Spis treści

Abstract

This thesis will cover the topic of efficiency optimization of web applications. It will be splited into main different areas of optimizations. First of all optimization of application architecture will be considered and then other issues like frontend optimization as well as database and webserver tuning will be covered. Also thesis needs to provide some useful tools and techniques that need to be used for mesuring of actual application performance and availability.

Rozdział 1

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.

Celem pracy 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 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*. Wybór takiej dziedziny 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 aplikacji stabilność

oraz wysoką dostępność.

1.1 Problematyka i zakres pracy

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

Innym ważnym kryterium jest wysoka dostępność usługi czyli wyeliminowanie do minimum wszelkiego rodzaju przerw wynikających z błędów lub konserwacji. Temat ten związany jest jednak głównie z odpowiednią konfiguracją sprzętową czyli wykorzystaniem równolegle działających instancji sprzętowych, które będą wykorzystywane w celu równoległego zrównoważenia ruchu, lub awaryjnie w wypadku uszkodzenia którejś z instancji.

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 jedynie wojskowym eksperymentem czy zaledwie miejscem na prezentacje własnej strony domowej. Stał się wyspecjalizowanym 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.

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 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 określonym środowisku. 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

1.2 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 developerski Yahoo. Serwis Yahoo ze względu na olbrzymie doświadczenie w kwestii skalowalności 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 aplikacja jest bardziej rozbudowana, a my potrzebujemy szybkiego i rzetelnego sposobu na wykrycie potencjalnych elementów do optymalizacji.

1.3 Układ pracy

Praca została podzielona na następujące rozdziały:

- **Rozdział pierwszy** opisuje narzędzia badawcze
- **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 2

Analiza jakościowa i wydajnościowa aplikacji

Projektując aplikacje, już od samego początku należy myśleć o zapewnieniu najlepszej możliwej jakości, a także problemach, które mogą się pojawić w po wdrożeniu oprogramowania. W celu zapewnienia oprogramowaniu najwyższej skuteczności pracy należy wziąć pod uwagę wiele cech, wśród których najważniejsze podane są poniżej.

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

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

Wydajność (ang. *Performance*) - przekłada się na możliwość faktycznego obsłużenia dużego ruchu sieciowego, przy jednoczesnym 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, jednak bez zapewnienia odpowiednich dróg, ten szybki samochód lub gorzej, ich zbiór nie jest w stanie rozwinąć maksymalnej prędkości, ponieważ jest blokowany przez inne pojazdy. Skalowalność jest więc zapewnieniem odpowiedniej infrastruktury gwarantującej właściwy rozrost systemu.

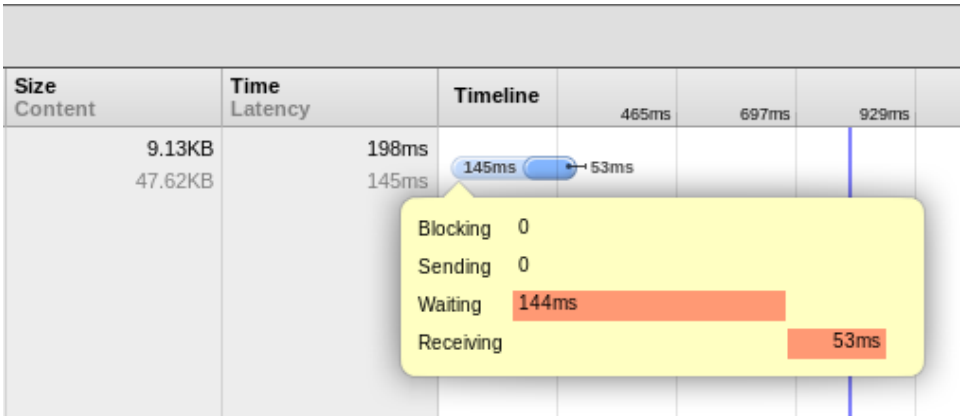
W celu zapewnienia możliwie najlepszej jakości tworzonej aplikacji, należy stale monitorować aktualny poziom wydajności aplikacji. Należy jednak

mieć na uwadze, że na wydajność aplikacji składa się wydajność poszczególnych węzłów systemu. Należy więc testować poszczególne z nich osobnymi metodami.

W celu rzetelnego określenia przyczyn błędów warto rozpocząć analizę od najbardziej ogólnego komponentu czyli serwera WWW. Kolejno należy dokonać dekompozycji wyróżniając kolejne węzły systemu, takie jak poszczególne instancje serwera WWW czy serwery bazodanowe.

2.1 Analiza wydajności serwera WWW

Zadaniem serwera WWW jest zwrócenie do przeglądarki wyniku przetwarzania zasobu opisanego adresem URL. W najprostszym wypadku analiza wydajności serwera polega na odpytaniu serwera o określony zasób i zmierzenie czasu od rozpoczęcia tej akcji do odebrania tego rezultatu. Taki proces możemy prześledzić i przeanalizować w większości popularnych przeglądarek np. Google Chrome (Rys. 2.1).



Rys. 2.1: Analiza czasu wykonywania strony http://ftims.edu.p.lodz.pl//

Ten 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 tejże komendy widoczny jest na listingu 2.1.

Listing 2.1: Analiza strony z wykorzystaniem narzędzia `ab`

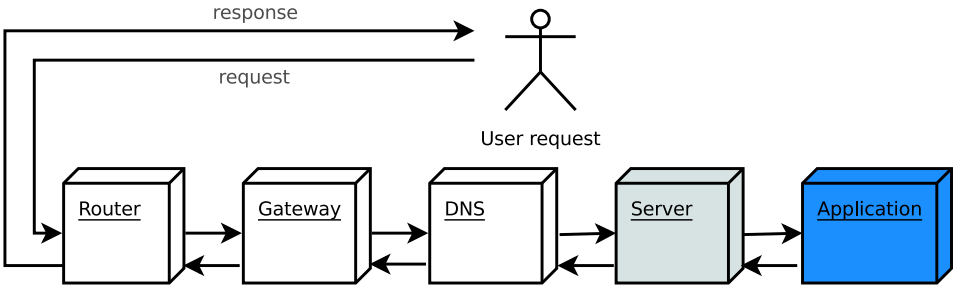
```
Server Software:        Apache/2.2.14
Server Hostname:        ftims.edu.p.lodz.pl
Server Port:            80

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, widać przede wszystkim, że czas oczekiwania na stronę przy 10 użytkownikach jest prawie trzykrotnie wyższy niż podczas jednego żądania wykonanego w przeglądarce (2.1).

Oczywiście na wyniki pomiarów ma też wpływ prędkość połączenia internetowego, dlatego w celu pominięcia dodatkowych czynników, testy docelowej aplikacji będziemy wykonywali przede wszystkim na lokalnym serwerze. Najbardziej miarodajną jednostką określającą wydajność aplikacji w wypadku narzędzia *ab* jest liczba zapytań na sekundę, określa ona w wypadku ustawienia większej ilości równoległych połączeń, maksymalną ilość jaką aplikacja jest w stanie obsłużyć.

Na rysunku 2.2 przedstawiono cykl życia żądania od użytkownika, kończąc na odpowiedzi serwera. Rysunek ten przedstawia drogę jaką pokonuje żądanie od użytkownika, kończąc na odebraniu odpowiedzi serwera pochodzącej z aplikacji WWW. Jak można zauważyć żądanie przebywa stosun-

kowo dużą drogę nim trafi do faktycznej aplikacji. Dlatego też wszelkie dodatkowe opóźnienia wynikają ze 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. 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. 2.2: Cykl życia żądania (*opracowanie własne*)

Nawiązując do listingu 2.1, 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 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 na dokument, 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 dosyć 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.

Rodzaj użytej przeglądarki. Nie bez znaczenia 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.

Konfiguracja serwera WWW. W zależności do użytej technologii istnieje wiele różnych serwerów WWW. Wśród najpopularniejszych prym wiedzie serwer HTTP Apache. Dla rozwiązań napisanych w technologii Java często wykorzystywane są serwery GlassFish, Tomcat, Jetty. Często wykorzystywane serwery, nie są odpowiednio skonfigurowane do obsługi aplikacji. 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 odwiedzin stron, pozwala to zaobserwować pewne trendy użytkowników. Często jest tak, że dane zasoby są intensywnie odpytywane przez użytkowników np. w czasie 10 minut stronę odwiedza nawet 1000 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 2.2: Test obciążenia czasowego

```
1 Benchmarking ftims.edu.p.lodz.pl (be patient)
2 Finished 504 requests
3
4 Server Software:        Apache/2.2.14
5 Server Hostname:        ftims.edu.p.lodz.pl
6 Server Port:            80
7
```

```
 8 Document Path:          /
 9 Document Length:        48759 bytes
10
11 Concurrency Level:      10
12 Time taken for tests:   40.180 seconds
13 Complete requests:      504
14 Failed requests:        0
15 Write errors:           0
16 Total transferred:      24822504 bytes
17 HTML transferred:       24574536 bytes
18 Requests per second:    12.54 [#/sec] (mean)
19 Time per request:       797.213 [ms] (mean)
20 Time per request:       79.721 [ms] (mean, across all requests)
21 Transfer rate:          603.31 [Kbytes/sec] received
22
23 Connection Times (ms)
24               min  mean[+/-sd] median   max
25 Connect:       48   65  14.3     61     145
26 Processing:   284  436 288.9    376    2957
27 Waiting:      119  199 281.5    151    2660
28 Total:        333  500 287.8    439    3007
29
30 Percentage of the requests served within a certain time (ms)
31   50%    439
32   66%    458
33   75%    477
34   80%    493
35   90%    563
36   95%    666
37   98%   1420
38   99%   2142
39  100%   3007 (longest request)
```

Listing 2.2 przedstawia wynik testów. Tym razem zamieszczono pełen rezultat wykonania tej komendy. 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 błędy 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 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

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. 2.1: Tabela z rezultatem działania aplikacji JMeter

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 testów właśnie w ramach tego zbioru poleceń. 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 2.1 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 samą sytuacje 500 razy. Na zakończenie dane możliwe są do wyeksportowania do formatu *CSV* lub wyświetlenia 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.

2.2 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 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, korzystali wyłącznie z terminali.

Obecnie wszyscy użytkownicy korzystają przynajmniej z jednej przeglądarki internetowej, dlatego dla większej precyzji, konieczne jest wyszczególnienie pewnego kluczowego komponentu jakim jest front-end aplikacji. W przypadku aplikacji internetowych, front-end to graficzna nakładka służąca do komunikacji z użytkownikiem i prezentacji danych opracowanych przez architekturę systemu (back-end) w sposób przystępny i zrozumiały dla użytkowników. Odpowiednia optymalizacja frontendu jest o tyle ważna, że jest to pierwsza technologia z jaką użytkownik ma kontakt w momencie przeglądania aplikacji [1].

Frontend jest miejscem analizy i przetwarzania wyniku odpowiedzi serwera. W większości wypadków miejscem osadzenia frontendu jest przeglądarka WWW, natomiast językiem obowiązującej komunikacji jest język HTML. Od kilku lat w wyniku rozwoju trendu WEB 2.0 istotnym elementem każdej strony staje się przeniesienie części logiki aplikacji na stronę przeglądarki (frontendu). Cienkie do tej pory aplikacje webowe (thin client), zaczynają dzięki zdobyczą technologicznym takim jak AJAX stawać się grubymi klientami. Oznacza to, że przetwarzanie danych może mieć miejsce w przeglądarce internetowej.

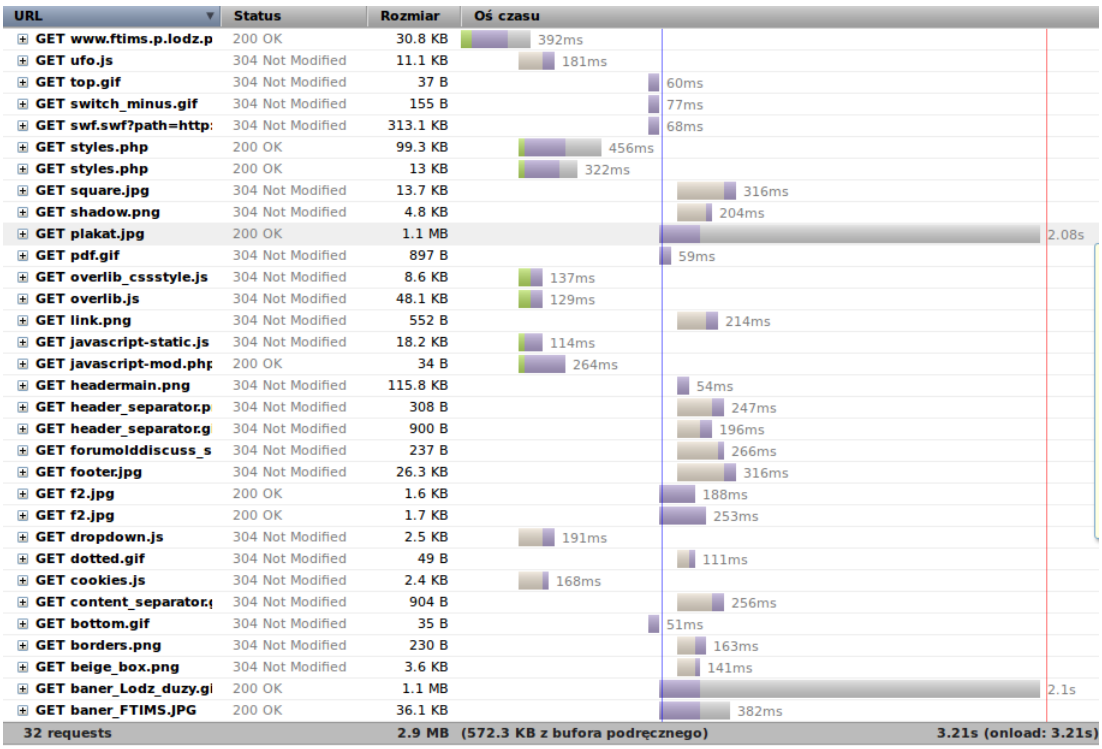
Dlatego też istotnym elementem analizy wydajnościowej jest analiza frontendu. Wśród istniejących na rynku rozwiązań, najpopularniejszymi są te, które są wbudowane bezpośrednio w interfejs przeglądarki internetowej. Jednym z pierwszych rozwiązań tego typu była wtyczka Firebug napisana dla przeglądarki Firefox. Jest to obecnie najbardziej zaawansowane narzędzie tego typu, rywalizujące jednocześnie z natywnymi narzędziami deweloperskimi dla przeglądarki Chrome.

Interfejs Firebuga pozwala na szczegółową inspekcję kodu HTML wraz z możliwością dynamicznej podmiany jego zawartości. 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 pokazywał 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, dokumentów JavaScript, apletów Java, dokumentów Flash.

Każda strona może zawierać 0 lub więcej 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 [2] istniej zasada, która mówi, że w większości wypadków tylko 10-20% czasu odpowiedzi jest spędzane na oczekiwanie na dokument HTML, natomiast pozostałe 80-90% to czas na pobieranie pozostałych zasobów.

Rysunek 2.3 jest najlepszym przykładem tej zasady. Strona wykonuje 32 zapytania do serwera, z czego tylko jedno to zapytanie o dokument 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 zeskalowane do odpowiednich rozmiarów. `Firebug` stanowi, więc cenne narzędzie przy diagnostyce frontendu strony. Staje 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 zbadanie każdego węzła.



Rys. 2.3: Analiza ruchu sieciowego na stronie http://ftims.p.lodz.pl

Rozdział 3

Wymagania i budowa aplikacji

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

Wymagania podzielono na część administracyjną i część użytkownika, natomiast zbiór funkcjonalnośc przedstawiony jest w tabeli 3.1.

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, wykonane zostanie żądanie do sprawdzenia przez serwer. Takie podejście powinno znacznie zredukować obciążenie np. podczas rejestracji użytkowników, po-

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. 3.1: Tabela z wykazem funkcjonalności

nieważ 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.

Rozdział 4

Architektura aplikacji

Architektura

Rozdział 5

Optymalizacja aplikacji

Optymalizacja

Rozdział 6

Optymalizacja kodu klienta

Optymalizacja kodu klienta

Rozdział 7

Metody rozproszenia aplikacji i usług

Rozproszenie

Podsumowanie

Podsumowanie

Bibliografia

[1] A. Padilla and T. Hawkins. *Pro PHP Application Performance Tuning PHP Web Projects for Maximum Performance*. Apress, 2010, address =.

[2] S. Souders. *High Performance Web Sites*. O'Reily, 2007, address =.

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: