Spis treści

Abstract

Nowadays Internet has been used as a wide range marketing and business tool. Most people is using Internet in various fields of life. For this reason sites and web applications becomes crowded and overloaded. Some times they can even stop working which will result in loss of money. That is why creating of high performance and stable applications is so important today. Also giving possibility to access page by huge amount of users can be cost-effective, when we put some commercials.

Last but not least web application architecture needs to be considered. It is very common that companies are using expensive hosting solutions, that doesn't actually fit their needs. On the another hand, when periodically massive traffic comes, their servers cannot handle this. Thats why cloud service providers are considered to be a good alternative in this situations, because companies are paying only for resources actually used and nothing more.

This thesis will cover the topic of efficiency optimization of web applications. It will be splitted into 3 main areas of optimizations. Firstly optimization of application architecture will be considered and then other issues like frontend optimization as well as database and webserver tuning will be discussed. Also this thesis needs to provide some useful tools and techniques that need to be used for mesuring of actual application performance and availability. Without this knowledge we cannot even start the main part of optimization, because we do not find the typical bottlenecks.

Wstęp

Niniejsza praca dyplomowa dotyczy zagadnień inżynierii oprogramowania oraz technologii baz danych wykorzystanych w dziedzinie *e-commerce*. Główny cel badań stanowi przedstawienie realnych korzyści biznesowych wynikających z zastosowania szerokiego spektrum usprawnień aplikacji internetowych.

W treści pracy przedstawiona jest analiza technik optymalizacji aplikacji internetowych z uwzględnieniem wykonywanych po stronie serwera (*server side*) oraz szeregu usprawnień po stronie przeglądarki *client side*. Równie istotny jest wybór metod dających najlepsze rezultaty w świetle obecnych technologii. Dodatkowo ważne jest wyróżnienie wszystkich pośrednich czynników, które mogą w jakimkolwiek stopniu wpłynąć na działanie aplikacji.

Do implementacji systemu wykorzystano technologie skryptowe PHP oraz Python. Pierwsza z nich pozwoli na szybkie przedstawienie obrazu typowej aplikacji e-commerce (olbrzymi odsetek takich aplikacji w internecie jest napisanych właśnie w tym języku). Python z kolei pozwoli wykorzystać zalety platformy Google Application Engine (GAE), która zapewnia skalowalną architekturę do późniejszych testów.

Obserwacja zostanie przeprowadzona na przykładzie aplikacji z dziedziny *e-commerce* - księgarni elektronicznej. Wybór takiej tematyki jest celowy, ponieważ najczęściej właśnie w takich aplikacjach występują problemy natury optymalizacyjnej. Spowodowane jest to najczęściej koniecznością obsłużenia wielu klientów, transakcji bazodanowych, czy przede wszystkim generowanie rozbudowanego wizualnie interfejsu użytkownika. Podczas generowania obrazów, wykonywania kodu serwera, czy interpretowania rozbudowanych struktur dokumentu *HTML*, czas oczekiwania na wynik może się zauważalnie wydłużyć.

Praca ma również na celu prezentację narzędzi badawczych umożliwiających znalezienie wąskich gardeł aplikacji. Tylko sukcesywne łączenie różnych narzędzi oraz ciągłe monitorowanie działania zapewni oprogramowaniu stabilność oraz wysoką dostępność.

Problematyka i zakres pracy

Wraz ze wzrostem popularności Internetu jako medium informacyjnego, istotnym problemem stało się obsłużenie napływającego ruchu sieciowego ze strony użytkowników. Pojęcie czas to pieniądz ma tutaj kluczowe znaczenie, ponieważ umiejętność przetworzenia jak największej ilości użytkowników w jak najkrótszym czasie będzie przekładała się na realne zyski.

Innym ważnym problemem dotyczącym stron jest zapewnienie wysokiej dostępności usługi, czyli wyeliminowanie do minimum wszelkiego rodzaju przerw wynikających z błędów aplikacji. Temat ten związany jest jednak głównie z odpowiednią konfiguracją sprzętową czyli wykorzystaniem równolegle działających instancji sprzętowych, które będą wykorzystywane w celu zrównoważenia ruchu, lub awaryjnie w wypadku uszkodzenia nośnika danych na jednym z urządzeń.

W internecie istnieje stosunkowo dużo publikacji związanych z tematyką optymalizacji aplikacji webowych, jednakże w większości wypadków omawiany jest tylko nikły procent wszystkich zagadnień. Zazwyczaj pomijane są aspekty związane z kodem po stronie klienta, a także studium narzędzi i metod badawczych. Najpopularniejsze na rynku są publikacje dotyczące optymalizacji samego kodu lub zapytań bazodanowych w zależności od użytego języka aplikacji lub bazy danych.

Praca ma na celu przedstawienie możliwie najszerszego wachlarzu technik optymalizacji. Należy przy tym zaznaczyć, że statystycznie tylko 20% czasu przetwarzania strony przez przeglądarkę, jest poświęcane na oczekiwanie na odpowiedź serwera. Implikuje to olbrzymie znaczenie optymalizacji kodu po stronie klienta w celu znacznego przyspieszenia odpowiedzi. Nie bez znaczenia są też czynniki takie jak lokalizacja geograficzna strony oraz konfiguracja serwera.

Obecnie Internet przestał już być tylko wojskowym eksperymentem, czy zaledwie miejscem na prezentacje własnej strony domowej. Wyewoluował on do medium globalnej komunikacji z gigantyczną ilością klientów docelowych. Większość firm, instytucji czy organizacji rządowych czuje się w obowiązku posiadania i utrzymywania strony internetowej, zazwyczaj spełniającej określone cele biznesowe. Pokazuje to jak bardzo powszechnym narzędziem codziennego użytku jest dzisiaj globalna pajęczyna.

Ze względu na niekwestionowaną popularność języka PHP oraz jego olbrzymią prostotę - w stosunkowo krótkim czasie od powstania języka, zaczęły pojawiać się proste strony, następnie aplikacje internetowe a kończąc na portalach i usługach sieciowych. Język PHP stał się narzędziem na tyle uniwersalnym, że zagadnienia modelowane przy jego użyciu, można z powo-

dzeniem przenieść na inne platformy takie jak *Java Enterprise* czy *.NET*. W sieci istnieje ponadto wiele gotowych implementacji systemów e-commerce: sklepów (np. Magento), systemów CRM (SugarCRM) czy np. platforma edukacyjna Moodle będąca częścią infrastruktury edukacyjnej Politechniki Łódzkiej np. dla Wydziału FTIMS. Wymienione przykłady zostały w całości zaimplementowane w języku PHP, a o ich popularności świadczą miliony ściągnięć i wdrażeń.

Jedną z wad gotowych rozwiązań jest fakt, że nie zawsze są one dostosowane do wszystkich stawianych przed projektem wymagań. Powoduje to, że, projekt docelowy w rezultacie otrzyma więcej funkcjonalności niż jest to wymagane. Z drugiej jednak strony możliwe jest, że gotowe rozwiązanie będzie wymagało szeregu usprawnień lub dodania nowych funkcjonalności. W większości wypadków, rozwiązania gotowe nie są jednak od początku dostosowane do bardziej zaawansowanych zastosowań lub wymagań wydajnościowych. Dlatego ważne jest by wykorzystując istniejące narzędzia wyskalować aplikacje do konkretnych potrzeb lub przewidzieć przyszłe obciążenie.

Projektowana w ramach pracy aplikacja stanowi studium przypadku analizy wydajnościowej aplikacji działającej w ściśle określonej architekturze. W ramach analizy omówione zostaną następujące zagadnienia:

- metody badawcze
- dobór odpowiedniej technologii,
- wybór właściwej architektury sprzętowej,
- projekt i optymalizacja bazy danych,
- optymalizacja kodu klienta,
- rozproszenie usług

Założenia wstępne

Treść pracy dyplomowej stanowi wypadkową informacji zawartych w dokumentacjach dotyczących użytych technologii, jak również wiedzy autora zdobytej podczas implementacji wielu zróżnicowanych aplikacji internetowych. Bardzo istotne dla publikacji były również informacje pochodzące ze sprawdzonych źródeł takich jak np. oficjalny blog programistyczny Yahoo. Serwis Yahoo ze względu na olbrzymie doświadczenie w kwestii skalowania aplikacji internetowych postanowił podzielić się tą wiedzą na łamach swoich stron internetowych oraz kilku specjalistycznych książek.

W kolejnych rozdziałach omawiane będą kolejne etapy drogi, jaką pokonuje żądanie od rozpoczęcia do zwrócenia i zinterpretowania przez przeglądarkę użytkownika. Często kolejne etapy są prawie niezauważalne ze względu

na małe różnice czasowe, jednak podczas wnikliwej analizy każdy z etapów trzeba rozpatrywać indywidualnie.

Analiza wydajnościowa przeprowadzana w wypadku prostych aplikacji, które nie będą w przyszłości podlegały intensywnemu obciążeniu nie ma w zasadzie sensu. Publikacja zyskuje na wartości w wypadku kiedy projekt jest bardziej rozbudowany, a my potrzebujemy szybkiego i rzetelnego sposobu na wykrycie potencjalnych elementów do optymalizacji.

Układ pracy

Praca została podzielona na następujące rozdziały:

- **Rozdział pierwszy** opisuje narzędzia badawcze, a także wyjaśnia teorię działania aplikacji opartych o protokół HTTP
- **W drugim rozdziale** przedstawiono wymagania stawiane przed aplikacją oraz jej budowę.
- **Trzeci rozdział** dotyczy optymalizacji związanych z wyborem odpowiedniej architektury.
- **Rozdział czwarty** dotyczy optymalizacji związanej z implementacją aplikacji
- **Rozdział piąty** dotyczy optymalizacji kodu klienckiego
- **Rozdział szósty** metody rozproszenia aplikacji i usług
- **Rozdział siódmy** podsumowanie i wnioski końcowe

Rozdział 1

Analiza jakościowa i wydajnościowa aplikacji

Projektując aplikacje, już w fazie projektowej, należy myśleć o zapewnieniu wysokiej wydajności, a także o potencjalnych problemach, które mogą się pojawić po wdrożeniu oprogramowania. W celu zapewnienia tworzonej aplikacji najwyższej skuteczności pracy, należy wziąć pod uwagę wiele cech, wśród których najważniejsze zdefiniowane są poniżej.

Skalowalność (ang. *Scalability*)
cecha aplikacji, określana jako zdolność do wzrostu wydajności aplikacji wraz ze zwiększeniem ilości dostępnych zasobów sprzętowych (serwery WWW, bazy danych, wydajniejsze procesory).

Niezawodność (ang. *High availability*)
stanowi projekt, jak i odpowiednią implementacje systemu, zapewniająca określony poziom ciągłości wykonywania operacji w czasie. Polega to na zapewnieniu jak największej dostępności usługi.

Wydajność (ang. *Performance*)
przekłada się na możliwość szybkiego wykonywania kodu aplikacji oraz utrzymaniu czasu odpowiedzi aplikacji na stosownym poziomie.

Często skalowalność jest mylona z wydajnością, jednak przekładając to na bardziej życiowy przykład, wydajność aplikacji można porównać do szybkiego samochodu. Z drugiej strony, bez zapewnienia odpowiednich dróg, ten szybki samochód lub ich grupa, nie jest w stanie rozwinąć maksymalnej prędkości. W najgorszym wypadku może nawet utkwić w korku, blokowany przez inne pojazdy. Skalowalność jest więc zapewnieniem **odpowiedniej infrastruktury** gwarantującej właściwy rozrost systemu.

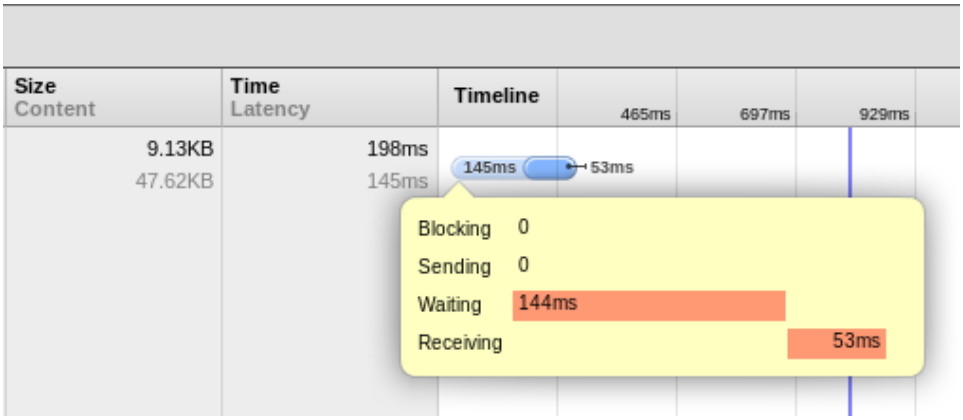
W celu zapewnienia możliwie najlepszej jakości tworzonej aplikacji, należy stale monitorować aktualny poziom wydajności aplikacji. Należy jednak

mieć na uwadze, że na wydajność aplikacji składa się czas przetwarzania poszczególnych węzłów systemu. Należy więc testować każde z nich osobnymi metodami, omówionymi w dalszej części pracy.

Szukając przyczyn błędów, warto rozpocząć analizę od najbardziej ogólnego komponentu czyli serwera WWW odpowiedzialnego za wysyłanie odpowiedzi na żądanie użytkownika. Następnie należy dokonać dekompozycji, wyróżniając kolejne węzły systemu, takie jak dalsze instancje serwera WWW czy serwery bazodanowe.

1.1 Analiza wydajności serwera WWW

Zadaniem serwera WWW jest wysłanie do inicjatora żądania wyniku przetwarzania zasobu określonego adresem URL. W najprostszym wypadku, analiza wydajności serwera, polega na odpytaniu go o określony zasób i zmierzenie czasu od rozpoczęcia tej akcji, do odebrania rezultatu. Taki proces możma prześledzić i przeanalizować w większości popularnych przeglądarek np. Google Chrome, które jest wyposażone w wiele przydatnych narzędzi analitycznych (Rys. 1.1).



Rys. 1.1: Analiza czasu wykonywania strony http://ftims.edu.p.lodz.pl// wykonana w przeglądarce Google Chrome

Taki sposób analizy, jest jednak przydatny jedynie w wypadku znacznych problemów z wydajnością aplikacji, ponieważ testowanie czasu odpowiedzi dla pojedynczego użytkownika, wykonującego pojedyncze żądanie, nie jest w żadnym stopniu miarodajne.

W celu zapewnienia bardziej rzetelnego testu, należy skorzystać z dedykowanych rozwiązań takich jak `ab` oraz `siege`. Są to typowe narzędzia przeznaczone do sprawdzania jak dobrze serwer radzi sobie z obsługą bardziej złożonego ruchu sieciowego. Przykładowo, dla wcześniej użytej strony, można zasymulować ruch równy wykonaniu 10 jednoczesnych żądań przez

10 niezależnych użytkowników. W tym celu należy wydać komendę `ab -n 10 -c 10 http://ftims.edu.p.lodz.pl/`. Rezultat działania komendy widoczny jest na listingu 1.1.

Listing 1.1: Analiza strony z wykorzystaniem narzędzia `ab`

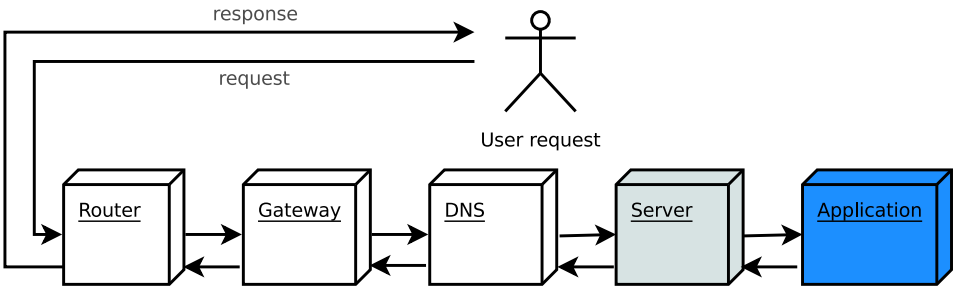
```
Server Software:        Apache/2.2.14
Server Hostname:        ftims.edu.p.lodz.pl
Server Port:            80

Document Path:          /
Document Length:        48759 bytes

Concurrency Level:      10
Time taken for tests:   0.695 seconds
Complete requests:      10
Failed requests:        0
Write errors:           0
Total transferred:      492510 bytes
HTML transferred:       487590 bytes
Requests per second:    14.38 [#/sec] (mean)
Time per request:       695.270 [ms] (mean)
Time per request:       69.527 [ms] (mean, across all concurrent
    requests)
Transfer rate:          691.77 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       52   63   7.4     63      74
Processing:   300  503  90.9    532     632
Waiting:      127  233  62.8    235     381
Total:        354  565  92.9    593     695

Percentage of the requests served within a certain time (ms)
  50%    593
  66%    612
  75%    625
  80%    628
  90%    695
  95%    695
  98%    695
  99%    695
 100%    695 (longest request)
```

Jak można wywnioskować z powyższych danych, narzędzie wykonuje wiele przydatnych analiz, a także wyświetla informacje o badanym zasobie. Widać przede wszystkim, że czas oczekiwania na stronę przy 10 użytkownikach jest prawie trzykrotnie dłuższy, niż podczas jednego żądania wykonanego w przeglądarce (1.1).

Oczywiście na wyniki pomiarów ma także wpływ prędkość połączenia internetowego, dlatego w celu pominięcia dodatkowych czynników, testy docelowej aplikacji będą wykonywane przede wszystkim na lokalnym serwerze. Najbardziej miarodajną jednostką określającą wydajność aplikacji w wypadku narzędzia *ab* jest liczba zapytań na sekundę (*req/s*). Określa ona maksy-

malną ilość żądań w jednostce czasu, jaką aplikacja jest w stanie obsłużyć. Oczywiście im większa wartość, tym lepsza ogólna wydolność aplikacji.

Na rysunku 1.2 przedstawiono cykl życia żądania od użytkownika je inicjującego, kończąc na odebraniu odpowiedzi serwera. Jak można zauważyć żądanie przebywa stosunkowo długą drogę, nim trafi do faktycznej aplikacji. Wynikiem tego są dodatkowe opóźnienia zależne od stopnia skomplikowania architektury trzech pierwszych węzłów. Dlatego też należy mieć na uwadze, że problemy z szybkością działania aplikacji nie muszą leżeć wyłącznie po stronie aplikacji lub serwera. Do najbardziej popularnych należą: niska przepustowość łącza internetowego klienta, wolny serwer DNS, daleka lokalizacja geograficzna serwera WWW, źle skonfigurowany router lub bardzo obciążona sieć lokalna.



Rys. 1.2: Cykl życia żądania

Nawiązując do listingu 1.1, wartościami związanymi ze wspomnianymi w poprzednim akapicie węzłami są `Connect` oraz `Waiting`, czyli odpowiednio czas oczekiwania na połączenie z zasobem i czas pobierania odpowiedzi z zasobu. Istnieje 5 głównych czynników wpływających na czas odpowiedzi serwera.

Położenie geograficzne i problemy z siecią komputerową.
Nie bez znaczenia dla czasu odpowiedzi, jaki użytkownik odczuwa jest też lokalizacja serwerów stron. Jeśli serwery są zlokalizowane w USA, a użytkownicy odwiedzający stronę są np. z Europy, dystans jaki musi pokonać żądanie od momentu dotarcia do zasobu, oczekiwania, aż do jego pobrania jest nie współmiernie większy niż w wypadku stron hostowanych dla tego samego położenia geograficznego. Stopień opóźnienia jest zazwyczaj uzależniony od ilości routerów, serwerów pośrednich, a nawet oceanów, które pokonuje żądanie od punktu początkowego do odbiorcy i z powrotem.

Wielkość dokumentu odpowiedzi serwera.
Zależność między wielkością dokumentu, a czasem odpowiedzi serwera jest oczywista, łatwo więc sprawdzić, że im większy dokument trzeba pobrać, tym więcej czasu potrzeba na zakończenie tego procesu.

Wykonywanie kodu aplikacji.
Najczęstsza przyczyna wolnego działania aplikacji wynika właśnie z braku optymalizacji kodu klienta. Długi czas wykonywania kodu aplikacji implikuje, długi czas łączny oczekiwania na odpowiedź serwera. Problem ten zostanie szczegółowiej poruszony w rozdziale 5.

Rodzaj użytej przeglądarki.
Nie bez znaczenia dla ogólnego czasu ładowania strony jest również rodzaj użytej przeglądarki. Często wbudowane w przeglądarkę wewnętrzne mechanizmy buforowania zasobów pozwalają w znaczny sposób zredukować ilość zapytań wysyłanych do serwera. Dotyczy to zwłaszcza danych statycznych takich jak arkusze CSS, pliki JavaScript czy zasoby graficzne, które nie zmieniają się zbyt często.

Konfiguracja serwera WWW.
W zależności od użytej technologii, istnieje wiele różnych serwerów HTTP. Wśród najczęściej używanych, prym wiedzie serwer HTTP *Apache*. Dla rozwiązań napisanych w technologii Java często wykorzystywane są serwery *GlassFish*, *Tomcat*, *Jetty*. W większości wypadków zaraz po instalacji, oprogramowanie serwera nie nadaje się jeszcze do wykorzystania w produkcji. Należy wyważyć ustawienia serwera do bieżących potrzeb, ponieważ w większości wypadków domyślne ustawienia mogą znacznie obniżyć ogólną wydajność. Innym ważnym działaniem jest dostosowywanie serwera do konkretnych zastosowań - do serwowania plików statycznych lepszym rozwiązaniem jest wykorzystanie bardziej oszczędnego pamięciowo i operacyjnie serwera *Ngnix*, natomiast do bardziej zaawansowanych zastosowań, w tym wykonywanie kodu aplikacji, serwera Apache lub osobnej instancji serwera *Ngnix*.

Administratorzy serwerów WWW mają bezpośredni dostęp do statystyk odwiedzalności stron, przez co pozwala to zaobserwować pewne trendy odwiedzin użytkowników. Często jest tak, że dane zasoby są dużo intensywniej odpytywane przez użytkowników np. w czasie 10 minut stronę odwiedza 100 użytkowników. Łatwo to sobie wyobrazić np. w wypadku premiery jakiejś nowej gry, lub publikacji wyników egzaminu na uczelni. Taki periodyczny, lecz bardzo wzmożony ruch może powodować pewne trudne do ustalenia problemy z działaniem aplikacji. Dlatego też twórcy narzędzia `ab`, zaimplementowali również możliwość testów czasowych (ang. *timed tests*). W ten sposób można zasymulować jak strona będzie się zachowywała również w takich nagłych wypadkach.

Wydając komendę `ab -c 10 -t 30 http://ftims.edu.p.lodz.pl/`, można sprawdzić, jak zachowa się aplikacja odwiedzana przez 10 użytkowników jednocześnie w czasie 30 sekund. Ta komenda pozbawiona jest parametru *-t ilość żądań*, oznacza to że symulacja zakończy się po 30 sekundach lub po osiągnięciu limitu 50 000 żądań.

Listing 1.2: Test obciążenia czasowego

```
Benchmarking ftims.edu.p.lodz.pl (be patient)
Finished 504 requests

Server Software:        Apache/2.2.14
Server Hostname:        ftims.edu.p.lodz.pl
Server Port:            80

Document Path:          /
Document Length:        48759 bytes

Concurrency Level:      10
Time taken for tests:   40.180 seconds
Complete requests:      504
Failed requests:        0
Write errors:           0
Total transferred:      24822504 bytes
HTML transferred:       24574536 bytes
Requests per second:    12.54 [#/sec] (mean)
Time per request:       797.213 [ms] (mean)
Time per request:       79.721 [ms] (mean, across all requests)
Transfer rate:          603.31 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       48   65  14.3     61     145
Processing:   284  436 288.9    376    2957
Waiting:      119  199 281.5    151    2660
Total:        333  500 287.8    439    3007

Percentage of the requests served within a certain time (ms)
  50%    439
  66%    458
  75%    477
  80%    493
  90%    563
  95%    666
  98%   1420
  99%   2142
 100%   3007 (longest request)
```

Listing 1.2 przedstawia wynik testów czasowych. Najważniejszą informacją z punktu widzenia optymalizacji jest ilość żądań na sekundę, która w tym wypadku wynosi `12.54`. Narzędzie `ab` pozwala również zdiagnozować potencjalne błędy aplikacji pod wpływem zbyt dużego ruchu. Pola takie jak `Failed requests` oraz `Write errors` ułatwiają określenie prawidłowości wykonywania żądań. W powyższym przykładzie, wartości są akceptowalne (średni czas żądania to 0.5 sekundy), co najważniejsze nie występują błędy na poziomie serwera WWW i z dużym prawdopodobieństwem również na poziomie aplikacji. Oczywiście zauważalny jest spadek wydajności, w porównaniu z pierwszym testem, co prawda wartości średnie są zbliżone, jednak widać większe rozbieżności między wartościami minimalnymi a maksymalnymi. Najdłuższe zapytanie zajęło ponad 3 sekundy.

W dokumentacji aplikacji `ab`, można znaleźć informację, że niektóre serwery mogą blokować wysyłane przez niego nagłówki HTTP. W tym celu

URL	Średnia	Min	Max	Błąd (%)	Req/Min
..14543704,wiadomosc.html	2299	415	94329	2.2	43.7
...1028235,wiadomosci.html	2485	335	94434	2.8	43.7
...14545325,wiadomosc.html	2023	394	94285	1.8	43.7
Łącznie	**2269**	**335**	**94434**	**2.27**	**131.1**

Tab. 1.1: Tabela z rezultatem działania aplikacji JMeter

można wykorzystać przełącznik umożliwiający podanie się za inną przeglądarkę. Np. chcąc zasymulować odwiedziny przy użyciu przeglądarki Chrome należy wykonać poniższą komendę. `ab -n 100 -c 5 -H "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US)AppleWebKit/534.2 (KHTML, like Gecko)Chrome/6.0.447.0 Safari/534.2" http://www.example.com.`

Pomimo wielu zalet wynikających z korzystania z narzędzia `ab`, istnieje jedna zasadnicza wada. Aplikacja nie daje możliwości przetestowania pewnego scenariusza lub jest to bardzo niewygodne, a przypadku aplikacji z wykorzystaniem technologii *JavaScript* i *AJAX* wręcz niewykonalne.

Dlatego też na przełomie lat wyspecjalizowały się bardziej zaawansowane narzędzia przeznaczone do prześledzenia logicznej kolejności działań wykonywanych na stronie (pewnego przypadku użycia) i wykonywanie analiz właśnie w ramach logicznego zbioru akcji. W ten sposób możliwe jest przeprowadzenie tzw. *testów funkcjonalnych*, a także sprawdzenie w jakim stopniu są one wrażliwe na zwiększony ruch sieciowy.

Jednym z przykładów takiego narzędzia o naprawdę olbrzymich możliwościach jest *Apache JMeter*. Z jego pomocą możliwe jest nagranie pewnego ciągu akcji wykonanych przy pomocy przeglądarki, a następnie przeprowadzenie ciągu analiz i testów na tak wyodrębnionym zbiorze. Tabela 1.1 pokazuje wynik działania przykładowego scenariusza polegającego na symulowaniu wejścia na stronę `http://www.wp.pl`, a następnie kliknięciu jednego z linków wiadomości, po czym kliknięciu na kolejny link z dostępnych na bieżącej stronie. Ostatnim elementem łańcucha akcji jest wysłanie komentarza do artykułu. JMeter podobnie jak `ab` umożliwia wykonywanie równoległych połączeń użytkowników określanych jako wątki (*threads*).

W zdefiniowanym przypadku użycia 5 użytkowników wykonuje jednocześnie tą logikę 500 razy. Na zakończenie dane możliwe są do wyeksportowania do formatu *CSV* lub wyświetlone bezpośrednio na ekranie. JMeter jest narzędziem bardzo rozbudowanym, przez co idealnie nadaje się do testowania zaawansowanych scenariuszy zarówno pod kątem poprawności działania, jak również ogólnej wydajności.

1.2 Analiza wydajności frontendu

W poprzednim podrozdziale została omówiona tematyka testowania czasu działania zasobów serwowanych przez serwer WWW. Wykorzystując wspomniane wcześniej narzędzia można jednoznacznie ustalić czy aplikacja funkcjonuje w sposób prawidłowy, czy nie występują błędy w pracy serwera oraz jak dobrze oprogramowanie radzi sobie ze wzmożonym obciążeniem.

Wydawać by się mogło, że problem testowania wydajności aplikacji został jednoznacznie omówiony. Nic bardziej mylnego, jedynie w idealnym świecie, strona WWW składała by się wyłącznie z tekstu, a użytkownicy do przeglądania Internetu, korzystaliby wyłącznie z terminali.

Współczesny użytkownik internetu, do przeglądania jego zasobów wykorzystuje przeglądarkę internetową, zdolną do wyświetlania zarówno tekstu, jak i mediów wszelakiego typu. Dlatego też, dla większej precyzji, konieczne jest wyszczególnienie kluczowego komponentu oprogramowania, jakim jest *front-end* aplikacji.

W przypadku aplikacji internetowych, *front-end* to graficzny interfejs służąca do komunikacji użytkownika ze stroną i prezentacji danych opracowanych przez zaplecze systemu (back-end) w sposób przystępny i zrozumiały.

Odpowiednia optymalizacja *front-endu* jest o tyle ważna, że jest to pierwsza technologia z jaką użytkownik ma kontakt w momencie korzystania z aplikacji webowej [2].

Frontend realizuje przetwarzanie i analizę wyniku odpowiedzi serwera, przy czym obowiązującym językiem komunikacji jest język HTML (*HiperText Markup Language*). Od kilku lat w wyniku rozwoju trendu WEB 2.0, istotnym zabiegiem, projektowanych aplikacji, staje się przeniesienie części logiki na stronę przeglądarki (frontendu). Cienkie do tej pory aplikacje webowe (*thin client*), zaczynają - dzięki zdobyczą technologicznym takim jak *AJAX* realizują znacznie szerszy zakres funkcjonalny niż dotychczas. Oznacza to, że przetwarzanie danych może mieć miejsce wykorzystując przeglądarkę internetową i język JavaScript. Dlatego też kolejnym istotnym elementem analizy wydajnościowej staje się analiza *frontendu*.

Na podstawie badań firmy Juniper, stwierdzono, że średni czas, jaki użytkownik jest w stanie poczekać na załadowanie strony to zaledwie 4 sekundy. Nie warto więc tracić potencjalnych klientów strony, tylko i wyłącznie z powodu braku optymalizacji po stronie przeglądarki.

Wśród istniejących na rynku rozwiązań służących do analizy po stronie klienta, najpopularniejszymi są te, które są wbudowane bezpośrednio w interfejs przeglądarki internetowej (jest to przecież najbardziej intuicyjne podejście). Jednym z pierwszych rozwiązań tego typu była wtyczka `Firebug`

napisana dla przeglądarki Firefox. Jest to obecnie najbardziej zaawansowane narzędzie tego typu, rywalizujące jednocześnie z natywnymi dodatkami deweloperskimi dla przeglądarki Chrome.

Interfejs Firebuga pozwala na szczegółową inspekcję kodu HTML wraz z możliwością dynamicznej operacji na węzłach DOM dokumentu HTML. Nie mniej ważnymi narzędziami są: możliwość wykonywania i debugowania kodu JavaScript na stronie, inspekcja związanego z dokumentem HTML obiektu DOM, edycja i rewizja kodu JavaScript oraz narzędzie do monitorowania ruchu sieciowego wykonywanego przez aplikację.

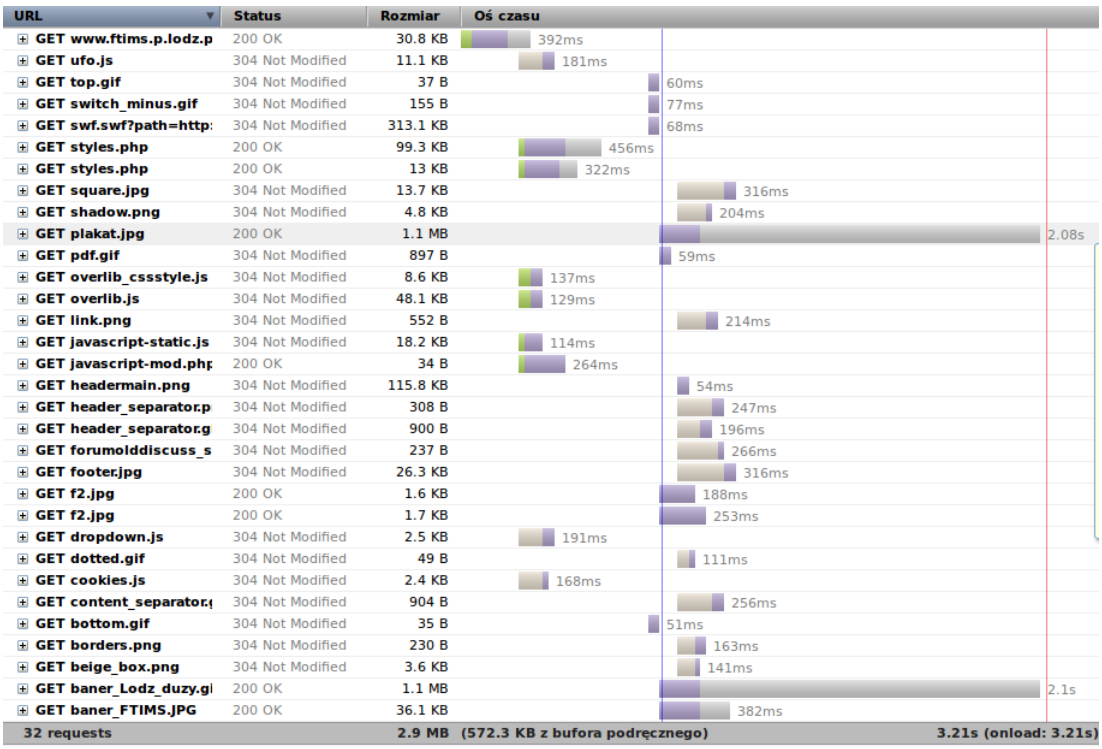
Ten ostatni moduł pełni podobną rolę do narzędzia `ab`, jednak wyświetla wszystkie zasoby, które mają bezpośrednie powiązanie z bieżącym dokumentem HTML. Omawiane wcześniej narzędzia pokazywały jedynie czas renderowania dokumentu HTML, jednak należy mieć na uwadze, że strona internetowa składa się z wielu różnych zasobów, wśród których nie sposób pominąć: grafik, arkuszy CSS, kodu JavaScript, apletów Java czy obiektów Adobe Flash.

Każda strona może zawierać zróżnicowaną ilość takich zasobów, dlatego na łączny czas ładowania strony składa się zarówno czas oczekiwania na dokument HTML, jak również czas konieczny na pobranie każdego z powiązanych z nim zasobów.

Nawiązując do [3] istniej zasada, która mówi, że tylko 10-20% czasu odpowiedzi jest spędzane na oczekiwanie dokumentu HTML, natomiast pozostałe 80-90% to czas na pobieranie pozostałych zasobów i ładowanie zawartości DOM.

Rysunek 1.3 jest najlepszym przykładem tej zasady. Strona wykonuje 32 zapytania do serwera, z czego tylko jedno to żądanie dokumentu HTML. Pobranie tego dokumentu zajęło około 400ms, tymczasem łączny czas wczytywania strony wyniósł **3.21 sekundy**. Oznacza to, że generowanie dokumentu zajęło zaledwie **12%** łącznego czasu oczekiwania.

Na podstawie danych z Firebuga łatwo stwierdzić pewne nieprawidłowości, bowiem w ramach strony wczytywane są 2 stosunkowo duże (1,1 MB) dokumenty graficzne, które prawdopodobnie nie zostały wymiarowane do odpowiednich rozmiarów. `Firebug` stanowi, więc cenne narzędzie przy diagnostyce *frontendu* strony. Narzędzie sstaje się jeszcze przydatniejszy przy pojawianiu się elementów dynamicznych JavaScript, ponieważ pozwala śledzić zarówno aktualnie wykonywany kod, jak również nasłuchiwać zapytań asynchronicznych wykonywanych przez AJAX. Narzędzie to może być również przydatne podczas śledzenia zmian dokonywanych w dokumencie HTML, za pomocą narzędzia inspekcji, umożliwia bowiem zbadanie każdego węzła.



Rys. 1.3: Analiza ruchu sieciowego na stronie http://ftims.p.lodz.pl

1.3 Analiza wydajności bazy danych

Bardzo często ogólna wydajność aplikacji uzależniona jest od szybkości operacji odczytu / zapisu bazy danych. W pewnym momencie twórcy aplikacji zaczynają oczekiwać od niej większej wydajności. Należy jednak zadać pytanie co należy tak naprawdę optymalizować? Konkretne zapytanie? Schemat bazy? Czy może sprzęt na którym baza danych pracuje? Jedynym sposobem na znalezienie jednoznacznej odpowiedzi, jest zmierzenie pracy wykonywanej przez bazę i sprawdzenie wydajności pod wpływem różnych czynników [1].

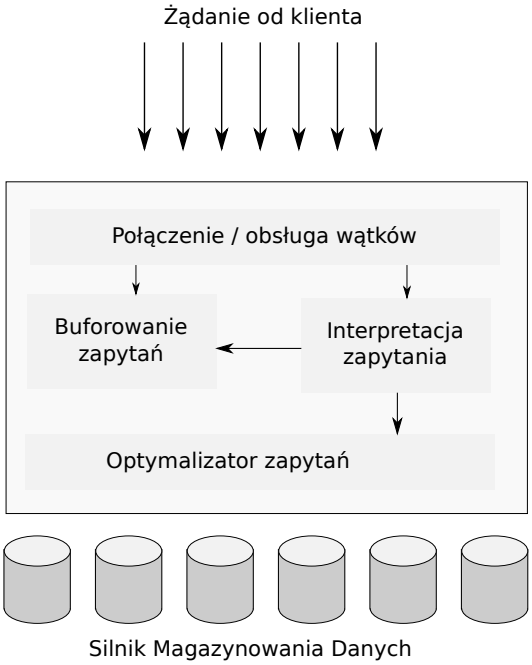
Bazy danych ewoluowały na przestrzeni kilkunastu lat, początkowo były to po prostu pliki o określonej strukturze, jednak wraz ze wzrostem wymagań zaczęto stosować równoległy dostęp do danych, a także dbając o spójność danych, zaimplementowano system transakcji.

Obecnie widać specjalizacji baz danych do konkretnych zastosowań, pomimo dominacji na rynku baz relacyjnych opartych o standard *SQL 93*, zaczęto również torować drogę nowym rozwiązaniom takim jak *NoSQL* czyli bazy danych o nieuporządkowanej strukturze, pozbawionych ściśle zdefiniowanych schematów, zyskując większą elastyczność. Dodatkowo w określonych zastosowaniach tego typu bazy danych okazują się dużo szybsze niż bazy relacyjne. Szybkość ta jest tym większa, im większa jest przechowywana kolekcja danych. Są to, więc bazy wysoce skalowalne, choć mniej spójne niż standardowe.

Wśród systemów bazodanowych wykorzystywanych w aplikacji e-commerce, bardzo dużą popularnością cieszy się oprogramowania MySQL. Pomimo dużo mniejszych możliwości niż np. komercyjne rozwiązania Oracle czy MS SQL Server, omawiana baza danych jest elastyczna i łatwo zaadaptować ją do własnych potrzeb. Od czasu wprowadzenia zgodności ze standardem ACID, MySQL zaczął być szeroko wykorzystywany w e-biznesie.

Rysunek 1.4 przedstawia, jak wygląda architektura systemu baz danych MySQL, z punktu widzenia funkcjonalny komponentów [2][str. 26]. Pierwsza warstwa zawiera usługi, które wbrew pozorom nie są unikalne tylko dla omawianego oprogramowania. Są to usługi charakterystyczne dla większości narzędzi w architekturze sieciowej. Wyróżniono więc obsługę połączenia, uwierzytelnianie itd. Druga warstwa wprowadza wiele zmian i komponentów specyficznych dla MySQL'a. Wyróżniamy więc komponenty odpowiedzialne za parsowanie zapytań, a także ich optymalizację, buforowanie oraz kod odpowiedzialny za implementacje wbudowanych funkcji (np. data, czas, funkcje matematyczne i kryptograficzne). Każda funkcjonalność oferowana przez którykolwiek z silników składowania danych (*storage engine*), ma tu swoje miejsce (np. procedury użytkownika, wyzwalacze oraz widoki).

Trzecia warstwa wyróżnia wszystkie silniki składowania, które odpowiedzialne są bezpośrednio za przechowywanie i pobieranie danych w MySQL'u.



Rys. 1.4: Schemat architektury MySQL

Każdy z silników, ma różne zastosowania (podobnie jak różne systemy plików w systemach operacyjnych). Komunikacja z każdym z nich odbywa się wykorzystując wewnętrzne API. Wspomniany interfejs ukrywa różnice w specyfice każdego z mechanizmów, przez co zapytania są bardziej abstrakcyjne i prostsze w użyciu dla użytkownika końcowego. Podobnie jak w wypadku serwerów WWW, między mechanizmem składowania a serwerem MySQL występuje zależność: żądanie - odpowiedź. Oznacza to, że serwer wysyła żądanie i oczekuje na odpowiedź.

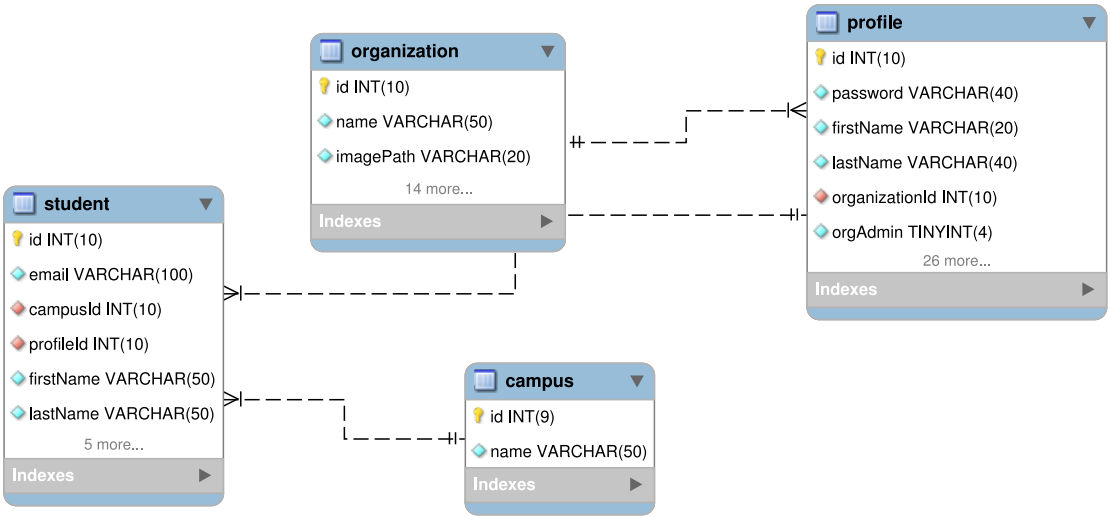
Najlepszą strategią optymalizacji, jest szukanie miejsc aplikacji, które działają najwolniej. Utworzono następujący schemat bazy danych (rys. 1.5). Na podstawie rysunku widać zależność studenta, który przynależy do 1 profilu (nauczyciela), który z kolei należy do organizacji. Podobnie student może należeć do jednego z istniejących kampusów.

Jak zachowa się baza danych przy próbie stworzenia alfabetycznego indeksu studentów, których nazwiska zaczynają się na daną literę (listing 1.3)?

Listing 1.3: Zapytanie do wyświetlenia menu książki adresowej uczniów

```
1 SELECT left(s.lastName, 1) letter, COUNT(s.id) students
2 FROM student s
3 WHERE s.lastName > 'A'
4 GROUP BY letter
```

Rezultat działania zapytania SQL widoczny jest na listingu 1.4. Jak widać, zapytanie wykonuje się w czasie 40 milisekund. Jest to dosyć krótko,



Rys. 1.5: Schemat testowej bazy danych

ale wynika to z głownie z rozmiarów kolekcji danych (20244 studentów). Wraz z rozrastaniem się tej tabeli, czas potrzebny na wykonanie tego zapytania będzie się systematycznie powiększał. Dzieje się tak, ponieważ tabela studentów nie posiada żadnego indeksu.

Listing 1.4: Wynik zapytania z listingu 1.3

```
+--------+----------+
| letter | students |
+--------+----------+
| A      |      999 |
| B      |     1749 |
| C      |      814 |
| D      |      670 |
| E      |      385 |
| F      |      918 |
| G      |     1599 |
| H      |      808 |
| I      |      245 |
| J      |      194 |
| K      |     1695 |
| L      |     1156 |
| M      |     1411 |
| N      |      492 |
| O      |      240 |
| P      |      686 |
| Q      |       14 |
| R      |     1259 |
| S      |     2662 |
| T      |      499 |
| U      |       57 |
| V      |      244 |
| W      |      698 |
| Y      |      316 |
```

```
| Z      |      400 |
+--------+----------+
25 rows in set (0.04 sec)
```

Czym jest indeks?

Indeks jest strukturą danych przeznaczoną do pomocy systemowi baz danych w efektywnym pobieraniu informacji z tabel. Są one często wymagane dla zapewnienia dobrej wydajności. Indeksy są szczególnie ważne w momencie kiedy baza danych się rozrasta, ponieważ ilość elementów do przeszukiwania wierszowego ulega zwielokrotnieniu.

Podczas zwykłego wyszukiwania wartości w bazie danych, program musi przeszukać każdą kolumnę, każdego wiersza w poszukiwaniu określonej wartości. W wypadku indeksów sprawa jest uproszczona ponieważ dysponujemy pewnym podzbiorem wartości np. z danej kolumny lub wyrażenia. W ten sposób silnik bazy danych wie, że dana wartość znaleziona w określonym indeksie powiązana jest z określonym rekordem, więc odpowiedź jest błyskawiczna.

Oczywiście wykorzystywanie indeksów wiąże się również z pewną zajętością danych, ponieważ oprócz danych w tabeli, trzeba dodatkowo przechowywać dane indeksów. Przeliczając jedna zyski do strat, większość przemawia jednak za stosowaniem indeksów.

1.3.1 Optymalizacja table przy użyciu indeksów

Optymalizacja zapytań przy użyciu indeksów jest stosunkowo prosta i polega na stworzeniu indeksu, który najlepiej pasuje do wyszukiwanej zawartości. W naszym wypadku potrzeba indeksu przechowujący pierwszą literę nazwiska. Z drugiej strony sortowanie po nazwisku lub nawet wyszukiwanie po nim jest dość częstą operacją dlatego warto stworzyć kompletny indeks dla pola `lastName` tabeli student (1.5). Na podstawie wyniku uzyskanego w listingu 1.6, widać znaczną poprawę czasu wykonywania. Wszystkie zapytania wykorzystujące kolumnę `lastName`, powinny wykonywać się zdecydowanie szybciej.

Listing 1.5: Utworzenie indeksu na polu nazwiska dla tabeli student

```
ALTER TABLE `student` ADD INDEX `lastName_idx`(`lastName`);
```

Listing 1.6: Wynik zapytania z listingu 1.3 po optymalizacji indeksu

```
+--------+----------+
| letter | students |
+--------+----------+
| A      |      999 |
| B      |     1749 |
| C      |      814 |
| D      |      670 |
```

```
| E       |       385 |
| F       |       918 |
| G       |      1599 |
| H       |       808 |
| I       |       245 |
| J       |       194 |
| K       |      1695 |
| L       |      1156 |
| M       |      1411 |
| N       |       492 |
| O       |       240 |
| P       |       686 |
| Q       |        14 |
| R       |      1259 |
| S       |      2662 |
| T       |       499 |
| U       |        57 |
| V       |       244 |
| W       |       698 |
| Y       |       316 |
| Z       |       400 |
+--------+-----------+
25 rows in set (0.01 sec)
```

Kolejność złączeń

Złączenia (ang. *joins*), są operacją spajającą ze sobą dwie lub więcej tabel. Dopuszczalne są łączenia 2 różnych tabel lub tej samej (*self-joins*). Używanie złączeń wynika w dużej mierze z normalizacji bazy danych, a co za tym idzie usunięcia nadmiarowości z rekordów tabel. Takie podejście w dużej mierze poprawia spójność danych, ale również może pogorszyć w dużej mierze wydajność zapytań.

Listing 1.7: Kilka możliwych do wykorzystania złączeń

```
1 SELECT SQL_NO_CACHE * FROM profile p inner join student s on s.
      profileId = p.id
2 SELECT SQL_NO_CACHE * FROM profile p left join student s on s.
      profileId = p.id
3 SELECT SQL_NO_CACHE * FROM profile p right join student s on s.
      profileId = p.id
4 SELECT SQL_NO_CACHE * FROM student s inner join profile p on s.
      profileId = p.id
5 SELECT SQL_NO_CACHE * FROM student s left join profile p on s.
      profileId = p.id
6 SELECT SQL_NO_CACHE * FROM student s right join profile p on s.
      profileId = p.id
```

Standard ANSI wyróżnia 4 rodzaje złączeń: INNER, OUTER, LEFT, RIGHT. W specjalnych okolicznościach tabela może być również połączona sama ze sobą. Zasadnicza różnica w specyfice polega na kryterium łączenia - w wypadku *INNER JOINÓW* wymagane jest istnienie odpowiadających sobie krotek po dwóch stronach relacji, natomiast *OUTER JOIN* wymaga spełnienia tego kryterium przynajmniej na jednej ze strony relacji - odpowiednio lewej lub prawej.

Typ	Czas wykonywania [sec]	Ilość rekordów
`inner join student`	0.2557	20244
`left join student`	**5.2125 / 0.5635**	20463
`right join student`	0.2634	20244
`inner join profile`	0.2726	20244
`left join profile`	0.3272	20463
`right join profile`	**5.2236 / 0.5704**	20463

Tab. 1.2: Tabela z rezultatem działania poszczególnych zapytań

Różnice między złączeniami przekładają się również na wydajność działania, najpopularniejsze ze złączeń *inner join'y* są najszybsze, podczas, gdy pozostałe przeznaczone są do bardziej specyficznych zastosowań.

Kolejny z przeprowadzanych testów, będzie polegał na złączeniu ze sobą studentów oraz profili, innymi słowy wyświetleniu wszystkich studentów przynależnych do któregoś z profili. Wynik działania poszczególnych testów (listing 1.7) został zestawiony w tabeli 1.2. Wyniki zapytań stanowią potwierdzenie ogólnie przyjętych zasad optymalizacji zapytań, *inner join* okazał się najszybszym ze złączeń, jednocześnie widać, że kolejność wykonywania złączeń również ma wpływ na wydajność. Zazwyczaj powinno się zaczynać od tabeli, która posiada miej rekordów (w tym wypadku tabela profile). Ogromna różnica między czasem wykonywania *left joina* oraz *right joina* w odwrotnej kolejności łączenia wynika z braku indeksu dla pola `profileId` tabeli `student`. Po dodaniu indeksów widać 10-krotną poprawę szybkości wykonywania tego zapytania.

1.3.2 Wykorzystanie polecenia `EXPLAIN` do optymalizacji zapytań.

Przedstawione dotychczas przypadki były stosunkowo proste do naprawy, często jednak zapytania są dużo bardziej rozbudowane i ciężkie do szybkiej dekompozycji. Warto wtedy skorzystać z udostępnianego przez MySQL narzędzia `EXPLAIN`. Oferuje on pomoc w zakresie dekompozycji bardziej skomplikowanych zapytań. Narzędzie to pokazuje m.in wykorzystane klucze dla łączeń, liczebność łączonych tabel, proponowane usprawnienia indeksów.

Jednym z zapytań, które może stanowić potencjalny problem w analizie jest zapytanie zaczerpnięte z istniejącej aplikacji opartej o przedstawiony na rysunku 1.5 diagram *ERD*. Przedstawione na listingu 1.8 zapytanie wyświetla szczegółowe statystyki liczebności, punktacji opartej o zsumowane wartości obecności na danych zajęciach, a także uczęszczanych programów, praktyk oraz wielu innych specjalistycznych kategorii.

Omawiane zapytanie jest prawdopodobnie jednym z trudniejszych, na ja-

kie można kiedykolwiek trafić podczas analizy bazodanowej. Głównym problemem takich zapytań jest istnienie wielu powiązanych z nim *podzapytań skorelowanych* czyli takich, których wynik zależy od zapytania głównego. W celu analitycznej analizy tego zapytania użyto narzędzie `EXPLAIN`. Składnia tej dyrektywy jest prosta i polega na dodaniu jej na początku wcześniejszego zapytana. Wynik takiego zapytania zwróci ilość wierszy równą ilości wszystkich podzapytań wykonywanych w trakcie przetwarzania (1.3).

Na podstawie tabeli widać, że na wynik zapytania składają się rezultaty łącznie 21 mniejszych zapytań. Za każdym razem kiedy złączamy jedną tabelę z drugą lub wykonujemy zapytanie wewnętrzne lub skorelowane ilość pracy do wykonania ulega zwiększeniu. Otrzymane zapytanie zawiera informacje podzielone na 6 kolumn. Każda z nich ma inne znaczenie i prezentuje określoną cechę badanego fragmentu zapytania [**?**].

Listing 1.8: Bardziej rozbudowane zapytanie SQL

```
SELECT o.id, o.name,

(SELECT group_concat(oc.campusId SEPARATOR ' ' ) FROM
    organizationCampus oc WHERE oc.organizationId = o.id) as
    campusIds,

(SELECT count(s5.id)
FROM profile p5
INNER JOIN student s5 on s5.profileId = p5.id
WHERE p5.organizationId = o.id) as students,

IFNULL((SELECT count(DISTINCT s3.id)
FROM profile p3
INNER JOIN student s3 on p3.id = s3.profileId
INNER JOIN studentIntensiveProgram sip on sip.studentId = s3.id
WHERE p3.organizationId = o.id
GROUP BY p3.organizationId
),0) as programs,

IFNULL((SELECT round(SUM(classes*1 + 1on1*3 + shabbaton*5 +
    socialEvents*0.5 + shabbosMeals*2))
FROM
student s2
INNER JOIN profile p2 on p2.id = s2.profileId
INNER JOIN `reportStudentAttendance` AS `r1` ON r1.studentId = s2.
    id
INNER JOIN `report` AS `ra`
ON ra.id = r1.reportId
and ((ra.month >= 9 and ra.year = 2011) or (ra.month <=8 and ra.
    year = 2012))
WHERE p2.organizationId = o.id
GROUP BY p2.organizationId ), 0) as score,

(SELECT ifnull(sum(datediff("2012-08-30", sy.startDate) BETWEEN 30
    and 90
      and sy.startDate between "2011-09-01" and "2012-08-30"),0)
FROM profile p4
INNER JOIN student s4 on s4.profileId = p4.id
INNER JOIN studentYeshiva sy on sy.studentId = s4.id
```

```
34 WHERE p4.organizationId = o.id) as yeshiva_1_3,
35
36 (SELECT ifnull(sum(datediff("2012-08-30", sy.startDate) BETWEEN 91
       and 180
37       and sy.startDate between "2011-09-01" and "2012-08-30"),0)
38 FROM profile p4
39 INNER JOIN student s4 on s4.profileId = p4.id
40 INNER JOIN studentYeshiva sy on sy.studentId = s4.id
41 WHERE p4.organizationId = o.id) as yeshiva_4_6,
42
43 (SELECT ifnull(sum(datediff("2012-08-30", sy.startDate) > 181
44       and sy.startDate > 2000),0)
45 FROM profile p4
46 INNER JOIN student s4 on s4.profileId = p4.id
47 INNER JOIN studentYeshiva sy on sy.studentId = s4.id
48 WHERE p4.organizationId = o.id) as yeshiva_6,
49
50 IFNULL((SELECT sum(case
51 when s1.beganSo = "spring/2011" then "2011-01-01"
52 when s1.beganSo = "summer/2011" then "2011-05-01"
53 when s1.beganSo = "fall/2011" then "2011-09-01"
54 else "2012-08-30"
55 end >= "2011-09-01" and right(s1.beganSo, 4) >= 2011 and s1.beganSo
       != "before")
56 FROM student s1
57 INNER JOIN profile p1 on s1.profileId = p1.id
58 WHERE p1.organizationId = o.id
59 GROUP BY o.id),0) AS `so`
60 FROM organization o
61 GROUP BY o.id
62 HAVING score > 0
63 ORDER BY o.name
```

Opis pól wynikowych polecenia `EXPLAIN`.

id
określa numer porządkowy zapytania w kontekście planu wykonywania.

select_type
określa rodzaj użyty rodzaj zapytania, dostępne wartości to `PRIMARY` (zapytanie główne), `UNION` (zapytanie typu `UNION`, drugie lub dalsze zapytanie w kolejności polegające na dodanie rezultatu jednego zapytania do drugiego, bez wyróżnienia kryteriów łączenia). `DEPENDENT UNION` zależy dodatkowo od zapytania głównego, `SUBQUERY` - pierwsze użycie `SELECT` w podzapytaniu. `DEPENDENT SUBQUERY` - podobnie jak w poprzednim przypadku, ale zależy od zapytania głównego.`DERIVED` - podzapytanie umieszczone w sekcji `FROM`. `UNCACHEABLE SUBQUERY / UNION`, to dwa najgorsze typy zapytania, ponieważ ich wynik musi być wykonywany dla każdego wiersza głównego zapytania osobno.

table
odnosi się do nazwy tabeli lub aliasu aktualnie przetwarzanego zapytania.

type

rodzaj zasobu użytego do złączenia, najpopularniejsze wartości to: `system` (złączenie z systemową tabelą, posiadającą jeden wiersz, specjalna odmiana złączenia `const`), najszybszy rodzaj złączenia, ale mało jest praktycznych zastosowań. `Const` - złączana tabela ma co najwyżej jeden pasujący element przez co pobierana jest tylko raz, przez co wiersz jest traktowany jako stała. Dotyczy porównania pewnej stałej wartości z **pełną** wartościami *klucza głównego* lub unikalnego indeksu.

`Eq_ref`, jest dostępny w sytuacjach kiedy porównujemy wartości z indeksów przy pomocy operatora `=` i istnieje dokładnie jedna wartość do pobrania dla każdego klucza z tabeli łączącej. Przyrównana wartość może być stałą lub wcześniej wczytaną kolumną innej tabeli. Kiedy istnieje więcej niż jedna wartość dla danego klucza tabeli łączącej, wtedy mamy do czynienia z typem `ref`. `Ref` występuje również wtedy, kiedy wykorzystano operatory `<=>`. Inną odmianą `ref` jest `ref_or_null`, występujące wówczas, gdy szukano również wierszy które są równe wartości `null`.

`ALL`, najgorszy możliwy przypadek złączenia, cała tabela jest przetwarzana i tworzone są wszystkie kombinacje wartości z tabeli łączącej. Trochę lepsza sytuacja jest w wypadku `INDEX`, kiedy to zapytanie wykorzystuje kolumny, które są częścią pojedynczego indeksu.

possible_keys
pokazuje proponowane klucze, które można wykorzystać podczas łączenia. W praktyce nie wszystkie z nich można faktycznie użyć.

key
wykorzystane do łączenia klucze, jeśli to pole jest puste, podobnie jak poprzednie, to wskazówka, że należy dodać indeks.

Na podstawie omówionej analizy, widać, że dla podzapytania nr 2 występuje rodzaj złączenia typu `index`, oznacza to, że można je usprawnić dodając indeks szczegółowy. Ponieważ tabela `organizationCampus`, do której odnosi się alias `oc` jest tabelą łącznikową, jest tam użyty klucz główny złączony. Dlatego też tylko część tego indeksu jest wykorzystana. Po dodaniu szczegółowego indeksu na pole `organizationId`, osiągnięto pożądany typ *ref*, a czas wykonywania zapytania zredukował się o 40ms.

1.3.3 Optymalizacja definicji table.

Nie bez znaczenia dla ogólnej wydajności aplikacji jest również wykorzystanie właściwych dla przechowywanych treści typów pól. Istnieje szereg reguł pomocnych w tworzeniu zoptymalizowanych schematów danych. Są to reguły niezależne od wykorzystanego systemu baz danych.

Małe znaczy lepsze.
Zazwyczaj, należy wykorzystywać możliwie najmniejszy typ danych, który

id	select_type	table	type	possible_keys	key
1	*PRIMARY*	o	ALL		
9	*DEPENDENT SUBQUERY*	p1	ref	PRIMARY,organizationId	organizationId
9	*DEPENDENT SUBQUERY*	s1	ref	profile_idx	profile_idx
8	*DEPENDENT SUBQUERY*	p4	ref	PRIMARY,organizationId	organizationId
8	*DEPENDENT SUBQUERY*	s4	ref	PRIMARY,profile_idx	profile_idx
8	*DEPENDENT SUBQUERY*	sy	ref	studentIdx	studentIdx
7	*DEPENDENT SUBQUERY*	p4	ref	PRIMARY,organizationId	organizationId
7	*DEPENDENT SUBQUERY*	s4	ref	PRIMARY,profile_idx	profile_idx
7	*DEPENDENT SUBQUERY*	sy	ref	studentIdx	studentIdx
6	*DEPENDENT SUBQUERY*	p4	ref	PRIMARY,organizationId	organizationId
6	*DEPENDENT SUBQUERY*	s4	ref	PRIMARY,profile_idx	profile_idx
6	*DEPENDENT SUBQUERY*	sy	ref	studentIdx	studentIdx
5	*DEPENDENT SUBQUERY*	p2	ref	PRIMARY,organizationId	organizationId
5	*DEPENDENT SUBQUERY*	s2	ref	PRIMARY,profile_idx	profile_idx
5	*DEPENDENT SUBQUERY*	r1	ref	student_idx,report_idx	student_idx
5	*DEPENDENT SUBQUERY*	ra	eq_ref	PRIMARY,month_idx,year_idx	PRIMARY
4	*DEPENDENT SUBQUERY*	p3	ref	PRIMARY,organizationId	organizationId
4	*DEPENDENT SUBQUERY*	s3	ref	PRIMARY,profile_idx	profile_idx
4	*DEPENDENT SUBQUERY*	sip	ref	student_idx	student_idx
3	*DEPENDENT SUBQUERY*	p5	ref	PRIMARY,organizationId	organizationId
3	*DEPENDENT SUBQUERY*	s5	ref	profile_idx	profile_idx
2	*DEPENDENT SUBQUERY*	oc	index		PRIMARY

Tab. 1.3: Wynik działania polecenia `EXPLAIN`

jest w stanie prawidłowo przechować powierzone dane. Mniejsze typy danych są szybsze ponieważ zużywają mniej miejsca na dysku, w pamięci i w pamięci podręcznej procesora. Dodatkowo mniejsza ilość cykli procesora jest potrzebna do ich przetworzenia.

Proste typu są wydajniejsze.
Podobnie jak w poprzednim przykładzie, proste typy danych wymagają mniejszych nakładów pracy procesora do przetworzenia. Dla porównania, typ całkowity jest szybszy w porównywaniu niż typ znakowy. Dzieje się tak ponieważ razek ze znakiem powiązane są informacje o kodowaniu oraz tzw. *collations* czyli regułach sortowania. Inne ważne przykłady dotyczą np. przechowywania daty i godziny w dedykowanych typach danych zamiast typu łańcuchowego. Z drugiej strony adres IP powinien być przechowywany jako liczba całkowita.

Unikanie wartości `NULL`
Tak często jak to tylko możliwe pola tabeli powinny być definiowane jako `NOT NULL`. Bardzo często schematy baz danych przechowują wartość `NULL`, nawet jeśli tego nie potrzebują. Należy mieć na uwadze, że optymalizator zapytań dużo gorzej radzi sobie z takimi kolumnami, ponieważ sprawiają one że indeksy, ich statystyki i porównywanie z wartościami jest bardziej skomplikowane. Nawet jeśli konieczne jest odnotowanie w polu pustej wartości, lepiej użyć jakiegoś arbitralnego symbolu np. 0 lub pusty łańcuch znakowy.

Dodatkowo istnieje kilka przesłanek dotyczących wykorzystywania pól typu `VARCHAR` i `CHAR`. Zazwyczaj lepiej jest wykorzystywać typ `VARCHAR` (typ znakowy o zmiennej długości), kiedy maksymalna szerokość kolumny jest dużo większa niż wartość średnia, a ilość aktualizacji tego pola jest mała, przez co fragmentacja nie jest problemem. Nie bez znaczenia jest też wybrane kodowanie, w standardzie UTF-8 każdy znak wykorzystuje różną wartość bajtów przestrzeni dyskowej.

Typ `CHAR` z kolei najlepiej spisuje się w sytuacjach kiedy przechowywane mają być bardzo krótkie łańcuchy lub każdy łańcuch jest praktycznie takiej samej długości. Dla przykładu, `CHAR` jest dobrym wyborem do przechowywania zahaszowanych wartości haseł użytkowników. W odróżnieniu od `VARCHAR`a, ten typ danych nie ulega fragmentacji ze względu na stałą szerokość pola.

Inną cenną radą jest stosowanie pól wyliczeniowych `ENUM` jeśli zbiór dostępnych wartości znakowych pola jest stały i nie będzie podlegał późniejszym zmianom.

1.4 Podsumowanie

Na łamach pierwszego rozdziału zostały przedstawione podstawowe metody optymalizacji aplikacji WWW. Omówienie miało na celu ogólne wprowadze-

nie w tematykę analizy oprogramowania działającego w sieci. Istnieje wiele technik optymalizacji front-endu, baz danych czy serwera WWW, większość z nich jest dostosowana do konkretnych problemów wydajnościowych lub skonfigurowanej infrastruktury. Omówione zagadnienia są jednak na tyle ogólne, że można je odnieść z powodzeniem do aplikacji w dowolnym języku programowania, a także dowolnym silniku bazodanowym. Znając jednak podstawy, z powodzeniem można wykonać dalsze etap optymalizacji na własną rękę. Przedstawione porady zapewnią natomiast solidny fundament tworzonej aplikacji.

Kolejny rozdział ma na celu przedstawienie architektury tworzonego oprogramowania. Będzie to rozwinięcie o praktyczne wskazówki rozdziału dotyczącego optymalizacji serwera WWW.

Rozdział 2

Wymagania i budowa aplikacji

Nie sposób rozpocząć implementacji rozwiązania, bez kompletnego opisu wymagań i zaplanowania budowy aplikacji. Jest to jeszcze ważniejsze, kiedy celem jest stworzenie wydajnej aplikacji.

Aplikacja powinna spełniać wszystkie oczekiwane cele biznesowe, jak również zapewniać prawidłowe funkcjonowanie bez względu na aktualnie panujące obciążenie. Aplikacja demonstracyjna będzie księgarnią internetową oferującą wiele typowych funkcji, charakterystycznych dla tej branży.

Głównym celem tworzonego oprogramowania jest odpowiednie wyważenie pracy wykonanej po stronie serwera, jak również po stronie przeglądarki. Dlatego też duży nacisk pracy został położony na utworzenie usług udostępniających interfejs dostępowy do bazy danych. W ten sposób wykorzystując architekturę REST kod JavaScript może dokonywać modyfikacji modelu, przy znikomym udziale serwera WWW.

2.1 Zakres funkcjonalny

Projektowana aplikacja stanowi wirtualną księgarnię, podobnie jak w wypadku jej realnego odpowiednika, zapewnia możliwość przeglądania zasobów, podzielonych na kategorie lub oznaczonych określonymi słowami kluczowymi. W odróżnieniu od prawdziwej księgarni, w internetowych aplikacjach konieczne jest zdefiniowanie pewnej tożsamości, która będzie później podstawą do zakupu książki, lub sprawdzenia stanu zamówienia.

Dlatego też istotne jest stworzenie spójnego systemu uwierzytelniania, zapewniającego zarówno bezpieczeństwo, jak również łatwość ewentualnego przypomnienia zapomnianego hasła.

Ponieważ oferta książkowa, ciągle się zmienia, na rynek trafiają nowe książki, a nawet tworzą się nowe gatunki, istotną rolę w tworzonej aplikacji stanowi współistnienie zarówno części przeznaczonej dla zwykłego użytkow-

Część Administracyjna	Sekcja użytkownika
Dodawanie książek	Rejestracja
Edycja książek	Edycja i podgląd konta
Usuwanie książek	Ocenianie książek
Moderacja wpisów	Komentowanie książek
Dodawanie kategorii	Przeglądanie kategorii
Dodawanie plików do książek	Pobieranie streszczeń książek
Nadzorowanie kont użytkowników	Wyszukiwanie książek w wyszukiwarce

Tab. 2.1: Tabela z wykazem funkcjonalności

nika, jak również części administracyjnej (*backend*). Sekcja ta powinna być również zabezpieczona przed potencjalnym włamaniem lub złośliwym atakiem ze strony intruzów.

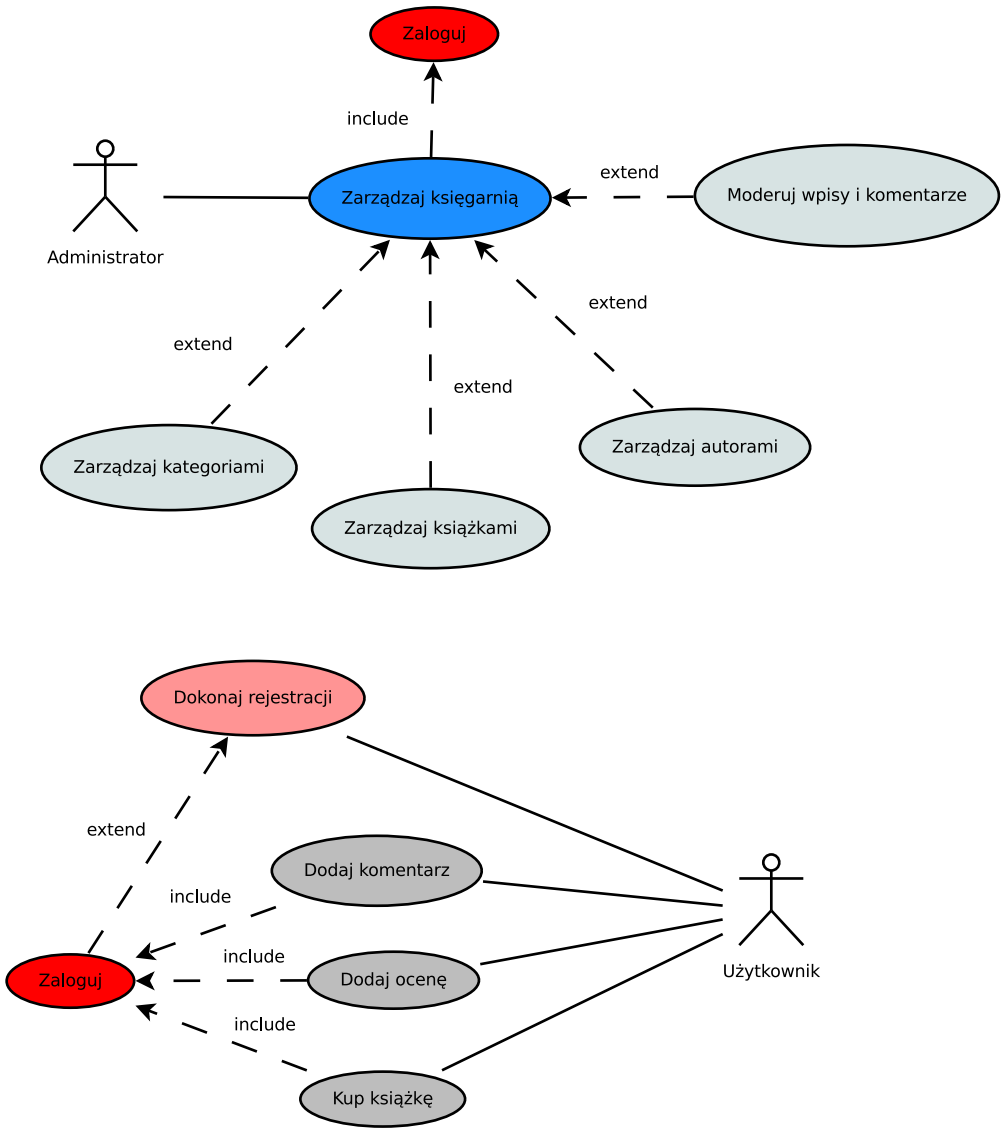
Ponieważ obecnie w dobie ery WEB 2.0 strony oferują dużo większą iterakcję, niż było to możliwe na początku ery Internetu. Dlatego też projektowana aplikacja daje również możliwość oceny i komentowania istniejących zbiorów. Jest to bardzo ważne z punktu widzenia biznesu i sprzedaży, ponieważ popularność i przychylne noty, z pewnością napędzą chętnych do kupna.

Wymagania podzielono na część administracyjną i część użytkownika, natomiast zbiór funkcjonalności przedstawiony jest w tabeli 2.1.

Rysunek 2.1 przedstawia przypadki użycia wyszczególnione dla tworzonej aplikacji. Warto nadmienić, że większość akcji użytkownika wymaga wcześniejszego zalogowania, a na początku rejestracji konta w wypadku użytkownika księgarni.

Aplikacja powinna w jak największym stopniu odciążać serwer WWW od niepotrzebnych zapytań, w tym celu, akcje usuwania, oceniania i komentowania książek będą realizowane przez kod JavaScript. Dodatkowo część walidacji danych będzie w pierwszej fazie wykonywana również przy użyciu frontendu, dopiero w wypadku wyłączenia wykonywania skryptów, wysłane zostanie żądanie do sprawdzenia przez serwer. Takie podejście powinno znacznie zredukować obciążenie np. podczas rejestracji użytkowników, ponieważ serwer wykonuje tylko minimum swojej pracy (*Lazy Loading*).

Należy jednak mieć na uwadze bezpieczeństwo oprogramowania, dlatego nie należy polegać w 100% na walidacji wykonywanej przez JavaScript, ponieważ użytkownik może bez problemu wyłączyć działanie skryptów. Dlatego też podczas tworzenia tego typu rozwiązań walidacja po stronie klienta powinna iść zawsze w parze z walidacją po stronie serwera.



Rys. 2.1: Przypadki użycia tworzonej aplikacji

Ze względu na konserwacje i rozwój tworzonej aplikacji, jest ona zarządzana przy pomocy systemu kontroli wersji git. W ten sposób tworzony kod jest pod ścisłą kontrolą, może być potem rozwijany przez więcej niż jednego programisty, eliminując konflikty podczas pracy na wspólnych zasobach.

Aplikacja wykorzystuje język HTML5, który staje się powoli standardem tworzenia kodu po stronie klienta. Zawiera bowiem wiele nowoczesnych mechanizmów i udogodnień w stosunku do poprzedniej wersji np. HTML4 oraz XHTML 1.0. Mając na uwadze prawidłowe funkcjonowanie w poszczególnych przeglądarkach i różną implementacje nowego standardu, aplikacja wykorzystuje dodatek `HTML5 Boilerplate` oraz `Twitter Bootstrap` jako metodę na otrzymanie podobnych rezultatów w szerokiej gamie urządzeń (w tym urządzenia mobilne).

Projektowana aplikacja wykorzystuje technologię AJAX (ang. Asynchronous JavaScript and XML, asynchroniczny JavaScript i XML), dzięki czemu możliwa jest symulacja zachowania aplikacji desktopowej. Reakcja na akcje użytkownika następuje bez potrzeby przeładowania strony. Pozwala to na dużą oszczędność przepustowości i czasu koniecznego do przetworzenia całej strony. Dzięki zastosowaniu frameworka jQuery dla języka JavaScript, możliwe jest stosowanie bardziej przyjaznego interfejsu użytkownika, przy jednoczesnym zapewnieniu kompatybilności wstecz z istniejącymi wersjami przeglądarek internetowych.

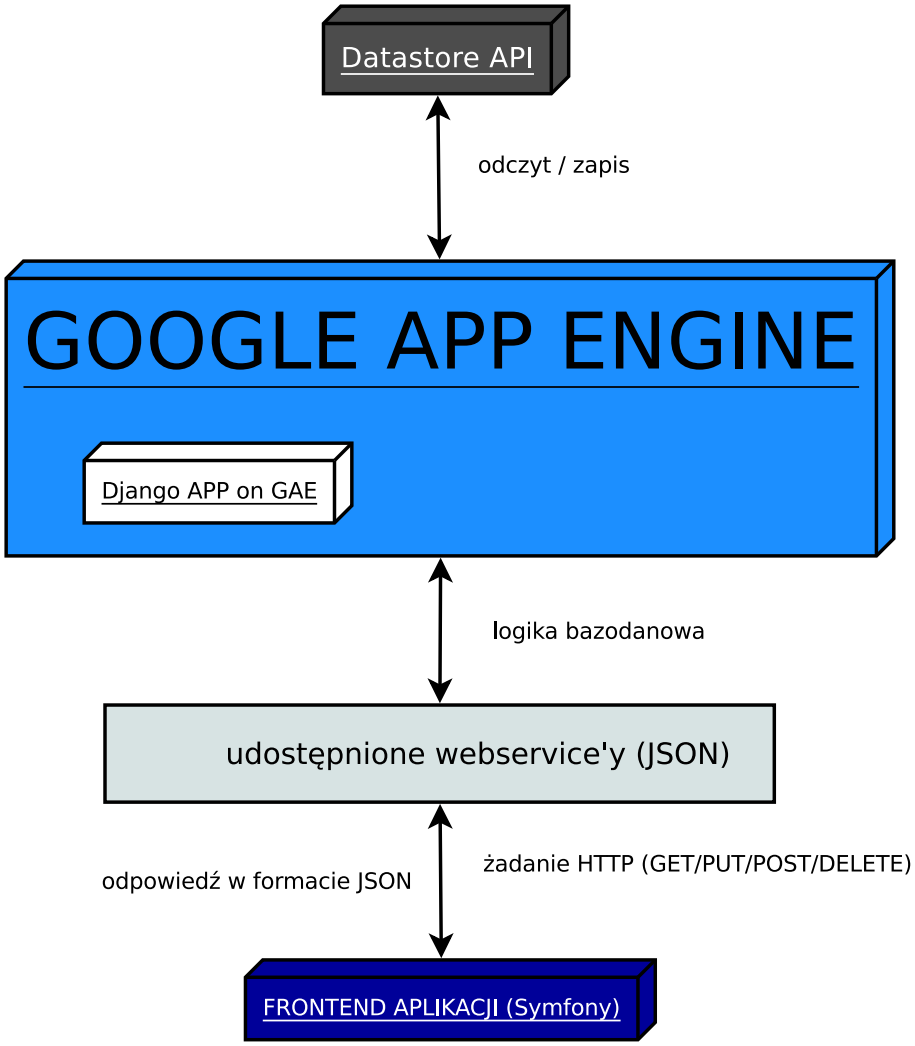
2.2 Budowa aplikacji

Projektowana aplikacja składa się tak naprawdę z 2 podaplikacji, jedna z nich hostowana jest wykorzystując usługę *Google Application Engine*, która zostanie omówiona w rozdziale 3. Aplikacja ta będzie udostępniała usługi oparte o architekturę *REST* (ang. *Representation State Transfer*). Aplikacja ta będzie zawierała również panel administracyjny do zarządzania księgarnią. Jest ona napisana w języku Python, wykorzystując framework Django, a także wtyczkę Piston umożliwiającą proste wystawienie webserwiceów na podstawie modelu bazy danych.

Druga aplikacja będzie zawierała zakres funkcjonalny zwykłego użytkownika, natomiast wszystkie dane będzie pobierała przy użyciu udostępnionych przez wcześniej omawianą aplikacje *webserviceów*. Aplikacja ta będzie napisana w języku PHP z wykorzystaniem frameworka symfony.

Obydwie aplikacje wykorzystują wzorzec MVC do separacji warstw logiki, prezentacji i danych, dodatkowo, wykorzystane frameworki są stworzone do szybkiego tworzenia oprogramowania, dzięki założeniom *scaffoldingu* oraz konwencją *DRY* (ang. *Don't repeat Yourself*) i *KISS* (ang. *Keep it simple stupid*).

Schemat budowy aplikacji przedstawia rysunek 2.2. Każdy element w systemie posiada szereg zależności, których nieprawidłowe działanie pociąga wiele daleko idących konsekwencji. Dlatego istotne jest, by kod aplikacji był spójny i posiadał modularną budowę. Umożliwi to łatwiejsze wykrywanie problematycznych aspektów aplikacji, istotnych dla prawidłowego działania całego projektu.



Rys. 2.2: Diagram budowy aplikacji

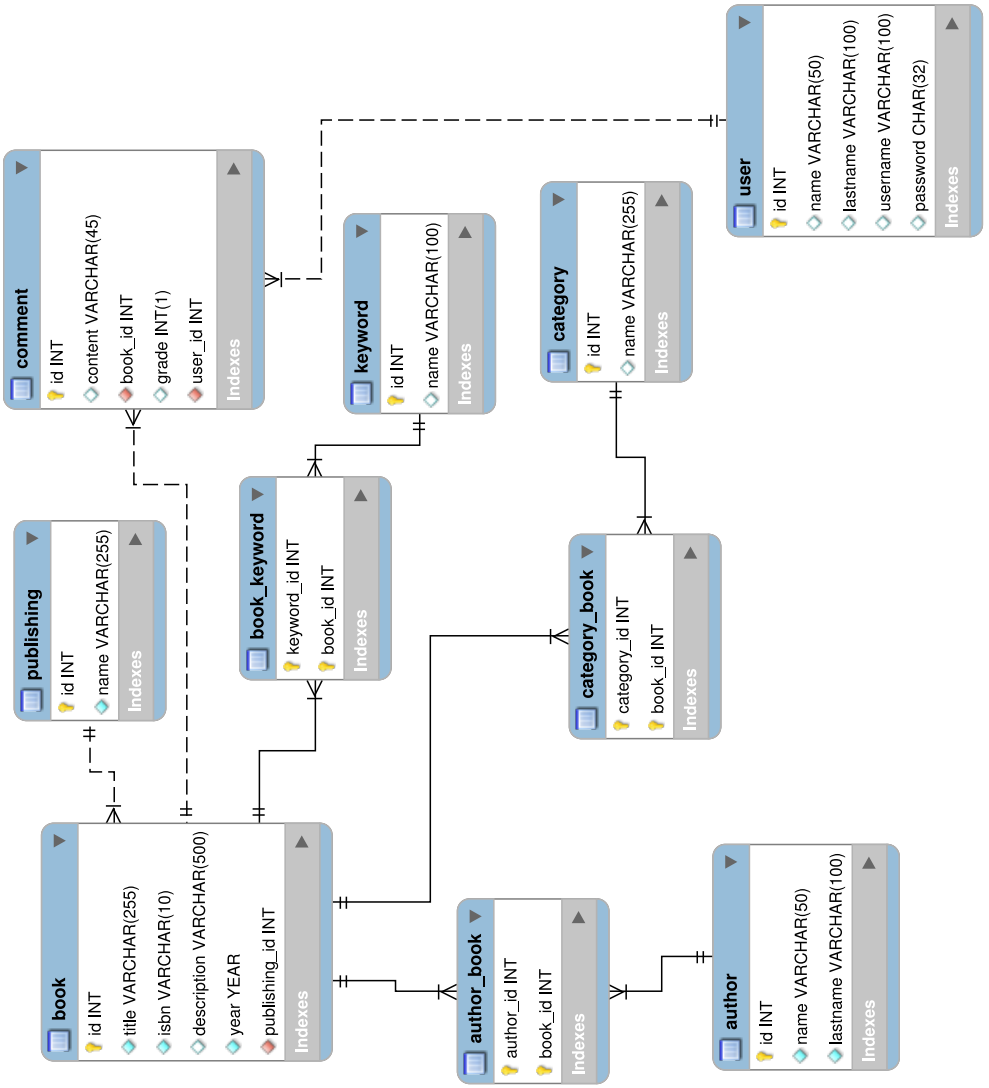
2.3 Projekt bazy danych

Rysunek 2.3 przedstawia projekt bazy danych wykorzystanej przy okazji projektu. Diagram ERD przedstawia jednak bazę w podejściu relacyjnym,

w późniejszej części pracy pokazana zostanie baza NoSQL, oraz proces denormalizacji, który musiał zostać dokonany. Na szczęście dzięki wspomnianemu w następnym rozdziale frameworkowi *Django-norel*, możliwe jest łatwe przechodzenie między istniejącymi implementacjami baz danych.

Baza zawiera książki, powiązane z nimi słowa kluczowe i kategorie. Każda książka ma swojego wydawcę. Dodatkowo użytkownicy mogą komentować wybrane książki, łącznie z wystawieniem oceny. Istotnym zamierzeniem projektowanego schematu była jak największa optymalizacja użytych pól. Np. ocena w komentarzu ma wartości między 1-5, więc wykorzystano typ danych `UNSIGNED INT(1) NOT NULL`, który przechowuje wartości od 0 do 255, bez składowania wartości pustej. Do przechowywania hasła wykorzystano typ `CHAR(32)` ponieważ w bazie przechowywana jest wartość skrótu hasła przy użyciu algorytmu *SHA1*.

Podczas projektowania należy mieć na uwadze specyfikę użycia poszczególnych tabel. Dla tabel kategorii i książek, dominować będą operacje odczytu, natomiast dla tabeli komentarzy operacje zapisu.



Rys. 2.3: Projekt bazy danych

Rozdział 3

Architektura aplikacji

Istotnym celem pracy jest zapewnienie możliwie najlepszej w danej chwili wydajności. Twierdzenie to powinno być prawdziwe również wówczas, kiedy aplikacja znajduje się pod silnym ruchem sieciowym. Aby to zapewnić, konieczne jest wykorzystanie odpowiedniej architektury aplikacji.

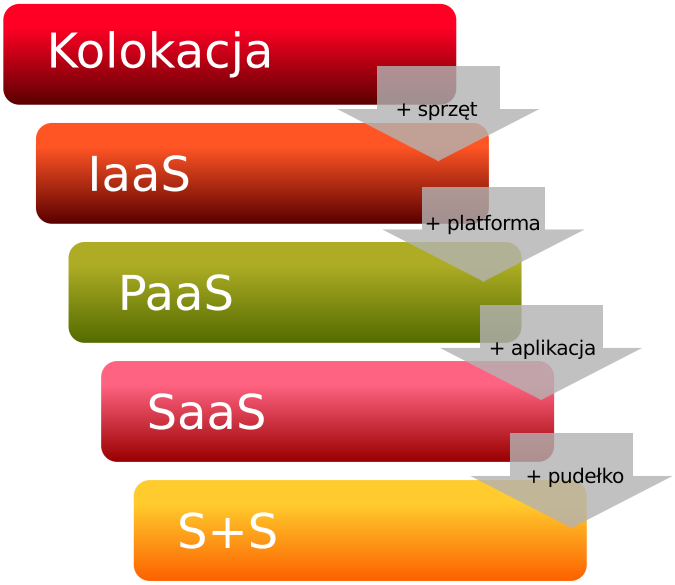
Od kilku lat na rynku IT można zaobserwować duże zainteresowanie związane z chmurami obliczeniowymi. Jeszcze większe zamieszanie na rynku spowodowało udostępnienie przez Google oraz Microsoft ich flagowych produktów znanych jako *Google Application Engine* oraz *Windows Azure*. Są to usługi kwalifikowane jako *PaaS* czyli *Platform as a Service*. Oznacza to, że korzystając z ich produktów otrzymujemy kompletne środowisko uruchomieniowe aplikacji oraz zaplecze technologiczne umożliwiające uruchamianie aplikacji. W wypadku GAE możliwe jest programowanie aplikacji z wykorzystaniem udostępnionego przez usługę zbioru bibliotek, napisanych w trzech językach: Java, Python oraz Go. Następnie przy pomocy udostępnionych narzędzi możliwe jest wdrożenie aplikacji (*Deployment*).

Takie podejście do tworzenia aplikacji internetowych zyskało olbrzymią ilość zwolenników, ponieważ pozwoliło na całkowite przeniesienie ciężaru zarządzania skomplikowaną infrastrukturą na producentów rozwiązań *Paas*. Innym ważnym powodem dla którego wielu ludzi zdecydowało się na wykorzystanie nowych usług, jest możliwość konfiguracji architektury do własnych potrzeb, a także (co było kluczowym powodem), do aktualnego obciążenia aplikacji.

3.1 Rodzaje chmur

Najogólniej chmury dzielą się na *prywatne* i *publiczne*. Prywatne wchodzą w skład części organizacji, ale jednoczesne stanowią autonomiczną usługę. Publiczne natomiast udostępniane są przez zewnętrznych dostawców usług.

Rysunek 3.1 pokazuje zasadnicze różnice między poszczególnymi rodzajami chmur. Chmura oferowana przez Google Application Engine, zapewnia wsparcie w zakresie architektury serwera, systemu plików, a także systemu bazodanowego. Dodatkowo GAE udostępnia również możliwość korzystania z usług w tle oraz serwerów mailowych.

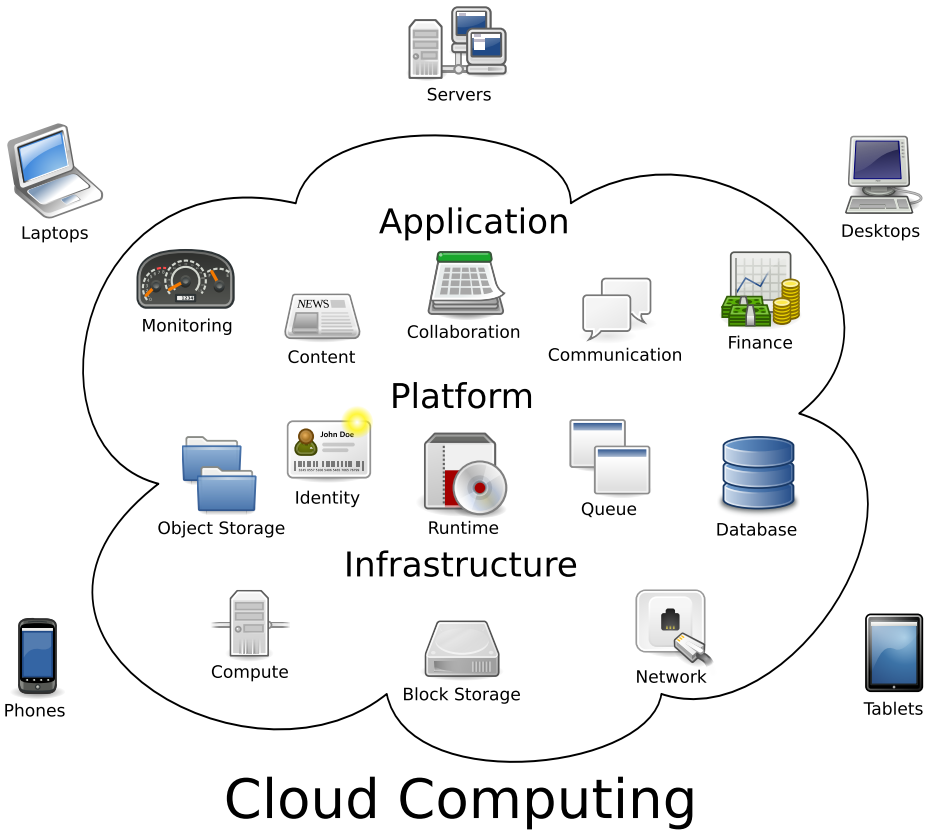


Rys. 3.1: Zestawienie rodzajów chmury ze względu na udostępniane zasoby

Jak, więc wynika z zestawienia rodzajów chmur, oferuje ona dużo więcej ponad standardową infrastrukturę sprzętową, jak np. Amazon EC2, który jako pierwszy oferował usługi typu *IAAS* (ang. *Infrastructure as a Service*).

Polityka chmur typu *PAAS* jest płacenie tylko za aktualne zużycie zasobów. Rozliczani jesteśmy wiec za ilość która faktycznie wykorzystujemy. Dodatkowo architektura dynamicznie zwiększa ilość zasobów, jak również instancji serwerów aktualnie wykorzystywanych, by jak najbardziej zmniejszyć czas odpowiedzi strony.

W wypadku GAE w ramach darmowych limitów do dyspozycji dostajemy 28h instancji, oznacza to, ze w wypadku pojedynczej najwolniejszej instancji będziemy mogli z niej korzystać 28h (600Mhz), w wypadku szybszej instancji (1200Mhz) czas ten zostanie 2-krotnie zmniejszony. W wypadku najszybszej instancji (2400Mhz) czas ten ulega 4-krotnemu skróceniu. Do decyzji użytkownika zostaje decyzja z jakiej instancji chce korzystać i oczywiście jak bardzo ilość serwerów będzie się zwiększała przy natężeniu ruchu. Jeśli darmowa oferta GAE jest niewystarczająca, w każdej chwili można przejść na opcje płatną, w której określamy tygodniowe / miesięczne limity płatności. Jedną z ważnych zmian w porównaniu z normalnymi hostingami



Rys. 3.2: Komponenty wchodzące w skład architektury w chmurze (źródło Wikipedia)

jest taryfikator operacji bazodanowych. Wyróżniono ceny za odczyt i zapis do bazy, odpowiednio 0.10$ za 100,0000 operacji zapisu i 0.07$ za podobną ilość operacji odczytu. Dodatkowo system liczenia operacji zapisu opera się na dodatkowych operacjach związanych z indeksowaniem danych. Przykładowo pobranie pojedynczego rekordu z bazy to koszt jednego odczytu, ale wykonanie zapytania zwracającego 20 rezultatów to koszt 21 operacji odczytu. W wypadku zapisu do bazy nowej encji, 2 operacji zapisu + 2 operacje zapisu na każdy indeks dla pola + 1 zapis dla wartości indeksu złożonego.

Rysunek 3.3 przedstawia poglądowy widok bieżących statystyk aplikacji udostępnianych przez GAE. Użytkownik ma dostęp do stanów bieżących limitów, a także szczegółowych wykresów wydajności aplikacji określanych na podstawie różnych czynników (zajętość pamięci, ilość operacji na sekundę, wykorzystanie instancji serwerów). Bardzo przydatny jest również wykres bieżącej aktywności aplikacji. W ten sposób można szybko ocenić, czy aplikacja jest obecnie pod dużym obciążeniem, a także porównać, czy np. wdrożone zmiany poprawiły jej wydajność.

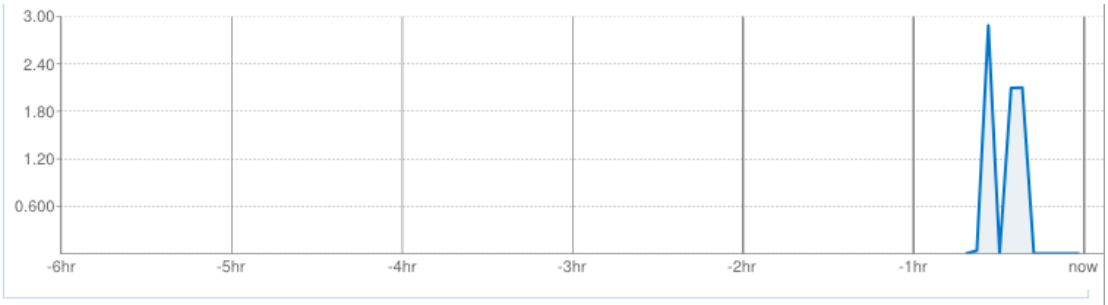
3.2 Nie relacyjne bazy danych NoSQL

Od początku ery informatyzacji informacja i jej przechowywanie było najważniejszym zadaniem powierzanym komputerom. Bazy danych muszą zarówno przechowywać dane (Zapis), jak i udostępniać interfejs służący ich odczytowi, a nawet wyszukiwaniu w oparci o zadane kryteria. Dotychczas wykorzystywano głównie relacyjne bazy danych, które łączyły pojedyncze tabele w różnego rodzaju asocjacje. Wczesnej jednak konieczne jest zdefiniowanie określonego schematu bazy danych, który będzie obowiązywał i nakładał ograniczenia na dane.

Obecnie jednak bardzo popularne staja się bazy bez określonej struktury, niesamowicie skalowalne, a przy tym dostęp do niech odbywa się przy pomocy *REST*owego API. Wyróżniono kilka konwencji:

- *Key Value Stores* - HDOOP, BigData
- *Graph Databases* - Neo'tj, AllegroGraph
- *BigTables* - HBase, Cassandra
- *Document Database* - MongoDB, CoachDB, SimpleDB

W projektach ecommerce największym zastosowaniem cieszą się bazy dokumentowe skupiające kolekcje obiektów - dokumentów. Każdy z nich może zawierać odpowiednie metadane i słowa kluczowe. Każdy dokument posiada dowolna definicje i może być dowolnie zagnieżdżamy i łączony z innymi dokumentami.



Instances

Number of Instances - Details	Average QPS	Average Latency	Average Memory
4 total	0.000	Unknown ms	50.0 MBytes

Billing Status: Free - Settings

Quotas reset every 24 hours. Next reset: 17 hrs

Resource		Usage
Frontend Instance Hours	7%	2.09 of 28.00 Instance Hours
Backend Instance Hours	0%	0.00 of 9.00 Instance Hours
Datastore Stored Data	0%	0.00 of 1.00 GBytes
Logs Stored Data	0%	0.00 of 1.00 GBytes
Task Queue Stored Task Bytes	0%	0.00 of 0.49 GBytes
Blobstore Stored Data	0%	0.00 of 5.00 GBytes
Datastore Write Operations	0%	0.00 of 0.05 Million Ops
Datastore Read Operations	15%	0.01 of 0.05 Million Ops
Datastore Small Operations	0%	0.00 of 0.05 Million Ops
Outgoing Bandwidth	0%	0.00 of 1.00 GBytes
Recipients Emailed	0%	0 of 100
Stanzas Sent	0%	0 of 10,000
Channels Created	0%	0 of 100

Current Load

URI	Requests last 7 hrs	Runtime MCycles last hr	Avg Latency last hr
/api/comments/2003/	1.82K	76	416 ms
/favicon.ico	14	20	40 ms
/api/comments/6006/	1	0	18 ms
/api/comments/6006	1		
/api/comments/2003	1		
/	1		

Errors

URI	Count	% Errors last 7 hrs
/favicon.ico	14	100%

Rys. 3.3: Wygląd panelu statystyk w ramach usługi GAE

Wraz z wykorzystaniem nie relacyjnych baz danych, zmienił się zupełnie pomysł na organizację danych. Siłą baz danych NoSQL jest brak ściśle określonej struktury, przez co schemat danych może się dowolnie zmieniać w trakcie rozwoju aplikacji. Innym ważnym powodem wykorzystania bazy Google Datastore oferowanej przez GAE jest szybkość i skalowalność. W odróżnieniu od zwykłego hostingu, baza danych w GAE może być rozproszona na dowolną ilość instancji.

Początkowo osoby korzystające z baz nierelacyjnych zarzucały takiemu rozwiązaniu następujące wady.

- Problem z ponownym wykorzystaniem kodu dla innych rozwiązań nie relacyjnym nawet kiedy implementacje są podobne
- Nie można wykorzystać kodu dla relacyjnej bazy danych, nawet jeśli schemat pozostał by nietknięty
- Konieczność zarządzania indeksami, a także problem denormalizacji itd.
- Konieczność samodzielnej implementacji łączenia rezultatów wielu zapytań
- Niektóre z rozwiązań są zintegrowane ze specyficznym dostawcą (App Engine, SimpleDB), więc użytkownik nie ma możliwości migracji na inną platformę bez dużego nakładu pracy

W projektowanej aplikacji wykorzystano framework *Django-nonrel* dodający obsługę baz NoSQL do frameworka *Django*. W ten sposób możliwe jest wykorzystanie abstrakcji bazodanowej i tworzenie rozwiązań niezależnie od technologii bazy. Wszystko to możliwe jest dzięki wbudowanemu narzędziu *ORM* (ang. Object-Relational Mapping), które udostępnia obiektowy interfejs dostępu do danych. W ten sposób pisana logika nie jest zależna od konkretnej platformy czy rozwiązania składującego.

3.3 Aplikacja zorientowana na usługi

Inną ważną cechą tworzonej aplikacji, jest wyszczególnienie usług (*webserviceów*), które pomogą w wykonywaniu operacji na modelu przy pomocy metod protokołu HTTP. Będą to więc popularne w dzisiejszych czasach usługi REST. Ze względu na specyfikę aplikacji i duży udział języka JavaScript w tworzonej logice, w większości wypadków usługi te będą zwracały w notacji JSON.

Kluczem i zaletą takiego podejścia, określanego później jako *Service Oriented Architecture* jest wydzielenie pewnych usług zawierających pewną

część / zakres funkcjonalny. W ten sposób aplikacja jest podzielona na grupę takich usług, z których każda realizuje tylko i wyłącznie swoje zadanie. Dzięki takiej dekompozycji, możliwym staje się wykorzystanie wspomnianych usług, wszędzie tam gdzie jest to potrzebne. Inną zaletą jest łatwość testowania takich usług ponieważ nie jest konieczne zestawienie całego zestawu wykonawczego, a jedynie test konkretnego działania, spełniającego określone cele.

Ponieważ komunikacja między frontendem aplikacji napisanym w języki PHP, a zapleczem umiejscowionym na platformie GAE odbywa się poprzez webservice'y, istotne było wyróżnienie następujących usług:

- usługa zwracająca dostępne kategorie, a także kategorie dla danej podkategorii,
- usługa wyświetlająca książki z danej kategorii,
- usługa wyświetlająca komentarze do danej książki,
- usługa dodająca komentarz z oceną do książki,
- usługa wyszukująca książki na podstawie słów kluczowych, tytułu lub kategorii

Każda usługa zwraca obiekt *JSON*, który następnie jest analizowany przez kod *JavaScript*. Takie podejście czyni przetwarzanie strony niezwykle wydajnym, bo obciążenie jest dzielone na dwie aplikacje.

3.3.1 AJAX problem same origin policy

Na początku projektowania zaistniał jeden problem wynikający z komunikacji przy użyciu technologii AJAX. Problem *same origin policy* jest bardzo istotnym założeniem bezpieczeństwa dotyczącym języków interpretowanych po stronie przeglądarki, takich jak *JavaScript*. Polityka bezpieczeństwa zezwala skryptom uruchamianym w ramach tej samej strony na dostęp do metod i właściwości powiązanych obiektów JavaScript bez specjalnych restrykcji. Z drugiej strony polityka ta odmawia dostępu do większości metod i własności podczas takich prób na innych stronach (inna domena, subdomena itp.).

Mechanizm ten odciska znaczące piętno dla nowoczesnych aplikacji internetowych, które w znacznym stopniu polegają np. na danych uwierzytelniających zapisanych w sesji *HTTP cookie*, jak również wykorzystujących technologię AJAX do pobierania rezultatów działania usług.

JSONP (JSON with padding)

JSONP stanowi dopełnienie bazowego formatu danych *JSON*. Jest sposobem na obejście problemu dostępu do danych z różnych domen związanych

z *same origin policy*. Zgodnie z zasadą, strona umieszczona pod adresem `server1.example.com` nie może normalnie połączyć się lub komunikować z serwerem innym niż `server1.example.com`. Jedynym wyjątkiem od tej reguły jest HTMLowy element `<script>`.

Obchodząc regułę bezpieczeństwa, możliwe jest pobranie zwracanego kodu *JavaScript*, który operuje na dynamicznie wygenerowanych danych w formacie *JSON*. Tak naprawdę takie żądanie nie zwraca wcale obiektu *JSON* tylko dynamicznie utworzony kod *JavaScript*, odnosi się do metod istniejących na stronie wykonującej żądanie.

Przykładowy kod obrazujący ten sposób widoczny jest na listingu 3.1 oraz 3.2. Pierwszy skrypt pokazuje frontendową część rozwiązania, drugie natomiast zwraca kod JavaScript, który wykona metodę `parseJSON` z przetworzonym obiektem JSON zawierającym wynik działania zapytania bazodanowego jako parametr funkcji. Następnie metoda `parseJSON` iterując w pętli po elementach obiektu JSON (w języku *JavaScript* obiekty są również tablicami więc można stosować iteratory tablicowe) wyświetli po kolei wartość dla każdego klucza obiektu w ramach poszczególnych wierszy wyniku zapytania.

Oczywiście omawiane zagadnienie sprawia troche problemów w praktyce, ale korzystając z wewnętrznej implementacji oferowanej przez jQuery, cała praca wykonywana jest za nas. Dodatkowo oprócz standardowych requestów typu `GET`, można wykonywać również inne metody protokołu *HTTP*.

Listing 3.1: Kod JavaScript realizujący JSONP

```
<html>
<head>
<script src="http://someserver.com/jsonService?category_id=5"></
    script>
</head>
<body>
<script>

function parseJSON(object)
{
 for(key in row)
 {
  document.write('Wiersz zapytania nr' + key + 1);
  for(subkey in row) {
    document.write(key + " = " + row[subkey]);
  }
 }
}

</script>
</body>
</html>
```

Listing 3.2: Logika po stronie zewnętrznego serwera

```
<?php
```

```
2
3 $categoryId = (int) $_GET['category_id'];
4
5 $sql = sprintf('SELECT * from category where category_id = %d',
      $categoryId);
6
7 $stmt = $con->query($sql); // dla uproszczenia pominięto łączenie
      się z bazą danych
8
9 $results = array();
10
11 while(false !== $row = $stmt->fetch())
12 {
13
14   $results[] = $row;
15
16 }
17
18
19 echo sprintf("parseJSON(%s);",  json_encode($results));
```

Podsumowując założeniem tworzonej aplikacji jest uzyskanie jak największej separacji logiki na zbiór wyspecjalizowanych usług. Usługi te zostaną opakowane w webservice'y w architekturze *REST*, co zapewni możliwość komunikacji z wykorzystaniem kodu *JavaScript*. W ten sposób aplikacja została rozbita na części, z których, każda możliwa ma zapewnioną skalowalność dzięki architekturze Google Application Engine.

Rozdział 4

Optymalizacja kodu klienta

Optymalizacja kodu klienta

Rozdział 5

Optymalizacja aplikacji

Na przestrzeni poprzednich rozdziałów prześledzono szczegółowo wszystkie istotne kwestie związane z optymalizacją aplikacji webowych. Czas, więc rozpocząć praktyczną implementację oprogramowania.

5.1 Framework Django

Framework *Django* powstał na bazie języka python i w niedługim czasie zrewolucjonizował proces tworzenia aplikacji internetowym w tymże języku.*Django* posiada wszystkie cechy, które charakteryzują narzędzia do szybkiego tworzenia oprogramowania (*ang. Rapid Application Development*). Podobnie jak *Ruby on Rails* czy *Symfony*, posiada bibliotekę do mapowania obiektowo - relacyjnego, w ten sposób definiując obiektowe modele i ich metody, można bez wiedzy z dziedziny baz danych, wykonywać na nich operacje. Oczywiście w niniejszej publikacji, istotna jest kompleksowa znajomość zagadnień bazodanowych, dlatego też z jednej strony wykorzystano to narzędzie w celu przyspieszenia procesu implementacji tworzonej aplikacji, z drugiej trzeba mieć na uwadze wszystkie omówione w poprzednich rozdziałach optymalizacje po stronie silnika bazodanowego.

Frameworki pokroju Ruby on Rails posiadały atut w postaci *scaffoldingu* czyli zdolności do generowania podstaw aplikacji przy użyciu odpowiednich narzędzi linii komend. *Django* nie jest pod tym względem wyjątkiem, ponieważ umożliwia generowanie zarówno bazy aplikacji, jak również kompletnego panelu administracyjnego, oferującego zaawansowane funkcje, przydatne np. do redagowania strony przez użytkownika końcowego bez znajomości technologii.

W przypadku projektowanej księgarni, za pomocą generatorów Django umożliwi prosty sposób zarządzania aplikacją oparty na wcześniej zdefiniowanych modelach. Oczywiście omówione w poprzednim rozdziale rozszerzenie *Django-Norel* umożliwi rozszerzenie zakresu dostępnych baz danych o

bazy nierelacyjne.

Django-Norel zapewnia również narzędzia umożliwiające łatwe wdrożenie tworzonego oprogramowania na platformę Google Application Engine. Aby jednak było możliwe umieszczenie projektu na tej platformie należy zarejestrować darmowe konto. Po zweryfikowaniu jego poprawności możliwe jest utworzenie do **10** darmowych aplikacji, każdej z własną bazą *Google Datastore* i zbiorem powiązanych usług.

5.2 Wystawienie usług jako webservice'y

Jednym z powodów wyboru frameworka *Django* jest łatwa możliwość tworzenia *REST*owych usług w oparciu o istniejące modele. Zapewnia to wtyczka *Django-Piston*, która dokonuje *serializacji* do kilku popularnych formatów, w tym do notacji *JSON*. Listing 5.1 przedstawia implementacje jednej z usług (Komentarzy). Jak widać implementacja takiej usługi jest dosyć prosta z wykorzystaniem wspomnianej wcześniej wtyczki. Wystarczy stworzyć nową klasę, która dziedziczy po bazowej `BaseHandler`. Następnie określono dozwolone metody HTTP, odpowiedzialne za pobieranie danych, ich tworzenie, aktualizacje, kończąc na usuwaniu. Piston oferuje możliwość zabezpieczania dostępu do konkretnych zasobów. W ten sposób tylko osoba znająca hasło może dodać komentarz czy ocenę. Z drugiej strony możliwe jest stworzenie osobnej wersji usługi udostępniającej tylko określone operacje. Jak widać na listingu odpowiedzialna jest za to klasa `AnonymousCommentHandler`. Wymieniona usługa pozwala na dodanie komentarza lub ich podgląd. Ostatnia czynność konieczna do wystawienia usługi, jest jej zmapowania na określony adres zasobu. Odpowiedzialny jest za to kod z listingu 5.2.

Listing 5.1: Implementacja usługi odpowiedzialnej za komentarze

```
1  class AnonymousCommentHandler(AnonymousBaseHandler):
2      model = Comment
3      fields = ('title', 'content', ('user', ('id','first_name', '
           last_name', 'username')), 'date', 'grade')
4
5      def read(self, request, book_id):
6          return self.model.objects.filter(book = book_id)
7
8  class CommentHandler(BaseHandler):
9      anonymous = AnonymousCommentHandler
10     allowed_methods = ('GET', 'PUT', 'POST')
11     model = Comment
12     fields = ('title', 'content', ('user', ('id','first_name', '
           last_name', 'username')), 'date', 'grade')
13
14     def read(self, request, book_id):
15
16         self.anonymous.read(request, book_id)
17
18     @validate(CommentForm)
19     def create(self, request, book_id):
20         data = request.data
```

```
21
22
23          em = self.model(
24                          title=data['title'],
25                          content=data['content'],
26                          grade = data['grade'],
27                          book_id = book_id,
28                          user_id = request.user.id
29                          )
30          em.save()
31
32
33          return rc.CREATED
```

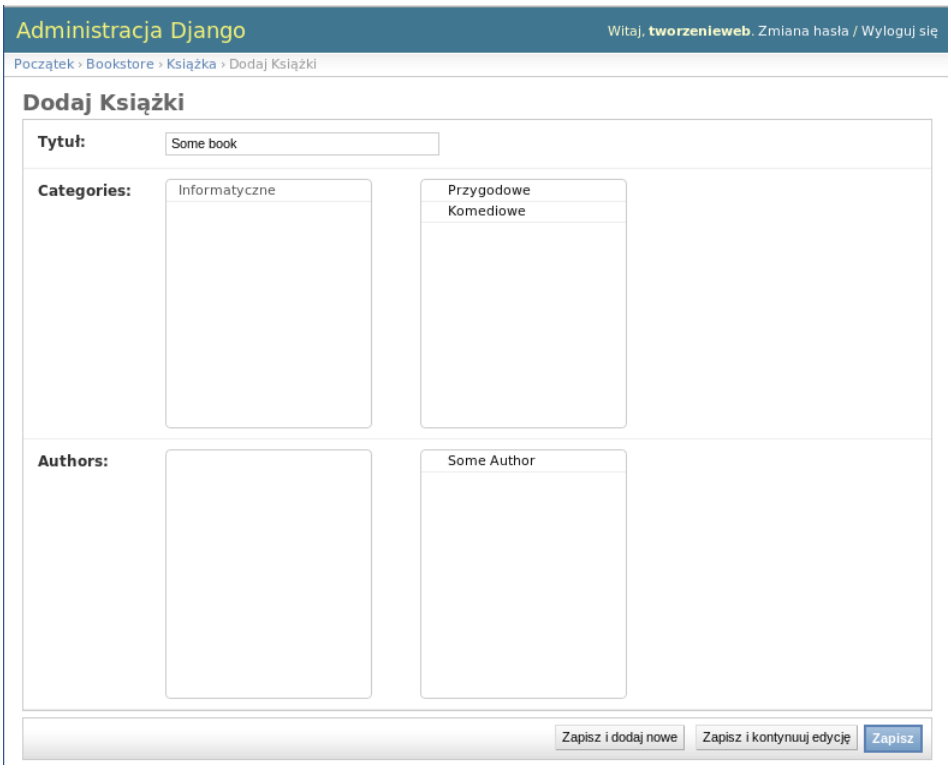
Listing 5.2: Wystawienie usługi komentarzy do publicznego dostępu

```
1 comment_handler = Resource(CommentHandler, authentication = auth)
2
3 urlpatterns = patterns('',
4     url(r'^comments/(?P<book_id>\d+)/', comment_handler, { '
          emitter_format': 'json' }),
5 )
```

5.3 Uruchomienie panelu administracyjnego aplikacji

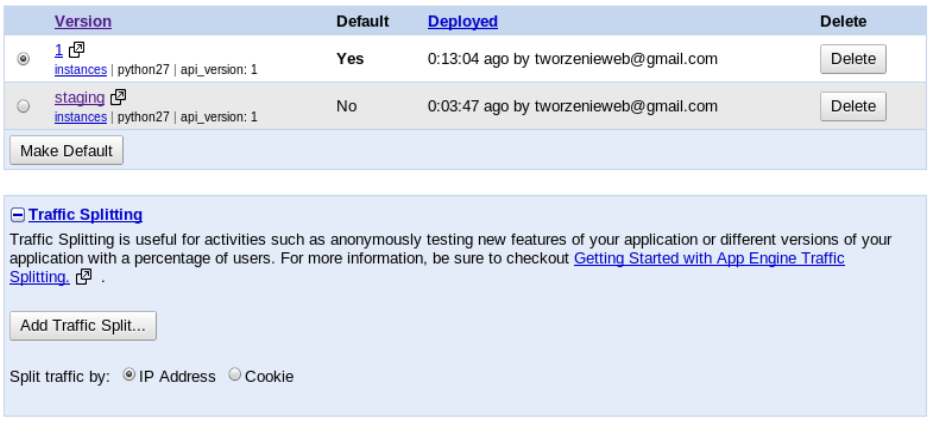
Zgodnie z omawianymi w podrozdziale 5.1 możliwościami frameworka *Django*, po utworzeniu odpowiednich modeli, możliwe jest dodanie opcji administracyjnych. Rysunek 5.1 pokazuje wygląd zaplecza administracyjnego. Natomiast rysunek 5.2 pokazuje przykładowy formularz dodawania książki. Ciekawym udogodnieniem jest wykorzystanie biblioteki *SelectMultiple* napisanej dla JavaScriptowego frameworka jQuery w celu wygodniejszej pracy z polami selectbox wielokrotnego wyboru. W ten sposób po lewej stronie są kategorie, autorzy nie wybrani, natomiast po prawej aktualnie zaznaczeni.

Rys. 5.1: Wygląd zaplecza administracyjnego



Rys. 5.2: Wygląd zaplecza administracyjnego

Jak zostało wykazane, framework Django w dużym stopniu przyspieszył proces implementacji docelowej aplikacji na platformie Google Application Engine. Jednym z ciekawych udogodnień oferowanych przez GAE jest system numeracji wersji. Przykładowo dodano do aplikacji nową funkcjonalność, która wymaga testów akceptacyjnych, edytując specjalny plik `app.yaml`. Parametr `version` pozwala np. na wdrożenie aplikacji pod specjalną subdomenę testową. Przykładowo `version: staging` spowoduje, że osobna wersja aplikacji dostępna będzie pod adresem `staging.tworzenieweb.appspot.com`. Dodatkowo w panelu administracyjnym możliwe jest ustawienie, która z wdrożonych aplikacji jest aplikacją domyślną (rysunek 5.3).



Rys. 5.3: Zarządzanie wersjami wdrożonego na platformę oprogramowania

5.4 Framework Symfony

Rozdział 6

Metody rozproszenia aplikacji i usług

Rozproszenie

Podsumowanie

Podsumowanie

Bibliografia

[1]

[2] A. Padilla and T. Hawkins. *Pro PHP Application Performance Tuning PHP Web Projects for Maximum Performance.* Apress.

[3] S. Souders. *High Performance Web Sites.* O'Reily.

Spis rysunków

Spis listingów

Płyta CD

Wraz z treścią pracy dyplomowej dołączono również płytę CD z kompletnym kodem źródłowym aplikacji. Dodatkowo kod można pobrać z poniższego repozytorium SVN: