Migracja serwisu WWW z techniki PHP na Ruby on Rails

Igor Rzegocki

2008

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

Spis treści

Rozdział 1

Wstęp

TODO

Rozdział 2

Technologie 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 technologiami są C#.NET, Java, Python, Ruby oraz PHP. Oczywiście nadal można wykorzystywać Perl czy C, ale dzisiaj mało kto wybiera te języki, ze względu na istnienie lepszych rozwiązań. 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 frameworki, które ułatwiają tworzenie i rozwój aplikacji.

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

2.1 Brama CGI

Na początku wszystkie strony internetowe były statyczne. To znaczy ich autorzy tworzyli osobny plik z kodem HTML dla każdej strony. Zmiana czegokolwiek wymagała manualnej 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]

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, te dane natomiast są wysyłane przez serwer do przeglądarki.

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

Co najważniejsze implementacja 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. Na-

[1] Źródło: `http://pl.wikipedia.org/wiki/CGI`

[2] Zobacz `http://www.w3.org/CGI/` i `http://hoohoo.ncsa.uiuc.edu/cgi/`

tomiast Perl jest językiem stworzonym do przetwarzania danych tekstowych, więc znakomicie nadaje się do generowania kodu HTML.

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

2.2 Java

Java jest darmową technologią rozwijaną pod kierunkiem firmy Sun Microsystems. Kod Javy jest kompilowany do tzw. bytecode'u, który następnie jest wykonywany przez maszynę wirtualną. Jej głównymi cechami są: pełna obiektowość, niezależność od architektury (raz stworzony bytecode może być wykonywany na wirtualnej maszynie uruchomionej na dowolnym systemie operacyjnym), sieciowość i obsługa programowania rozproszonego, niezawodność i bezpieczeństwo.

Wczesne wersje Javy 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 klienta. Jedną z alternatyw jest open sourceowy framework Struts[3], wspomagany przez także open sourceową bibliotekę Hibernate[4] ułatwiającą obsługę bazy danych. Popularny jest także framework Spring[5]. Java posiada także swój serwer WWW Tomcat[6]. Budowę aplikacji ułatwiają bardzo dobre zintegrowane środowiska programistyczne (IDE) Eclipse lub NetBeans dedykowane dla tego języka.

Dużą zaletą Javy jest to, że technologia 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. Java jest językiem kompleksowym jeżeli chodzi o możliwości zastosowania. Dlatego sprawdza się wszędzie tam gdzie jeden duży system, składający się z mniejszych aplikacji i ma za zadanie pełnić wiele ról przy jednoczesnej integracji. Taki przykładem może być system ERP IFS[7], który jest w całości napisany w Javie. Zarówno cała logika biznesowa, klient okienkowy prze-

[3]Zobacz `http://struts.apache.org/`
[4]Zobacz `http://www.hibernate.org/`
[5]Zobacz `http://www.springframework.org/`
[6]Zobacz `http://tomcat.apache.org/`
[7]Zobacz `http://www.ifsworld.com/pl/`

znaczony dla stacji roboczych, jak i część webowa w której zrealizowany jest moduł zamówień dla klientów firmy.

- Plusy:
  - stabilność i wydajność,
  - cena (za darmo lub open source),
  - międzyplatformowość,
  - mnogość możliwych rozwiązań.
- Minusy:
  - mnogość możliwych rozwiązań,
  - potrzeba przekompilowania kodu po każdej zmianie,
  - rozwój aplikacji w Javie jest stosunkowo pracochłonny.

2.3 C#.NET

C# jest odpowiedzią firmy Microsoft na Javę Sun'a. Podobnie jak Java jest obiektowym językiem wysokopoziomowym, kompiluje się do języka Common Intermediate Language (CIL) (podobnie jak bytecode Javy), który następnie wykonywany jest w odpowiednim środowisku uruchomieniowym. Do budowy aplikacji webowych wykorzystuje się framework .NET[8]. Framework .NET może współpracować również z innymi językami programowania takimi jak: Visual Basic, J#, Jscript.Net, COBOL, Fortran, Lisp, Pyton, Perl i kilka innych. .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.

Podobnie jak Java, framework .NET z językiem C# ma także szersze spektrum zastosowań. Do znanych rozwiązań należy system obsługi banku PKO BP S.A. czy Getin BANK S.A zbudowane przez krakowską firmę Vsoft[10]. Obydwa systemy

[8]Zobacz `http://www.microsoft.com/net/`
[9]Zobacz `http://www.iis.net/`
[10]Zobacz `http://www.vsoft.pl/`

obsługują większą cześć procesów w banku i posiadają także moduł banokości internetowej. Dobrym przykładem zastosowań webowych jest serwis społecznościowy MySpace[11], który posiada ponad 100 milionów użytkowników na całym świecie.

- Plusy:
  - pełne wsparcie techniczne ze strony Microsoftu,
  - sprawdzona i stabilna technologia.
- Minusy:
  - cena,
  - całkowita zależność od platformy Microsoft.

2.4 PHP

Język PHP[12] (ang. PHP: Hypertext Preprocessor, PHP: przetwornik hipertekstu) był od początku projektowany do wykorzystania na potrzeby aplikacji webowych. Pierwotnie PHP był raczej zestawem narzędzi niż językiem, dopiero od wersji 3 można mówić o języku programowania. PHP jest językiem interpretowanym, to znaczy przy każdym uruchomieniu interpreter czyta kod źródłowy i wykonuje go. Dzięki temu każda zmiana w aplikacji ma natychmiastowe skutki. 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 (open source), a jego interpreter jest dostępny na wiele platform. Do PHP istnieje wiele frameworków (np. Zend Framework, Zoop Framework, Symfony, Prado, CakePHP etc.), wiele bibliotek do obsługi baz danych (np. ADOdb, Propel, PEAR::DB ect.). Jednak wydaje się, że PHP nie ma przed sobą przyszłości.

- 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

[11] Zobacz `http://www.myspace.com/`
[12] Zobacz `http://www.php.net/`

Skrypty PHP są serwowane najczęściej za pomocą serwera Apache2.

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

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

2.5 Python

Python[13] to interpretowany język programowania stworzony jako następca języka ABC[14]. Jest to język zaprojektowany z myślą o jak największej produktywności oraz o 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:

- Paradygmat programowania strukturalnego
- Paradygmat programowania funkcjonalnego
- Paradygmat programowanie obiektowego

[13]Zobacz `http://python.org/`
[14]Zobacz `http://idhub.com/abc/`

Kolejną jego cechą jest rozszerzalność. Zamiast wbudowywać wszystko w rdzeń, można łatwo dokładać napisane w C lub C++ moduły. Dzięki temu sam środek języka 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 frameworków – najpopularniejsze to Django[16], Zope[17], TurboGears[18] i Pylons[19].

- Plusy:
  - elastyczność i rozszerzalność,
  - wydajność,
  - wiele frameworków.
- Minusy:
  - duża swoboda programowania powoduje bałagan w kodzie,
  - wymóg stosowania wcięć znacznie utrudnia stworzenie dobrego systemu szablonów.

2.6 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 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 frameworku Ruby On Rails (dosł. rubin na szynach). Rails posiadają wszystko co jest niezbędne do budowy aplikacji webowych i znakomicie nadają się do

[15] Zobacz `http://www.artima.com/intv/pythonP.html`
[16] Zobacz `http://www.djangoproject.com/`
[17] Zobacz `http://www.zope.org/`
[18] Zobacz `http://www.turbogears.org/`
[19] Zobacz `http://www.pylonshq.com/`
[20] Porównaj http://www.informit.com/articles/article.aspx?p=18225&rl=1

bardzo szybkiej budowy małych aplikacji a także budowania prototypów aplikacji. Interpretery Rubiego istnieją na każdą platformę. Jedyną wadą języka zdaje się być jego wydajność (podobnie jak we wczesnych wersjach Javy), 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 CGI lub rozszerzenia mod_ruby.

Największą aplikacja napisaną całkowicie w technologii RoR jest Basecamp[21]. Basecamp jest aplikacją wspomagającą zarządzanie projektami, która obecnie obsługuje ponad milion użytkowników. Drugą duża aplikacją jest Twitter[22]. Twitter jest z kolei platformą służącą do „mikroblogowania" (czyli pisania wpisów nie dłuższych niż 140 znaków). Obecnie cała społeczność Ruby on Rails śledzi z zapartym tchem dzieje Twittera, gdyż ma on ogromne problemy wydajnościowe. Od tego jak je rozwiąże, zależy jak inwestorzy komercyjni ocenią przydatność samego frameworka.

Obecnie RoR jest wykorzystywany głównie do budowy małych aplikacji (takich jak ta prezentowana w tej pracy), oraz prototypowania aplikacji, które docelowo zostaną przepisane na bardziej wydajną platformę.

- Plusy:
  - darmowy
  - pełna obiektowość
  - duże perspektywy rozwoju,
  - przejrzystość kodu,
  - bardzo prosty w nauce
  - duża aktywna społeczność zawsze chętna do pomocy
  - pisanie w Rubym to naprawdę świetna zabawa
- Minusy:
  - niska wydajność
  - brak znanych udanych wdrożeń

[21]Zobacz http://www.basecamphq.com/
[22]Zobacz http://www.twitter.com/

Rozdział 3

Technologie po stronie klienta wykorzystywane w aplikacjach internetowych

3.1 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 URL (ang. Uniform Resource Locator, jednoznaczny wskaźnik zasobu),
- *HEAD* - pobiera informacje o tym czy dana informacja istnieje

[1] Źródło: `http://pl.wikipedia.org/wiki/HTTP`

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

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

3.2 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.

„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]

[2] Źródło `http://pl.wikipedia.org/wiki/Hipertekst`
[3] Źródło `http://pl.wikipedia.org/wiki/HTML`

Wraz z rozwojem sieci WWW pojawiła się potrzeba rozwoju języka HTML aby posiadał on możliwość dołączania do testów 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 i postanowiło dostosować język do XML (ang. eXtensible Markup Language). W wyniku powstał XHTML, dla którego istnieje tryb zgodności z HTML, który to umożliwia wyświetlenie 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], a dla niego arkusze 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.

3.3 CSS

„Kaskadowe arkusze stylów (ang. Cascading Style Sheets, CSS) to język służący do opisu formy prezentacji (wyświetlania) stron WWW. 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]”

CSS jest wykorzystywany aby wspomóc czytniki 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

[4]Zobacz `http://www.w3c.org/`
[5]Zobacz `http://www.w3.org/TR/xhtml11/`
[6]Zobacz `http://www.w3.org/TR/REC-CSS2/`
[7]Zobacz `http://www.w3.org/html/wg/html5/`
[8]Zobacz http://www.w3.org/People/howcome/p/cascade.html

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). 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 druki). CSS określa też priorytety z jakimi należy nakładać kolejne właściwości. Najnowszą wersją CSS jest obecnia wersja 2.1 – trwają jednak prace, nad wersją trzecią[9].

3.4 JavaScript

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

Nazwa nasuwa skojarzenie z językiem Java, jednak 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 technologią 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, kliknięcie myszką, naciśnięcie klawisza na klawiaturze etc.). Połączenie HTML ze skryptami języka JavaScript nazywane jest Dynamic HTML (DHTML), czyli dynamiczny HTML, aby uwydatnić różnicę ze statycznymi stronami HTML. Przełomem w DHTML było udostępnienie obiektu XMLHttpRequest (XHR). XHR umożliwia wysyłanie żądań HTTP z poziomu języka JavaScript już po załadowaniu się 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 udostęp-

[9]Zobacz `http://www.w3.org/Style/CSS/current-work`

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

[11]Zobacz `http://www.w3.org/DOM/`

nia XHR. 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. AJAX 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).

3.5 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, podcasty etc. Dokument RSS zawiera zazwyczaj podsumowanie przypisanej mu strony WWW lub jej pełną zawartość.

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. Oznacza to więc 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 nieprzeczytana.

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 „rodziny" RSS:

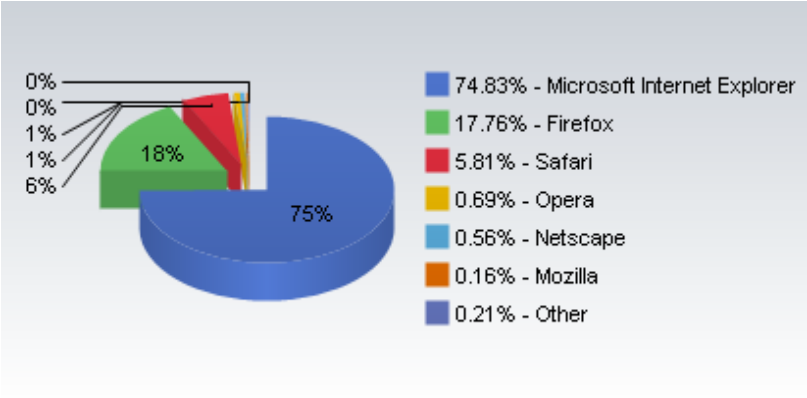
- Really Simple Syndication (RSS 2.0)
- RDF Site Summary (RSS 1.0 i RSS 0.90)
- Rich Site Summary (RSS 0.91).

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

3.6 Przeglądarki Internetowe

Przeglądarka internetowa jest to aplikacja, mająca na celu pobieranie i wyświetlanie 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 etc.), KHTML/WebKit (np. Konqueror, Safari), oraz Presto (Opera). Z ciekawostek 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 3.1



Rysunek 3.1: Udział procentowy w rynku przeglądarek internetowych.
Źródło: http://marketshare.hitslink.com/report.aspx?qprid=0

Niestety silniki nie są ze sobą ze sobą kompatybilne, każdy trochę inaczej interpretuje HTML, CSS, JavaScript, nie wszystkie mają wbudowane najnowsze technologie w tym samym stopniu. Największe trudności sprawia programistom 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 (wersja 7) pojawiła się dopiero pod koniec 2006 roku. Zanim wersja 6 zniknie z rynku minie

[13]Zobacz `http://en.wikipedia.org/wiki/Browser_wars`

jeszcze kilka lat, w czasie których rozwój aplikacji po stronie klienta będzie ograniczony możliwościami tej przeglądarki.

Rozdział 4

Koncepcje spotykane w aplikacjach internetowych

4.1 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. W MVC 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 etc. Kontroler spaja wszystko w całość pośrednicząc między Modelem a Widokiem i zajmując się sterowaniem.

4.1.1 Krótka Historia

Wzorzec został po raz pierwszy opisany w 1979 r.[1] przez Trygve Reenskaug wtedy pracującegp 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 wariacji na temat koncepcji, dość wspomnieć o wzorcu Model-view-presenter, który powstał początkiem lat 90 i został zaprojek-

[1]Zobacz `http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html`
[2]Zobacz `http://www.smalltalk.org/`
[3]Zobacz `http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html`

towany jako następca MVC. Nie zmienia to faktu, że MVC trzyma się mocno i wciąż znajduje szerokie zastosowanie.

4.1.2 Opis wzorca

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

- *Model* - Specyficzna dla każdego projektu reprezentacja informacji, na których działa aplikacja. Jest to DSL (ang. Domain Specific Language, zob. r. 4.4 str. 28), który opisuje i operuje na surowych danych (np. obliczenie czy dzisiaj użytkownik obchodzi urodziny, kosztu wszystkich towarów w koszyku etc.). Wiele aplikacji wykorzystuje ściśle określony mechanizm przechowywania danych (w przypadku aplikacji internetowych, prawie zawsze jest to baza danych). 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. MVC nie definiuje ścisłej relacji ilościowej pomiędzy Modelami a Widokami, co oznacza, że jeden Model może obsługiwać wiele widoków jak i jeden Widok może otrzymywać dane od wielu Modeli.

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

4.1.3 Zastosowanie w aplikacjach internetowych

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 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 MVC jest dostarczany w wielu frameworkach, z których każdy ma odmienny pomysł na jego implementację, generalna zasada jest taka sama:

1. Użytkownik wykonuje jakąś akcję w UI (np. wciska guzik).
2. Kontroler obsługuje nadchodzące wywołanie, często wykorzystując jakąs 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 nic nie wie o Widoku.
5. UI oczekuje na kolejną akcję, co rozpoczyna cały cykl od początku.

Rozdzielając Modele od Widoków, MVC znacznie zmniejsza skomplikowanie architektury programu i zwiększa przejrzystość i elastyczność.

4.2 Object-relational mapping

Mapowanie obiektowo-relacyjne (z ang. Object-relational mapping) jest techniką programistyczną stosowaną do konwersji danych przy niekompatybilnych ze sobą relacyjnych baz danych i 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 rozwiązanie ORM, niemniej sporo programistów wciąż tworzy swoje rozwiązania.

4.2.1 Opis problemu

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 telefonów i kilka adresów (zameldowania, zamieszkania etc.). Obiekt reprezentujący taką osobę miałby kilka „slotów", które zawierałyby kolejno, dane osobowe, listę telefonów i listę adresów. Oczywiście, lista telefonów może z kolei zawierać obiekty numerów telefonów (z osobnym slotem, na operatora, numer kierunkowy etc.), 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 przenaszalne między kodem a bazą danych.

4.2.2 Implementacje

Najpowszechniej stosowanym rozwiązaniem jest relacyjna baza danych, która poprzedziła narodziny 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.

ORM powinien systemacznie przewidywać jakie tabele będą potrzebne do zapisania określonych obiektów i generować odpowiednie zapytania SQL. Różnice między sposobem prezentacji 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. 4.5, str. 30) dla bardziej skomplikowanych relacji,
- Uproszczenie i spójność kodu konieczna przy szybkim tworzeniu aplikacji,
- Zarządzanie i elastyczność kodu.

Prawdziwą wartością ORM jest przedewszystkim oszczędność czasu i uproszczenie kodu (wszystkie skomplikowane operacje bazodanowe ORM załatwia za programistę). Dodatkowo ORM zwiększa wydajność i skalowalność oraz minimalizuje

problemy niekompatybilności architektur (dobry ORM obsługuje większość obecnie stosowanych systemów baz danych).

Powstało sporo aplikacji ORM, które dostarczając zestaw klas i bibliotek autmatyzują proces mapowania i odciążają programistę. Po podaniu im listy tabel w bazie, same wygenerują odpowiednie obiekty i relacje 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 zmapowanie wszystkich możliwych kombinacji zapytań. Wszystkie 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).

4.2.3 Zastosowanie w aplikacjach internetowych

Koncepcja ORM jest wykorzystywana we wszystkich nowoczesnych frameworkach webowych, ponieważ ściąga z programisty konieczność dbania o spójność danych w bazie, utrzymywania i rozłączania połączeń etc. Nie bez znaczenia jest też wsparcie dla różnych architektur bazodanowych. Często dzieje się tak, że po przeniesieniu aplikacji na nowy serwer okazuje się, że baza danych zmieniła się. Wtedy wystarczy tylko jedna zmiana w pliku konfiguracyjnym, a całość powinna zadziałać automatycznie.

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 niespójności i nieścisłości – wszyscy trzymają się jednej konwencji.

4.3 Representational State Transfer

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 na rozprawie doktorskiej, którą przeprowadzał Roy Fielding[4] – jeden z głównych autorów specyfikacji protokołu HTTP. Zwroty te, szybko rozpowszechniły się wśród sieciowej społeczności.

REST ściśle odnosi się do założeń sieciowych, które określają jak zasoby 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. 4.4, str. 28) poprzez protokół HTTP wraz z dodatkową warstwą wiadomości, taką jak SOAP czy też kontrolą sesji wykorzystującą „ciasteczka". Oba te określenia mogą budzić konflikt, jak i się uzupełniać. 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 pewnego zakłopotania w technicznych dyskusjach.

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

4.3.1 Założenia

Najważniejszym założeniem w REST jest istnienie zasobów (źródeł określonej informacji), z których każdy może być zaadresowany używając 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 lężąc na tym okręgu definiują go jednoznacznie.

[4]Zobacz http://www.ics.uci.edu/ fielding/pubs/dissertation/rest_arch_style.htm

Orędownicy REST utrzymują, że skalowalność i rozwój sieci są bezpośrednią przyczyną kilku kluczowych założeń:

- Stany i funkcjonalności aplikacji są podzielone między zasoby,
- Każdy zasób jest unikalnie adresowalny używając uniwersalnej składni wykorzystywanej do tworzenia odnośników,
- Wszystkie zasoby dzielą jednolity interfejs do transportu 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,
  - Cacheowalność,
  - Warstwowość.

4.3.2 Zastosowanie w aplikacjach internetowych

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

HTML może zawierać JavaScript i applety aby wspierać kod na żądanie, oraz posiada wbudowaną obsługę hyperlinków.

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

Najważniejszymi metodami HTTP są POST, GET, PUT i DELETE (patrz rozdz. 3.1, str. 14). Są one często porównywane do CREATE, READ, UPDATE i DELETE (CRUD, patrz rozdz. 4.5, str. 30), które to pojęcia mają zastosowanie w bazach danych.

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

Tablica 4.1: Porównanie zapytań HTTP z koncepcją CRUD

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

Ta bezstanowość może być naruszona w HTTP, kiedy 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 tak naprawdę pozwala tylko na użycie zapytań typu „GET". Dodatkowo jest jeszcze dostępny „POST" dzięki formularzom. Pozostałe metody HTTP nie są wspierane ani przez HTML 4.01 ani XHTML 1.0.[5]

4.4 Domain Specific Language

DSL (z ang. Domain Specific Language, język określonego przeznaczenia) to język zaprojektowany w ściśle określonym celu. W przeciwieństwie do języków ogólnego przeznaczenia takich jak Java czy C, których zadaniem jest uniwersalność i możliwość zastosowania w jak największej liczbie przypadków, DSLe są projektowane tylko w jednym, ściśle określonym celu. Przykładem 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. Powodem tego jest znaczny wzrost mocy obliczeniowej komputerów oraz elastyczność nowych języków generalnego przeznaczenia, która umożliwia pisanie własnych DSL.

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

4.4.1 Przykłady zastosowań

Można wykazać conajmniej kilka przykładów zastosowań dla DSL:

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

2. 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. DSL 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 zrozumieć i przetworzyć. Generalnie, taki DSL można rozumieć jako język programowania szczególnego przeznaczenia napisany w języku programowania ogólnego przeznaczenia.

4. DSL, które są częścią aplikacji w której pracują – 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 DSL może być używane w kilku powyższych przypadkach naraz.

4.4.2 Zastosowanie w aplikacjach internetowych

W aplikacjach internetowych DSL znalazły szerokie zastosowanie przy zadaniach często powtarzających się, takich jak na przykład tworzenie modeli, testy czy obsługa baz danych. Nie można również zapomnieć o najważniejszym z DSL z punktu widzenia WWW – o SQL. Bez niego istnienie większości serwisów internetowych byłoby praktycznie niemożliwe. DSL świetnie też się sprawdzają w całej otoczce okołoaplikacyjnej. Używając ich, 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 developerskiego na produkcyjny, jak np.:
  - Kompresja i obfuskacja (zaciemnienie) plików JavaScript
  - Kompresja CSS
  - Usuwanie informacji o gammie 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 cacheowania plików przez przeglądarkę)
  - Czyszczenie różnych pozostałości developerskich (stare, skompilowane pliki szablonów, logi, etc.)

4.5 Create, Read, Update and Delete

CRUD (Create, read, update and delete – utwórz, odczytaj, popraw i skasuj) są to cztery podstawowe funkcje dotyczące operacji na danych, najważniejszej czynności w całej informatyce. Czasami w skrócie tym, zamiast „read" używa się „restore" a zamiast „delete" – „destroy". Często jest stosowany przy opisywaniu operacji na bazach danych lub przy opisie interfejsów.

4.5.1 CRUD w relacyjnych bazach danych

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

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

Tablica 4.2: Mapa akronimu CRUD na zapytania SQL

odnosi się do relacyjnych baz danych, często jest też stosowany w odniesieniu do obiektorych baz danych, XMLowych baz danych, plików tekstowych, nośników informacji etc.

4.5.2 Zastosowanie w aplikacjach internetowych

W aplikacjach internetowych, CRUD odnosi się do samego 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, też jest podwaliną każdego dobrego frameworka, gdyż zautomatyzowanie jej, znacznie skraca czas tworzenia programu.

4.6 Don't Repeat Yourself

DRY (z ang. Don't Repeat Yourself, nie powtarzaj się) jest filozofią, której celem nadrzędnym jest zredukowanie duplikatów. Występuje głównie w informatyce. DRY zakłada, że ta sama informacja nie powinna być powielana w ramach jednego projektu, gdyż taka duplikacja znacznie utrodnia zmiany, zmniejsza czytelność i prowadzi do niespójności. Don't Repeat Yourself jest głównym przykazaniem książki *The Pragmatic Programmer*[6], 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 dokumentacje"[7]. 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.

4.6.1 Kiedy DRY może się nie sprawdzić

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

[6]TODO: BIBLIOGRAFIA

[7]Wywiad Billa Vennersa z Dave Thomasem *Orthogonality and the DRY Principle*, zob. `http://www.artima.com/intv/dry.html`

- W małych projektach, albo prostych skryptach, wysiłek jaki trzebaby było włożyć aby trzymać się reguły DRY, byłby znacznie większy niż proste napisanie dwóch kopii tej samej danej w różnych miejscach,

- Wymuszanie DRY 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, bez wykorzystywania DRY. Dla przykładu, bardzo dobrym nawykiem jest tworzenie trzech środowisk dla jednej aplikacji – produkcyjnego, testowego i deweloperskiego. 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 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.

4.6.2 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 desktopu – 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 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 w stanie szybko przystosować się do zmiennych warunków rynkowych.

4.7 Convention over Configuration

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

Sformułowanie samo w sobie oznacza, ze programista musi rozpatrywać tylko niekonwencjonalne aspekty aplikacji. Dla przykładu, jeśli dana jest klasa *Sprzedaż* obsługująca model, to przypisana jej tabela jest domyślnie również 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.

4.7.1 Geneza

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

Dobrym przykładem jest tutaj ORM (zob. rozdz. 4.2, str. 23) 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 deweloperów Hibernate i narzucona użytkownikom, tj. programistom, którzy używają tego frameworka. Oczywiście nic na siłę i jeśli któryś programista potrzebuje, aby nazwy się różniły, zawsze może sobie ustalić wszystkie wyjątki w odpowiednim pliku konfiguracyjnym.

4.7.2 Zastosowanie w aplikacjach internetowych

Convention over Configuration, jest koncepcją stosunkowo młodą w światku internetowym, ale z miejsca zdobyła sobie rzesze zwolenników. Wynika to przedewszystkim z tego, że upraszcza wiele aspektów i sprawia, że z każdym nowym projektem de-

weloperzy 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 5

Aplikacja

Serwis WWW zadany w temacie pracy był od początku z góry ustalony. Jest to strona Katedry Elektroniki i Telekomunikacji[1]. Motywem przewodnim 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. Dlaczego więc nie zdecydowane się na przepisanie serwisu od nowa, również w PHP? Głównym powodem była „samoopisywalność" kodu stworzonego w Ruby on Rails. Dzięki filozofii DRY (zob. rozdz. 4.6, str. 31) oraz Convention over Configuration (zob. rozdz. 4.7, str. 33) każdy kawałek kodu ma ściśle określone miejsce w aplikacji. Dodatkowo sama struktura plików projektu Railsowego jest klarowna i przejrzysta. Zawsze w tym samym miejscu przechowywane są modele, widoki, biblioteki, kontrolery itd. Dzięki temu, kiedy autor niniejszej pracy przekaże serwis w ręce nowego programisty, już na starcie będzie on doskonale wiedział, gdzie czego szukać. Dodatkowo czytelność kodu Ruby i łatwość tworzenia DSL (zob. rozdz. 4.4, str. 28) sprawia, że taki kod od razu przekazuje „co autor miał na myśli". I stąd właśnie bierze się jego samoopisywalność.

5.1 Opis funkcjonalny aplikacji

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

- System aktualności i najświeższych informacji dla studentów,

[1] Zobacz. http://www.eit.agh.edu.pl/

- Wszelkie informacje pochodzące z dziekanatu, m.in. informacje o:
  - planach zajęć,
  - praktykach,
  - przedmiotach obieralnych,
  - zajęciach w języku angielskim,
  - międzynarodowym programie edukacyjnym,
  - wydarzeniach obowiązkowych w kalendarzu studenta (ferie, sesje etc.).
- System zarządzania materiałami i ocenami,
- Informacje o prowadzących,
- Wszelkie dane dotyczące studiów podyplomowych.

Dodatkowo należało jeszcze zatroszczyć się o całą warstwę administracyjną. Grupa ludzi opiekujących się stroną (zwanych dalej Administratorami) powinna mieć specjalne uprawnienia, do dodawania informacji, usuwania niepotrzebnych treści, zarządzania studentami (z poziomu systemu) itd. Dodatkowo, wszyscy prowadzący powinni mieć możliwość dodawania aktualności i materiałów.

Wymienione powyżej funkcjonalności autor tej pracy wspólnie z prowadzącym uznał za najważniejsze i umieścił je w serwisie. Stara strona EiT miała jeszcze dodatkowo forum i katalog linków. Autor wspólnie z opiekunem, uznali je jednak za zbyteczne. W obu przypadkach głównym powodem było znikome zainteresowanie studentów. Katalog linków nie sprawdził się, ponieważ tak naprawdę wszystkie informacje były dostępne z poziomu serwisu i nie były potrzebne żadne referencje do innych witryn. Forum z kolei (zwane Hydeparkiem) przestało pełnić swoją pierwotną funkcję. W dzisiejszych czasach istnieją setki serwisów pozwalające założyć darmowe forum jednym kliknięciem myszki. I właśnie na takie biuletyny przenieśli się studencji, tworząc fora swoich roków 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 ludzi technicznych, którzy doskonale orientują w świecie nowoczesnych technologii. Najmniejsze niedopatrzenie może się zakończyć kompromitacją serwisu. 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 witryny był szczegółowo rozważany pod tym kątem. Nie bez znaczenia jest też fakt, że 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, która linijka w którym skrypcie go wywołała. Zamiast tego, wyświetli się informacja dla użytkownika, że „coś poszło nie tak" 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 jakąś wartościową informacją samo w sobie nie jest.

Należy też pamiętać o interfejsie użytkownika. 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 menu nie odświeża strony tylko od razu rozwija listę podmenu. Zmniejsza to znacznie ilość zapytań (wcześniej student musiał otworzyć nową stronę, tylko po to, żeby zobaczyć rozwinięte menu i przejść do kolejnej strony) oraz skraca czas interakcji użytkownika ze stroną. A wszystko to, dzięki odrobinie JavaScriptu.

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

5.2 Przygotowanie środowiska pracy

Po przygotowaniu założeń należało wybrać środowisko pracy.

[2]Zobacz `http://mongrel.rubyforge.org/`

5.2.1 Ruby i pochodne

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

Slackware 12.0 ma interpreter Ruby w standardzie (przy maksymalnej instalacji). W razie jego braku, należy pobrać odpowiednią paczkę – Ruby 1.8.6[3]. Do tego będą jeszcze protrzebne RubyGems 0.9.2[4]. Ponieważ Slackware nie obsługuje zależności, w razie problemów należy doinstalować brakujące biblioteki.

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

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,
- Mongrel.

Aby zainstalować wymagane gemy, z poziomu powłoki należy wykonać polecenia: Polecenia te, zainstalują pakiety wraz wszystkimi zależnościami. Po ich wykonaniu, zadane aplikacje są gotowe do użycia.

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

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

Rysunek 5.1: Lista poleceń niezbędnych do zainstalowania wymaganych gemów

5.2.2 Dodatkowe pakiety

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

MySQL powinno być standardowo w Slackware. Jeśli go nie ma, należy ściągnąć odpowiednią paczkę[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].

Aby baza danych mogła działać poprawnie należy zainicjalizować katalog przechowywujący dane i stworzyć tabele systemowe niezbędne do funkcjonowania bazy. Dodatkowo, dobrze jest ustawić sobie, aby daemon bazy danych uruchamiał się przy każdym starcie systemu. Wszystko to osiągniemy, wykonując poniższe polecenia.

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

Rysunek 5.2: Lista poleceń niezbędnych do zainicjalizowania MySQL

Iconv nie wymaga żadnych dodatkowych przygotowań.

5.2.3 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

[5]Zobacz `http://www.mysql.com/`

[6]Zobacz `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]Zobacz `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`

bazą danych oraz przeglądarkę wraz z zestawem dodatków wspomagających pracę z gotowym kodem.

Środowisko programistyczne

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

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

Jako dodatkowy bonus, autor zaleca zainstalowanie wtyczki do NetBeans – *Extra Ruby Color Themes* (Zakładka *Tools→Plugins*), a następnie włączenie schematu kolorów *Ruby Dark Pastels* (Zakładka *Options→Fonts & Colors*). Jest to schemat kolorów wzięty z TextMate, dużo czytelniejszy i mniej męczący wzrok.

Aplikacje wspomagające pracę z bazą danych

Aby nie musieć wykonywać wszystkich niezbędnych działań na bazie z domyślnego klienta tekstowego, zalecane jest zainstalowanie graficznych narzędzi ułatwiających obsługę bazy. Te narzędzia to:

- MySQL Query Browser[14] - pakiet MySQL-Query-Browser[15],
- MySQL Administrator[16] - pakiet mysql-administrator[17],

[10]Zobacz `http://macromates.com/`
[11]Zobacz `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`
[14]Zobacz `http://www.mysql.com/products/tools/query-browser/`
[15]Do pobrania z `http://linuxpackages.telecoms.bg/Slackware-10.1/X11/MySQL-Query-Browser/mysql-query-browser-1.1.12-i486-1JL.tgz`
[16]Zobacz `http://www.mysql.com/products/tools/administrator/`
[17]Do pobrania z `http://linuxpackages.telecoms.bg/Slackware-10.2/X11/`

- DBDesigner[18] - do pobrania z `http://downloads.mysql.com/DBDesigner4/DBDesigner4.0.5.4.tar.gz`

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

Przeglądarki internetowa i dodatki

Z powodu niedoskonałości silników renderujących przeglądarek (zob. rozdz. 3.6, str. 3.6), podczas pisania pracy, autor musiał korzystać aż z czterech przeglądarek internetowych, aby przetestować stronę pod czterema najpopularniejszymi silnikami renderującymi:

- *Trident* – Microsoft Internet Explorer 6.0 (pod Linuksem dostępny, dzięki pakietowi *ies4linux*[19])
- *Gecko* – Mozilla Firefox 2.0.0.14[20]
- *Presto* – Opera 9.50[21]
- *WebKit* – Konqueror 3.5.8[22]

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

- *Web Developer*[23] – pasek narzędzi, który posiada wiele opcji i udogodnień (np. zarządzanie ciasteczkami, zaznaczanie wybranych elementów dokumentu, podgląd bieżącego źródła, walidację HTML/CSS etc.).
- *FireBug*[24] – 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

`mysql-administrator/mysql-administrator-1.1.6-i586-1gds.tgz`

[18] Zobacz `http://www.fabforce.net/dbdesigner4/`
[19] Zobacz `http://www.tatanka.com.br/`
[20] Zobacz `http://www.getfirefox.com/`
[21] Zobacz `http://www.opera.com`
[22] Zobacz `http://www.konqueror.org`
[23] Zobacz `http://chrispederick.com/work/webdeveloper/`
[24] Zobacz `http://www.getfirebug.com/`

wykonywania instrukcji tego języka i obserwacji ich rezultatów, debugowanie skryptów, przeglądanie elementów DOM, a także obserwowanie wywołań XMLHttpRequest (zob. rozdz. 3.4, str. 17).

- *ColorZilla*[25] – rozszerzenie bardzo użyteczne podczas projektowania wyglądu strony i tworzenia arkuszy CSS. Udostępnia ono „pipetę" (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 layoutu.

- *HTML Validator*[26] – dodatek, który rozszerza nam podgląd źródła dokumentu o numerację linii i walidator dokumentu HTML (na bazie silnika TIDY[27]). Dzięki temu, od razu widać, gdzie popełniono błąd w strukturze dokumentu (niedomknięty tag, brak cudzysłowa przy wartości, etc.).

- *LiveHTTPHeaders*[28] – narzędzie służące do podglądu nagłówków HTTP wysyłanych i odbieranych podczas każdego zapytania. Świetnie się sprawdza, w przypadku błędnych przekierowań, kiedy tak naprawdę nie wiemy, w którym momencie doszło do zapętlenia skryptu.

5.3 Projekt bazy danych

W pierwszej kolejności należy zaprojektować bazę danych tak aby spełniała wymogi projektu funkcjonalnego.

Zasada działania relacyjnej baz danych (jaką jest MySQL wykorzystywany w aplikacji , jak i technika odwzorowywania w niej rzeczywistych danych jest szerokim zagadnieniem. Czytelników zainteresowanych tym tematem autor odsyła do znakomitego tutorialu1 udostępnionego przez firmę Microsof[29]

Projektując bazę programista powinien trzymać się konwencji nazewnictwa tabel oraz pól tabeli przyjętej przez Ruby on Rails. Zgodnie z konwencją nazwy tabel powinny być liczbą mnogą nazwy modelu, który przechowuje informacje o danej tabeli. Klucz podstawowy tabeli powinien nosić nazwę id, klucze obce powinny być

[25]Zobacz `http://www.iosart.com/firefox/colorzilla/`
[26]Zobacz `http://users.skynet.be/mgueury/mozilla/`
[27]Zobacz `http://tidy.sourceforge.net/`
[28]Zobacz `http://livehttpheaders.mozdev.org/`
[29]Zobacz `http://office.microsoft.com/pl-pl/access/CH062526791045.aspx`

nazwane zgodnie ze schematem {nazwa obiektu w liczbie pojedynczej}_id. Schemat bazy danych, przedstawia rysunek 5.3.

Rysunek 5.3: Schemat bazy danych EiT.

5.3.1 Opis tabel

Tabela *users*

Pole	Typ	Opis
id	int(11)	
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 5.1: Tabela *users*

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 5.4: Struktura pola *privileges* w tabelce *users*

- Tabelka ta zawiera zbiorcze informacje o użytkownikach mogących dostać się do systemu (zarówno studentach, jak i prowadzących).
- Pole *login* może być puste. Wraz z początkiem każdego roku akademickiego,

- Hasło użytkownika jest zahashowane algorytmem SHA-1[30]. Zabezpiecza to hasła użytkowników przed odczytem w przypadku skompromitowania bazy danych.
- Struktura pola *privileges* to maska, w której każdy bit określa inne uprawnienia, jak opisuje to rysunek 5.4. 1 – oznacza przyzwolenia, 0 – brak dostępu
- 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.
- Pole *signature* nie jest nigdzie używane – zachowano je w celu uzyskania wstecznej zgodności i na poczet przyszłych zastosowań.

Tabela *cathedrals*

Pole	Typ	Opis
id	int(11)	
name	varchar(50)	Nazwa katedry

Tablica 5.2: Tabela *cathedrals*

- Tabelka ta zawiera wszystkie katedry obsługiwane przez system.

Tabela *specialities*

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

Tablica 5.3: Tabela *specialities*

- Tabelka ta zawiera wszystkie kierunki obsługiwane przez system. Pierwszy wiersz ($id = 1$) jest pusty i zarezerwowany dla studentów, którzy jeszcze nie dokonali wyboru.

[30]Standard dostępny pod `http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf`

Tabela *users_ students*

Pole	Typ	Opis
id	int(11)	
user_id	int(11)	
speciality_id	int(11)	
sgroup	varchar(7)	Grupa dziekanatowa
sindex	int(6)	Numer indeksu
gadu_gadu	int(15)	Numer Gadu-Gadu
icq	int(15)	Numer ICQ
cell	varchar(15)	Telefon kontaktowy

Tablica 5.4: Tabela *users_ students*

K	R	R	S

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

Rysunek 5.5: Struktura pola grupy

$$rok = \begin{cases} obecny\_rok - 2000 - RR & gdy\ miesiac \leq 9 \\ obecny\_rok - 2000 - RR + 1 & gdy\ miesiac > 9 \end{cases} \tag{5.1}$$

- Tabelka zawiera informacje o studentach. Prowadzący nie mają w niej rekordów.
- *sgroup* i *sindex* mają „s” na początku w związku z obostrzeniami MySQL (*index* i *group* są słowami zastrzeżonymi).
- Na podstawie grupy dziekanatowej, obliczany jest rok studiów wg. wzoru 5.1

Tabela *users_ lecturers*

Pole	Typ	Opis
id	int(11)	
cathedral_id	int(11)	
user_id	int(11)	
place	varchar(30)	Pokój
title	varchar(50)	Tytuł naukowy
consultations	varchar(50)	Terminy konsultacji
phone	varchar(15)	Telefon kontaktowy
photo	varchar(50)	Zdjęcie

Tablica 5.5: Tabela *users_ lecturers*

- Tabelka zawiera informacje o prowadzących. Studenci nie są do niej dopisywani.
- Pole *photo* zawiera tylko nazwę pliku ze zdjęciem. Model w aplikacji sam ustala ścieżkę.

Tabela *subjects_ types*

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

Tablica 5.6: Tabela *subjects_ types*

- Tabelka zawiera kategorie (rodzaje) przedmiotów. Istotne przy grupowaniu po kategoriach.

Tabela *subjects*

Pole	Typ	Opis
id	int(11)	
subjects_type_id	int(11)	
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 5.7: Tabela *subjects*

- Tabelka zawiera informacje o wykładanych przedmiotach. Przedmiot liczy się jako całość (nie ma podziału na wykłady, laboratoria etc.)
- Pole *season* oznacz semestr na którym dany przedmiot jest prowadzony. Jeśli wynosi 0, oznacza to przedmiot specjalny (np. Dziekanat).

Tabela *news_types*

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

Tablica 5.8: Tabela *news_types*

- Tabelka ta zawiera kategorie aktualności.

Tabela *news*

Pole	Typ	Opis
id	int(11)	
news_type_id	int(11)	
user_id	int(11)	
subject_id	int(11)	
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 5.9: Tabela *news*

- Tabelka ta zawiera aktualności.
- Pole *for_year* to maska roczników których dotyczy aktualność, przy czym najmłodszy bit to pierwszy rok, najstarszy – piąty. Jeżeli pole ma wartość „0" to aktualność dotyczy wszystkich.

Tabela *uploaded_files*

Pole	Typ	Opis
id	int(11)	
subject_id	int(11)	
user_id	int(11)	
uploader_id	int(11)	
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 5.10: Tabela *uploaded_files*

- Tabelka ta zawiera informacje o plikach wgranych do systemu (materiałach lub ocenach). Jest także tabelką pośredniczącą dla relacji wiele-do-wielu między prowadzącymi a przedmiotami (zob. rozdz. **??**, str. **??**).

- Pole *filename* 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*

Pole	Typ	Opis
id	int(11)	
user_id	int(11)	
head	varchar(255)	Nazwa grupy
body	text	Opis grupy

Tablica 5.11: Tabela *groups*

- Tabelka zawiera grupy użytkowników. Podstawowe typy grup to:
  - Administratorzy,
  - Zalogowani,
  - Specjalności (Elektronik, Telekomunikacja),
  - Lata (Rok I – Rok V),
  - Grupy dziekanatowe (1 – 10).
- 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 Elektroniki.

Tabela *polls_ questions*

Pole	Typ	Opis
id	int(11)	
user_id	int(11)	
question	varchar(200)	Pytanie
start_time	datetime	Czas rozpoczęcia
end_time	datetime	Czas zakończenia
anonymous	tinyint(1)	Czy anonimowa?

Tablica 5.12: Tabela *polls_ questions*

- Tabelka zawiera informacje o ankietach wraz z pytaniem.
- 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* informuja czy ankieta jest anonimowa. Na stronie maksymalnie mogą być aktywne naraz dwie ankiety – jedna anonimowa, druga nie.

Tabela *polls_ answers*

Pole	Typ	Opis
id	int(11)	
polls_question_id	int(11)	
answer	varchar(200)	Odpowiedź
quantity	int(11)	Ilość udzieleń

Tablica 5.13: Tabela *polls_ answers*

- Tabelka zawiera odpowiedzi do pytań zadanych w ankietach. Jedna ankieta może mieć maksymalnie sześć odpowiedzi (baza tego nie sprawdza).
- Pole *quantity* zawiera ilość użytkowników którzy wybrali tą odpowiedź. Na podstawie tego pola, później jest liczony procentowy udział tego wyboru.

Tabela *exams_ names*

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

Tablica 5.14: Tabela *exams_ names*

- Tabelka ta zawiera nazwy odpowiednich kombinacji typu studiów, roku i kierunku przyporządkowane do ich kodu.
- 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*

Pole	Typ	Opis
id	int(11)	
subject_id	int(11)	
user_id	int(11)	
exams_name_id	varchar(3)	
examiner	varchar(255)	Dane egzaminatora
date	datetime	Data egzaminu
place	varchar(255)	Miejsce egzaminu
term	int(11)	Termin

Tablica 5.15: Tabela *exams*

- Tabelka ta zawiera informacje o egzaminach w najbliższej sesji.
- Pole *examiner* zawiera dane egzaminatora wpisane na sztywno, a nie referencję do odpowiedniego rekordu. Bierze się to stąd, że czasami egzamin może przeprowadzać osoba spoza katedry.
- Pole *term* określa termin egzaminy i powinno zawierać się w przedziale <0;3>.

Tabela *events*

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

Tablica 5.16: Tabela *events*

- Tabelka ta zawiera kalendarz studenta.
- Pole *for_ year* funkcjonuje na identycznej zasadzie, jak to z tabelki *news* (zob. rozdz. 5.3.1, str. 5.3.1).
- Pole *url* nie jest obowiązkowe. Jeśli jednak zostanie podane, cały tytuł będzie linkiem do podanego w *url* adresu.

Tabela *declarations*

Pole	Typ	Opis
id	int(11)	
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 5.17: Tabela *declarations*

- Tabelka ta zawiera informacje o samych deklaracjach jako takich. Nie zawiera informacji o wyborach dokonanych przez studentów.
- Pola *available_from* i *available_ to* zawierają przedział czasowy w którym dana deklaracja jest dostępna do wypełnienia. Rok nie jest brany pod uwagę.
- Pole *year* określa, którego roku akademickiego taka deklaracja dotyczy. Nie działa tu mechanizm znany z *for_ year* – jedna deklaracja jest tylko dla jednego roku.

- Pole *how_many* dotyczy tylko deklaracji odnoszących się do wyboru przedmiotów obieralnych. Jest to liczba dwucyfrowa. Liczba dziesiątek oznacza ilość przedmiotów dla Elektroniki, jedności – dla Telekomunikacji.

Tabela *declarations_subjects*

Pole	Typ	Opis
id	int(11)	
declaration_id	int(11)	
subject_id	int(11)	
user_id	int(11)	
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)	Kierunek
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 5.18: Tabela *declarations_subjects*

- Tabelka ta zawiera szczegółowe informacje o deklaracjach, a także informacje wypełniane przez studentów. Tabelkę tę można dowolnie modyfikować. Więcej na ten temat zostało przedstawione w dziale Deklaracje (zob. rozdz. 5.7.4, str. 5.7.4).

- Pole *price* dotyczy tylko deklaracji dotyczących 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 model oferuje metodę *average*, która obliczą średnią ocen danego studenta.

- Pole *year* dotyczy tylko deklaracji wyboru kierunku 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 kierunku studiów. Zawiera id kierunku (referencja do specialities) na którym dany student chciałby kontynuuować swoją edukację. Jest wypełniane przez studentów.

- Pole *date* zawiera dotyczy wszystkich deklaracji. Zawiera datę ich złożenia. Jest wypełniane przez system automatycznie.

- Pole *language* dotyczy tylko deklaracji wyboru języka przedmiotów. Zawiera dwuliteroww kody języków wybrane przez studenta w formacie WWDD, gdzie WW to kod dla wykładu, a DD to kod dla zajęc dodatkowych (laboratoriów, ćwiczeń, etc.). Jest wypełniane przez studentów.

- 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.

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

Tabela *declarations_ experiences*

Pole	Typ	Opis
id	int(11)	
declaration_id	int(11)	
firstname	varchar(255)	Imię studenta
lastname	varchar(255)	Nazwisko studenta
sindex	int(6)	Numer indeksu
speciality_id	int(11)	
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 5.19: Tabela *declarations_ experiences*

- Tabelka ta zawiera szczegółowe informacje dotyczące praktyki zawodowej danego studenta.
- 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.

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 5.6: Struktura pola *confirmations*

- 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.
- Pole *confirmations* zawiera potwierdzenia dla opiekuna praktyk, a jego struktura została przedstawiona na ryskunku 5.6.

5.3.2 Opis relacji

Relacja *users→users_students*

- Typ relacji: 1 do 1
- Pole: *users_students.user_id*
- Klucz obcy: *users_students_ibfk_1*
- Obowiązkowa
- Dodaje 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 kierunku studiów studenta. Jeśli jeszcze nie został wybrany, pole przyjmuje wartość 1 (zob. tab. 5.3.1, str. 46).

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*
- Tabelka pośrednicząca: *uploaded_files*
- Klucze obce: *uploaded_files_ibfk_1*, *uploaded_files_ibfk_2*
- Obowiązkowa
- Relacja wykorzystywana do określenia, które przedmioty prowadzi jaki wykładowca. Aby uniknąć z każdym semestrem ręcznego ich dodawania, wszystko dzieje automatycznie. Kryterium bycia prowadzącym, jest dodanie pliku do systemu i przypisanie siebie jak prowadzącego do określonego przedmiotu. Wtedy taka osoba (oczywiście, jeśli dodatkowo jest wykładowcą, czyli jest wpisana do tabelki *users_lecturers*) automatycznie tworzy sobie relację z tym przedmiotem poprzez tabelkę *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 na poczet 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 kto jest właścicielem danej grupy.

Relacja *groups↔users*

- Typ relacji: wiele do wielu
- Tabelka 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
- Tabelka 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 starym systemie funkcjonalność praktycznie nieużywana. W nowym porzucona, zachowana tylko na poczet przyszłych ustaleń.

Relacja *groups↔uploaded_files*

- Typ relacji: wiele do wielu
- Tabelka 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 grup do określonych plików. Bardzo ważna funkcjonalność, która zabezpiecza np. materiały dla studentów określonego roku, przed ściągnięciem ich przez studentów innych lat.

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 kierunku studiów. Wszystkie egzaminy w systemie są pogrupowane wg. tych kodów, a te relacja służy ich rozwinięciu 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 jakiego 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. 5.7.4 str. 78)

Relacja *specialities→declarations_ subjects*

- Typ relacji: 1 do wielu
- Pole: *declarations_ subjects.speciality_ id*
- Klucz obcy: *declrations_ subjects_ ibfk_ 4*
- Nieobowiązkowa
- Określa kierunek 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.

5.4 Konwersja bazy danych

TODO

5.5 Mapa serwisu

Kolejnym krokiem będzie zbudowanie mapy serwisu, tak aby spełniała wymogi projektu funkcjonalnego. 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.

Rysunek 5.7: Mapa strony kierunku Elektronika i Telekomunikacja.

5.6 Opis poszczególnych kontrolerów i akcji

5.6.1 *IndexController*

Główny kontroler, uruchamiany przy pierwszym wejściu na stronę.

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

5.6.2 *AboutController*

Kontroler wyświetlający informacje o samej stronie EiT i ludziach, którzy ją tworzyli.

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

5.6.3 *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

5.6.4 *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 i moduły.
- *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 wszystkich aktualnie dostępnych deklaracji a także zawiera informacje na ich temat. Więcej informacji w sekcji „Deklaracje" rozdz. 5.7.4, str. 78.
- *calendar* – kalendarz studenta. Wszystkie istotne wydarzenia z punktu widzenia studenta (sesje, ferie, terminy składania podań, etc.).

5.6.5 *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. Bierze się 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 przerabiane przez skrypt. Przerabianie to polega 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.

5.6.6 *LecturersController*

Kontroler wyświetlający informacje o wszystkich prowadzących dopisanych do systemu.

- *index* – wyświetla listę wszystkich prowadzących, wraz z linkami do ich profili.
- *electronics* – j.w., ale tylko prowadzący Elektroniki
- *telecommunication* – j.w., ale tylko prowadzący Telekomunikacji

5.6.7 *MaterialsController*

Kontroler odpowiedzialny za wyświetlanie i wyszukiwanie materiałów i 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 w formularza i przetwarza je na zapytanie GET. W takiej postaci przesyłane jest ono do akcji *search*. Całe to przetwanie zrobione jest po to, aby można było również podawać zapytania do wyszukiwarki bezpośrednio w postaci linków – w wielu miejscach, są np. linki do materiałów konkretnego prowadzącego, które nie są niczym innym jak odpowiednio spreparowanym zapytaniem dla akcji *search*.

- *search* – pobiera zapytanie, stanowiące pięć 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.

5.6.8 *NewsController*

Kontroler odpowiadający za wyświetlanie aktualności.

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

- *search* – wyszukiwarka aktualności. Przyjmuje parametry rozdzielone przecinkami, przez 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 odpowiedni request GET dla akcji search a następnie wysyła go do akcji search; akcja search wyświetla posegregowana wyniki wyszukiwania (stronami, po 10).

- *read* – akcja, która jako parametr przyjmuje zakodowane ID wiadomości (zob. rozdz. 5.7.1, str. 75), odkodowuje je, a następnie wyświetla aktualność o tym numerze.
- *exams* – akcja wyświetlające informacje o egzaminach w aktualnej sesji. Dodatkowo, jeżeli użytkownik posiada stosowne uprawnienia, umożliwia dodawanie, edycję i usuwanie egzaminów.

5.6.9 *PostgraduateController*

Kontroler odpowiadający za wyświetlanie informacji na temat studiów podyplomowych.

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

5.6.10 *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.

5.6.11 *RedirectController*

Kontroler odpowiedzialny za odpowiednie przekierowywanie ze starych linków na nowe (zob. rozdz. 5.8, str. 80)

- *index* – przekierowuje na adres podany w parametrze

5.6.12 *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 ludzie, znajdujący się na liście studentów corocznie dostarczanej przez dziekanat. Takie osoby są dopisywane do bazy, bez przydzielonego logina i hasła. Proces rejestracj polega więc, wpierw na uwierzytelnieniu się swoim imieniem, nazwiskiem i numerem indeksu, a następnie na podaniu pozostałych informacji o sobie. Po pomyślnym jego przejściu, student staje się użytkownikiem systemu.

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

5.6.13 *RssController*

Kontroler generujący treść dla czytników RSS (zob. rozdz. 3.5, str. 18).

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

5.6.14 *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.

5.6.15 *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. tyle, że oceny
- *groups* – administrowanie grupami studentów. Nie można zmienić domyślnie zadeklarowanych, można jednak dodawać, usuwać i edytować swoje.
- *polls* – dodawanie ankiet. W systemie mogą być maksymalnie dwie – jedna anonimowa i jedna nie, dodanie każdej nowej spowoduje zamknięcie poprzedniej.
- *declarations* – wyświetlenie informacji o deklaracjach wypełnionych przez studentów oraz zarządzanie nimi (zob. rozdz. 5.7.4, str. 78.
- *calendar* – zarządzanie kalendarzem studenta
- *profile* – zarządzanie profilem aktualnie zalogowanego użytkownika. Z poziomu tej akcji, można zmienić swój login, hasło i kilka innych danych.

5.6.16 *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.

5.6.17 *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 użytkownicy, 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 istnieje).

5.6.18 *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.

5.7 Komponenty

Aby aplikacja działała poprawnie, niezbędne okazało się stworzenie kilku komponentół – małych klas, stworzonych w jednym, skonkretyzowanym celu. Dzięki temu, zgodnie z regułą DRY (zob. rozdz. 4.6, str. 31), wszystko jest w jednym miejscu.

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

Schemat kodowania jest następujący:

$$id_{XOR} = id \oplus hash \tag{5.2}$$

$$salt_M = salt_{MARKER} = id_{XOR} \; mod \; 10 \tag{5.3}$$

$$id_{ENC} = \left(\frac{salt \; mod \; 10^{20-salt_M} - salt \; mod \; 10^{20-salt_M-10}}{10^{20-salt_M-10}} * 10 + salt_M \right)_{(28)} \tag{5.4}$$

Rozkodowywanie odbywa się w analogiczny sposób, tylko w drugę stronę.

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

Użycie klasy jest bardzo proste:

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

Rysunek 5.8: Użycie polecenia IdEncoder

5.7.2 Pager

Klasa *Pager* jest wykorzystywana wszędzie tam, gdzie zwracana ilość 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). Konstrukcja konstruktora jest następująca:

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

Rysunek 5.9: Struktura konstruktora klasy Page

Gdzie parametry to:

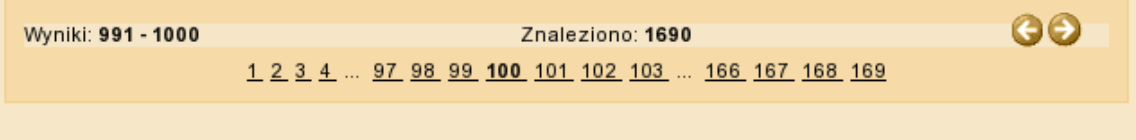
- *params* – tablica params z kontrolera. Z racji tego, że komponenty są oddzielone od warstwy aplikacji, nie widzą domyślnych tablic Ruby on Rails (m.in. params), dlatego należy je przekazać. Jest to konieczne, aby Pager mógł poprawnie budować odnośniki.
- *results* – zawiera obiekt modelu danych wyciągniętych bezpośrednio z bazy, które mają być podzielone na strony.
- *params_key* – domyślnie *:id*. Określa który parametr routera 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 ilość stron,
- *array_of_pages* – zwraca pasek nawigacyjny Pagera. To te cyfry z numerami stron i strzałkami „poprzednia”, „następna” etc. (zobacz rysunek **??**). 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 **??**
- *actual_page* – zwraca numer aktualnej strony
- *next_page* – zwraca numer następnej strony
- *previous_page* – zwraca numer poprzedniej strony



Rysunek 5.10: Ilustracja wykorzystania *array_of_pages*, gdzie *page_number* = 100, a *pages_padding* = 3. Dodatkowo na górze po lewej, efekt działania *results_range* z domyślnym separatorem

Dodatkowo Pager ma przygotowany, predefiniowany dla serwisu EiT widok wyświetlający nawigację, który można znaleźć w *app/views/_partials/_pager.rhtml*.

5.7.3 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.

5.7.4 Deklaracje

System deklaracji to najdynamiczniej zmieniająca się część serwisu. 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 plugin. Dzięki temu, łatwo w przyszłości będzie dodawać nowe deklaracje, bez naruszenia integralności serwisu.

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 klasy:

- *@logged_user* – obiekt zalogowanego użytkownika
- *@declarations_subjects* – obiekt modelu DeclarationsSubject (tabela *declarations_subjects*, zob. rozdz. 5.3.1, str. 55) 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 przepuścić je przez 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, 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. 5.3.1, str. 54)
- *@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. 5.3.1, str. 54 (Tabela *declarations*)
- Rozdz. 5.3.1, str. 55 (Tabela *declarations_subjects*)
- Rozdz. 5.3.2, str. 64

Po utworzeniu nowego plugina należy dopisać odpowiednie deklaracje do tabeli *declarations*, oraz ewentualnie utworzyć nowe kolumny w tabeli *declarations_columns* i wypełnić je danymi początkowymi.

5.8 Przemapowanie hiperłączy

5.9 Testowanie i instalacja

Rozdział 6

Zakończenie