także szersze spektrum zastosowań. Do bardziej znanych rozwiązań należy system obsługi banku PKO BP S.A. czy Getin BANK S.A zbudowane przez krakowską firmę Vsoft[10]. Obydwa systemy obsługują większą cześć procesów w banku i posiadają także moduł bankowości internetowej. Dobrym przykładem zastosowań webowych jest serwis społecznościowy MySpace[11], który posiada ponad 100 milionów użytkowników na całym świecie.

- Zalety:
  - pełne wsparcie techniczne ze strony Microsoftu,
  - sprawdzona i stabilna technika.
- Wady:
  - cena,
  - całkowita zależność od platformy Microsoft.

1.4 PHP

Język PHP[12] (ang. PHP: Hypertext Preprocessor) był od początku projektowany do wykorzystania dla potrzeb aplikacji webowych. Pierwotnie był raczej zestawem

[8] *Projekt .NET*, zob. http://www.microsoft.com/net/
[9] *Internet Information Services*, zob. http://www.iis.net/
[10] *Firmowa wizytówka VSoft*, zob. http://www.vsoft.pl/
[11] *myspace.com - a place for friends*, zob. http://www.myspace.com/
[12] *Projekt PHP*, zob. http://www.php.net/

narzędzi niż językiem, dopiero od wersji 3, można mówić o języku programowania. Jest on językiem interpretowanym, to znaczy przy każdym uruchomieniu interpreter czyta kod źródłowy i wykonuje go. Dzięki temu każda zmiana w aplikacji ma natychmiastowy efekt. Chociaż udostępnia on możliwość programowania obiektowego, tak naprawdę jest językiem proceduralnym. Jego popularność jest bardzo duża ze względu na łatwość jego użycia. Jest całkowicie darmowy (otwarty kod), a jego interpreter jest dostępny na wiele platform. Istnieje tu wiele platform (np. Zend Framework, Zoop Framework, Symfony, Prado, CakePHP), wiele bibliotek do obsługi baz danych (np. ADOdb, Propel, PEAR::DB). Jednak wydaje się, że język PHP nie ma przed sobą przyszłości. Lista zarzutów jest poważna:

- jest wiele nieścisłości w nazwach funkcji i kolejności argumentów,
- część standardowych bibliotek jest udostępniana proceduralnie, a część obiektowo,
- kompatybilność wsteczna nie pozwala na uporządkowanie języka,
- brak możliwości obsługi niektórych błędów powoduje zatrzymanie wykonywania aplikacji.

Skrypty PHP są obsługiwane najczęściej za pomocą serwera Apache2.

Język nadaje się zarówno do budowy dużych aplikacji (np. http://www.yahoo.com/, http://www.interia.pl/, http://facebook.com), jak i do budowy małych komponentów (np. licznik odwiedzin, księga gości). Jest on bardzo często pierwszym językiem od którego programiści zaczynają przygodę z aplikacjami webowymi.

- Zalety:
  - łatwość użycia,
  - popularność,
  - dostępność,
  - cena (open source).
- Wady:
  - język proceduralny,
  - brak perspektyw rozwoju,
  - niespójne API (ang. Application Programming Interface)

1.5 Python

Python[13] to interpretowany język programowania stworzony jako następca języka ABC[14]. Jest to język zaprojektowany z myślą o jak największej produktywności, prostocie i czytelności kodu. Obsługuje też typy dynamiczne oraz automatyczne zarządzanie pamięcią. Ciekawą cechą Pythona jest to, że mimo iż różne części języka są opisane i ustandaryzowane, to język sam w sobie wciąż nie ma specyfikacji.

Python jest językiem, który wspiera różne paradygmaty programowania:

- programowania strukturalnego,
- programowania funkcjonalnego,
- programowania obiektowego.

Kolejną jego cechą jest rozszerzalność. Zamiast wbudowywać wszystko w rdzeń języka, można łatwo dokładać napisane w C lub C++ moduły. Dzięki temu sam rdzeń pozostaje mały, elastyczny i wydajny. Takie rozwiązanie wzięło się z frustracji autora językiem ABC, w którym założenia były dokładnie odwrotne [15].

Python bywa często wykorzystywany w aplikacjach internetowych. Najbardziej znana to http://www.youtube.com/, a z polskich http://www.grono.net/. Doczekał się również wielu platform programistycznych – najpopularniejsze to: Django[16], Zope[17], TurboGears[18] i Pylons[19].

- Zalety:
  - elastyczność i rozszerzalność,
  - wydajność,
  - wiele środowisk programistycznych.
- Wady:
  - duża swoboda programowania powoduje bałagan w kodzie,
  - wymóg stosowania wcięć znacznie utrudnia stworzenie dobrego systemu szablonów.

[13] *Strona domowa projektu Python*, zob. http://python.org/
[14] *ABC Programmers Handbook*, zob. http://idhub.com/abc/
[15] *Wywiad z Guido van Rossumem*, zob. http://www.artima.com/intv/pythonP.html
[16] *Strona domowa projektu Django*, zob. http://www.djangoproject.com/
[17] *Strona domowa projektu Zope*, zob. http://www.zope.org/
[18] *Strona domowa projektu TurboGears*, zob. http://www.turbogears.org/
[19] *Strona domowa projektu Pylons*, zob. http://www.pylonshq.com/

1.6 Ruby

Ruby (dosł. rubin) to interpretowany, w pełni obiektowy język programowania stworzony w 1995 roku przez Yukihiro Matsumoto. Jak twierdzi jego autor[20], Ruby został zaprojektowany dla optymalizacji wydajności programistów, przy zachowaniu zasady dobrego interfejsu. Ze względu na pochodzenie języka (Japonia), początkowo był on mało popularny z powodu braku angielskiej dokumentacji. Jednak dzięki Internetowi rosła jego popularność i z czasem zaczęły się pojawiać artykuły w języku angielskim a wraz z nimi społeczność skupiona wokół tego języka. Przełomem była prezentacja w 2003 roku platformy Ruby On Rails (dosł. rubin na szynach). Posiada ona wszystko, co jest niezbędne do budowy aplikacji webowych i znakomicie nadaje się do bardzo szybkiej budowy małych aplikacji, a także do budowania prototypów aplikacji. Interpretery języka istnieją na każdą platformę. Jedyną wadą zdaje się być jego niska wydajność (podobnie jak we wczesnych wersjach języka Java), którą jednak można zwiększyć wykorzystując JRuby (implementację interpretera w języku Java). Ruby posiada swoje dwa serwery WWW – Mongrel oraz Webrick. Można używać także serwera Apache za pośrednictwem interfejsu CGI lub rozszerzenia *mod_ruby*.

- Zalety:
  - darmowy,
  - pełna obiektowość,
  - duże perspektywy rozwoju,
  - przejrzystość kodu,
  - bardzo prosty w nauce,
  - duża aktywna społeczność zawsze chętna do pomocy.
- Wady:
  - niska wydajność,
  - brak znanych udanych wdrożeń.

1.7 Ruby on Rails

Współdziałanie środowiska Rails i języka Ruby ma umożliwiać programiście szybkie, wydajne i łatwe tworzenie aplikacji internetowych. Efekt taki uzyskano łącząc

[20] *Artykuł Yukihiro Matsumoto*, zob. http://www.informit.com/articles/article.aspx?p=18225

cechy dynamicznego i obiektowego języka ze sprawdzonymi wzorcami i schematami dotyczącymi projektowania aplikacji internetowych.

„Często ludzie, zwłaszcza inżynierowie oprogramowania skupiają się na maszynach. Sądzą oni iż robiąc to tak sprawimy, iż maszyna będzie pracowała szybciej. Jeśli znów zmienimy coś innego to znów maszyna przyśpieszy. Skupiają się tylko na procesorach i wydajności. Tak naprawdę potrzebujemy skupić się na programistach. Na tym jak ludzie chcą aby ich aplikacje były zaprojektowane i jak działały. To my jesteśmy władcami. One są niewolnikami."[21]. Słowa te najlepiej obrazują filozofię języka, jak i to co można uzyskać używając połączonych sił Ruby i Rails.

Cechy Ruby on Rails to przede wszystkim:

- Szybkość, łatwość i przyjemność z pisania,
- Reguła DRY – Don't Repeat Yourself (zob. rozdz. 3.6, str. 29),
- Reguła Convention over Configuration (zob. rozdz. 3.7, str. 31),
- Architektura MVC (Model-view-controller) jako sprawdzony wzorzec budowania aplikacji internetowych (zob. rozdz. 3.1, str. 20),
- Testy, jako integralna część aplikacji, do pomocy i automatyzacji procesu wyszukiwania błędów,
- Prostota w dodawaniu wtyczek (rozszerzają aplikację o całą gamę gotowych funkcjonalności od prostego logowania do całych modułów zarządzających).

Platforma Rails zbudowany jest z trzech zasadniczych elementów:

- *Active Record* – mechanizm ORM (Object-relational Mapping) (zob. rozdz. 3.2, str. 22) odpowiada za odwzorowanie obiektowej architektury obiektów na relacyjną strukturę bazy danych. Różni się on od zwyczajnych systemów ORM minimalizacją potrzebnych konfiguracji ze strony programisty. Odpowiada za generowanie modeli.
- *Action Pack* – zestaw klas, w którego skład wchodzą klasy obsługujące widoki oraz kontrolery. Dzięki uniwersalnemu podejściu widoki nie muszą tylko i wyłącznie ograniczać się do produkowania kodu HTML, który potem wyświetlany jest w przeglądarce. W bardzo prosty sposób możemy przekształcać nasze widoki do XML'a , generować dynamiczny JavaScript , używać protokołu SOAP oraz wiele innych. Kontrolery natomiast są logiką całej naszej aplikacji. Zarządzają interakcjami między użytkownikami a systemem, modelami a widokami itd.

[21] *Wywiad z Yukihiro Matsumoto*, zob. http://www.artima.com/intv/ruby.html

- Mechanizm „scaffolding" – służy bardzo szybkiemu tworzeniu gotowych powiązań między modelem a kontrolerem i widokami. W praktyce jest to po prostu generator gotowego kodu źródłowego dla danego modelu. Tworzone są wszystkie podstawowe akcje jak stwórz, czytaj, edytuj, kasuj (zob. rozdz. 3.5, str. 28). Zwalnia to programistę z pisania po raz kolejny tych samych wspólnych zachowań dla modelu i pozwala skupić się na tym jak aplikacja ma działać, a nie na wymyślaniu kolejnego systemu walidacji danych.

Kolejnym ważnym elementem przemawiającym na korzyść platformy, jest ilość dostępnych darmowych wtyczek, które automatyzują/ułatwiają wykonywanie określonych zadań. Główny serwis `http://rubyforge.org`, który zajmuje się zbieraniem i informowaniem o wtyczkach, posiada obecnie 5912[22] różnych wtyczek, które możemy użyć w naszym projekcie.

Środowisko Rails, znacznie upraszcza również proces tworzenia tabel w bazie danych, dzięki tzw. migracjom. Są to skrypty w języku Ruby, które odpowiadają za tworzenie nowych tabel, zakładanie kluczy itd. Skrypty te następnie uruchamia się poleceniem *rake db:migrate*.

Dzięki uniwersalności i prostocie możemy stosować środowisko Ruby on Rails wszędzie tam, gdzie potrzebujemy szybko, przyjemnie i bezproblemowo stworzyć aplikację internetową. W bardzo krótkim czasie możemy przygotować gotowe elementy naszego systemu. Każdy kawałek kodu ma swoje jedno dokładnie pasujące miejsce. Tworząc tylko podstawowy szkielet aplikacji mamy już bardzo dużo rzeczy zrobionych za nas, czekających tylko do wykorzystania.

Największą aplikacją napisaną całkowicie w technice Ruby on Rails jest Basecamp[23]. Basecamp jest aplikacją wspomagającą zarządzanie projektami, która obecnie obsługuje ponad milion użytkowników. Drugą duża aplikacją jest Twitter[24]. Twitter jest z kolei platformą służącą do „mikroblogowania" (czyli pisania wpisów nie dłuższych niż 140 znaków). Obecnie cała społeczność Ruby on Rails śledzi losy Twittera, gdyż ma on ogromne problemy wydajnościowe. Od tego jak je rozwiąże, zależy jak inwestorzy komercyjni ocenią przydatność samego środowiska.

Struktura katalogów

Środowisko Ruby on Rails już od początku narzuca programiście strukturę katalogów. W jednym, zawsze tym samym i jasno określonym miejscu umieszczone są modele, w innym widoki itd. Struktura katalogów jest następująca:

- *app* – zawiera katalogi tworzące rdzeń aplikacji, ułożone zgodnie z metodologią MVC (zob. rozdz. 3.1, str. 20),

[22] Stan na 24 maja 2008 r.
[23] *Zarządzanie projektami Basecamp*, zob. http://www.basecamphq.com/
[24] *Mikroblogowanie Twitter*, zob. http://www.twitter.com/

  - *controllers* – kontrolery,
  - *helpers* – funkcje pomocnicze (są to proste funkcje tworzone przez programistę, mające automatyzować czynności często powtarzane w widoku, jak np. tworzenie list)
  - *models* – modele,
  - *views* – widoki (dodatkowo może zawierać katalog *_partials*, w którym umieszcza się kawałki kodu HTML wykorzystywanego w wielu miejscach, tzw. partiale)

- *components* – komponenty (zob. rozdz. 4.7, str. 74),
- *config* – pliki konfiguracyjne,
- *db* – automatycznie generowany schemat bazy danych oraz migracje,
- *doc* – automatycznie wygenerowana dokumentacja projektu,
- *lib* – dodatkowe biblioteki,
- *log* – wszystkie akcje występujące podczas działania aplikacji (wejście na stronę, zapytania SQL, błędy, itd.) zapisywane są w logach umieszczonych w tym katalogu,
- *public* – katalog widoczny z poziomu serwera WWW, zawiera wszystkie pliki, które mogą być dostępne dla przeglądarki WWW (obrazki, arkusze stylów, pliki języka JavaScript, itd.),
- *script* – skrypty środowiska, takie jak serwer WWW czy konsola wiersza poleceń,
- *test* – testy,
- *tmp* – wszystkie pliki tymczasowe, takie jak identyfikatory sesji, czy tymczasowe pliki serwera WWW,
- *vendor* – wszystkie rozszerzenia aplikacji, w przeważającej większości są to wtyczki (ang. plugins).

Taki sposób organizacji kodu wymusza porządek zgodny z konwencją i w każdym projekcie (tak samo zorganizowanym) wiadomo, gdzie znajdują się odpowiednie elementy.

Rozdział 2

Techniki po stronie klienta

2.1 HTTP

„HTTP (HyperText Transfer Protocol - protokół przesyłania dokumentów hypertekstowych) to protokół sieci WWW. Obecną definicję HTTP stanowi RFC 2616. Za pomocą protokołu HTTP przesyła się żądania udostępnienia dokumentów WWW, informacje o kliknięciu odnośnika oraz informacje z formularzy. Zadaniem stron WWW jest publikowanie informacji – natomiast protokół HTTP właśnie to umożliwia."[1]

HTTP jest to protokół zapytań i odpowiedzi pomiędzy klientem a serwerem. Jest on bardzo użyteczny, ponieważ udostępnia znormalizowany sposób komunikowania się komputerów ze sobą. Określa on formę żądań klienta dotyczących danych oraz formę odpowiedzi serwera na te żądania. Jest to protokół bezstanowy ponieważ nie zachowuje żadnych informacji o poprzednich transakcjach z klientem. Protokół definiuje osiem metod, które mogą być użyte w żądaniu HTTP:

- *GET* – pobranie informacji wskazanej przez identyfikator URL (ang. Uniform Resource Locator, jednoznaczny wskaźnik zasobu),
- *HEAD* – pobiera informacje o tym czy dana informacja istnieje,
- *PUT* – informacja potwierdzająca pobranie danych w postaci pliku od klienta przez serwer,
- *POST* – analogicznie jak powyżej tyle, że informacja nie musi być plikiem (np. formularze),
- *DELETE* – żądanie usunięcia informacji – wymagane odpowiednie uprawnienia,

[1] *Definicja protokołu HTTP*, zob. http://pl.wikipedia.org/wiki/HTTP

- *OPTIONS* – informacje o opcjach i wymaganiach istniejących w kanale komunikacyjnym,
- *TRACE* – analiza i diagnostyka kanału komunikacyjnego,
- *CONNECT* – żądanie wykorzystywane w tunelujących serwerach proxy.

Dla potrzeb przeglądania stron WWW używane są metody GET oraz POST. Natomiast cały zestaw metod protokołu HTTP daje znacznie większe możliwości i może posłużyć do kompleksowej wymiany informacji pomiędzy komputerami w sieci Internet.

2.2 HTML

Zawartość strony internetowej jest hipertekstem, znaczy to, że użytkownik oglądając stronę internetową może podążać za hiperłączami, które przenoszą go do innych stron internetowych w ramach tego samego serwera internetowego lub innych dostępnych w ramach sieci. Znaczy to też obecność grafiki i innych multimediów.

„Hipertekst to organizacja danych w postaci niezależnych leksji połączonych hiperłączami. Hipertekst cechuje nielinearność i niestrukturalność układu leksji. Oznacza to, że nie ma z góry zdefiniowanej kolejności czytania leksji, a nawigacja między nimi zależy wyłącznie od użytkownika."[2]

Implementacją hipertekstu na potrzeby WWW jest język HTML.

„HTML (HyperText Markup Language, hipertekstowy język znaczników), to język składający się ze znaczników oraz reguł ich poprawnego stosowania (gramatyki, semantyki), stosowany do pisania stron WWW. HTML jest teoretycznie aplikacją SGML, tzn. został zdefiniowany za pomocą SGML, będącego tzw. metajęzykiem (językiem służącym do definiowania innych języków)."[3]

Wraz z rozwojem sieci WWW pojawiła się potrzeba rozwoju języka HTML tak, aby posiadał on możliwość dołączania do tekstów danych tabelarycznych, grafik czy plików multimedialnych. Kolejne wersje języka rozwijane były niezależnie przez producentów przeglądarek internetowych, co doprowadziło do częściowej niekompatybilności wersji HTML zaimplementowanych w przeglądarkach różnych producentów. Próbą odpowiedzi na tę sytuację było stworzenie W3C[4] czyli World Wide Web Consortium, organizacji, która zajmuje się ustanawianiem wspólnych standardów HTML, a także innych spraw związanych z pisaniem stron WWW. Ostatnią wersją HTML jest wersja 4.01, która próbuje wydzielić zarządzanie wyglądem strony do kaskadowych arkuszy stylów (CSS). Na jakiś czas W3C zaprzestało rozwoju HTML

[2] *Definicja hipertekstu*, zob. http://pl.wikipedia.org/wiki/Hipertekst
[3] *Definicja HTML*, zob. http://pl.wikipedia.org/wiki/HTML
[4] *World Wide Web Consortium*, zob. http://www.w3c.org/

i postanowiło dostosować język do formatu XML (ang. eXtensible Markup Language). W wyniku tego powstał XHTML, dla którego istnieje tryb zgodności z HTML, do wyświetlenia kodu XHTML w przeglądarkach zgodnych z HTML 4.01. Zmiana ta ma zapewnić większą rozszerzalność i dostępność języka. Z tego powodu właśnie XHTML jest obecnie zalecanym standardem tworzenia stron WWW. Obecną najnowszą wersją hipertekstu na potrzeby stron WWW jest XHTML 1.1[5], wraz z arkuszami styli CSS 2.1[6]. Kolejną wersją języka (X)HTML miał być XHTML 2.0, jednak ta droga rozwoju została ostatecznie uznana za nietrafioną i rozwój XHTML 2.0 został zarzucony na rzecz HTML 5[7], nad rozwojem którego trwają obecnie intensywne prace.

2.3 CSS

„Kaskadowe arkusze stylów (ang. Cascading Style Sheets, CSS) to język służący do opisu formy prezentacji (wyświetlania) stron WWW. Standard CSS został opracowany przez organizację W3C w 1996 r. jako potomek języka DSSSL przeznaczony do używania w połączeniu z SGML-em. Pierwszy szkic CSS zaproponował w 1994 r. Håkon Wium Lie.[8]”

Język CSS jest wykorzystywany do wspomagania czytników stron WWW w definiowaniu kolorów, czcionek, układów i innych aspektów prezentacji dokumentu. Głównym założeniem projektowym była separacja warstwy treści (zapisanej w HTML albo w podobnym języku znaczników) od warstwy prezentacji (zapisanej w CSS). Taka separacja znacznie podnosi dostępność treści, zapewnia dużo większą elastyczność i kontrolę nad aspektami prezentacyjnymi dokumentu. Dodatkowo redukuje powtarzalność w warstwie dokumentu (jeżeli chcemy np. aby wszystkie nagłówki były koloru zielonego, nie musimy go nadawać każdemu z nich z osobna). Arkusz CSS pozwala również na to, aby ta sama treść była wyświetlana w różny sposób na różnych urządzeniach (np. inaczej będziemy wyświetlać stronę na ekranie komputera, inaczej na telefonie komórkowym, a jeszcze inaczej w wersji do druku). Określa się też priorytety z jakimi należy nakładać kolejne właściwości. Najnowszą wersją standardu CSS jest obecnie wersja 2.1 – trwają jednak prace, nad wersją trzecią[9].

[5] *Specyfikacja standardu XHTML 1.1*, zob. http://www.w3.org/TR/xhtml11/
[6] *Specyfikacja standardu CSS 2*, zob. http://www.w3.org/TR/REC-CSS2/
[7] *Specyfikacja standardu HTML 5*, zob. http://www.w3.org/html/wg/html5/
[8] *Pierwszy projekt CSS*, zob. http://www.w3.org/People/howcome/p/cascade.html
[9] *Specyfikacja trzeciej, rozwojowej wersji CSS*, zob. http://www.w3.org/Style/CSS/current-work

2.4 JavaScript

JavaScript jest skryptowym językiem programowania, który został opracowany przez firmę Netscape dla potrzeb stron internetowych. JavaScript jest wersją standaryzowanego języka ECMAScript[10].

Nazwa nasuwa skojarzenie z językiem Java. Oba języki nie mają ze sobą wiele wspólnego, oprócz tego że ich składnia jest zaczerpnięta z języka C. W połączeniu z Document Object Model[11], JavaScript stał się o wiele bardziej potężną techniką niż przewidywali jej twórcy. Po załadowaniu strony programista ma możliwość manipulacji strukturą dokumentu (załadowanej do przeglądarki strony), może także tworzyć funkcje, które reagują na zdarzenia występujące w oknie przeglądarki (ruchy myszką, klikniecie, naciśnięcie klawisza na klawiaturze itd.). Połączenie języka HTML ze skryptami języka JavaScript nazywane jest Dynamic HTML (DHTML), czyli dynamiczny HTML, aby uwydatnić różnicę w stosunku do statycznych stron HTML.

Przełomem w rozwoju techniki DHTML było udostępnienie obiektu XMLHttpRequest (XHR). Umożliwia on wysyłanie żądań HTTP z poziomu języka JavaScript już po załadowaniu strony internetowej w trakcie interakcji z użytkownikiem. Otrzymane odpowiedzi serwera są wówczas wykorzystywane do modyfikacji załadowanego dokumentu. Możliwość asynchronicznego wykonywania żądań sprawia, że są one wykonywane w tle i nie przerywają interakcji użytkownika ze stroną, dynamicznie ją zmieniając. Technika wykorzystania obiektu XHR została nazwana AJAX (ang. Asynchronous JavaScript and XML, asynchroniczny JavaScript i XML). Dzisiaj każda przeglądarka udostępnia metody XHR. Technika AJAX sprawdza się tam gdzie nie ma potrzeby przeładowywania całej strony, a tylko jej części lub gdzie w ogóle nie trzeba dokonywać przeładowania a jedynie wysłać informację do serwera WWW. Rozwiązanie to, oszczędza czas użytkownika (w trakcie doładowywania informacji może on przeglądać stronę, która dalej jest dla niego dostępna) oraz zmniejsza obciążenie serwera (doładowywanie tylko części informacji zamiast całej strony ogranicza ilość danych wysyłanych do przeglądarki).

2.5 RSS

RSS (ang. Really Simple Syndication, bardzo proste rozpowszechnianie artykułów) jest to format języków znacznikowych używany do publikowania często zmieniającej się treści, takiej jak wpisy na blogach, nagłówki wiadomości, podkasty itd. Dokument RSS zawiera zazwyczaj podsumowanie przypisanej mu strony WWW lub jej

[10] *Pełny standard ECMAScript*, zob. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

[11] *Informacje na temat DOM*, zob. http://www.w3.org/DOM/

pełną zawartość.

Technika RSS umożliwia ludziom, bycie na bieżąco ze stronami internetowymi dzięki automatycznym programom zwanymi „czytnikami RSS". Aplikacje te, zbierają wiadomości z witryn wybranych przez użytkownika, a następnie przedstawiają w skondensowanej i przystępnej formie. Dzięki temu, nie ma potrzeby odwiedzania każdej witryny – wszystkie można przeglądać z poziomu jednej aplikacji. W praktyce oznacza to znaczną oszczędność czasu. Zasada działania agregatora RSS jest prosta. Co jakiś czas (aplikacja działająca w tle, lub agregator internetowy[12]) lub przy każdym uruchomieniu następuje sprawdzenie wszystkich wpisanych kanałów RSS, pod kątem nowej zawartości. Jeśli zostanie odnaleziona, następuje jej ściągnięcie i oznaczenie jako nieprzeczytanej.

Oczywiście wiadomość RSS nie jest w żaden sposób ograniczona w stosunku do normalnej strony WWW. Z racji tego, że jest to zwykły plik XML, może zawierać odnośniki, grafikę a nawet informacje multimedialne. Każda wiadomość RSS zawiera także hiperlink do oryginału (co jest istotne w przypadku skrótów).

Wyróżnia się następujące wersje protokołu RSS:

- Really Simple Syndication (RSS 2.0)
- RDF Site Summary (RSS 1.0 i RSS 0.90)
- Rich Site Summary (RSS 0.91).

Wszystkie powyższe wersje są wstecznie zgodne.

2.6 YAML

Format YAML to łatwy w odczycie dla człowieka sposób serializacji danych, który odziedziczył rozwiązania z języków takich jak XML, C, Python czy Perl. YAML po raz pierwszy zaprezentował Clark Evans w 2001 roku.

YAML to akronim rekursywny oznaczający „YAML Ain't a Markup Language" (YAML nie jest językiem znaczników). Na początku rozwoju słowo YAML oznaczało „Yet Another Markup Language" (kolejny język znaczników), ale rozwinięcie zostało zmienione głównie po to aby zwrócić uwagę, że YAML koncentruje się na samej informacji, a nie jej reprezentacji i semantyczności.

YAML został zaprojektowany w taki sposób, aby był łatwy do przenoszenia na typy danych stosowane w językach wysokiego poziomu – listy, hashe i tablice. Język ten świetnie nadaje się do zastosowań, w których niezbędna jej ingerencja człowieka nie znającego rozwiązań zastosowanych w aplikacji – czyli do plików konfiguracyjnych, nagłówków, informacji zwrotnych i kontrolnych. Jego prosta i klarowna

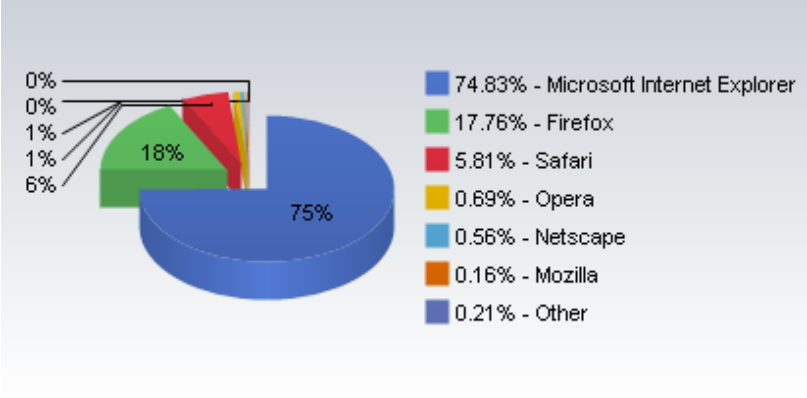
[12]Na przykład http://reader.google.com/

struktura znacznie ułatwia projektowanie takich plików i późniejsze przenoszenie informacji między systemami.

2.7 Przeglądarki internetowe

Przeglądarka internetowa jest to aplikacja, służąca do pobierania i wyświetlania treści tzw. stron internetowych. Strony internetowe to dokumenty w jednym z języków SGML (najczęściej HTML, XHTML lub XML), zawierające tekst, grafikę i pliki multimedialne.

Na rynku istnieje wiele przeglądarek, jednak tak naprawdę nie jest istotna ich nazwa, a silnik jaki wykorzystują do wyświetlania stron. Cztery najważniejsze silniki to: Trident (Internet Explorer), Gecko (np. Firefox, Netscape, Flock, SeaMonkey, Camino), KHTML/WebKit (np. Konqueror, Safari), oraz Presto (Opera). Oprócz przeglądarek graficznych warto jeszcze wspomnieć o przeglądarkach w trybie tekstowym. Są to Lynx, Links i eLinks – z czego ta ostatnia, potrafi renderować JavaScript. Jeżeli chodzi o udział przeglądarek w rynku przedstawia się jak na rysunku 2.1 (*Źródło: http://marketshare.hitslink.com/report.aspx?qprid=0*). W polsce, dobrym miernikiem popularności jest serwis http://www.ranking.pl/.



Rysunek 2.1: Udział procentowy w rynku przeglądarek internetowych.

Niestety silniki przeglądarek nie są ze sobą kompatybilne. Każdy trochę inaczej interpretuje HTML, CSS, JavaScript. Nie wszystkie mają wbudowane najnowsze techniki w tym samym stopniu. Największe trudności sprawia twórcom stron przeglądarka Internet Explorer w wersji 6, która zajmuje największą część rynku i zarazem jest najbardziej przestarzałą (2001 r.). Ta wersja jest niezgodna ze standardami zatwierdzonymi przez W3C i posiada wiele błędów, często nazywana jest „zmorą programistów". Niestety przeglądarka IE 6 wygrała pierwszą „wojnę przeglądarek"[13] i niemalże zmonopolizowała rynek. Następna wersja Internet Explorera

[13] *Definicja wojen przeglądarek*, zob. http://en.wikipedia.org/wiki/Browser_wars

(wersja 7) pojawiła się dopiero pod koniec 2006 roku. Zanim wersja 6 zniknie z rynku minie jeszcze kilka lat, w czasie których rozwój aplikacji po stronie klienta będzie ograniczony możliwościami tej przeglądarki.

Rozdział 3

Koncepcje w aplikacjach internetowych

3.1 Model-view-controller

MVC (ang. Model-view-controller, model-widok-kontroler) to wzorzec projektowy stosowany szeroko w informatyce, niemniej największą popularność zyskał w Internecie. Odpowiednie użycie wzorca oddziela logikę biznesową od warstwy prezentacji, co w efekcie przekłada się na możliwość zmiany wyglądu aplikacji bez wpływu na logikę biznesową i odwrotnie. We wzorcu MVC komponent o nazwie Model reprezentuje informacje o aplikacji i założeniach biznesowych użytych w celu obróbki danych. Widok, odpowiada za elementy UI (ang. User Interface, interfejs użytkownika) takie jak tekst, listy, pola wyboru itd. Kontroler spaja wszystko w całość pośrednicząc między Modelem a Widokiem i zajmując się sterowaniem sposobem działania aplikacji.

3.1.1 Krótka historia

Wzorzec MVC został po raz pierwszy opisany w 1979 r.[1] przez Trygve Reenskaug pracującegp wtedy w XEROX PARC nad językiem Smalltalk[2]. Oryginalna implementacja została dokładnie opisana w dokumencie *Applications Programming in Smalltalk-80: How to use Model-View-Controller.*[3]

Po pewnym czasie powstało wiele odmian koncepcji. Warto wspomnieć o wzorcu Model-view-presenter, który powstał początkiem lat 90 i został zaprojektowany jako następca koncepcji MVC. Nie zmienia to faktu, że wzorzec MVC trzyma się mocno i wciąż znajduje szerokie zastosowanie.

[1] *Koncepcja MVC*, zob. http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html
[2] *Język Smalltalk*, zob. http://www.smalltalk.org/
[3] Dostępny pod adresem http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html

3.1.2 Opis wzorca

Dość powszechnie stosowanym rozwiązaniem jest dzielenie aplikacji na warstwę prezentacji i warstwę logiczną. Wzorzec MVC idzie o krok dalej, dzieląc warstwę prezentacji na Widok i Kontroler. Podział kładzie większy nacisk na architekturę aplikacji niż typowy wzorzec projektowy. Trzy jego główne elementy to:

- *Model* – Specyficzna dla każdego projektu reprezentacja informacji, z którymi działa aplikacja. Jest to język DSL (ang. Domain Specific Language, zob. rozdz. 3.4 str. 27), który opisuje zależności i operuje na surowych danych (np. obliczenie czy dzisiaj użytkownik obchodzi urodziny, kosztu wszystkich towarów w koszyku). Wiele aplikacji wykorzystuje ściśle określony mechanizm przechowywania danych (w przypadku aplikacji internetowych, prawie zawsze jest to baza danych). Wzorzec MVC sam w sobie, nie określa metody przechowywania danych, ponieważ założenie jest takie, że danymi zajmuje się model.

- *Widok* – Przedstawia dane z modelu w postaci czytelnej dla użytkownika. Podejście MVC nie definiuje ścisłej relacji ilościowej pomiędzy Modelami a Widokami, co oznacza, że jeden Model może obsługiwać wiele widoków, jak również jeden Widok może otrzymywać dane od wielu Modeli.

- *Kontroler* – Odpowiada za proces interakcji między aplikacją a użytkownikiem. Pobiera „zapytania" od użytkownika i przekazuje je Modelowi. Następnie zwraca użytkownikowi nowy widok z nowymi informacjami. Może również dokonywać zmian w danych opisywanych przez Model.

3.1.3 Zastosowanie w aplikacjach internetowych

Wzorzec MVC często znajduje zastosowanie w aplikacjach internetowych, gdzie widokiem jest aktualnie wyświetlana strona WWW, a kontroler to kod, który pobiera dane dynamiczne i wypełnia nimi szablony HTML. Model z kolei reprezentuje dane (zazwyczaj przechowywane w bazie danych lub plikach XML) oraz reguły biznesowe, które zamieniają te dane na informację przedstawianą potem użytkownikowi.

Mimo, że wzorzec jest dostarczany w wielu środowiskach, z których każde zawiera odmienny pomysł na jego implementację, generalna zasada jest taka sama:

1. Użytkownik wykonuje jakąś akcję w interfejsie użytkownika (np. wciska guzik).

2. Kontroler obsługuje nadchodzące wywołanie, często wykorzystując jakąs ustaloną metodę lub odwołanie.

3. Kontroler informuje model o akcji użytkownika, czasami również modyfikuje jego stan (np. Kontroler uaktualnia zawartość koszyka).

4. Widok wykorzystuje model (pośrednio) do wygenerowania odpowiedniego interfejsu użytkownika (np. wyświetla tabelkę z zakupami). Widok pobiera dane z modelu, ale sam model nie wchodzi w interakcję z Widokiem.

5. Interfejs użytkownika oczekuje na kolejną akcję, co rozpoczyna cały cykl od początku.

Rozdzielając elementy należące do Modelu od Widoków, koncepcja MVC znacznie zmniejsza skomplikowanie architektury programu i zwiększa przejrzystość i elastyczność.

3.2 Mapowanie ORM

Mapowanie obiektowo-relacyjne (z ang. Object-relational mapping) jest techniką programistyczną stosowaną do konwersji danych przy użyciu niekompatybilnych ze sobą relacyjnych baz danych i przy obiektowo zorientowanych językach programowania. W efekcie otrzymywany jest „wirtualny obiekt bazy danych", który może być następnie użyty na poziomie kodu zadanego języka programowania. Istnieją zarówno darmowe jak i komercyjne rozwiązania mapowania ORM, niemniej sporo programistów wciąż stara się wdrażać swoje koncepcje.

3.2.1 Opis problemu mapowania

Zarządzanie danymi w programowaniu obiektowym zazwyczaj jest rozwiązane jako manipulowanie różnego rodzaju obiektami, które prawie nigdy nie są wielkościami skalarnymi. Dla przykładu, dany jest obiekt reprezentujący osobę w książce teleadresowej. Osoba taka, może mieć kilka numerów telefonów i kilka adresów (zameldowania, zamieszkania itd.). Obiekt reprezentujący taką osobę miałby kilka rekordów, które zawierałyby kolejno, dane osobowe, listę numerów telefonów i listę adresów. Oczywiście, lista telefonów może z kolei zawierać obiekty numerów telefonów (z osobnym wpisem, na operatora, numer kierunkowy itd.), lista adresów – obiekt adresu itd. Dodatkowo, wszystkie te obiekty miałyby swoje własne metody, np. do zwracania telefonu domowego, preferowanego adresu itd.

Z drugiej strony, bazy danych mogą przechowywać tylko dane skalarne zorganizowane w tabelach.

Programista musi więc albo zorganizować podobne wartości obiektów jako dane dla tabeli, albo całkowicie uprościć strukturę kodu, do poziomu wartości skalarnych

aby potem bezproblemowo umieszczać je w bazie. Podejście ORM wykorzystuje pierwsze rozwiązanie.

Cały problem polega na takiej organizacji tych obiektów, aby potem dało się je łatwo zapisać do bazy, jednocześnie nie tracąc samej struktury obiektu (w celu późniejszego odczytu). Obiekty takie muszą być więc łatwe do przenoszenia między kodem, a bazą danych.

3.2.2 Implementacje

Najpowszechniej stosowanym rozwiązaniem jest relacyjna baza danych, która poprzedziła narodziny koncepcji ORM w 1990 r. Relacyjne bazy danych, wykorzystują serie tabel do organizacji danych. Dane w różnych tabelach są ze sobą powiązane przy pomocy kluczy założonych na całe kolumny, a nie na poszczególne wartości. Może się okazać, że dane przechowywane w jednym obiekcie muszą być rozłożone na kilka tabel.

Model ORM powinien systematycznie przewidywać jakie tabele będą potrzebne do zapisania określonych obiektów i generować odpowiednie zapytania języka SQL. Różnice między sposobem organizacji danych w modelu obiektowo zorientowanym (w takich językach jak Java, C# czy Ruby) a sposobem ich przechowywania w relacyjnej bazie danych (takiej jak Oracle czy PostgreSQL), pociągają za sobą następujące problemy:

- wydajność,
- skalowalność,
- zarządzanie operacjami typu CRUD (patrz rozdz. 3.5, str. 28) dla bardziej skomplikowanych relacji,
- uproszczenie i spójność kodu konieczna przy szybkim tworzeniu aplikacji,
- zarządzanie i elastyczność kodu.

Prawdziwą wartością podejścia ORM jest przede wszystkim oszczędność czasu i uproszczenie kodu (wszystkie skomplikowane operacje bazodanowe wzorzec ORM wykonuje za programistę). Dodatkowo mapowanie ORM zwiększa wydajność i skalowalność oraz minimalizuje problemy niekompatybilności architektur (dobry wzorzec obsługuje większość obecnie stosowanych systemów baz danych).

Powstało sporo aplikacji, które dostarczając zestaw klas i bibliotek automatyzują proces mapowania i odciążają programistę. Po podaniu im listy tabel w bazie, same wygenerują odpowiednie obiekty i zależności między nimi. Wracając do poprzedniego przykładu – pytając taki obiekt o numer telefonu, w tle nastąpi utworzenie

zapytania SQL, wysłanie go do bazy, przetworzenie wyniku, a następnie konwersja do postaci zgodnej z obiektem.

Z punktu widzenia programisty, nie powinno go zajmować jak to wszystko się odbywa – powinien tylko wiedzieć, że zapisanie danej do obiektu, skutkuje zapisaniem jej w bazie.

W praktyce nie jest to takie proste. Nie jest możliwe stuprocentowe odwzorowanie wszystkich możliwych kombinacji zapytań. Wszystkie implementacje ORM mają margines błędu, który użytkownik może zniwelować tylko poprzez bezpośrednie wysłanie zapytania SQL do bazy. Dodatkowo, cały proces mapowania i unifikacji zapytań pociąga za sobą spadek wydajności (zapytania SQL z poziomu ORM są tworzone z myślą o jak największej elastyczności i uniwersalności, więc niekoniecznie muszą być najszybsze dla konkretnego przypadku).

3.2.3 Zastosowanie w aplikacjach internetowych

Koncepcja ORM jest wykorzystywana we wszystkich nowoczesnych środowiskach webowych, ponieważ zdejmuje z programisty konieczność dbania o spójność danych w bazie, utrzymywania i rozłączania połączeń itd. Nie bez znaczenia jest też wsparcie dla różnych architektur bazodanowych. Często dzieje się tak, że po przeniesieniu aplikacji na nowy serwer okazuje się, że zmienił się silnik bazy danych. Wtedy wystarczy tylko jedna zmiana w pliku konfiguracyjnym, a całość powinna nadal działać.

Drugą, znaczącą zaletą jest to, że ORM wymusza spójność kodu (przynajmniej w kwestii obsługi bazy danych) – oznacza to, że przy projektach pisanych przez wielu programistów, jest znacznie mniej nieścisłości – wszyscy trzymają się jednej konwencji.

3.3 Metodologia REST

REST (ang. REpresentional State Transfer, reprezentacja stanu transferu) jest to styl w architekturze programowania przeznaczony dla multimedialnych systemów rozproszonych takich jak World Wide Web. Określenia „Representional state transfer" i „REST" zostały pierwszy raz zaprezentowane w 2000 roku w rozprawie doktorskiej, której autorem był Roy Fielding[4] – jeden z głównych twórców specyfikacji protokołu HTTP. Zwroty te, szybko rozpowszechniły się wśród sieciowej społeczności.

Termin REST ściśle odnosi się do założeń sieciowych, które określają jak zasoby

[4] *Representational State Transfer (REST)*, zob. http://www.ics.uci.edu/ fielding/pubs/dissertation/rest_arch_style.htm

są definiowane i adresowane. Określenie to, jest też często używane w luźnym sensie, do opisania jakiegokolwiek prostego interfejsu, który przesyła informacje określone w danej domenie (patrz rozdz. 3.4, str. 27) poprzez protokół HTTP wraz z dodatkową warstwą opisu wiadomości, taką jak SOAP (ang. Simple Object Access Protocol) czy też kontrolą sesji wykorzystującą „ciasteczka". Zastosowanie protokołu HTTP może uzupełniać projekt REST, lub zburzyć jego filozofię. Jest możliwe zaprojektowanie dużej, skomplikowanej aplikacji w metodologii REST, jednocześnie nie używając HTTP i nie łączac jej z WWW. Jednakże, jest także możliwe zbudowanie prostej aplikacji XML+HTTP, która zupełnie nie spełnia założeń REST. Te różnice w używaniu terminu „REST", często prowadzą do nieporozumień w dyskucjach technicznych.

Systemy, które spełniają założenia REST, często bywają określane jako „RESTful".

3.3.1 Założenia projektowe

Najważniejszym założeniem w metodologii REST jest istnienie zasobów (źródeł określonej informacji), z których każdy może być zaadresowany przy użyciu globalnego odnośnika URI (ang. Uniform Resource Identifier, jednoznaczny identyfikator zasobu). Aby modyfikować te zasoby, komponenty sieci (klienci i serwery) komunikują się między sobą przez zestandarazywany interfejs (np. HTTP) i wymieniają między sobą reprezentacje tych zasobów (dokumenty zawierające określone informacje). Dla przykładu zasób „okrąg" może przyjmować i zwracać reprezentację w postaci współrzędnych środka i promienia, skonwertowaną do formatu SVG, ale może też to być plik tekstowy zawierający trzy współrzędne po średniku, które leżąc na tym okręgu definiują go jednoznacznie.

Zwolennicy techniki REST utrzymują, że skalowalność i możliwość rozwoju sieci są bezpośrednim wynikiem kilku kluczowych założeń:

- stany i funkcjonalności aplikacji są podzielone między zasoby,
- każdy zasób jest unikalnie adresowalny z użyciem uniwersalnej składni do tworzenia odnośników,
- wszystkie zasoby dzielą jednolity interfejs do transferu stanów pomiędzy klientem a zasobem, zawierający:
  - ograniczony zestaw dobrze opisanych rozkazów,
  - ograniczony zestaw typów danych, może jednak być wysłany w ramach „kodu na żądanie",
- protokół, którego cechami są:

- klient-serwer,
- bezstanowość,
- możliwość zapisywania stanu,
- warstwowość.

Tak sformułowane założenia stanowią bazę do projektu zgodnego z metodologią REST.

3.3.2 Zastosowanie w aplikacjach internetowych

World Wide Web jest kluczowym przykładem projektu typu „RESTful". Większość założeń systemu WWW, pokrywa się z założeniami REST. Sieć opiera się na protokole HTTP, prezentacji treści w postaci dokumentów HTML oraz różnych innych technikach internetowych, jak np. system DNS (Domain Name System, system nazw domenowych).

Plik HTML może zawierać kod języka JavaScript i applety Java, aby wspierać kod na żądanie, oraz posiada wbudowaną obsługę hiperlinków.

Protokół HTTP jest jednolitym interfejsem dostępu do zasobów, który zawiera adres URI, metody, kody stanu, nagłówki i zawartość opisaną typem MIME (ang. Multipurpose Internet Mail Extensions, rozszerzenia poczty internetowej stosowane do różnych celów).

Najważniejszymi metodami protokołu HTTP są POST, GET, PUT i DELETE (patrz rozdz. 2.1, str. 13). Są one często porównywane do CREATE, READ, UPDATE i DELETE (CRUD, patrz rozdz. 3.5, str. 28), które to pojęcia mają szczególne zastosowanie w bazach danych.

HTTP	CRUD
POST	Create, Update, Delete
GET	Read
PUT	Create, Update
DELETE	Delete

Tablica 3.1: Porównanie zapytań HTTP z koncepcją CRUD

Protokół HTTP spełnia jedno z założeń metodologii REST – jest bezstanowy. Każda wiadomość zawiera wszystkie informacje niezbędne do identyfikacji zapytania. W efekcie czego, ani klient, ani serwer nie muszą przekazywać żadnych stanów informacyjnych pomiędzy wiadomościami. Każdy stan, w którym jest serwer, może być zamodelowany jako zasób.

Ta bezstanowość może być naruszona gdy używamy ciasteczek do podtrzymywania sesji. Używanie ciasteczek często wiąże się z naruszeniami polityki prywatności

i problemami bezpieczeństwa. Dodatkowo, pojawia się wiele nieścisłości i błędów jeśli do problemów z ciasteczkami dołożymy jeszcze obecność przycisku „wstecz" w przeglądarce.

Nie bez znaczenia jest też fakt, że hipertekst HTML pozwala tylko na użycie zapytań typu „GET". Dodatkowo dzięki użyciu formularzy jest jeszcze dostępny „POST". Pozostałe metody HTTP nie są wspierane ani przez format HTML 4.01 ani XHTML 1.0.[5]

3.4 Język DSL

Język DSL (z ang. Domain Specific Language, język określonego przeznaczenia) jest zaprojektowany do wykonywania tylko jednego, dokładnie określonego rodzaju zadań. Stanowi przeciwieństwo do języków ogólnego przeznaczenia takich jak Java czy C, których założeniem jest uniwersalność i możliwość zastosowania w jak największej liczbie przypadków. Przykładem realizacji DSLa jest np. język SQL – zaprojektowany tylko i wyłącznie do obsługi baz danych. Termin DSL nie jest terminem nowym, jednak dopiero ostatnio zyskał sobie sporą popularność, szczególnie w projektach webowych. Powodem tego jest znaczny wzrost mocy obliczeniowej komputerów oraz elastyczność nowych języków generalnego przeznaczenia, która umożliwia pisanie własnych aplikacji DSL.

3.4.1 Przykłady zastosowań

Można wykazać conajmniej kilka przykładów zastosowań dla języków typu DSL:

1. wykorzystywanie samodzielnych aplikacji wywoływanych przez bezpośrednią akcję użytkownika (najczęściej z poziomu linii poleceń lub pliku Makefile), np. zestaw narzędzi GraphViz,

2. Języki DSL, które są zaimplementowane przy użyciu systemów makr, a następnie są konwertowane lub rozszerzane do trybu zgodności z nadrzędnym językiem programowania ogólnego przeznacznia, podczas fazy kompilacji lub interpretowania

3. Realizacje w całości napisane w języku programowania ogólnego przeznaczenia (takim jak C lub Perl), mające na celu wykonywanie określonych operacji i zwracanie danych, które język nadrzędny jest w stanie zinterpretować i przetworzyć. Generalnie, można to rozumieć jako język programowania szczególnego przeznaczenia napisany w języku programowania ogólnego przeznaczenia.

[5] Więcej informacji na http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626/#_http_binding_default_rule_method

4. samodzielne funkcjonalnie części aplikacji – dobrym przykładem są tu różnego rodzaju systemy makr w arkuszach kalkulacyjnych, wykorzystywane do uruchamiania specyficznych funkcji napisanych przez użytkownika, lub tworzone dynamicznie przez samą aplikację.

Wiele języków DSL może być używane w kilku powyższych przypadkach naraz.

3.4.2 Zastosowanie w aplikacjach internetowych

W aplikacjach internetowych systemy DSL znalazły szerokie zastosowanie przy zadaniach często powtarzających się, takich jak na przykład tworzenie modeli, testy czy obsługa baz danych – głównie dzięki językowi SQL. Używając systemów DSL, można zautomatyzować procesy takie jak:

- wysyłanie aplikacji na serwer,
- synchronizacja z systemem kontroli wersji np. SVN (z ang. SubVersioN),
- testowanie aplikacji oraz jej poszczególnych komponentów,
- wiele różnych akcji związanych z przejściem z trybu rozwojowego na produkcyjny, jak np.:
  - kompresja i obfuskacja (zaciemnienie) plików JavaScript,
  - kompresja CSS,
  - usuwanie informacji o składowej gamma z plików PNG (ang. Portable Network Graphics, przenośna grafika sieciowa),
  - zmiany nazw plików, aby zostały ponownie pobrane (dotyczy to w szczególności serwisów, gdzie wymuszany jest nieskończony czas składowania plików przez przeglądarkę),
  - czyszczenie różnych pozostałości procesu rozwijania kodu (stare, skompilowane pliki szablonów, logi, itd.).

3.5 Metody CRUD

Akronim CRUD (Create, read, update and delete – utwórz, odczytaj, popraw i skasuj) oznacza cztery podstawowe funkcje dotyczące operacji na danych. Czasami w skrócie tym, zamiast „read” używa się „restore” a zamiast „delete” – „destroy”. Termin ten często jest stosowany przy opisywaniu operacji na bazach danych lub przy opisie interfejsów.

3.5.1 CRUD w relacyjnych bazach danych

Skrót CRUD jest często odnoszony do relacyjnych baz danych, ponieważ te cztery operacje determinują zupełność operacji na takiej bazy danych. Każda litera akronimu, może zostać przedstawiona jako odpowiednia funkcja SQL. Pomimo iż pojęcie

Operacja	SQL
Create	INSERT
Read (Retrieve)	SELECT
Update	UPDATE
Delete (Destroy)	DELETE

Tablica 3.2: Mapa akronimu CRUD na zapytania SQL

CRUD głównie odnosi się do relacyjnych baz danych, często jest też stosowany w odniesieniu do baz danych obiektowych (lub w formacie XML), plików tekstowych, nośników informacji itd.

3.5.2 Zastosowanie w aplikacjach internetowych

W aplikacjach internetowych, termin CRUD odnosi się wyłącznie do interfejsu. Dla przykładu, jeśli tworzona jest książka teleadresowa, podstawową jednostką danych jest informacja o kontakcie. Jako niezbędne minimum, aplikacja musi:

- utworzyć lub dodać nowy kontakt (Create),
- wyświetlić istniejące wpisy (Read),
- zmieniać niektóre informacje (Update),
- usuwać niepotrzebne kontakty (Delete).

Bez którejkolwiek z powyższych operacji, aplikacja nie może być uznana za kompletną. Zasada ta, jest też podwaliną każdego dobrego zintegrowanego środowiska programistycznego, gdyż zautomatyzowanie jej znacznie skraca czas tworzenia programu.

3.6 Reguła DRY

Reguła DRY (z ang. Don't Repeat Yourself, nie powtarzaj się) jest filozofią, której celem nadrzędnym jest zredukowanie duplikatów każdego rodzaju. Idea DRY zakłada, że ta sama informacja nie powinna być powielana w ramach jednego projektu, gdyż taka duplikacja znacznie utrudnia zmiany, zmniejsza czytelność i prowadzi do niespójności. Don't Repeat Yourself jest głównym przykazaniem książki *The*

Pragmatic Programmer[4], której autorami są Andy Hunt i Dave Thomas. W publikacji tej, zalecają, aby używać jej w jak najszerszym zakresie, włączając „schematy baz danych, plany testów, budowany system, a nawet dokumentację”[6]. Kiedy reguła DRY zostanie pomyślnie wdrożona w system, jakakolwiek zmiana jakiegoś elementu w kodzie nie wpłynie w żaden sposób na elementy nie będące z nim logicznie połączone. Dodatkowo, wszystkie elementy logicznie połączone ze zmienionym, zmienią się w sposób przewidywalny i wciąż będą ze sobą zsynchronizowane.

3.6.1 Ograniczenia idei DRY

Reguła DRY nie zawsze się sprawdza i w niektórych przypadkach, lepiej z niej zrezygnować.

- W małych projektach, albo prostych skryptach, wysiłek jaki trzeba włożyć, aby trzymać się reguły DRY, byłby znacznie większy niż proste napisanie dwóch kopii tej samej danej w różnych miejscach.
- Wymuszanie filozofii w projektach, w których najważniejszą rolę w tworzeniu treści pełni społeczność (np. wikipedia), mogłoby znacznie ostudzić zapał ludzi do tworzenia nowych wpisów.
- Zarządzanie projektami i kontrolą wersji pozwala (a nawet) zaleca tworzenie kilku kopii. Dla przykładu, bardzo dobrym nawykiem jest tworzenie trzech środowisk dla jednej aplikacji – produkcyjnego, testowego i rozwojowego. Dzięki temu mamy pewność, że rozwojowy albo testowy kod, nie będzie miał wpływu na stabilne wersje produkcyjne,
- Dokumentacja dla użytkowników końcowych (od komentarzy w kodzie, do drukowanych instrukcji) zawiera typowe informacje opisujące pewnie funkcjonalności kodu przeznaczone dla ludzi, którzy nie potrzebują lub nie mają czasu analizować całego listingu programu. Niestety zasada DRY zakłada, że jeśli taka dokumentacja nie stanowi jakiejś wartości dla samego kodu (a nie dla użytkowników), to powinna być generowana a nie pisana,
- Generatory kodu źródłowego - niepowtarzalność będzie skuteczna dla generatora samego w sobie, ale nie musi już być konieczna dla samego wygenerowanego źródła, które jest używane automatycznie i nikt nie będzie do niego zaglądał ani go zmieniał.

[6] *Orthogonality and the DRY Principle*, zob. http://www.artima.com/intv/dry.html

3.6.2 Zastosowanie w aplikacjach internetowych

Reguła DRY świetnie sprawdza się we wszystkich projektach programistycznych, ale największe uznanie znalazła właśnie wśród programistów aplikacji internetowych. Specyfika Internetu jest inna niż środowiska izolowanego – wszystkie zmiany zachodzą dużo szybciej. Czas życia przeciętnego projektu internetowego to kilka lat i w tym czasie musi on podlegać ciągłemu rozwojowi i przemianom. Gdyby nie zasada DRY, zmiany zachodziły by dużo wolniej, byłyby dużo trudniejsze do przeprowadzenia i pochłaniałyby więcej środków. Filozofia DRY (oczywiście w pełni respektowana) wymusza na programistach WWW porządek w kodzie, dzięki czemu taka aplikacja jest łatwa do przystosowania do zmiennych warunków rynkowych.

3.7 Zasada Convention over Configuration

Convention over Configuration (konwencje ponad konfiguracje) jest jedną z metodologii projektowania oprogramowania, która dąży do osiągnięcia minimalizacji liczby decyzji jakie muszą podjąć programiści zyskując dzięki temu prostotę bez utraty elastyczności.

Sformułowanie samo w sobie oznacza, że programista musi rozpatrywać tylko niekonwencjonalne aspekty aplikacji. Dla przykładu, klasie *Sprzedaż* przypisana jest tabela domyślnie nazwana „sprzedaż". Jeżeli teraz nie zostanie naruszona konwencja (czyli np. tabela nie zostanie nazwana „sprzedaż_produktów"), to programista nie musi nigdzie określać nazwy tabeli i potem odwoływać się do niej w innych miejscach.

Im więcej dobrych konwencji w kodzie, tym na wyższym poziomie abstrakcji może się skupić programista, co oznacza, że mocniej zajmuje się aspektami logicznymi zagadnienia, a nie technicznymi.

3.7.1 Geneza

Tradycyjnie, systemy internetowe potrzebowały ogromnej ilości różnych plików konfiguracyjnych, z których każdy opisywał inne ustawienia. Pliki te zawierały specyficzne informacje dla każdego projektu, zawierające wszystko – od mapowania adresów URL, poprzez klasy, aż po konfiguracje tabel w bazie danych. Przy większej złożoności aplikacji rozmiar i liczba tych plików powiększała się.

Dobrym przykładem reguły Convention over Configuration jest system ORM (zob. rozdz. 3.2, str. 22) Hibernate. Mapuje on tabele w bazach danych na świat obiektowy, wykorzystując w tym celu pliki XML. Większość informacji jest konwertowanych w myśl z góry ustalonych zasad. Nazwy klas są mapowane na identycznie nazwane tabele w bazie danych, a ich wartości – na kolumny. Jest to konwencja ustalona przez autorów Hibernate i narzucona użytkownikom, tj. programistom, którzy

używają tego projektu. Jeśli któryś programista potrzebuje, aby nazwy się różniły, zawsze może sobie ustalić wszystkie wyjątki w odpowiednim pliku konfiguracyjnym.

3.7.2 Zastosowanie w aplikacjach internetowych

Convention over Configuration jest koncepcją stosunkowo młodą, ale już zdobyła sobie rzesze zwolenników. Wynika to przede wszystkim z tego, że upraszcza wiele aspektów i sprawia, że z każdym nowym projektem twórcy nie muszą wciąż skupiać się na tych samych problemach. Poza tym, przy większych aplikacjach, nad którymi pracują dziesiątki programistów konwencje sprawiają, że każdy pisze w ten sam sposób i nie ma problemów w rodzaju – „Co ta metoda robi?". Dodatkowo złamanie konwencji wiąże się ze sporą nadprodukcją kodu, co niejako „samo z siebie" wymusza w programistach dyscyplinę.

Rozdział 4

Realizacja serwisu WWW

Temat pracy z góry narzucał cel projektu. Jest nim strona Kierunku Elektronika i Telekomunikacja[1]. Motywem takiego wyboru była przestarzałość serwisu i niespójność kodu. Przez kilka lat od momentu stworzenia, aplikację rozwijało wiele różnych osób, skutkiem czego kod zatracił swoją początkową „myśl przewodnią" oraz znacznie stracił na czytelności. Głównym powodem było SCP (ang. Speaking Code Principle) kodu stworzonego w Ruby on Rails. Dzięki filozofii DRY (zob. rozdz. 3.6, str. 29) oraz Convention over Configuration (zob. rozdz. 3.7, str. 31) każdy kawałek kodu ma ściśle określone miejsce w aplikacji. Dodatkowo sama struktura plików projektu Rails jest klarowna i przejrzysta. Zawsze w tym samym miejscu przechowywane są modele, widoki, biblioteki, kontrolery itd. Dzięki temu przekazanie serwisu w ręce nowego programisty, powinno odbyć się bezproblemowo. Dodatkowo czytelność kodu Ruby i łatwość tworzenia DSL (zob. rozdz. 3.4, str. 27) sprawia, że taki kod od razu przekazuje intencje autora. I stąd właśnie bierze się jego samoopisywalność.

4.1 Opis funkcjonalny aplikacji

Głównym założeniem serwisu projektu Kierunku Elektroniki i Telekomunikacji (zwanego dalej EiT) jest stworzenie portalu zawierającego wszelkie niezbędne studentowi informacje. Na funkcjonalności te składają się przede wszystkim:

- system aktualności i najświeższych informacji dla studentów,
- dziekanatowe informacje o:
  - planach zajęć,
  - praktykach,

[1] *Strona domowa EiT*, zob. http://eit.agh.edu.pl/

  - przedmiotach obieralnych,
  - zajęciach w języku angielskim,
  - międzynarodowym programie edukacyjnym,
  - wydarzeniach obowiązkowych w kalendarzu studenta (ferie, sesje itd.).
- system zarządzania materiałami i ocenami,
- informacje o prowadzących,
- wszelkie dane dotyczące studiów podyplomowych.

Dodatkowo istotnym elementem jest warstwa administracyjna. Grupa ludzi opiekujących się stroną powinna mieć specjalne uprawnienia do dodawania informacji, usuwania niepotrzebnych treści, zarządzania danymi studentów (z poziomu systemu) itd. Dodatkowo, wszyscy prowadzący powinni mieć możliwość dodawania aktualności, materiałów i ocen.

Wymienione powyżej funkcjonalności uznane zostały za najważniejsze i umieszczone w serwisie. Dotychczasowa strona EiT miała jeszcze dodatkowo forum i katalog linków. Uznano je jednak za zbyteczne. W obu przypadkach głównym powodem było znikome zainteresowanie studentów. Forum, z kolei, (zwane Hydeparkiem) przestało pełnić swoją pierwotną funkcję. W dzisiejszych czasach istnieją setki serwisów pozwalające założyć wìasne, darmowe forum jednym kliknięciem myszki. I właśnie na takie biuletyny przenieśli się studenci, tworząc fora swoich lat bądź grup dziekanatowych. Nie bez znaczenia jest też fakt, że do Hydeparku dostęp mieli prowadzący co z miejsca ucinało niektóre rodzaje konwersacji.

Kolejną niesłychanie istotną cechą serwisu jest jego bezpieczeństwo. W przypadku serwisu EiT sprawa jest o tyle poważna, że jest to aplikacja przeznaczona dla osób z dużą wiedzą techniczną, którzy doskonale orientują w świecie nowoczesnych technik. Najmniejsze niedopatrzenie może się zakończyć kompromitacją witryny. Autor niniejszej pracy od wielu lat zajmuje się tematyką szeroko pojętego bezpieczeństwa. I to zarówno od strony aplikacyjnej (pisanie bezpiecznego kodu) jak i serwerowej (zabezpieczanie maszyn przed intruzami). W związku z tym, od początku każdy element strony był szczegółowo analizowany pod tym kątem. Nie bez znaczenia jest też fakt, że system Ruby on Rails przychodzi z wieloma rozwiązaniami ochronnymi w standardzie. Dla przykładu, jeśli nastąpi jakiś błąd w działającej aplikacji, to w odróżnieniu od PHP nie pojawi się komunikat dokładnie informujący, który wiersz w którym pliku go wywołał. Zamiast tego, wyświetli się informacja dla użytkownika, a sam błąd zostanie zapisany do logów administratora. Dzięki temu, potencjalny atakujący nie otrzymuje żadnej informacji o serwisie, poza tym, że działając tak i tak można uszkodzić aplikację – co nie jest wartościową informacją samo w sobie.

Sama szata graficzna pozostała niezmieniona, natomiast musiał zmienić się jej kod. Wynika to z zupełnie nowej filozofii tworzenia widoków. Poza tym, stary kod HTML nie był w ogóle semantyczny, co znacznie utrudniało pracę robotom wyszukiwarek i skutkowało niższymi pozycjami w rankingach. Cały HTML został przepisany na nowo (już jako XHTML) i stworzone zostały nowe arkusze stylów. Zmieniła się też organizacja menu – kliknięcie w pozycję nie odświeża strony, tylko od razu rozwija listę podmenu. Zmniejsza to znacznie liczbę zapytań (wcześniej użytkownik musiał otworzyć nową stronę, tylko po to, żeby zobaczyć rozwinięte menu i przejść do kolejnej strony) oraz skraca czas interakcji ze stroną. A wszystko to, dzięki zastosowaniu języka JavaScript.

Jak już nadmienione zostało wcześniej – Ruby nie jest zbyt szybkim językiem. Dlatego wszelkie działania mające na celu optymalizację kodu są wysoce pożądane. Bardzo istotne jest zwracanie uwagi na takie szczegóły, jak wyciąganie deklaracji zmiennych poza pętle, optymalizacje warunków (ważna jest tu znajomość praw de-Morgana), przypisywanie obiektów do zmiennych itd. Zastosowany też został dużo wydajniejszy serwer WWW aniżeli dostarczany w standarcie WebRick — Mongrel[2]

4.2 Środowisko narzędziowe

W kolejnych podrozdziałach opisane zostały użyte narzędzia i środki do implementacji aplikacji.

4.2.1 Ruby i pochodne

Ruby jest językiem międzyplatformowym, co oznacza, że jego interpreter jest dostępny pod prawie każdym systemem operacyjnym. Niestety implementacja pod systemem Windows pozostawia wiele do życzenia. Bierze się to stąd, że Ruby został pierwotnie stworzony na systemach Uniksowych a następnie przeniesiony na platformę Windows. Przy takiej skali skomplikowania języka, brak optymalizacji pod konkretny system daje się odczuć. Różnice w czasach wykonania aplikacji potrafiły przekroczyć 100%! Do realizacji niniejszej pracy korzystano z Linuksa a konkretniej z dystrybucji Slackware Linux. Natomiast do pracy z Ruby zaleca się jakąkolwiek dystrybucje UNIXopodobną (włączając to MAC OS X).

System operacyjny Slackware 12.0 ma interpreter Ruby w standardzie (przy maksymalnej instalacji). W razie jego braku, należy pobrać odpowiednią paczkę – Ruby 1.8.6[3]. Dodatkowo potrzebny jest pakiet RubyGems 0.9.2[4].

[2] *Strona domowa projektu mongrel*, zob. http://mongrel.rubyforge.org/

[3] Do pobrania m.in. na http://linuxpackages.telecoms.bg/Slackware-11.0/ken/ruby-1.8.6-i486-1kjz.tgz

[4] Do pobrania m.in. na http://linuxpackages.telecoms.bg/Slackware-11.0/ken/

Po instalacji użytkownik otrzymuje w pełni funkcjonujący interpreter języka Ruby, w którym jednak brakuje jeszcze kilku bibliotek. Do ich instalacji należy posłużyć się narzędziem RubyGems.

Moduł Gem to spakowana aplikacja lub biblioteka języka Ruby. Zarządzanie gemami odbywa się za pośrednictwem polecenia *gem*. Daje ono możliwość instalacji, usuwania i zadawania pytań o pakiety. Narzędzie jest podobne do APT (ang. Advanced Packaging Tool, zaawansowane narzędzie paczkujące), systemu zarządzania pakietami używanym przez system Debian GNU/Linux.

Repozytorium gemów jest otwarte dla każdego, to znaczy że każdy może pobierać a także publikować swoje gemy.

W niniejszej pracy wykorzystano dwa gemy:

- Ruby on Rails – środowisko programistyczne w wersji 2.0.2,
- Mongrel – serwer WWW w wersji 1.1.3.

```
# gem install rails --include-dependencies
# gem install mongrel --include-dependencies
```

Listing 4.1: Lista poleceń niezbędnych do zainstalowania wymaganych gemów

Aby zainstalować wymagane gemy, z poziomu powłoki należy wykonać następujące polecenia (zob. list. 4.1), które zainstalują pakiety wraz ze wszystkimi zależnościami. Po ich wykonaniu, wymienione aplikacje są gotowe do użycia.

4.2.2 Dodatkowe pakiety

Poza środowiskiem Ruby on Rails aplikacja EiT korzysta jeszcze z dwóch zewnętrznych aplikacji – serwera MySQL 5.0.37[5] oraz pakietu iconv[6]. Pierwsza z nich, to relacyjna baza danych, druga – pakiet konwertujący różne strony kodowe.

Serwer MySQL powinien być standardowo w dystrybucji Slackware. Jeśli go nie ma, należy ściągnąć odpowiedni pakiet[7]. Iconv stanowi część pakietu GLIBC (ang. GNU C Library, biblioteka GNU C)[8] i jego status w systemie paczek jest *required*, co oznacza, że na pewno jest zainstalowany. W razie jakichkolwiek problemów, należy ściągnąć najświeższą paczkę[9].

`rubygems-0.9.2-noarch-1kjz.tgz`

[5] *Strona domowa projektu MySQL*, zob. http://www.mysql.com/

[6] *Informacje o bibliotece iconv*, zob. http://www.gnu.org/software/libiconv/

[7] MySQL można pobrać z ftp://ftp.slackware.com/pub/slackware/slackware-12.1/slackware/ap/mysql-5.0.51b-i486-1.tgz

[8] *Biblioteka GLIBC*, zob. http://www.gnu.org/software/libc/

[9] GLIBC można pobrać z ftp://ftp.slackware.com/pub/slackware/slackware-12.1/slackware/l/glibc-i18n-2.7-noarch-10.tgz

Aby baza danych mogła działać poprawnie należy stworzyć katalog przechowywujący dane i tabele systemowe niezbędne do funkcjonowania bazy. Dodatkowo, dobrze jest ustawić, aby proces obsługujący bazę danych uruchamiał się przy każdym starcie systemu. Wszystko to osiągniemy, wykonując polecenia z listingu 4.2.

```
# chmod +x /etc/rc.d/rc.mysqld
# /etc/rc.d/rc.mysqld start
# su - mysql
$ mysql_install_db
```

Listing 4.2: Lista poleceń niezbędnych do zainicjalizowania MySQL

Moduł iconv nie wymaga żadnych dodatkowych przygotowań.

4.2.3 Aplikacje wspomagające tworzenie projektu

Po przygotowaniu wszystkiego co niezbędne od strony serwera, trzeba jeszcze przygotować narzędzia wspomagające pisanie kodu – tj. środowisko programistyczne (ang. IDE, Integrated Development Environment), aplikacje wspomagające pracę z bazą danych oraz przeglądarkę wraz z zestawem dodatków wspomagających pracę z gotowym kodem.

Środowisko programistyczne

Najlepszym, zdaniem autora, edytorem wspomagającym tworzenie aplikacji Ruby on Rails jest TextMate[10]. Niestety jest on dostępny tylko dla systemów Mac OS X. Alternatywnie istnieje zintegrowane środowisko programistyczne NetBeans[11]. Jest ono stworzone w języku Java i istnieją jego wersje dla systemów Linux, Windows, Mac OS i Solaris. Ta praca powstała przy użyciu NetBeans w wersji 6.0beta1, niemniej obecnie jest już dostępna wersja 6.1[12].

Do poprawnego działania edytora NetBeans potrzebny jest jeszcze pakiet JDK (ang. Java Development Kit, Zestaw narzędzi deweloperskich dla języka Java)[13]. Należy pamiętać, że dla dystrybucji Slackware należy pobrać wersję binarną z instalatorem, a nie paczkę RPM (ang. RPM Package Manager, zarządca pakietów RPM). Po ściągnięciu z sieci, należy ją uruchomić i odpowiedzieć na kilka pytań (m.in. o ścieżkę docelową). Analogicznie instaluje się pakiet NetBeans.

Jako dodatek użyteczny przy rozwijaniu kodu Rails, autor zaleca zainstalowanie wtyczki do NetBeans o nazwie *Extra Ruby Color Themes* (Zakładka *To-*

[10] *Macromates.com*, zob. http://macromates.com/

[11] *Środowisko NetBeans*, zob. http://www.netbeans.org/

[12] Do pobrania z http://download.netbeans.org/netbeans/6.1/final/bundles/netbeans-6.1-ruby-linux.sh

[13] Do pobrania z http://java.sun.com/javase/downloads/?intcmp=1281

ols→Plugins), a następnie włączenie schematu kolorów *Ruby Dark Pastels* (Zakładka *Options→Fonts & Colors*). Jest to schemat kolorów wzięty z TextMate, dużo czytelniejszy od domyślnego i mniej męczący wzrok.

Aplikacje wspomagające pracę z bazą danych

W celu uniknięcia wykonywania wszystkich niezbędnych działań na bazie z domyślnego interfejsu tekstowego, zalecane jest zainstalowanie graficznych narzędzi ułatwiających obsługę bazy. Te polecane narzędzia to:

- MySQL Query Browser[14],
- MySQL Administrator[15],
- DBDesigner[16].

Pakiet MySQL Query Browser ułatwia wykonywanie zapytań i przeglądanie danych, natomiast MySQL Administrator – odpowiada za zarządzanie użytkownikami w bazie MySQL. DBDesigner jest narzędziem do graficznego projektowania bazy danych.

Przeglądarki internetowe i dodatki

Z powodu niekompatybilności silników renderujących przeglądarek (zob. rozdz. 2.7, str. 18), podczas pisania pracy, autor musiał korzystać aż z czterech przeglądarek internetowych, aby przetestować stronę pod najpopularniejszymi silnikami renderującymi:

- *Trident* – Microsoft Internet Explorer 6.0 (pod Linuksem dostępny, dzięki pakietowi *ies4linux*[17]),
- *Gecko* – Mozilla Firefox 2.0.0.14[18],
- *Presto* – Opera 9.50[19],
- *WebKit* – Konqueror 3.5.8[20].

W czasie projektowania i rozwijania aplikacji, autor korzystał z przeglądarki Mozilla Firefox, która obecnie posiada najlepszy silnik renderujący. Dodatkowo można do niej zainstalować kilka, ułatwiających pracę dodatków:

[14] Do pobrania z witryny http://www.mysql.com/products/tools/query-browser/
[15] Do pobrania z witryny http://www.mysql.com/products/tools/administrator/
[16] Do pobrania z witryny http://www.fabforce.net/dbdesigner4/
[17] *Strona projektu ies4Linux*, zob. http://www.tatanka.com.br/
[18] *Mozilla Firefox*, zob. http://www.getfirefox.com/
[19] *Opera Software*, zob. http://www.opera.com
[20] *Konqueror web browser*, zob. http://www.konqueror.org

- *Web Developer*[21] – pasek narzędzi, który posiada wiele opcji i udogodnień (np. zarządzanie ciasteczkami, zaznaczanie wybranych elementów dokumentu, podgląd bieżącego źródła, walidację HTML/CSS).

- *FireBug*[22] – narzędzie pozwalające na przeglądanie drzewa dokumentu HTML, podglądu i edycji jego poszczególnych elementów, przeglądanie i dynamiczną edycję arkuszy styli CSS, przeglądanie skryptów JavaScript oraz możliwości wykonywania instrukcji tego języka i obserwacji ich rezultatów, debugowanie skryptów, przeglądanie elementów DOM, a także obserwowanie wywołań XMLHttpRequest (zob. rozdz. 2.4, str. 16).

- *ColorZilla*[23] – rozszerzenie bardzo użyteczne podczas projektowania wyglądu strony i tworzenia arkuszy CSS. Udostępnia ono narzędzie o nazwie pipeta (ang. color picker), która pozwala pobrać kolor dowolnego piksela w obrębie przeglądarki. Dzięki temu, nie trzeba za każdym razem zaglądać do styli, podczas tworzenia nowego arkusza CSS na bazie starego projektu.

- *HTML Validator*[24] – dodatek, który rozszerza podgląd źródła, dokumentu o numerację wierszy i walidator dokumentu HTML (na bazie silnika TIDY[25]). Dzięki temu, od razu widać, gdzie popełniono błąd w strukturze dokumentu (niedomknięty tag, brak cudzysłowu przy wartości, itd.).

- *LiveHTTPHeaders*[26] – narzędzie służące do podglądu nagłówków z protokołu HTTP wysyłanych i odbieranych podczas każdego zapytania. Świetnie się sprawdza w przypadku błędnych przekierowań, kiedy nie wiadomo, w którym momencie doszło do zapętlenia programu.

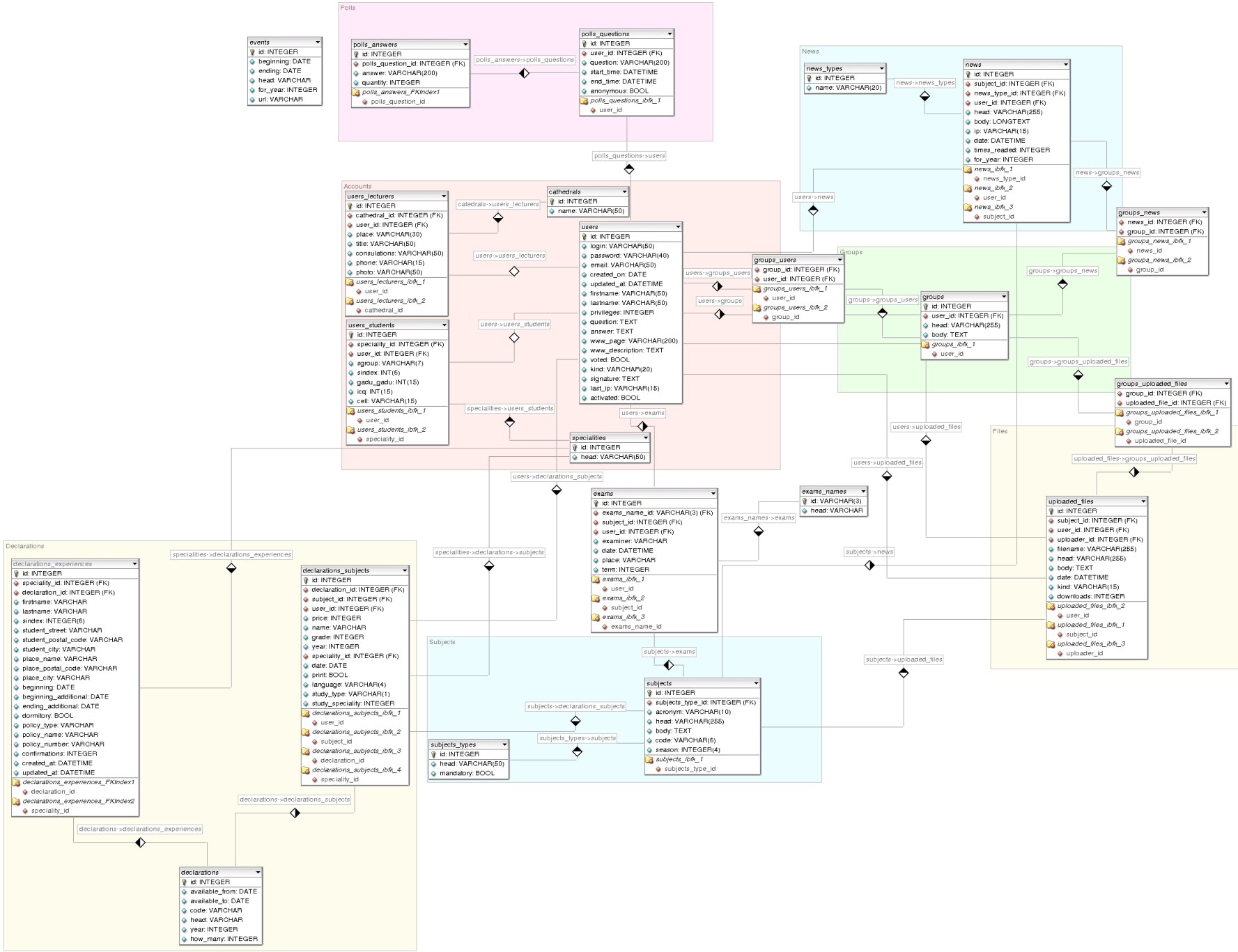
4.3 Projekt bazy danych

Zasada działania relacyjnej baz danych (jaką jest MySQL wykorzystywany w aplikacji), jak i technika odwzorowywania w niej rzeczywistych danych jest obszernym zagadnieniem. Czytelników zainteresowanych tym tematem odsyła się do znakomitego samouczka udostępnionego przez firmę Microsoft[27].

Projektując bazę programista powinien trzymać się konwencji nazewnictwa tabel oraz pól tabeli przyjętej przez Ruby on Rails. Zgodnie z konwencją, nazwy tabel

[21] Do pobrania z witryny http://chrispederick.com/work/webdeveloper/
[22] Do pobrania z witryny http://www.getfirebug.com/
[23] Do pobrania z witryny http://www.iosart.com/firefox/colorzilla/
[24] Do pobrania z witryny http://users.skynet.be/mgueury/mozilla/
[25] *Silnik sprawdzający poprawność HTML – Tidy*, zob. http://tidy.sourceforge.net/
[26] Do pobrania z witryny http://livehttpheaders.mozdev.org/
[27] *Projektowanie baz danych*, zob. http://office.microsoft.com/pl-pl/access/CH062526791045.aspx

powinny być w liczbie mnogiej w stosunku do nazwy modelu, który przechowuje informacje o danej tabeli. Klucz podstawowy tabeli powinien nosić nazwę *id*, klucze obce powinny być nazwane zgodnie ze schematem {nazwa obiektu w liczbie pojedynczej}_id. Dla potrzeb realizacji niniejszej pracy stworzono zgodnie z powyższymi wytycznymi nowy schemat bazy danych, przedstawiony na rysunku 4.1.



Rysunek 4.1: Schemat bazy danych EiT.

4.3.1 Opis tabel

W niniejszym rozdziale omówione zostały wszystkie tabele wraz ze znaczeniami poszczególnych kolumn (pól) oraz z rysunkiami poglądowymi. Rozdział ten stanowi dokumentację systemu dla osób mających zająć się jego utrzymaniem.

Tabela *users*

Tabela ta zawiera zbiorcze informacje o użytkownikach mających dostęp do systemu (zarówno studentach, jak i prowadzących).

Pole	Typ	Opis
id	int(11)	identyfikator użytkownika
login	varchar(50)	login użytkownika
password	varchar(40)	hasło użytkownika
email	varchar(50)	email użytkownika
created_on	date	data utworzenia konta
updated_at	datetime	data ostatniego zalogowania
firstname	varchar(50)	imię
lastname	varchar(50)	nazwisko
privileges	int(11)	przywileje w systemie
question	text	pytanie ratunkowe
answer	text	odpowiedź na pytanie ratunkowe
www_page	varchar(200)	adres strony WWW użytkownika
www_description	text	opis strony WWW
voted	tinyint(1)	czy użytkownik zagłosował już w ankiecie?
signature	text	sygnaturka użytkownika
last_ip	varchar(15)	ostatnie IP z którego logował się użytkownik
activated	tinyint(1)	czy konto jest aktywne?

Tablica 4.1: Tabela *users*

- Pole *login* może być puste. Wraz z początkiem każdego roku akademickiego, dziekanat dostarcza listę nowych studentów. Osoby te są dopisywane do bazy danych. Później, jeśli któraś z nich zapragnie założyć konto podaje login, hasło oraz numer indeksu, które z miejsca stają się aktywne i włączają użytkownika do systemu. Takie rozwiązanie zapobiega założeniu konta przez osobę niepowołaną.

- Hasło użytkownika (*password*) jest zahaszowane algorytmem SHA-1[28]. Zabezpiecza to hasła użytkowników przed odczytem w przypadku skompromitowania (nieuprawnionego odczytu) bazy danych.

- Struktura pola *privileges* to maska, w której każdy bit określa inne uprawnienia jak opisuje to rysunek 4.2. Wartość 1 – oznacza przyzwolenie, 0 – brak dostępu do danego zasobu.

- Pola *question* i *answer* nie są obowiązkowe, zabezpieczają jednak użytkownika przed utratą hasła. Po podaniu odpowiedzi na pytanie użytkownik jest z powrotem logowany do systemu, gdzie może zmienić swoje hasło.

X	O	D	A	G	M	K	P

- X_{128} – nieużywany
- O_{64} – dodawanie i edycja ocen
- D_{32} – odczyt deklaracji studenckich
- A_{16} – dodawanie i edycja aktualności
- G_8 – konfiguracja grup studentów
- M_4 – dodawanie i edycja materiałów
- K_2 – konfiguracja wydarzeń w kalendarzu studenckim
- P_1 – administracja ankietami

Rysunek 4.2: Struktura pola *privileges* w tabeli *users*

Tabela *cathedrals*

Tabela ta zawiera wszystkie katedry obsługiwane przez system. Tablica 4.2 pokazuje, że w tej tabeli znajdują się jedynie nazwy katedr.

Pole	Typ	Opis
id	int(11)	Identyfikator katedry
name	varchar(50)	Nazwa katedry
action	varchar(255)	Nazwa akcji
menu_name	varchar(255)	Nazwa menu

Tablica 4.2: Tabela *cathedrals*

Pole *action* określa nazwę akcji pod jaką będzie widziana dana katedra. Na chwilę obecną wykorzystywane w *LecturersController* (zob. rozdz. 4.6.6, str. 69). Pole *menu_name* określa pod jaką nazwą, wpis będzie widoczny w menu na stronie.

Tabela *specialities*

Tabela ta zawiera wszystkie moduły obsługiwane przez system według struktury w tablicy 4.3. Jeden rekord ($id = 1$) nie ma opisu i zarezerwowany jest dla studentów, którzy jeszcze nie dokonali wyboru.

Pole	Typ	Opis
id	int(11)	Identyfikator modułu
head	varchar(50)	Nazwa modułu

Tablica 4.3: Tabela *specialities*

Tabela *users_students*

Tabela zawiera uzupełniające informacje o studentach. Nie ma w niej rekordów odpowiadających prowadzącym. Jej strukturę obrazuje Tablica 4.4.

Pole	Typ	Opis
id	int(11)	Identyfikator studenta
user_id	int(11)	*users*→*users_students*
speciality_id	int(11)	*specialities*→*users_students*
sgroup	varchar(7)	Grupa dziekanatowa (rys. 4.3)
sindex	int(6)	Numer indeksu
gadu_gadu	int(15)	Numer Gadu-Gadu
icq	int(15)	Numer ICQ
cell	varchar(15)	Telefon kontaktowy

Tablica 4.4: Tabela *users_students*

K	R	R	S

- K – kod modułu (E dla Elektroniki, K dla Telekomunikacji)
- RR – dwie ostatnie cyfry roku rozpoczęcia studiów przez daną grupę
- S – kod specjalności

Rysunek 4.3: Struktura pola grupy

Na podstawie grupy dziekanatowej, obliczany jest rok studiów wg. wzoru 4.1 oraz zgodnie z rys. 4.3.

$$rok = \begin{cases} obecny\_rok - 2000 - RR & gdy\ miesiac \leq 9 \\ obecny\_rok - 2000 - RR + 1 & gdy\ miesiac > 9 \end{cases} \tag{4.1}$$

Tabela *users_lecturers*

Tabela zawiera informacje o prowadzących w.g. tablicy 4.5. Studenci nie są do niej dopisywani.

Pole	Typ	Opis
id	int(11)	Identyfikator prowadzącego
cathedral_id	int(11)	*cathedrals→users_lecturers*
user_id	int(11)	*users→users_lecturers*
place	varchar(30)	Pokój
title	varchar(50)	Tytuł naukowy
consultations	varchar(50)	Terminy konsultacji
phone	varchar(15)	Telefon kontaktowy
photo	varchar(50)	Zdjęcie

Tablica 4.5: Tabela *users_lecturers*

Pole *photo* zawiera tylko nazwę pliku ze zdjęciem. Model w aplikacji sam ustala ścieżkę.

Tabela *subjects_types*

Tabela zawiera kategorie (rodzaje) przedmiotów. Jest to istotne przy grupowaniu po kategoriach zgodnie z tablicą 4.6.

Pole	Typ	Opis
id	int(11)	Identyfikator rodzaju przedmiotu
head	varchar(50)	Nazwa rodzaju
mandatory	tinyint(1)	Czy obowiązkowy?

Tablica 4.6: Tabela *subjects_types*

Tabela *subjects*

Tabela zawiera informacje o prowadzonych przedmiotach. Przedmiot liczy się jako całość (nie ma podziału na wykłady, laboratoria itd.) w obrębie jednego semestru.

Pole	Typ	Opis
id	int(11)	Identyfikator przedmiotu
subjects_type_id	int(11)	*subjects_types→subjects*
acronym	varchar(10)	Skrót nazwy przedmiotu
head	varchar(255)	Nazwa przedmiotu
body	text	Opis przedmiotu
code	varchar(6)	Kod dziekanatowy
season	int(4)	Semestr

Tablica 4.7: Tabela *subjects*

Pole *season* oznacz semestr na którym dany przedmiot jest prowadzony. Jeśli wynosi 0, oznacza to przedmiot specjalny (np. Dziekanat).

Tabela *news_types*

Tabela ta zawiera kategorie aktualności.

Pole	Typ	Opis
id	int(11)	Identyfikator rodzaju aktualności
name	varchar(20)	Rodzaj aktualności

Tablica 4.8: Tabela *news_types*

Tabela *news*

Tabela ta zawiera wykaz aktualności sparametryzowanych jak to pokazuje tablica 4.9

Pole	Typ	Opis
id	int(11)	Identyfikator aktualności
news_type_id	int(11)	*news_types*→*news*
user_id	int(11)	*users*→*news*
subject_id	int(11)	*subjects*→*news*
head	varchar(255)	Tytuł aktualności
body	text	Treść aktualności
ip	varchar(15)	IP z której dodano aktualność
date	datetime	Data dodania
times_readed	int(11)	Ilość przeczytań
for_year	int(11)	Dla których lat

Tablica 4.9: Tabela *news*

Pole *for_year* to maska bitowa roczników, których dotyczy aktualność, przy czym najmłodszy bit to pierwszy rok, najstarszy – piąty. Jeżeli pole ma wartość „0" lub „31" to aktualność dotyczy wszystkich.

Tabela *uploaded_files*

Tabela ta zawiera informacje o plikach (materiałach lub ocenach) załadowanych do systemu. Jest także tabelą pośredniczącą dla relacji wiele-do-wielu między prowadzącymi a przedmiotami (zob. rozdz. 4.3.2, str. 56).

Pole	Typ	Opis
id	int(11)	Identyfikator pliku
subject_id	int(11)	*users*↔*subjects*
user_id	int(11)	*users*↔*subjects*
uploader_id	int(11)	*users*→*uploaded_files*
filename	varchar(255)	Nazwa pliku
head	varchar(255)	Tytuł pliku
body	text	Opis pliku
date	datetime	Data dodania
kind	varchar(15)	Rodzaj
downloads	int(11)	Ilość pobrań

Tablica 4.10: Tabela *uploaded_files*

Pole *filename* (tab. 4.10) zawiera tylko nazwę pliku – system sam określi ścieżkę. Pole *kind* określa rodzaj pliku. Możliwe są dwa stany:

- *material* – plik z materiałami,
- *grade* – plik z ocenami.

Tabela *groups*

Tabela zawiera grupy użytkowników (tab 4.11). Podstawowe typy grup to:

- Administratorzy,
- Zalogowani,
- Moduły (Elektronika, Telekomunikacja),
- Lata (Rok I – Rok V),
- Grupy dziekanatowe (1 – 10).

Pole	Typ	Opis
id	int(11)	Identyfikator grupy
user_id	int(11)	*users*→*groups*
head	varchar(255)	Nazwa grupy
body	text	Opis grupy

Tablica 4.11: Tabela *groups*

Każdy prowadzący może dodawać swoje grupy. Grupy pełnią istotną rolę przy określaniu uprawnień do plików. Dzięki nim możliwe jest dodawanie materiałów np. tylko dla III. roku albo tylko dla modułu Elektronika.

Tabela *polls_ questions*

Tabela (tab. 4.12) zawiera podstawowe informacje o każdej ankiecie.

Pole	Typ	Opis
id	int(11)	Identyfikator pytania
user_id	int(11)	*users→polls_ questions*
question	varchar(200)	Pytanie
start_time	datetime	Czas rozpoczęcia
end_time	datetime	Czas zakończenia
anonymous	tinyint(1)	Czy anonimowa?

Tablica 4.12: Tabela *polls_ questions*

- Pole *question* zawiera pytanie, jakie będzie zadane w ankiecie.
- Pole *end_ time* określa czas zakończenia ankiety. Po jego upływie ankieta zostaje automatycznie zamknięta. Jeśli pozostanie puste, ankieta jest bezterminowa.
- Pole *anonymous* informuje czy ankieta jest anonimowa. W systemie maksymalnie mogą być aktywne naraz dwie ankiety – jedna anonimowa, druga nie.

Tabela *polls_ answers*

Tabela (tab. 4.13) zawiera odpowiedzi do pytania zadanego w ankiecie. Jedna ankieta może mieć maksymalnie sześć odpowiedzi. Wynika to z ograniczenia systemu.

Pole	Typ	Opis
id	int(11)	Identyfikator odpowiedzi
polls_question_id	int(11)	*polls_ questions→polls_ answers*
answer	varchar(200)	Odpowiedź
quantity	int(11)	Ilość udzieleń

Tablica 4.13: Tabela *polls_ answers*

Pole *quantity* zawiera ilość użytkowników którzy wybrali tą odpowiedź. Na podstawie tego pola później jest liczony procentowy udział tego wyboru.

Tabela *exams_names*

Tabela ta zawiera nazwy odpowiednich kombinacji typu studiów, roku i modułu przyporządkowane do ich kodu (tab. 4.14).

Pole	Typ	Opis
id	varchar(3)	Kod
head	varchar(255)	Nazwa

Tablica 4.14: Tabela *exams_names*

Pole *id* wyjątkowo nie jest tu wartością liczbową, a kombinacją trzech znaków. Jest jednak jednoznacznie identyfikowalne i ma odnoszący się do niego klucz obcy w tabeli *exams*.

Tabela *exams*

Tabela ta zawiera informacje o egzaminach w najbliższej sesji.

Pole	Typ	Opis
id	int(11)	Identyfikator egzaminu
subject_id	int(11)	*subjects*→*exams*
user_id	int(11)	*users*→*exams*
exams_name_id	varchar(3)	*exams_names*→*exams*
examiner	varchar(255)	Dane egzaminatora
date	datetime	Data egzaminu
place	varchar(255)	Miejsce egzaminu
term	int(11)	Termin

Tablica 4.15: Tabela *exams*

- Pole *examiner* zawiera dane egzaminatora wpisane w postaci tekstu (tab. 4.15), a nie referencję do odpowiedniego rekordu. Wynika to stąd, że czasami egzamin może przeprowadzać osoba spoza katedry.
- Pole *term* określa termin egzaminu i powinno zawierać się w przedziale <0;3>.

Tabela *events*

Tabela ta zawiera kalendarz studenta (tab. 4.16).

Pole	Typ	Opis
id	int(11)	Identyfikator wydarzenia
beginning	date	Początek
ending	date	Koniec
head	varchar(255)	Opis
for_year	int(11)	Dla których lat
url	varchar(255)	Link

Tablica 4.16: Tabela *events*

- Pole *for_ year* ma identyczne znaczenie jak to z tabeli *news* (zob. rozdz. 4.3.1, str. 47).
- Pole *url* nie jest obowiązkowe. Jeśli jednak zostanie podane, to cały tytuł będzie linkiem do podanego w *url* adresu.

Tabela *declarations*

Tabela ta zawiera informacje konfiguracyjne deklaracji. Nie zawiera informacji o wyborach dokonanych przez studentów.

Pole	Typ	Opis
id	int(11)	Identyfikator deklaracji
available_from	date	Dostępna od...
available_to	date	Dostępna do...
code	varchar(255)	Kod deklaracji
head	varchar(255)	Nazwa deklaracji
year	int(11)	Dla którego roku
how_many	int(11)	Ile opcji

Tablica 4.17: Tabela *declarations*

- Pola *available_from* i *available_ to* określają przedział czasowy w którym dana deklaracja jest dostępna do wypełnienia (tab. 4.17).
- Pole *year* określa, którego roku studiów taka deklaracja dotyczy. Jedna deklaracja jest przewidziana tylko dla jednego roku. Podobne deklaracje dla różnych lat, wymagają osobnych rekordów w bazie.
- Pole *how_ many* dotyczy tylko deklaracji odnoszących się do wyboru przedmiotów obieralnych. Jest to liczba dwucyfrowa. Liczba dziesiątek oznacza liczbę przedmiotów do wybrania dla modułu Elektronika, jedności – dla modułu Telekomunikacja.

Tabela *declarations_subjects*

Tabela (tab. 4.18) ta zawiera szczegółowe informacje o deklaracjach, a także informacje wypełniane przez studentów. Tabelę tę można dowolnie modyfikować. Więcej na ten temat zostało przedstawione w części Deklaracje (zob. rozdz. 4.7.4, str. 77).

Pole	Typ	Opis
id	int(11)	Identyfikator parametru deklaracji
declaration_id	int(11)	*declarations→declarations_subjects*
subject_id	int(11)	*subjects→declarations_subjects*
user_id	int(11)	*users→declarations_subjects*
price	int(11)	Cena materiałów
name	varchar(255)	Nazwa po angielsku
grade	decimal(2,1)	Ocena
year	int(11)	Rok rozpoczęcia studiów
speciality_id	int(11)	*specialities→declarations_subjects*
date	date	Data złożenia
language	varchar(4)	Język przedmiotu
print	tinyint(1)	Drukować materiały?
study_type	varchar(1)	Typ studiów
study_speciality	int(11)	Specjalność

Tablica 4.18: Tabela *declarations_subjects*

- Pole *price* dotyczy tylko druku materiałów. Oznacza ona cenę materiałów w złotówkach pomnożoną przez 10. Nie wypełniają go studenci.
- Pole *name* dotyczy tylko deklaracji wyboru języka przedmiotów. Zawiera angielski odpowiednik nazwy przedmiotu. Nie wypełniają go studenci.
- Pole *grade* dotyczy tylko deklaracji wyboru kierunku studiów. Oznacza ono ocenę z danego przedmiotu. Jest wypełniane przez studentów. Dodatkowo w modelu Rails jest metoda *average*, która obliczą średnią ocen danego studenta.
- Pole *year* dotyczy tylko deklaracji wyboru modułu studiów. Oznacza ono rok, w którym dany student rozpoczął studiowanie na Elektronice i Telekomunikacji. Jest wypełniane przez studentów.
- Pole *speciality_id* dotyczy tylko deklaracji wyboru modułu studiów. Zawiera *id* kierunku (referencja do tabeli *specialities*) na którym dany student chciałby kontynuuować swoją edukację. Jest wypełniane przez studentów.
- Pole *date* dotyczy wszystkich deklaracji. Zawiera datę ich złożenia. Jest wypełniane przez system automatycznie.

- Pole *language* dotyczy tylko deklaracji wyboru języka przedmiotów. Zawiera dwuliterowe kody języków wybrane przez studenta w formacie WWDD, gdzie WW to kod dla wykładu, a DD to kod dla zajęć dodatkowych (laboratoriów, ćwiczeń, itd.). Jest wypełniane przez studentów. Np. ENEN – wszystko po angielsku, zaś opcja PLEN jest zabroniona.

- Pole *print* dotyczy tylko deklaracji dotyczących druku materiałów. Jeśli jest ustawione na 1 oznacza, że dany student życzy sobie materiały w formie papierowej dla danego przedmiotu. Jest wypełniane przez studentów.

- Pole *study_type* dotyczy tylko deklaracji wyboru specjalności. Określa ono rodzaj studiów – M to studia magisterskie, I to studia inżynierskie. Jest wypełniane przez studentów. Obecnie nieużywane.

- Pole *study_speciality* dotyczy tylko deklaracji wyboru specjalności. Zawiera ono cyfrowy kod specjalności. Jest wypełniane przez studentów.

Tabela *declarations_ experiences*

Tabela ta (tab. 4.19) zawiera szczegółowe informacje dotyczące praktyki zawodowej danego studenta.

Pole	Typ	Opis
id	int(11)	Identyfikator deklaracji praktyki
declaration_id	int(11)	*declarations→declarations_ experiences*
firstname	varchar(255)	Imię studenta
lastname	varchar(255)	Nazwisko studenta
sindex	int(6)	Numer indeksu
speciality_id	int(11)	*specialities→declarations_ experiences*
student_street	varchar(255)	Adres zamieszkania
student_postal_code	varchar(255)	Kod pocztowy
student_city	varchar(255)	Miasto
place_name	varchar(255)	Miejsce praktyki
place_street	varchar(255)	Adres zakładu pracy
place_postal_code	varchar(255)	Kod pocztowy
place_city	varchar(255)	Miasto
beginning	date	Termin rozpoczęcia
beginning_additional	date	Dodatkowy termin
ending_additional	date	Koniec dodatkowego terminu
dormitory	tinyint(1)	Czy chce akademik?
policy_type	varchar(255)	Rodzaj ubezpieczenia
policy_name	varchar(255)	Nazwa polisy
policy_number	varchar(255)	Numer polisy
confirmations	int(11)	Potwierdzenia
created_at	datetime	Utworzono
updated_at	datetime	Poprawiono

Tablica 4.19: Tabela *declarations_ experiences*

- Pola *student_ street*, *student_ postal_ code*, *student_ city* dotyczą adresu zameldowania studenta.
- Pola *place_ name*, *place_ street*, *place_ postal_ code*, *pace_ city* dotyczą informacji o zakładzie pracy, w którym będzie odbywać się praktyka.
- Pole *beginning* to termin rozpoczęcia praktyki.
- Pola *beginning_ additional* i *ending_ additional* określają dodatkowy termin praktyki w przypadku gdyby praktyka była dwuetapowa.
- Pole *dormitory* określa czy student jest zainteresowany pobytem w akademiku na czas praktyki.
- Pola *policy_ type*, *policy_ name* i *policy_ number* dotyczą obowiązkowego dokumentu potwierdzającego ubezpieczenie na czas praktyki.

E	D	C	B	A

– A – dostarczone zaświadczenie z miejsca praktyki
– B – dostarczona informacja o ubezpieczeniu
– C – dostarczone potwierdzenie odbycia praktyki i sprawozdanie
– D – praktyka bezpłatna
– E – wpis do indeksu

Rysunek 4.4: Struktura pola *confirmations*

- Pole *confirmations* zawiera potwierdzenia dla opiekuna praktyk (w formie maski bitowej), a jego struktura została przedstawiona na ryskunku 4.4.

4.3.2 Opis relacji

W poprzednim rozdziale opisano wszystkie tabele. Są one połączone ze sobą za pomocą relacji przedstawionych w tej części.

Relacja *users→users_students*

- Typ relacji: 1 do 1
- Pole: *users_students.user_id*
- Klucz obcy: *users_students_ibfk_1*
- Obowiązkowa
- Dodatkowe informacje o studencie, specyficzne tylko dla niego.

Relacja *specialities→users_students*

- Typ relacji: 1 do wielu
- Pole: *users_students.speciality_id*
- Klucz obcy: *users_students_ibfk_2*
- Obowiązkowa
- Informacje o module studiów studenta. Jeśli jeszcze nie został wybrany, pole przyjmuje wartość 1 (zob. tab. 4.3.1, str. 43).

Relacja *cathedrals→users_lecturers*

- Typ relacji: 1 do wielu
- Pole: *users_lecturers.cathedral_id*
- Klucz obcy: *users_lecturers_ibfk_1*
- Obowiązkowa
- Katedra do której dany prowadzący należy.

Relacja *users→users_lecturers*

- Typ relacji: 1 do 1
- Pole: *users_lecturers.user_id*
- Klucz obcy: *users_lecturers_ibfk_2*
- Obowiązkowa
- Dodatkowe informacje o prowadzącym, specyficzne tylko dla niego.

Relacja *subjects_types→subjects*

- Typ relacji: 1 do wielu
- Pole: *subjects.subjects_type_id*
- Klucz obcy: *subjects_ibfk_1*
- Nieobowiązkowa
- Informacje o rodzaju danego przedmiotu.

Relacja *users↔subjects*

- Typ relacji: wiele do wielu
- Pola: *uploaded_files.user_id*, *uploaded_files.subject_id*
- Tabela pośrednicząca: *uploaded_files*
- Klucze obce: *uploaded_files_ibfk_1*, *uploaded_files_ibfk_2*
- Obowiązkowa

- Relacja wykorzystywana do określenia, które przedmioty prowadzi dana osoba. Aby uniknąć z każdym semestrem ręcznego ich dodawania, wszystko dzieje automatycznie. Kryterium bycia prowadzącym jest dodanie pliku do systemu i przypisanie kogoś jako prowadzącego do określonego przedmiotu. Wtedy opis takiej osoby (oczywiście, jeśli dodatkowo jest wpisana do tabeli *users_lecturers*) automatycznie tworzy relację z tym przedmiotem poprzez tabelę *uploaded_files*, co system odczytuje również jako prowadzenie tychże zajęć.

Relacja *news_types→news*

- Typ relacji: 1 do wielu
- Pole: *news.news_type_id*
- Klucz obcy: *news_ibfk_1*
- Nieobowiązkowa
- Informacje o rodzaju aktualności. Obecnie niewykorzystywane. Zachowane dla wstecznej zgodności i dla przyszłych zastosowań.

Relacja *users→news*

- Typ relacji: 1 do wielu
- Pole: *news.user_id*
- Klucz obcy: *news_ibfk_2*
- Obowiązkowa
- Określa autora danej aktualności.

Relacja *subjects→news*

- Typ relacji: 1 do wielu
- Pole: *news.subject_id*
- Klucz obcy: *news_ibfk_3*
- Nieobowiązkowa
- Określa przedmiot, którego dotyczy dana aktualność.

Relacja *users→uploaded_files*

- Typ relacji: 1 do wielu
- Pole: *uploaded_files.uploader_id*
- Klucz obcy: *uploaded_files_ibfk_3*
- Obowiązkowa
- Informacja o tym kto dodał dany plik dla danego przedmiotu. Nie jest to jednoznaczne z byciem prowadzącym. Podczas dodawania pliku można określić, kto prowadzi dany przedmiot i może to być osoba inna od osoby dodającej plik.

Relacja *users→groups*

- Typ relacji: 1 do wielu
- Pole: *groups.user_id*
- Klucz obcy: *groups_ibfk_1*
- Nieobowiązkowa
- Informacja o założycielu danej grupy.

Relacja *groups↔users*

- Typ relacji: wiele do wielu
- Tabela pośrednicząca: *groups_users*
- Pola: *groups_users.group_id*, *groups_users.user_id*
- Klucze obce: *groups_users_ibfk_1*, *groups_users_ibfk_2*
- Obowiązkowa
- Informacje o przypisaniu użytkowników do grup.

Relacja *groups↔news*

- Typ relacji: wiele do wielu
- Tabela pośrednicząca: *groups_news*
- Pola: *groups_news.group_id*, *groups_news.news_id*

- Klucze obce: *groups_news_ibfk_1*, *groups_news_ibfk_2*
- Obowiązkowa
- Informacje o uprawnieniach do odczytu określonych wiadomości przez grupy. W poprzednim systemie funkcjonalność praktycznie nieużywana. W obecnym porzucona, zachowana tylko na poczet przyszłych ustaleń.

Relacja *groups↔uploaded_files*

- Typ relacji: wiele do wielu
- Tabela pośrednicząca: *groups_uploaded_files*
- Pola: *groups_uploaded_files.group_id*, *groups_uploaded_files.uploaded_file_id*
- Klucze obce: *groups_uploaded_files_ibfk_1*, *groups_uploaded_files_ibfk_2*
- Obowiązkowa
- Uprawnienia członków grup do określonych plików. Bardzo ważna funkcjonalność, która zabezpiecza np. materiały dla studentów określonego roku, przed ściągnięciem ich przez studentów innych lat, modułów czy innych grup.

Relacja *users→polls_questions*

- Typ relacji: 1 do wielu
- Pole: *polls_questions.user_id*
- Klucz obcy: *polls_questions_ibfk_1*
- Obowiązkowa
- Określa autora ankiety.

Relacja *polls_questions→polls_answers*

- Typ relacji: 1 do wielu
- Pole: *polls_answers.polls_question_id*
- Klucz obcy: *polls_answers_ibfk_1*
- Obowiązkowa
- Przypisuje daną odpowiedź do określonej ankiety.

Relacja *subjects→exams*

- Typ relacji: 1 do wielu
- Pole: *exams.subject_id*
- Klucz obcy: *exams_ibfk_1*
- Obowiązkowa
- Przedmiot z którego odbędzie się egzamin.

Relacja *users→exams*

- Typ relacji: 1 do wielu
- Pole: *exams.user_id*
- Klucz obcy: *exams_ibfk_2*
- Obowiązkowa
- Osoba, która dodała informację o egzaminie.

Relacja *exams_names→exams*

- Typ relacji: 1 do wielu
- Pole: *exams.exams_name_id*
- Klucz obcy: *exams_ibfk_3*
- Obowiązkowa
- Specyficzna relacja, gdyż kluczem obcym nie jest tu liczba a trzyznakowy kod. Kod ten – to kombinacja typu studiów, roku i modułu. Wszystkie egzaminy w systemie są pogrupowane wg. tych kodów, a ta relacja służy do rozwinięcia ich na pełną nazwę.

Relacja *declarations→declarations_subjects*

- Typ relacji: 1 do wielu
- Pole: *declarations_subjects.declaration_id*
- Klucz obcy: *declrations_subjects_ibfk_1*
- Obowiązkowa
- Relacja przypisująca parametry, danej deklaracji.

Relacja *subjects→declarations_ subjects*

- Typ relacji: 1 do wielu
- Pole: *declarations_ subjects.subject_ id*
- Klucz obcy: *declrations_ subjects_ ibfk_ 2*
- Nieobowiązkowa
- Przedmiot, którego dotyczy dana deklaracja.

Relacja *users→declarations_ subjects*

- Typ relacji: 1 do wielu
- Pole: *declarations_ subjects.user_ id*
- Klucz obcy: *declrations_ subjects_ ibfk_ 3*
- Nieobowiązkowa
- Użytkownik, którego dotyczy deklaracja. Zawsze powinna być ustalona, jeśli student dodaje jakąś deklarację. Jest nieobowiązkowa tylko w przypadku, gdy wpis określa parametry samej deklaracji (zob. rozdz. 4.7.4 str. 77)

Relacja *specialities→declarations_ subjects*

- Typ relacji: 1 do wielu
- Pole: *declarations_ subjects.speciality_ id*
- Klucz obcy: *declrations_ subjects_ ibfk_ 4*
- Nieobowiązkowa
- Określa moduł wybrany przez studenta. Istotna tylko dla deklaracji wyboru modułu.

Relacja *declarations→declarations_ experiences*

- Typ relacji: 1 do wielu
- Pole: *declarations_ experiences.declaration_ id*
- Klucz obcy: *declrations_ experiences_ ibfk_ 1*
- Obowiązkowa
- Relacja parametrów deklaracji praktyki, do określonej deklaracji praktyki.

Relacja *specialities→declarations_ experiences*

- Typ relacji: 1 do wielu
- Pole: *declarations_ experiences.speciality_ id*
- Klucz obcy: *declrations_ experiences_ ibfk_ 2*
- Obowiązkowa
- Moduł studiów studenta zgłaszającego się na praktykę.

4.4 Konwersja bazy danych

Najważniejszym problemem jaki należało rozwiązać była konwersja istniejącej bazy danych na format zgodny z Active Record. Dodatkowym utrudnieniem był fakt, że dotychczasowa baza miała rozwiązane w różny sposób relacje – raz były to klucze liczbowe, innym razem tekstowe. Dodatkowo nazwy kolumn często były mylące. Kolejnym problemem było kodowanie polskich znaków – różnie rozwiązane w różnych tabelach.

Rozwiązaniem tych problemów jest specjalny konwerter. Został on zaprojektowany jako skrypt typu *rake task*, który dokona automagicznie konwersji bez jakiejkolwiek ingerencji użytkownika. Jedyne co trzeba zrobić, to przygotować wcześniej specjalny plik konfiguracyjny *config/convert_ map.yml*. Jest to plik formatu YAML (zob. rozdz. 2.6, str.17), zawierający wszelkie wytyczne informujące konwerter jak ma przeprowadzić proces przeniesienia.

4.4.1 Łączenie konwertera z bazami

Do połączenia z bazami konwerter wykorzystuje standardowy plik konfiguracyjny w Ruby on Rails o nazwie *config/database.yml* (list. 4.3).

```
1  original:
2    adapter: mysql
3    database: eit_old
4    username: eit
5    password: haslo
6    encoding: latin2
7  converted:
8    adapter: mysql
9    database: eit_new
10   username: eit
```

```
11    password: haslo
12    encoding: utf8
```

Listing 4.3: Przykładowe wpisy konwertera w pliku *config/database.yml*

Należy zdefiniować dwie nowe reguły *original* (dotyczącą połączenia ze starą bazą) i *converted* (połączenie do nowej bazy). Bardzo ważne jest, aby pamiętać o określeniu typu kodowania. Jeśli kodowanie w *original* i *converted* będą się różnić aplikacja sama zrzutuje jeden standard znaków na drugi. Dlatego należy przypilnować, aby w systemie była zainstalowana biblioteka *iconv*, który odpowiada właśnie za to rzutowanie.

4.4.2 Plik konfiguracyjny konwersji

Struktura pliku konfiguracyjnego konwersji (wraz z przykładem) jest przedstawiona na list. 4.4 i list. 4.5.

```
<liczba porzadkowa>_<nowa nazwa tabeli>:
  <stara nazwa tabeli>:
     <... opcje tabeli ...>
    <stara nazwa kolumny>:
       <... opcje kolumny ...>
    <kolejna stara kolumna>:
       <... opcje kolumny ...>
<liczba porzadkowa>_<kolejna tabela>:
   <... itd. ...>
```

Listing 4.4: Struktura pliku *config/convert_map.yml*

```
57 03_users_students:
58   users_k1:
59     __BEFORE: INSERT INTO @ (id) SELECT id FROM users WHERE id NOT IN (
60 D login IS NOT NULL AND kind = 1
61     __AFTER: ALTER TABLE users_k1 DROP band2
62     id:
63       from: users
64       name: user_id
65     band:
66       name: sgroup
67     band2:
68       name: speciality_id
69       function: IF((SELECT (SUBSTRING(NOW(),1,4) - CONCAT('20', SUBSTRI
70     indeks:
```

```
71        name: sindex
72      gg:
73        name: gadu_gadu
74      module:
75        action: remove
76      modified:
77        action: remove
78      nband:
79        action: remove
```

Listing 4.5: Fragment pliku *config/convert_map.yml*

Opcje w pliku można podzielić na dwie grupy – opcje tabeli i opcje kolumny. Opcje tabeli to:

- *__BEFORE* – określa zapytanie SQL, które ma zostać wykonane przed rozpoczęciem konwersji na starej tabeli. Jest to użyteczne, gdy trzeba wcześniej poprawić jakieś niespójne dane, aby później ułatwić w ten sposób pracę konwerterowi. Znak „@" (at) przy składaniu zapytania zostanie zamieniony na nazwę starej tabeli.

- *__AFTER* – analogicznie jak powyżej, tylko zapytanie zostanie wykonane po konwersji. Zazwyczaj używane do odwrócenia zmian wykonanych w *__BEFORE*, aby pozostawić starą bazę danych w nienaruszonym stanie.

- *__WHERE* – ta opcja spowoduje dodanie klauzuli *WHERE* do zapytania wyciągającego dane ze starej tabeli. Dzięki temu konwersji mogą zostać poddane tylko określone wiersze (ponieważ np. pozostałe są już przestarzałe i nie ma dla nich miejsca w nowej bazie, albo chcemy rozbić konwersję na kilka etapów i każdy z nich przeprowadzić osobno).

Opcje kolumn stanowią:

- *action* – akcja, jaką należy podjąć na tej kolumnie. Dostępne są dwie akcje: *copy* i *remove*. Pierwsza kopiuje dane z kolumny do nowej tabeli, druga nie bierze pod uwagę tej kolumny przy konwersji. Domyślna akacja to *copy*.

- *name* – nowa nazwa kolumny. Domyślnie kopiowana jest stara.

- *from* – definicja funkcji mappera wartości. Często zdarza się tak, że przy kopiowaniu starych tabel na nowe niektóre wartości nie spełniają kryteriów nowej bazy danych i są odrzucane. Skutkiem tego zmienia się ułożenie wierszy w tabeli, a co za tym idzie, także wartości pól *id*. Podczas pracy konwertera, równolegle działa również specjalny kod mapujący, który odwzorowuje stary

numer *id* na nowy. Dzięki temu zachowane zostają relacje i spójność bazy danych. Opcja *from* określa, mapa której tabeli ma zostać wykorzystana przy konwersji obcych *id*. Na przykład, przenoszona jest tabelka *users* i zniknęło kilkudziesięciu użytkowników. Zmieniły się więc numery *id* użytkowników bezpośrednio po nich. Teraz następuje przenoszenie tabelki *users_students*, która ma jednak referencję do tabelki *users*. Gdyby nie zastosowano mapowania, referowane *user_id* przeniosłyby się tak jak były, skutkiem czego wskazywałyby na zupełnie innych użytkowników. Ważna uwaga – parametr *from* jako nazwę mapy ustawia sobie nazwę nowej tabeli. Jeśli więc przenosiliśmy np. tabelkę *specs* na *subjects*, to mapa będzie miała nazwę *subjects* i taką należy podać.

- *field* – polecenie określające po jakim polu odbywa się mapowanie. Domyślnie jest to *id* i w większości przypadków nie należy tego zmieniać.
- *function* – funkcja SQL, której działaniu zostanie poddana nowa wartość zanim zostanie wpisana do nowej tabeli. Samą daną reprezentuje „@" (at). Na przykład, *function: SHA1(@)* spowoduje, że wszystkie dane z tej kolumny będą poddawane haszowaniu SHA1, zanim zostaną wpisane do nowej bazy.
- *connect* – służy do łączenia wartości kilku kolumn w starej tabeli w jedną w nowej. Nazwy kolumn należy oddzielić przecinkiem, np. *"dt_year, dt_month, dt_day, dt_hour, dt_min"*
- *pattern* – wymagane razem z connect. Określa schemat połączenia kolumn. Np. *"{dt_year}-{dt_month}-{dt_day} {dt_hour}:{dt_min}"* stworzy z pięciu kolumn reprezentujących datę, jedną zgodną z formatem DATETIME.

4.4.3 Uruchamianie konwertera

Przed rozpoczęciem konwersji warto wcześniej wyczyścić nową bazę danych (jeżeli dokonujemy zmian). Konwerter uruchamia się, wpisując w głównym katalogu aplikacji polecenie przedstawione na list. 4.6

```
$ rake eit:db:convert
```

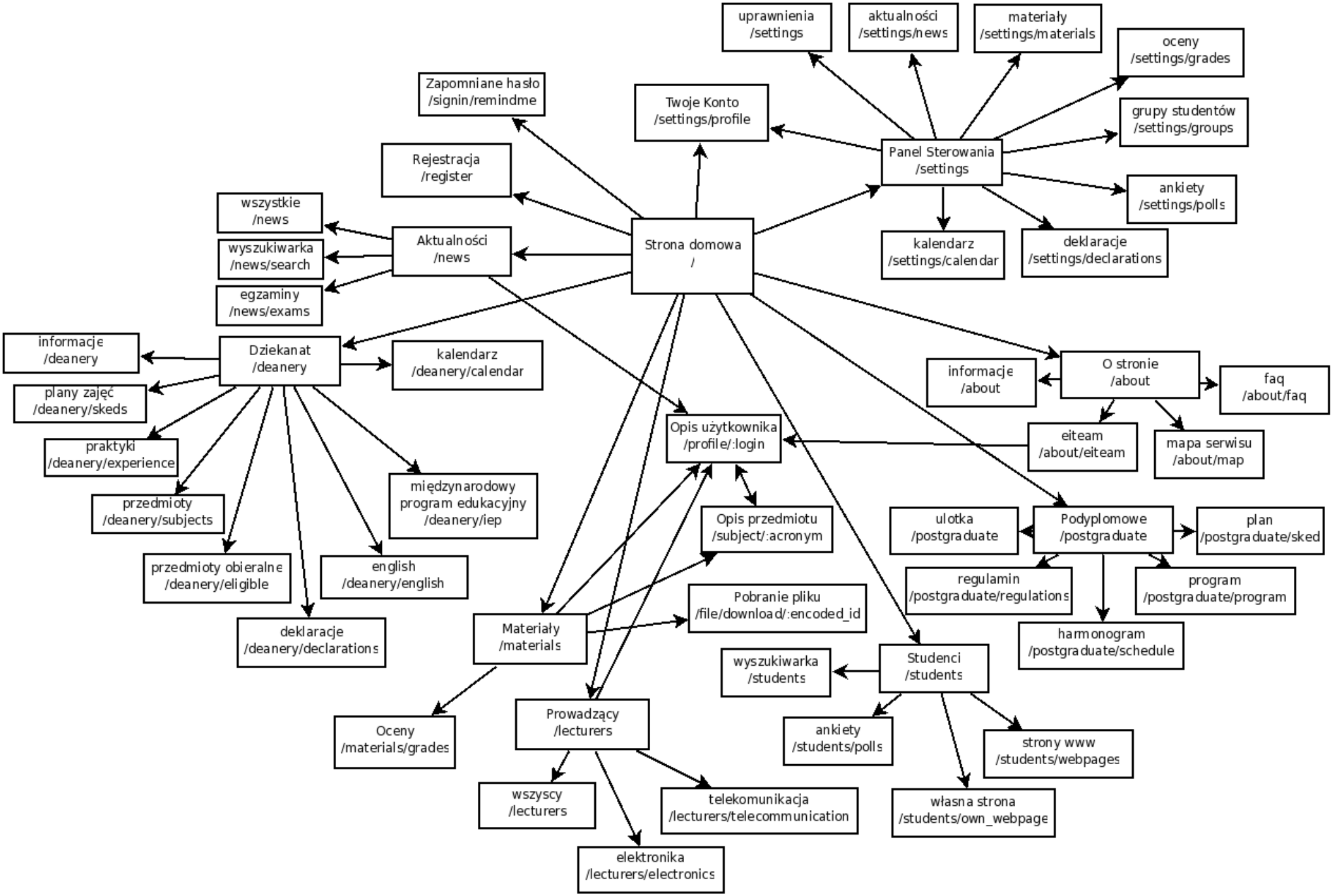
Listing 4.6: Polecenie wykonujące konwersję baz danych

Wszystkie błędy w czasie działania konwertera są zapisywane w pliku *log/conversion.log*. Wpisy w nim niekoniecznie oznaczają, że coś poszło w sposób nieplanowany. Informują one o tym, które wiersze nie zostały przeniesione do nowej bazy – zazwyczaj powodem jest nie spełnianie one nowych kryteriów (np. w nowej tabeli kolumna ma atrybut *NOT NULL* skutkiem czego wszystkie dane, które były *NULL* nie zostaną przeniesione). Po automatycznej konwersji powinna być zatem

wykonana dodatkowa praca, sprawdzenia odrzuconych rekordów i przenoszenia ich w razie potrzeby.

4.5 Mapa serwisu

Kolejnym krokiem było zbudowanie mapy serwisu. Bardzo ważne jest, aby na etapie projektu mapy zaprojektować także adresy URL. Użytkownicy używają ich na różne sposoby. Wysyłają je do innych osób poprzez e-mail, publikują je na forach internetowych, dyktują przez telefon. Najczęściej nie są to adresy strony głównej, ale adresy podstrony. Warto więc zadbać aby adres był krótki i opisowy, ułatwi to w znaczny sposób korzystanie z serwisu. Na rysunku 4.5 przedstawiono zastosowany schemat mapy strony. Znajdują się tu tytuły stron wraz z odpowiadającymi im ścieżkami.



Rysunek 4.5: Mapa strony kierunku Elektronika i Telekomunikacja.

4.6 Opis poszczególnych kontrolerów i akcji

W tej części pracy opisano kontrolery systemu MVC wraz z przypisanymi im akcjami.

4.6.1 *IndexController*

Główny kontroler, uruchamiany przy pierwszym wejściu na stronę. Zawiera tylko jedną akcję – *index*.

- *index* – akcja odpowiedzialna za wyświetlanie pierwszej, „powitalnej" strony EiT. Ładuje siedem ostatnich aktualności, siedem ostatnio dodanych materiałów, ostatnie ankiety oraz ostatnio dodane oceny.

4.6.2 *AboutController*

Kontroler wyświetlający informacje o samej stronie EiT i osobach, które ją tworzyły.

- *index* – rys historyczny powstawania strony Elektroniki i Telekomunikacji,
- *eiteam* – zespół odpowiedzialny za stronę,
- *map* – mapa serwisu,
- *faq* – najczęściej zadawane pytania odnośnie witryny.

4.6.3 *CsvController*

Kontroler odpowiedzialny za generowanie plików *.csv* i wysyłanie ich użytkownikom. Głównie stosowany przy podsumowaniach deklaracji. Każda akcja wysyła określony plik csv.

- *language* – informacje o językach prowadzenia przedmiotów, wybranych przez poszczególnych studentów,
- *print* – informacje o zainteresowaniu wydrukiem materiałów,
- *module* – kto wybrał jaki moduł,
- *subjects* – przypisanie studentów, do poszczególnych przedmiotów obieralnych,
- *experience* – zbiorcze deklaracje praktyk,
- *speciality* – informacje o wyborze kierunku studiów i specjalności.

4.6.4 *DeaneryController*

Kontroler odpowiedzialny za wyświetlanie informacji z dziekanatu.

- *index* – wyświetla podstawowe informacje o osobach pracujących w dziekanacie

- *subjects* – lista wszystkich przedmiotów z podziałem na lata, kierunki, moduły i specjalności,
- *eligible* – lista przedmiotów obieralnych dla poszczególnych modułów i semestrów, wraz z liczbą studentów, którzy je wybrali,
- *english* – informacje o przedmiotach prowadzonych w języku angielskim,
- *iep* – informacje o międzynarodowym programie edukacyjnym,
- *declarations* – sekcja wymagająca zalogowania się, dostępna tylko dla studentów. Umożliwia wypełnienie wszystkich aktualnie dostępnych deklaracji, a także zawiera informacje na ich temat. Więcej informacji w sekcji „Deklaracje" rozdz. 4.7.4, str. 77,
- *calendar* – kalendarz studenta. Wszystkie istotne wydarzenia z jego punktu widzenia (sesje, ferie, terminy składania podań, itd.).

4.6.5 *FileController*

Kontroler odpowiedzialny za pobieranie plików z serwisu.

- *download* – zgodnie z polityką bezpieczeństwa, wszystkie pliki dodawane przez użytkowników nie są trzymane w katalogu widocznym z poziomu serwera WWW. Wynika to stąd, że wtedy każdy mógłby podać bezpośredniego linka do takiego pliku i cały mechanizm logowania i zabezpieczania materiałów byłby niepotrzebny. Zamiast tego pliki leżą w specjalnym katalogu (tu */files*) i są przetwarzane przez skrypt. Polega to na sprawdzeniu czy plik istnieje oraz czy użytkownik, który się do niego odwołuje ma do niego prawa dostępu. Jeśli tak, następuje pobranie, jeśli nie, strona zwraca błąd braku uprawnień.

4.6.6 *LecturersController*

Kontroler wyświetlający informacje o wszystkich prowadzących dopisanych do systemu.

- *index* – wyświetla listę wszystkich prowadzących, wraz z linkami do ich profili.
- *method_missing* – nazwa metody interpretowana jest jako nazwa katedry. Jeśli w bazie istnieje katedra dla której pole *action* jest zgodne z nazwą metody, to wyświetlana jest lista prowadzących przypisanych do tej katedry.

4.6.7 *MaterialsController*

Kontroler odpowiedzialny za wyświetlanie i wyszukiwanie materiałów oraz ocen dodanych do serwisu.

- *index* – wyświetla 10 ostatnich materiałów oraz stanowi punkt wyjścia przy wyszukiwaniu kolejnych. Kiedy użytkownik wypełni pola i uruchomi wyszukiwanie, akcja ta pobiera zapytanie POST z formularza i przetwarza je na zapytanie GET. W takiej postaci przesyłane jest ono do akcji *search*. Całe to przetwarzanie zrobione jest w celu podawania zapytania do wyszukiwarki bezpośrednio w postaci linków – w wielu miejscach są np. linki do materiałów konkretnego prowadzącego, które nie są niczym innym, jak odpowiednio przygotowanym zapytaniem dla akcji *search*,

- *search* – pobiera zapytanie, złożone z pięciu kryteriów oddzielonych przecinkami. Są to kolejno:

  – zakodowane ID przedmiotu,

  – zakodowane ID prowadzącego,

  – numer semestru,

  – treść wyszukiwania,

  – rodzaj sortowania wyników.

  W przypadku braku którejkolwiek z wartości nie jest ona brana pod uwagę przy wyszukiwaniu,

- *grades* – dostępne tylko dla zalogowanych użytkowników listy ocen. Kryterium wyszukiwania jest grupa – jeśli użytkownik i plik z oceną należą do tej samej, to link do pliku jest wyświetlany.

4.6.8 *NewsController*

Kontroler odpowiadający za wyświetlanie aktualności.

- *index* – akcja wyświetlająca wszystkie aktualności (stronami po 10) a także robiąca za pośrednika dla wyszukiwarki (patrz niżej),

- *search* – wyszukiwarka aktualności. Przyjmuje parametry rozdzielone przecinkami, metodą GET. W przypadku ich braku wyświetla formularz. Mechanizm działania jest następujący – użytkownik wpisuje szukaną informacje i określa kryteria wyszukiwania; następnie całość jest składana w zapytanie typu POST

i zostaje przekierowane do akcji *index*; akcja *index* rozpakowuje zapytanie, buduje odpowiednie żądanie GET dla akcji *search*, a następnie go do niej wysyła; akcja *search* wyświetla posegregowane wyniki wyszukiwania (stronami po 10),

- *read* – akcja, która jako parametr przyjmuje zakodowane ID wiadomości (zob. rozdz. 4.7.1, str. 74), odkodowuje je, a następnie wyświetla aktualność zgodną z tym numerem,
- *exams* – akcja wyświetlająca informacje o egzaminach w aktualnej sesji. Dodatkowo, jeżeli użytkownik posiada stosowne uprawnienia, umożliwia dodawanie, edycję i usuwanie egzaminów.

4.6.9 *PostgraduateController*

Kontroler odpowiadający za wyświetlanie informacji na temat studiów podyplomowych.

- *index* – ulotka reklamowa,
- *regulations* – regulamin,
- *schedule* – harmonogram,
- *program* – program,
- *sked* – plan zajęć.

4.6.10 *ProfileController*

Kontroler odpowiedzialny za wyświetlanie informacji o użytkownikach.

- *view* – wyświetla informacje o użytkowniku, którego zakodowane ID przyjmuje jako parametr. Zalogowany użytkownik może wyświetlić informacje o każdym, niezalogowany – tylko o prowadzących.

4.6.11 *RedirectController*

Kontroler odpowiedzialny za odpowiednie przekierowywanie ze starych linków na nowe (zob. rozdz. 4.8, str. 79)

- *index* – przekierowuje na adres podany w parametrze

4.6.12 *RegisterController*

Kontroler odpowiedzialny za rejestrację. Cały mechanizm rejestracji jest dość specyficzny, z racji tego, że nie każdy może dołączyć do serwisu. Dopisać się mogą tylko osoby znajdujące się na liście studentów corocznie dostarczanej przez dziekanat. Takie osoby są dopisywane do bazy bez przydzielonego loginu i hasła. Proces rejestracji polega więc, wpierw na uwierzytelnieniu się swoim imieniem, nazwiskiem i numerem indeksu, a następnie na podaniu pozostałych informacji o sobie. Po pomyślnym przejściu procesu student staje się użytkownikiem systemu.

- *index* – przeprowadzenie procesu rejestracji, tylko dla niezalogowanych użytkowników.

4.6.13 *RssController*

Kontroler generujący treść dla czytników RSS (zob. rozdz. 2.5, str. 16).

- *index* – zwraca 15 ostatnich aktualności w formacie zgodnym z RSS.

4.6.14 *RtfController*

Kontroler stworzony w celu serwowania predefiniowanych, dynamicznie uzupełnianych dokumentów *.rtf*.

- *experience* – wysyła użytkownikowi (oczywiście, jeśli ma odpowiednie uprawnienia) komplet gotowych umów o praktykę między studentami (którzy wypełnili deklaracje praktyk) a ich zakładami pracy. Obecnie zamieniony funkcjonalnością przez korespondencję seryjną w formacie CSV.

4.6.15 *SettingsController*

Kontroler zbiorczy, odpowiedzialny za wszystkie zadania związane z administracją serwisu.

- *index* – wyświetlenie strony powitalnej zawierającej odnośniki do wszystkich sekcji administracyjnych wraz z określeniem praw dostępu (zielona kropka – dostęp dozwolony, czerwona – zabroniony),
- *news* – administracja aktualnościami,
- *materials* – dodawanie materiałów oraz ustalanie do nich praw dostępu,
- *grades* – j.w. dla ocen,

- *groups* – administrowanie grupami studentów; nie można zmienić domyślnie zadeklarowanych, można jednak dodawać, usuwać i edytować swoje,
- *polls* – dodawanie ankiet,
- *declarations* – wyświetlenie informacji o deklaracjach wypełnionych przez studentów oraz zarządzanie nimi (zob. rozdz. 4.7.4, str. 77,
- *calendar* – zarządzanie kalendarzem studenta,
- *profile* – zarządzanie profilem aktualnie zalogowanego użytkownika. Z poziomu tej akcji, można zmienić swój login, hasło i kilka innych danych.

4.6.16 *SigninController*

Kontroler obsługujący wszelkie akcje związane z logowaniem użytkownika do systemu.

- *index* – akcja odpowiadająca za logowanie użytkownika,
- *signout* – wylogowuje zalogowanego,
- *remindme* – uruchamia procedure odzyskania zapomnianego hasła. Wpierw użytkownik proszony jest o podanie imienia, nazwiska i numeru indeksu. Jeżeli dane te zostaną pomyślnie zweryfikowane użytkownikowi wyświetlone jest pytanie kontrolne. Odpowiedź na nie jest przekierowywana do akcji *answer*. W przypadku, kiedy użytkownik nie zadeklarował pytania ratunkowego wyświetlana jest stosowna informacja,
- *answer* – ta akcja sprawdza odpowiedź podaną przez użytkownika na pytanie kontrolne. Jeśli jest poprawna, następuje zalogowanie do systemu.

4.6.17 *StudentsController*

Kontroler zawierający wszelkie informacje dotyczące samych studentów.

- *index* – wyszukiwarka studentów, dostępna tylko dla zalogowanych użytkowników. Pozwala na wybór kryteriów wyszukiwania, wpływ na wyświetlane wyniki i określenie kolumny po której odbędzie się końcowe sortowanie. Całe zapytanie POST z formularza jest przesyłane do akcji *results*,
- *results* – przetwarza zapytanie POST z wyszukiwarki i wyświetla pasujące do kryteriów wyszukiwania wyniki – odpowiednio posortowane i opisane,

- *own_ webpage* – informacje dla studentów chcących sobie założyć własną stronę WWW na serwerach EiT,
- *webpages* – lista wszystkich stron WWW na serwerze EiT. Lista jest tworzona poprzez przeszukanie wszystkich katalogów domowych użytkowników pod kątem katalogu *public_ html*. Wszyscy, którzy mają taki katalog a ich loginy zgadzają się z loginami w bazie zostają wyświetleni. Dodatkowo, w rubryce „Opis strony domowej" pojawia się zawartość pliku *.webinfo* znajdującego się w katalogu ze stroną (o ile taki plik istnieje).

4.6.18 *SubjectController*

Kontroler wyświetlający informacje o przedmiocie.

- *method_ missing* – jest to specjalna metoda w Rubym. Jest ona zawsze wywoływana, gdy obiekt próbuje odwołać się do nieistniejącej metody. Jako pierwszy parametr przyjmuje jej nazwę. W tym przypadku, nazwa ta jest interpretowana jako akronim nazwy przedmiotu i jeśli zostanie odnaleziony w bazie, wszelkie informacje na temat tych zajęć są wyświetlane użytkownikowi.

4.7 Komponenty

Do poprawnego działania aplikacji, niezbędne okazało się stworzenie kilku komponentów – małych klas, stworzonych w jednym, określonym celu. Dzięki temu, zgodnie z regułą DRY (zob. rozdz. 3.6, str. 29) pomocne procedury biblioteczne są zaimplementowane jednokrotnie.

4.7.1 IdEncoder

IdEncoder to klasa krytyczna z punktu widzenia bezpieczeństwa aplikacji. Jej zadaniem jest kodowanie cyfr (zazwyczaj numerów *id* wierszy, stąd nazwa *IdEncoder*) na trudny do złamania ciąg znaków. Głównym celem takiego postępowania jest zabezpieczenie się przed wszelkiego rodzaju botami, które chodząc po stronie mogłyby inkrementować liczniki co 1 i w ten sposób powyciągać wszystkie informacje z bazy. Dodatkowo zabezpiecza to przed sprytnymi użytkownikami, którzy mając dostęp do materiału o *id* np. 3, a nie mając do 4, próbowaliby podmieniać cyfry w adresach URL. Po instalacji aplikacji należy w klasie IdEncoder ustawić dwie zmienne:

- *hash* – jest to liczba z zakresu <0;4294967295>, z którą *id* będzie poddane opearacji XOR na początku kodowania,

- *salt* – jest to dwudziestocyfrowa liczba, z której 10 cyfr będzie użyte do zaciemnienia wyniku w końcowym etapie kodowania.

Schemat kodowania jest przedstawiono we wzorach 4.2 — 4.4.

$$id_{XOR} = id \oplus hash \tag{4.2}$$

$$salt_M = salt_{MARKER} = id_{XOR} \ mod \ 10 \tag{4.3}$$

$$id_{ENC} = \left(\frac{salt \ mod \ 10^{20-salt_M} - salt \ mod \ 10^{20-salt_M-10}}{10^{20-salt_M-10}} * 10 + salt_M \right)_{(28)} \tag{4.4}$$

Liczba (28) we wzorze 4.4, oznacz konwersję wyniku na format dwudziestoósemkowy – dzięki czemu otrzymujemy w zakodowanym *id* zestaw liter i cyft. Rozkodowywanie odbywa się symetrycznie.

Dodatkowo, algorytm został tak pomyślany, że nawet znając schemat postępowania niemożliwe jest rozkodowanie liczby bez znajomości *hash* i *salt*.

Użycie klasy jest bardzo proste i zostało zilustrowane na list. 4.7.

```
IdEncoder.encode(id)          # zakoduj id
IdEncoder.decode(encoded_id) # zdekoduj id
```

Listing 4.7: Użycie polecenia IdEncoder

4.7.2 Pager

Klasa *Pager* jest wykorzystywana wszędzie tam, gdzie zwracana liczba wyników jest zbyt duża i umieszczenie ich na jednej stronie znacznie zmniejszyłoby czytelność, lub wręcz ją uniemożliwiło (wygenerowana strona byłaby tak duża, że ściągnięcie jej zajęłoby zbyt dużo czasu). Budowa została przedstawiona na list. 4.8.

```
def initialize params, results, params_key = :id, actual_page = nil,
                results_per_page = 10
  ...
end
```

Listing 4.8: Struktura konstruktora klasy *Pager*

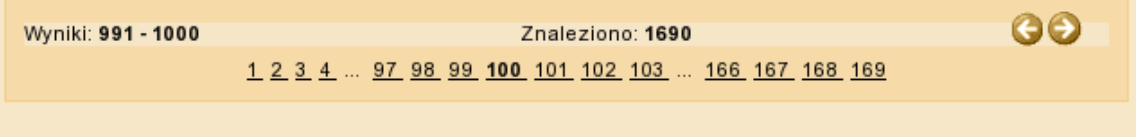
Gdzie parametry to:

- *params* – tablica params z kontrolera. Z racji tego, że komponenty są oddzielone od warstwy aplikacji, nie mają dostępu do domyślnych tablic Ruby on Rails (m.in. params), dlatego należy je przekazać. Jest to konieczne, aby w klasie Pager można było poprawnie budować odnośniki.
- *results* – zawiera obiekt modelu danych wyciągniętych bezpośrednio z bazy, które mają być podzielone na strony.

- *params_key* – domyślnie *:id*. Określa który parametr rutera Rails ma być interpretowany jako numer kolejnej strony z wynikami (czasami zdarza się, że *:id* jest już wykorzystane do czegoś innego, wtedy ta opcja się przydaje).
- *actual_page* – wymuszenie zwrócenia wyników dla określonej strony. Domyślnie brana jest strona z *params[params_key]* (lub pierwsza, jeśli podana wartość jest błędna lub nieistniejąca).
- *results_per_page* – określa ile wyników ma być zareprezentowanych na jednej stronie. Domyślnie jest to 10.

Dodatkowo Pager zawiera wiele przydatnych metod, użytecznych przy budowaniu widoków. Są to:

- *count* – zwraca liczbę stron,
- *array_of_pages* – zwraca pasek nawigacyjny klasy Pager. Są to cyfry z numerami stron i strzałkami „poprzednia", „następna" itd. (zobacz rysunek 4.6, gdzie *page_number* = 100, a *pages_padding* = 3. Dodatkowo na górze po lewej, efekt działania *results_range* z domyślnym separatorem.).



Rysunek 4.6: Przykład wykorzystania *array_of_pages*.

  Przyjmuje dwa parametry:

  - *page_number* – względem której strony ma być wyskalowany. Domyślnie jest to aktualna strona.
  - *pages_padding* – odnośniki do ilu stron wokół aktywnej mają zostać wyświetlone. Domyślnie trzy.

- *last_page* – zwraca numer ostatniej strony,
- *results* – zwraca wyniki w zakresie odpowiednim dla aktualnej strony,
- *all_results* – zwraca wszystkie wyniki (czyli tak naprawdę obiekt *results* przekazany do konstruktora),
- *results_count* – zwraca liczbę wszystkich wyników,
- *results_range* – zwraca pełny zakres dostępnych stron oddzielonych separatorem przekazanym jako parametr. Czyli np. „1 - 123", zobacz rysunek 4.6,

- *actual_page* – zwraca numer aktualnej strony,
- *next_page* – zwraca numer następnej strony,
- *previous_page* – zwraca numer poprzedniej strony.

Dodatkowo klasa Pager ma przygotowany, predefiniowany dla serwisu EiT widok wyświetlający nawigację, który można znaleźć w *app/views/_partials/_pager.rhtml* (zob. rozdz. 1.7, str. 11).

4.7.3 Imieniny

Klasa *NameDay* to bardzo prosta, statyczna klasa o jednej metodzie – *get*. Przyjmuje ona dwa parametry: dzień i miesiąc, a następnie zwraca gotowy łancuch znaków zawierający oddzielone przecinkiem imieniny osób w tym dniu.

4.7.4 Deklaracje

System deklaracji to najbardziej dynamiczna część serwisu z punktu widzenia aplikacji. Niektóre deklaracje stają się przestarzałe, czasami dochodzą nowe. Projektując system deklaracji, od początku trzeba było mieć to na uwadze. Oznacza to, że deklaracji nie można było po prostu „zaszyć" w kontrolerze. Dlatego też autor wybrał inne rozwiązanie – system deklaracji, jako system wtyczek. Każdy rodzaj deklaracji ma swój własny oddzielny komponent. Dzięki temu łatwo w przyszłości będzie dodawać nowe deklaracje, bez naruszenia integralności serwisu. Jest to również powód, dla którego zostały one szerzej opisane.

Pliki

Pliki do systemu deklaracji znajdują się w trzech miejscach:

- *components/declarations/* – tutaj znajdują się klasy deklaracji. Każdy rodzaj deklaracji ma swoją oddzielną klasę. Więcej szczegółów odnośnie klas znajduje się w dalszej części tego rozdziału.
- *app/views/deanery/declarations/* – widoki deklaracji przeznaczone dla studentów. To właśnie one są wyświetlane, w momencie kiedy student wypełnia deklarację. Należy trzymać się konwencji *nazwa deklaracji→nazwa widoku*. Jeśli widoków dla jednej deklaracji będzie więcej, informację identyfikującą należy dodawać po twardej spacji.
- *app/views/settings/declarations/* – analogicznie jak powyżej, tyle że widoki przeznaczone są dla prowadzących mających dostęp do ustawień deklaracji.

Klasa główna

Wszystkie klasy deklaracji powinny dziedziczyć z klasy *Declarations*, która znajduje się w *components/declarations.rb*. Klasa ta, dostarcza klasom potomnym następujące zmienne:

- *@logged_user* – obiekt zalogowanego użytkownika,
- *@declarations_subjects* – obiekt modelu *DeclarationsSubject* (tabela *declarations_subjects*, zob. rozdz. 4.3.1, str. 52) zawierający tylko przedmioty dotyczące deklaracji potomnej,
- *@merged_subjects* – początkowo pusta. Domyślnie powinna zawierać przedmioty podane w deklaracji przez użytkownika (jeżeli jest to konieczne). Wcześniej należy uruchomić specjalny skrypt, który usuwa wszystkie podejrzane lub błędne wpisy (zabezpieczenie przed fałszerstwem) – skrypt ten, to *merge_subjects*,
- *@flash_notice* – wiadomość, jaka ma zostać wyświetlona w czasie wypełniania deklaracji. Domyślnie jest pusta, ale nic nie stoi na przeszkodzie, aby klasa potomna ją ustawiła w razie potrzeby,
- *@template* – nazwa widoku, który ma zostać wyrenderowany. Aplikacja sama dobierze ścieżkę,
- *@declaration_name* – nazwa deklaracji (pole *head* w tabeli *declarations*, zob. rozdz. 4.3.1, str. 51),
- *@declaration_id* – numer *id* deklaracji.

Klasy potomne

Każda klasa potomna powinna mieć dwie metody – *execute* oraz *setup*. Obie powinny przyjmować jeden parametr – będzie to *params* z Ruby on Rails. Metoda execute wywołuje się w kontrolerze *DeaneryController*, jest więc przeznaczona dla studentów wypełniających deklaracje. Metoda *setup* wywoła się w kontrolerze *SettingsController*, jest więc przeznaczona dla prowadzących zarządzających deklaracjami. Najważniejsze o czym należy pamiętać w klasach potomnych, to ustawianie zmiennej *@template* z odpowiednim widokiem oraz o tym, aby później inicjalizacje klas potomnych dopisać do metod *declarations* w wyżej wymienionych kontrolerach.

Struktura bazy danych i relacji

Wszystkie informacje odnośnie deklaracji po stronie bazy danych zostały opisane w:

- Rozdz. 4.3.1, str. 51 (Tabela *declarations*)

- Rozdz. 4.3.1, str. 52 (Tabela *declarations_ subjects*)
- Rozdz. 4.3.2, str. 60

Po utworzeniu nowego komponentu należy dopisać odpowiednie deklaracje do tabeli *declarations* oraz ewentualnie utworzyć nowe kolumny w tabeli *declarations_ columns* i wypełnić je danymi początkowymi.

4.8 Przemapowanie hiperłączy

Ostatnim, ważnym problemem jaki należało rozwiązać był problem dotychczasowych odsyłaczy. Nowa aplikacja posiada zupełnie inaczej rozwiązany system odnośników. Nie są to już trzyliterowe, nic nie mówiące skróty, tylko pełne nazwy angielskie. Takie podejście ułatwia pracę robotom wsyszukiwarek, pozycjonuje wyżej stronę, ułatwia ludziom zapamiętanie adresu i przygotowuje możliwość przyszłego tłumaczenia serwisu. Niestety obecnie cała strona EiT jest zaindeksowana pod starymi adresami i uruchomienie nowej aplikacji skutkowałoby ogromną liczbą martwych odsyłaczy. Użytkownik wyszukiwarki po kliknięciu na takie hiperłącze zostałby oddelegowany w próżnię. Aby tego uniknąć stworzony został mapper do linków. Jest to po prostu zestaw regułek w pliku *config/routes.rb*, który tłumaczy stare linki na nowe.

Niestety same wpisy w *routes.rb* nie wystarczą, gdyż wtedy skończylibyśmy z nową treścią renderowaną pod starym adresem URL. Nie jest to zachowanie oczekiwane, gdyż w ten sposób stare odsyłacze nigdy nie zniknęłyby z wyszukiwarek. Dlatego trzeba jeszcze dodać przekierowanie – czyli zastosować kod HTTP 301 *Moved permanently*. Niestety system rutingu w Ruby on Rails nie przewiduje takiego zastosowania, dlatego trzeba posłużyć się pewnym trikiem. Mianowicie, wszystkie zapytania o stare adresy są wpierw kierowane do kontrolera *RedirectController*, a jako parametr podawany jest nowy adres. Kontroler ten, wysyła następnie kod 301 i przekierowuje użytkownika na odpowiednią stronę.

Rozwiązanie takie ma dwie zalety. Po pierwsze, użytkownik klikając w wyszukiwarce na odnośnik, znajduje się już na nowej stronie i od razu widzi w pasku adresu nowy link, dzięki temu nie przywiązuje się do starych oznaczeń. Po drugie, wyszukiwarki dużo szybciej usuwają stare linki i przeindeksowują je sobie na nowe, jeśli napotykają kod *Moved Permamently*.

4.9 Testowanie i instalacja

W ramach niniejszej pracy implementacja projektu została zakończona. Ruby on Rails posiada także wbudowaną funkcjonalność pozwalającą na kompleksowe testowa-

nie aplikacji. Zarówno na poziomie jednostkowym, jak i funkcjonalnym. Nie da się ukryć, że tworzenie testów bardzo pomaga w rozwoju aplikacji i chociaż jest bardzo pracochłonne w dłuższej perspektywie pozwala zaoszczędzić czas. Jednak aplikacja opisana w pracy obecnie jest mała, a całość kodu została napisana przez jednego programistę. Dopisanie testów zajęłoby sporo czasu. Autor jest w stanie samodzielnie kompleksowo ją przetestować, dlatego zrezygnował z pisania testów.

Instalacja i uruchomienie ogranicza się do przekopiowania całości kodu na docelowy serwer, który będzie posiadał wszystkie niezbędne komponenty (wymienione w podrozdziale Przygotowanie środowiska pracy, zob. str. 35). Następnie wystarczy skonfigurować odpowiednie połączenia z bazami w pliku *config/database.yml* (zob. rozdz. 4.4.1, str. 62), przygotować strukturę w nowej bazie danych (uruchamiając polecenie *rake db:migrate* – list. 4.9) i przekonwertować starą bazę na nową (polecenie *rake eit:db:convert*).

```
rake db:migrate
rake eit:db:convert
mongrel_rails start -p 80 -l ./log/mongrel.log -e production)
```

Listing 4.9: Kompletny zestaw poleceń, niezbędny do uruchomienia aplikacji

Ostatnim etapem przygotowania jest uruchomienie serwera *mongrel* w trybie produkcyjnym.

Rozdział 5

Zakończenie

Celem niniejszej pracy było stworzenie nowego serwisu dla Kierunku Elektroniki i Telekomunikacji. Cel ten został zrealizowany – aplikacja została ukończona. W rozdziałach 1, 2 i 3 zawarta została cała wiedza konieczna, aby zacząć tworzyć lub rozwijać aplikacje w środowisku Ruby on Rails. Rozdział nr 4 zawiera całą dokumentację niezbędną do obsługi danej aplikacji. Może też posłużyć jako wzorzec projektowy dla innych tego typu witryn.

Rozwiązanie Ruby on Rails zdecydowanie się sprawdziło jako narzędzie programistyczne. Cały proces tworzenia aplikacji skoncentrowany był tylko na problemach koncepcyjncyh. Nie było potrzeby koncetrowania się na aspektach stricte technicznych, takich jak obsługa bazy danych czy budowanie logiki widoków. Dzięki temu czas poświęcony na naukę języka zwraca się z nawiązką podczas tworzenia aplikacji.

Dodatkowo, tworząc kod w środowisku Ruby on Rails, mamy gwarancję (jeśli trzymamy się wytycznych) jego czytelności i elastyczności. Wszystkie kawałki kodu są na swoim miejscu i łatwo je zmienić nie naruszając struktury aplikacji. Dzięki temu, jeśli tylko programiści dbający o serwis będą przestrzegać reguł takich jak DRY czy Convention over Configuration, czytelność kodu nie ulegnie zmianie.

Pozostaje jeszcze pytanie o stabilność i wydajność. Odpowiedź przyjdzie po uruchomieniu serwisu, kiedy pojawią się na nim użytkownicy. Zaczną wtedy generować ruch, który będzie podstawą do wyciągnięcia odpowiednich wniosków. Ponieważ serwis EiT nie jest duży, a dodatkowo jest przeznaczony dla konkretnej grupy odbiorców, nie zachodziła potrzeba stosowania rozwiązań zwiększających wydajność takich jak memcached[1]. Nic nie stoi jednak na przeszkodzie, aby wykorzystać je w przyszłości.

[1] *Projekt memcached*, zob. http://www.danga.com/memcached/

Podziękowania

Serdeczne podziękowania dla dr inż. Krzysztofa Juszkiewicza, za pomoc i wiele cennych rad i wskazówek podczas realizacji tej pracy.

Podziękowania składam też Radosławowi Bułatowi, za wielokrotną pomoc w czasie pisania kodu.

Dziękuję Bartłomiejowi Głownii i Krzysztofowi Klusce za ogromną ilość ciekawych materiałów.

Podziękowania należą się też Marcinowi Pyle i firmie Leftbrain B.V. za udostępnienie lokalu i sprzętu, na którym bez przeszkód mogłem stworzyć tę pracę.

Bibliografia

[1] Dan Cederholm. *Bulletproof Web Design: Improving flexibility and protecting against worst-case scenarios with XHTML and CSS, Second Edition.* New Riders Press, 2007.

[2] Chad Fowler. *Rails Recipes.* Pragmatic Bookshelf, 2006.

[3] Dave Thomas Chad Fowler and Andrew Hunt. *Programming Ruby: The Pragmatic Programmers Guide, Second Edition.* Pragmatic Bookshelf, 2004.

[4] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master.* Addison-Wesley Professional, 1999.

[5] Patrick Lenz. *Build Your Own Ruby on Rails Web Applications.* SitePoint, 2007.

[6] Russ Olsen. *Design Patterns in Ruby.* Addison-Wesley Professional, 2007.

[7] Kevin Marshall Chad Pytel and Jon Yurek. *Pro Active Record: Databases with Ruby and Rails.* Apress, 2007.

[8] Leonard Richardson and Sam Ruby. *RESTful Web Services.* O'Reilly Media, Inc., 2007.

[9] Dave Shea. *The Zen of CSS Design: Visual Enlightenment for the Web.* Peachpit Press, 2005.

[10] Dave Thomas and David Heinemeier Hansson. *Agile Web Development with Rails, Second Edition.* Pragmatic Bookshelf, 2006.

[11] Jeffrey Zeldman. *Designing with Web Standards, Second Edition.* Peachpit Press, 2006.