JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 17 (2020) 351–357
Received: 2 September 2020
Accepted: 21 September 2020

Comparative analysis of web application performance testing tools

Analiza porównawcza narzędzi do badania wydajności aplikacji internetowych

Agata Kołtun*, Beata Pańczyk

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
Recent years have brought the rise of importance of quality of developed software. Web applications should be functional, user friendly as also efficient. There are many tools available on the market for testing the performance of web applications. To help you choose the right tool, the article compares three of them: Apache JMeter, LoadNinja and Gatling. They were analyzed qualitatively in terms of a user-friendly interface, parameterization of the requests and creation of own testing scripts. The research was carried out using a specially prepared application. The summary indicates the most important advantages and disadvantages of the selected tools.

Keywords: performance; Apache JMeter; LoadNinja; Gatling

Streszczenie
Ostatnie lata pokazują, jak ważna jest jakość wytwarzanego oprogramowania. Aplikacje internetowe, oprócz funkcjonalności przyjaznej użytkownikowi, muszą być również wydajne. Na rynku oferowanych jest wiele narzędzi do badania wydajności aplikacji internetowych. Aby ułatwić wybór odpowiedniego narzędzia, w artykule porównano trzy z nich: Apache JMeter, LoadNinja oraz Gatling. Przeanalizowano je jakościowo pod kątem przyjaznego interfejsu użytkownika, możliwości parametryzacji oraz tworzenia własnych skryptów. Do badań wykorzystano autorską aplikację, a w podsumowaniu wskazano najważniejsze zalety i wady wybranych narzędzi.

Słowa kluczowe: wydajność; Apache JMeter; LoadNinja; Gatling

*Corresponding author
Email address: **agata.koltun95@gmail.com** (A. Kołtun)

1. Wstęp

W związku z kluczową rolą, jaką pełni odporność aplikacji na zadane obciążenie, istotnym aspektem pracy nad rozwojem aplikacji jest wybór właściwego narzędzia wspomagającego testowanie wydajności aplikacji. Wybór ten nie jest prosty, zważywszy na to, że dostępne w sieci artykuły nie dają jednoznacznej odpowiedzi, które z narzędzi powinno zostać wybrane do przeprowadzenia takich testów.

Na rynku oferowanych jest wiele narzędzi (w tym płatnych), za pomocą których developerzy są w stanie napisać symulacje odpowiadające działaniom wielu użytkowników – na przykład tysiąca kupujących przeglądających asortyment i dokonujących zakupów w sklepie internetowym. Rozwiązania bazujące na odpowiednio przygotowanym, sparametryzowanym skrypcie wydają się znacznie wygodniejszym i tańszym rozwiązaniem (zwłaszcza w kontekście powtarzania testów w wielu iteracjach). Koszt pracy developera niezbędny do przygotowania i utrzymania takich skryptów należy również uwzględnić podczas wyboru odpowiedniego narzędzia. Przygotowanie skryptu nie powinno wymagać zaawansowanych umiejętności technicznych, a ewentualne zmiany w testowanej aplikacji powinny być raczej kosmetyczne. Inspekcja kodu (ang. *code review*) dokonywana przez innego programistę nie powinna być zbyt czasochłonna.

Większość publikacji odnosi się do JMeter [1, 2]. Dodatkowo pozycje internetowe, porównujące takie narzędzia dotyczą z reguły zestawienia rozwiązania firmy Apache z innymi narzędziami [3, 4, 5].

2. Testy wydajnościowe

„Testy wydajnościowe to rodzaj testów, które mają na celu scharakteryzowanie responsywności, niezawodności, przepustowości, interoperacyjności i skalowalności systemu oraz/lub aplikacji po zadaniu danego obciążenia. Może również określać prędkość lub efektywność komputera, sieci, aplikacji lub urządzenia” [1].

Testy obciążeniowe skupione są na trzech aspektach określających jakość systemu:

- **prędkości** (ang. *speed*) – jak szybko odpowiada aplikacja,
- **stabilności** (ang. *stability*) - czy aplikacja działa stabilnie przy dużym obciążeniu,
- **skalowalności** (ang. *scalability*) – cecha określająca maksymalną liczbę użytkowników, którą aplikacja jest w stanie obsłużyć.

Jedną z podstawowych pobudek do przeprowadzenia testów wydajności jest *Service Level Agreement*, czyli umowa o gwarantowanym poziomie świadczenia usług, którą wprowadza się w celu określenia czy spełnione są wymagania funkcjonalne określone przez usługobiorcę.

Niewątpliwą zaletą, którą mogą ukazać takie testy, jest wskazanie pojedynczych punktów błędów w przypadku szczytowego obciążenia oraz określenia

tak zwanego wąskiego gardła, które definiuje się jako element procesu, który ma ograniczoną wydajność, a zatem ogranicza przepustowość systemu jako całości [6].

W przypadku aplikacji internetowych test odnosi się zwykle do pojedynczego modułu, a nawet zapytania (np. przeprowadzanej skomplikowanej transakcji bazodanowej), które powoduje, że cały system staje się mniej efektywny.

Najczęściej spotykanymi **wąskimi gardłami** są [7]:

- wysokie obciążenie procesora,
- wykorzystanie pamięci,
- wysokie obciążenie sieci,
- ograniczenia systemu operacyjnego,
- wykorzystanie dysku.

3. Proces przeprowadzania testów

Liczne źródła internetowe, a także pozycje literaturowe opisują następujący proces testowania wydajności [8]:

1. Wybranie środowiska, które będzie testowane (np. środowisko testowe, które odpowiada parametrom środowiska produkcyjnego).
2. Określenie wymaganej, akceptowanej wydajności.
3. Zaplanowanie testów – opracowanie scenariuszy testowych, wybranie liczby użytkowników końcowych.
4. Konfiguracja środowiska testowego – przygotowanie zasobów przed uruchomieniem testów.
5. Implementacja testów.
6. Przeprowadzenie testów – uruchomienie i monitorowanie przebiegu testów.
7. Analiza wyników i ponowne przeprowadzenie testów po poprawkach w przypadku zdiagnozowania źródeł wąskiego gardła lub innych problemów.

Zaznaczyć należy, że każdy krok jest jednakowo istotny. Właściwe zaplanowanie pracy jest bardzo ważnym działaniem, które determinuje jakość przeprowadzonych badań.

4. Przegląd literatury

Autor pozycji [8] opisuje proces zrozumienia charakterystyk wydajności aplikacji oraz jej modułów jako bezcenny. W swojej pracy dokonuje porównania rozwiązań wspierających wspomniane badanie. Pierwszym z opisywanym narzędzi jest Apache Benchmark, program dedykowany do testowania obciążeniowego serwerów HTTP, w szczególności serwera Apache. Narzędzie określone jest tam jako mało elastyczne, lecz cechujące się satysfakcjonującą prostotą działania. Jako alternatywne, wyróżniające się rozwiązanie, zaproponowany jest Gatling, który oferuje wiele sposobów konfiguracji interakcji wirtualnych użytkowników z witryną oraz współbieżnością działań. W pozycji [8] Gatling określony jest jako potężne narzędzie ze względu na język dziedzinowy DSL (ang. *domain specific language*) [9].

DSL umożliwia:

- określenie asercji (dotyczących np. statusów HTTP),
- zawartości fragmentu przesyłanych danych (ang. *payload*),
- konfigurację dopuszczalnych opóźnień.

Autor skrótowo podsumowuje również Apache JMeter, określając go jako narzędzie dobre, lecz mniej elastyczne od Gatling. Stosowanie Apache JMeter wskazane jest w przypadku, gdy programista nie chce uczyć się nowego DSL.

Pozycja [1] również poświęcona jest testowaniu wydajności za pomocą JMeter. Wspomniano tu także o innych rozwiązaniach, np. LoadRunner, LoadUI, Gatling, OpenSTA oraz Apache Benchmark. Jednak podstawową ich wadą jest komercyjność, brak dojrzałości rozwiązania, niewystarczającą przenośność i rozszerzalność rozwiązania w stosunku do Apache JMeter. HP LoadRunner jest kosztowny, lecz oferuje lepszy interfejs użytkownika oraz ciekawsze możliwości monitorowania w stosunku do JMeter. Gatling z kolei jest opisany jako projekt nowy, „wciąż w powijakach", choć obiecujący. Główną zaletą Gatling jest łatwiejsze testowanie języka specyficznego dla domeny (w porównaniu do dużych plików XML JMeter) oraz atrakcyjne wizualizacje rezultatów. Za wadę podano niewielką liczbę użytkowników narzędzia oraz konieczność znajomości języka Scala.

Większość porównań dotyczących poszczególnych narzędzi stanowią jednak publikacje dostępne w Internecie. W [5] jako najlepsze podano płatne rozwiązanie WebLOAD, oparte na elastycznej platformie obsługującej setki technologii i umożliwiającej integrację z większością najważniejszych narzędzi. LoadNinja zajmuje trzecie miejsce w rankingu. Opisane jest jako rozwiązanie umożliwiające tworzenie zaawansowanych testów bez użycia skryptów, skracające czas testowania o 50% i zastępujące emulatory obciążenia prawdziwymi przeglądarkami. Dzięki LoadNinja zespoły projektowe mogą bardziej skupić się na tworzeniu i rozwijaniu aplikacji, niż na tworzeniu skryptów je testujących. W [5] Apache JMeter wskazano jako szóste z najlepszych narzędzi, które oferuje przyjazny interfejs użytkownika, wysoką przenośność i generujące proste wykresy (wystarczające do analizy kluczowych statystyk monitorowania obciążenia aplikacji). Pozycja [5] nie wspomina o Gatling.

5. Wybór narzędzi do porównania

Narzędzia można podzielić na dwie podstawowe grupy: bazujące na płatnej licencji oraz te typu open source [10]. Pierwsze z nich zwykle oferują wersję próbną, umożliwiającą sprawdzenie możliwości narzędzia przed podjęciem decyzji, a poniesiony koszt zależy od liczby wirtualnych użytkowników dostępnych dla scenariusza testu. Liczba testów, które należy przeprowadzić, rodzaj testu i możliwości raportowania często nie wpływają na wycenę procesu.

W niniejszym artykule do porównania jakościowego wybrano trzy najpopularniejsze rozwiązania służące do badania wydajności aplikacji internetowych, które reprezentują odmienne podejście twórców narzędzi do interfejsu użytkownika:

- Apache JMeter - rozwiązanie darmowe, bardzo popularne i dobrze opisane w wielu publikacjach,

posiada interfejs użytkownika w formie aplikacji okienkowej;
- Gatling - narzędzie darmowe, coraz bardziej popularne [3] i ciekawe ze względu na brak interfejsu użytkownika;
- LoadNinja - płatne rozwiązanie platformowe.

5.1. Apache JMeter

Apache JMeter [11] jest narzędziem stworzonym przez Apache Software Foundation, które może zostać wykorzystane do analizy obciążenia różnych serwisów, w szczególności aplikacji internetowych. Na dzień 19 stycznia 2020 roku najnowszą, stabilną, wydaną wersją jest 5.2 na licencji Apache License 2.0.

JMeter jest narzędziem darmowym bazującym na licencji typu open source, co oznacza, że można nie tylko korzystać z niego za darmo (również w przypadku zastosowań komercyjnych), ale też kopiować, analizować, modyfikować i rozbudowywać dostępne moduły do własnego użytku.

Najważniejszymi komponentami, które służą do budowania testu są [12]:
- kontrolery typu Samplers (generowanie prostych zapytań HTTP, FTP, zapytań do bazy danych, wywołanie implementacji JavaSamplerClient lub wysyłanie/odczyt wiadomości z kolejki typu Java Message Service),
- kontrolery Logic Controllers (generowanie zapytań po spełnieniu określonych warunków),
- asercje (weryfikacja poprawności odpowiedzi na zapytanie),
- timers (umożliwiają opóźnienia lub synchronizację zapytań, szczególnie przydatne podczas symulacji czasu, jaki użytkownik spędza przeglądając stronę internetową, bez wykonywania interakcji z nią),
- listeners (elementy umożliwiające wygenerowanie reprezentacji m.in. graficznych wykresów przedstawiających wyniki testów, zapisu odpowiedzi na zapytanie lub spełnienia asercji).

Od wersji 3, domyślnym językiem pisania skryptów jest Groovy. Narzędzie posiada prosty graficzny interfejs, który może być przydatny dla mniej zaawansowanych użytkowników. HTTP Proxy umożliwia nagranie działań użytkownika na stronie internetowej – przez co utworzenie podstawowych testów może sprowadzić się do podania parametrów dla gotowych, wygenerowanych komponentów oraz przygotowania liczby wątków wykonujących dany test.

5.2. LoadNinja

LoadNinja [13] jest rozwiązaniem chmurowym, które zostało opracowane przez SmartBear Software. Umożliwia testy symulujące działania użytkownika aplikacji (ang. *UI Testing*) oraz testy API, symulując żądania klienta przesyłającego zapytania HTTP do serwera.

Wersja trial jest edycją, która nie nadaje się do użytku komercyjnego. Na dzień 10 lipca 2020 roku dostępne są trzy rodzaje licencji, każda oferująca 8 godzin testów dla wariantu miesięcznego oraz 100 godzin dla rocznego. Najtańsza subskrypcja, która umożliwia przeprowadzenie testu na 1000 wirtualnych użytkownikach to koszt 739€.

Najważniejsze komponenty tego narzędzia to:
- Recorder InstaPlay (tworzenie testu i jego późniejsze odtwarzanie),
- VU Debugger (umożliwia debugowanie testu w czasie rzeczywistym),
- VU Inspector (monitorowanie aktywności wirtualnych użytkowników w czasie rzeczywistym).

Narzędzie nie wymaga instalowania żadnego oprogramowania na urządzeniu – nie jest wymagana nawet instalacja Javy, co wyróżnia je na tle pozostałych porównywanych narzędzi. Potrzebne jest jedynie konto użytkownika oraz jedna z przeglądarek internetowych: Chrome, Firefox, Safari lub Microsoft. Czyni to LoadNinja narzędziem polecanym do przeprowadzania testów przez analityków biznesowych, bez wsparcia technicznego.

5.3. Gatling

Gatling [14] jest narzędziem do badania wydajności i testowania obciążenia, wyprodukowanym przez firmę Gatling (Gatling FrontLine – w przypadku wersji Enterprise). Na dzień 14 kwietnia 2020 roku ostatnią, stabilną wersją był Gatling 3.1.3, wydany w 2019 roku.

Narzędzie opiera się o licencję typu open source. Oprócz oficjalnie wydanych dodatków, wokół narzędzia skoncentrowana jest spora społeczność, której rezultatem pracy jest m.in. darmowy dodatek integrujący Gatling z systemem kolejkowym RabbitMQ.

Podstawowymi komponentami są: Recorder oraz Domain Specific Language [10]. Język dziedzinowy sprawia, że testy są łatwiejsze w odbiorze, proste do utrzymania oraz zrozumiałe biznesowo.

Skrypty pisane są w języku Scala, który działa na wirtualnej maszynie Javy i ma cechy języków obiektowych i funkcyjnych.

Narzędzie nie udostępnia graficznego interfejsu użytkownika. Nagrywarka wyzwalana jest z aplikacji desktopowej. Testy modyfikowane są za pomocą edytora tekstowego, w szczególności polecane jest IDE, które można rozszerzyć o podpowiadanie składni języka Scala np. IntelliJ. Uruchomienie testów należy przeprowadzić z poziomu konsoli, raport generowany jest samoistnie.

6. Porównanie wybranych narzędzi

Do badań wykorzystano specjalnie przygotowaną aplikację internetową. W każdym z narzędzi opracowano skrypty do przeprowadzenia testów zgodnie z zadanym scenariuszem.

Scenariusz testowy zakłada wykonanie przez wirtualnego użytkownika następujących akcji:
- rejestrację konta użytkownika,
- zalogowanie się do aplikacji,
- wypełnienie anonimowej ankiety,
- wypełnienie formularza,
- wylogowanie się z systemu.

6.1. Aplikacja testowa

Na potrzeby badań utworzono prostą aplikację webową, której część serwerowa napisana została w języku Java 8. Jako narzędzie automatyzujące budowanie projektu wybrano Apache Maven. Wykorzystano Spring Framework, a także Spring Boot – w celu szybkiego uruchomienia gotowej aplikacji, niewymagającej dodatkowej konfiguracji. Warstwa wizualna aplikacji oparta jest o formularze JSP. Pracę z danymi obsługuje PostgreSQL v42.2.6.

Baza danych przechowuje informacje o użytkownikach systemu, czyli wirtualnych użytkownikach, których akcje symulowane są w ramach scenariusza oraz wynikach formularza osobowego i ankiety badającej poziom satysfakcji użytkownika.

Realizacja scenariusza, którego przebieg opisano w punkcie 6, wymagała utworzenia formularza JSP oraz odpowiedniej akcji HTTP dla każdego z opisywanych kroków.

6.2. Realizacja badań

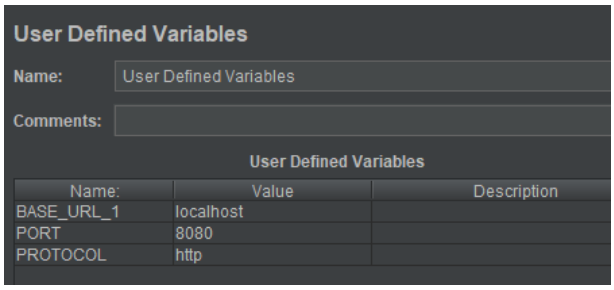
Testy wydajnościowe przygotowano i przeprowadzono za pomocą trzech, wcześniej opisanych narzędzi.

Apache JMeter

Test można utworzyć „ręcznie”, wyklikując kolejne komponenty w graficznym interfejsie użytkownika lub tworząc własny plik w formacie JMX. O wiele efektywniejsze jest jednak nagranie testu za pomocą nagrywarki, a następnie parametryzacja zapytań.

Jedną z możliwości nagrania scenariusza jest wykorzystanie szablonu *Recording with Think Time*, które wymaga wygenerowania certyfikatu i zaimportowania go do przeglądarki. Łatwiejszym rozwiązaniem jest skorzystanie z narzędzia BlazeMeter [12], które jest dodatkiem do przeglądarki Chrome.

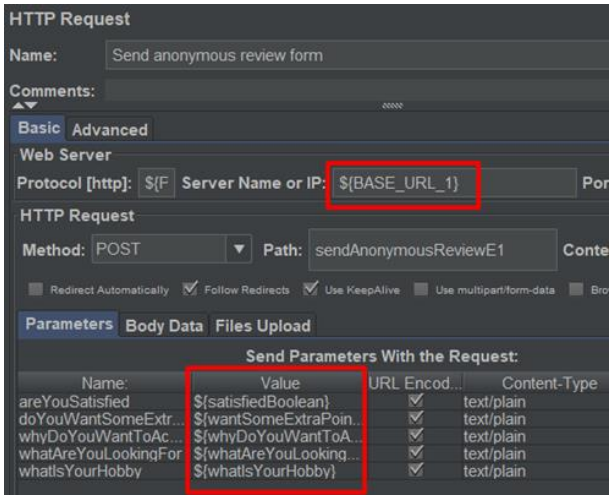
Są trzy podstawowe sposoby parametryzacji scenariusza. Blok User Definded Variables (Rys. 1) umożliwia definiowanie parametrów globalnych np. IP serwera i rodzaju protokołu sieciowego.



Rysunek 1: Parametryzacja za pomocą bloku User Defined Variables

Obsługa plików płaskich (CSV oraz TSV) jest przydatna do przeprowadzania testów wymagających unikalnych danych, które są wykorzystywane przez każdego wirtualnego użytkownika. Przekazywanie parametrów z poziomu konsoli może zostać wykorzystane np. do ustalenia liczby wirtualnych użytkowników wskazanych przy przeprowadzeniu konkretnej iteracji testów. Niezależnie od typu parametru i zastosowanej metody, sposób użycia parametru (Rys. 2) jest taki sam, co czyni testy elastycznymi, a parametry można łatwo nadpisać.

Narzędzie oferuje różnego typu wizualizacje rezultatów testów, zarówno w formie wykresów, jak i tabel. Wykresy nie są atrakcyjne wizualnie (Rys. 3). Na szczególną uwagę zasługuje View Results Tree, które umożliwia weryfikację każdego wykonanego zapytania osobno (wartości parametrów oraz odpowiedzi na zadane żądanie) w trakcie i po przeprowadzeniu testu.



Rysunek 2: Przykład parametryzacji zapytań Apache JMeter

LoadNinja

Aby rozpocząć pracę z narzędziem LoadNinja [13] konieczne jest utworzenie konta użytkownika. W przypadku testowania aplikacji na hoście lokalnym (localhost) należy lokalnie uruchomić aplikację LoadNinja Tunnel.

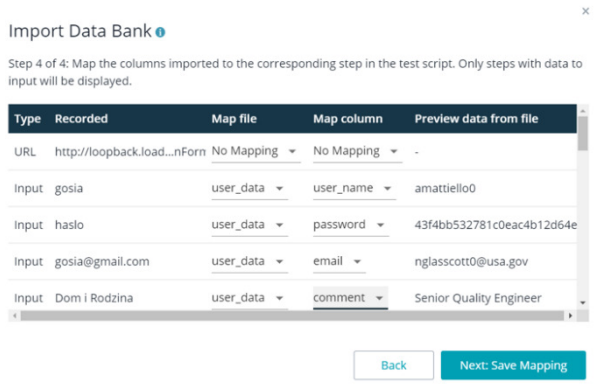
Utworzenie projektu i uruchomienie nagrywarki powoduje, że na ekranie wyświetlany jest podgląd testowanej aplikacji. Każde wysłanie formularza prezentowane jest jako osobny krok, na który składają się akcje użytkownika takie jak: bezczynność, kliknięcie pola tekstowego, przewijanie strony, użycie przycisku *backspace* lub wprowadzenie tekstu.

Narzędzie umożliwia parametryzację zapytań poprzez pliki płaskie CSV lub TSV. Podgląd (Rys. 3) oryginalnej wartości wprowadzonej podczas nagrywania scenariusza oraz przykładowa wartość z wybranej kolumny pliku płaskiego czyni parametryzację testu dosyć przystępną.

Podczas przeprowadzania bardziej skomplikowanych testów, brak możliwości tworzenia wartości wyliczeniowych parametrów może okazać się problematyczny.

Wersja próbna oferuje ograniczoną liczbę wizualizacji rezultatów testów, choć są nowoczesne i interaktywne (Rys. 4).

LoadNinja oferuje wizualizację statystyk czasowych pojedynczych kroków wyłącznie w formie tabelarycznej (Apache JMeter i Gatling umożliwiały wygenerowanie ich również w postaci diagramów graficznych) (Rys. 5).



Rysunek 3: Parametryzacja formularzy LoadNinja.

Gatling

Utworzenie testu Gatling możliwe jest przez napisanie własnego skryptu lub wykorzystanie nagrywarki, a następnie dokonanie modyfikacji (np. w celu parametryzacji zapytań).

Nagrywarka wymaga konfiguracji proxy, jednak prostszym rozwiązaniem jest zaimportowanie pliku HAR, wygenerowanego podczas nagrywania logów sieciowych (dostępne w zakładce *Network* w przeglądarce Chrome).

Skrypt może zostać sparametryzowany w dowolnej formie, nie ma konieczności korzystania z rozwiązań oferowanych przez bibliotekę, podnosi to jednak koszty czasowe realizacji testu. Przykładowo biblioteka oferuje możliwość zdefiniowania opóźnień pomiędzy wykonaniem kolejnych kroków, jednak wprowadzenie bardziej losowych wartości wymaga dodatkowej implementacji (Listing 1).

Listing 1: Funkcja generująca losowy czas z przedziału

```
def randomTimeInSeconds(from: Int, to: Int): Int = {
    val random = new Random()
    random.nextInt(to - from + 1) + from
}
val between30and40 = randomTimeInSeconds(from = 30, to = 40)
```

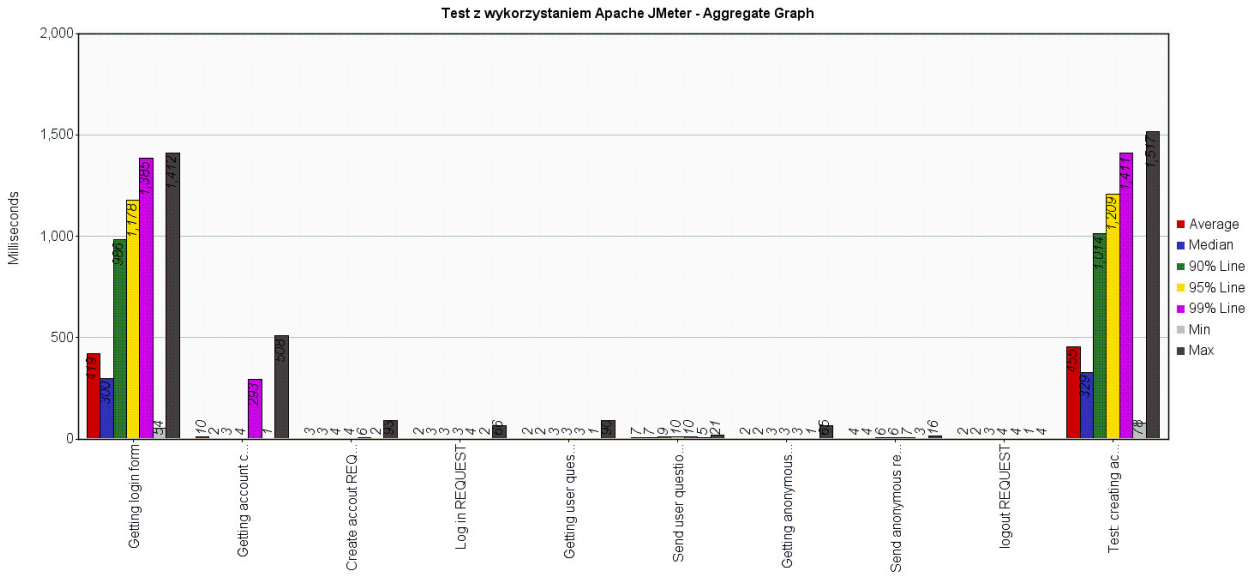
Podstawowym sposobem parametryzacji testu jest przekazywanie zmiennej uruchomieniowej programu. Jest to odpowiedni sposób do przekazywania podstawowych parametrów takich jak liczba wirtualnych użytkowników czy tryb uruchomienia testu. Komponent Feeders [15] (Listing 2) umożliwia parametryzację zapytań m.in. przez pliki płaskie, JSON lub źródło bazodanowe.

Na zakończenie testu generowane są nie tylko globalne statystyki, ale również te dotyczące poszczególnych kroków. Wizualizacje graficzne są nowoczesne i atrakcyjne (Rys. 6).

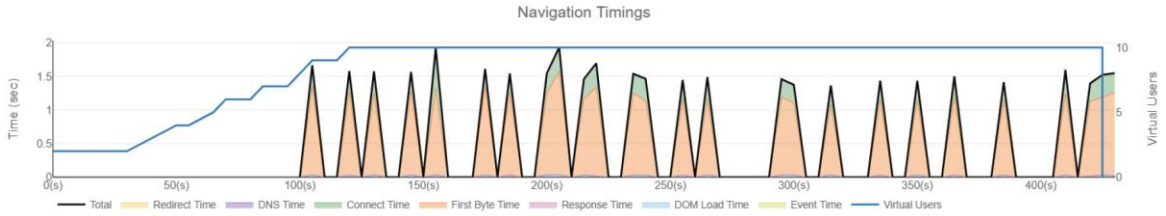
Brak graficznego interfejsu użytkownika jest największą wadą Gatling.

Listing 2: Parametryzacja scenariusza za pomocą Feeders.

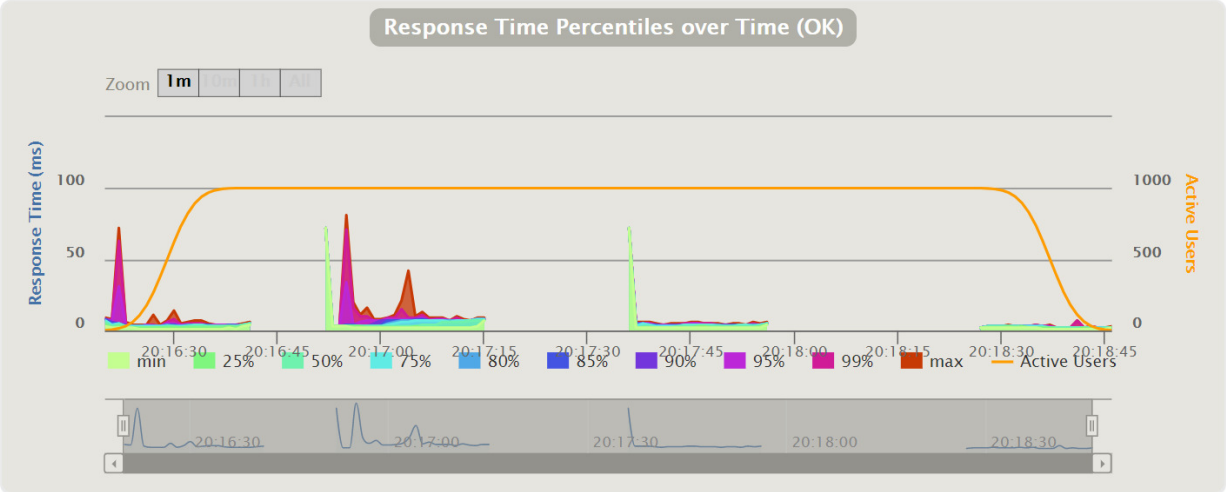
```
val usersCsv = csv(resourcePath + "user_data.csv")

val scn = scenario("GatlingTest")
    .feed(usersCsv)
    .exec(http("create_account_request")
              .post("/createAccountA")
              .headers(headers_1)
              .formParam("login", "${user_name}")
              .formParam("password", "${password}")
              .formParam("email", "${email}")
              .formParam("comment", "${comment}"))
    .pause(randomTimeInSeconds(20, 30))
```



Rysunek 4: Wynik wizualizacji Aggregate Graph eksperymentu Apache JMeter.



Rysunek 5: Wykres statystyk czasowych realizowanych żądań na przestrzeni czasu eksperymentu LoadNinja.



Rysunek 6: Liczba realizowanych żądań na przestrzeni czasu eksperymentu Gatling.

7. Wyniki

Wybrane narzędzia były znacząco różne, każde z nich wydaje się skierowane do grupy docelowej, o różnych oczekiwaniach. Przeprowadzone testy pozwoliły na opracowanie wyników, zestawionych w tabeli 1.

Tabela 1: Wyniki eksperymentu

Apache JMeter	LoadNinja	Gatling
Tworzenie nowych testów		
Trudne. Narzędzie posiada nagrywarkę, wymaga instalacji oraz importu certyfikatów.	Łatwe. Narzędzie posiada nagrywarkę, nie wymaga żadnej konfiguracji.	Trudne. Narzędzie posiada nagrywarkę, wymaga instalacji oraz importu certyfikatów.
Modyfikacja istniejących testów		
Łatwe. W przypadku podziału testu na fragmenty wymieniane są całe bloki testu (możliwe jest nagranie pojedynczego fragmentu).	Trudne. Możliwa jest tylko podstawowa modyfikacja fragmentów kroków np. zmiana wartości parametru w formularzu.	Umiarkowanie łatwe. Kod czytelny i samo-opisujący się dzięki DSL biblioteki pod warunkiem znajomości zmian np. REST API.
Inspekcja zmian (ang. *Code review*)		
Umiarkowanie trudne. Kod ma znaczne rozmiary, format JMX jest mało popularny.	Umiarkowanie łatwe. Brak kodu do weryfikacji, każda zmiana wymaga sprawdzenia spełnienia założeń biznesowych.	Umiarkowanie łatwe. Skrypt ma małe rozmiary, jest samo-opisujący się, pisany w języku obiektowym.
Wersjonowanie testu		
Wymaga systemu kontroli wersji np. Git.	Nie wymaga systemu kontroli wersji.	Wymaga systemu kontroli wersji np. Git.
Intuicyjność interfejsu		
Interfejs nie przyjazny użytkownikowi.	Interfejs przejrzysty i nowoczesny.	Brak interfejsu użytkownika.
Język pisania skryptów		
Groovy i Bash.	Brak.	Scala.
Możliwości parametryzacji		
Satysfakcjonujące. Możliwa jest parametryzacja adresu i treści zapytania poprzez m.in. pliki płaskie, wartości pobrane z bazy danych, zmienne wyliczeniowe poprzez gotowe komponenty.	Niesatysfakcjonujące. Narzędzie obsługuje tylko pliki płaskie, brak możliwości pobrania danych z bazy danych, brak możliwości wprowadzenia zmiennych wyliczeniowych.	Bardzo duże. Możliwa jest parametryzacja testu poprzez gotowe komponenty, które obsługują liczne formaty danych lub przygotowanie własnej implementacji zasilania danymi.
Metryki rezultatów		
Umiarkowanie satysfakcjonujące. Zawiera szereg gotowych komponentów generujących gotowe zestawienia. Wykresy graficzne są mało atrakcyjne wizualnie. Możliwy eksport wyników do wybranej formy. Przykładowymi metrykami są statystyki czasowe konkretnych żądań HTTP w formie tabelarycznej (View Results in Table) i graficznej (Aggregate Graph).	Umiarkowanie satysfakcjonujące. Wersja próbna zawiera ograniczoną liczbę metryk. Są dość atrakcyjne wizualnie. Dostępny eksport rezultatów do niskiej jakości pliku PDF. Przykładowymi metrykami są statystyki czasowe konkretnych żądań http w formie tabelarycznej (Statistics) i graficznej (Response Time Percentiles over Time) oraz statystyki zbiorcze (np. Number of responses per second).	Satysfakcjonujące. Wizualizacje są nowoczesne i interaktywne, generowane w formacie HTML. Statystyki dostępne również w formie pliku płaskiego, co umożliwia np. import wyników do bazy danych. Przykładowymi metrykami są statystyki czasowe konkretnych żądań HTTP wyłącznie w formie tabelarycznej (Durations and Errors).

8. Wnioski

Przeprowadzenie eksperymentu umożliwiło sformułowanie oceny każdego z badanych narzędzi.

Apache JMeter jest rozwiązaniem bardzo kompleksowym, co stanowi o jego dużej popularności. Posiada interfejs użytkownika, który w przeciwieństwie do opinii z pozycji [5], w niniejszym artykule określono jako nieatrakcyjny. Umożliwia pisanie własnych skryptów,

jednak jego złożoność oraz wymogi do konfiguracji stacji roboczej, pozwalają twierdzić, że przygotowanie testów jest łatwe dla programisty, ale bardzo trudne dla analityka biznesowego.

LoadNinja jest narzędziem bardzo prostym, chmurowym. Zaspokaja podstawowe potrzeby biznesowe. Zgodnie z opisem z pozycji [5], utworzenie testu jest bardzo szybkie, jednak nie da się tu utworzyć bardzo skomplikowanych testów, opartych o własne skrypty. Badania potwierdziły opinię przytoczoną w pozycji [1] - LoadNinja jest płatne, ale posiada bardzo atrakcyjny interfejs użytkownika, co wyróżnia go znacząco spośród pozostałych badanych narzędzi.

Gatling nie posiada interfejsu użytkownika, czym wyróżnia się spośród pozostałych analizowanych narzędzi. Przygotowanie w nim testów było najszybsze, dzięki DSL, którego rolę podkreślono w pozycji [8] - utworzony kod był bardzo czytelny, a potencjalne modyfikacje, które należałoby poddać ocenie przez innego dewelopera zrozumiałe. To darmowe rozwiązanie ma dodatkowo niezwykle rozwiniętą społeczność. Żadne z pozostałych narzędzi nie posiada takiego wachlarza rozszerzeń do wykorzystania bezpłatnie. Dla autorów nie było problemu z nieznajomością języka Scala (na co zwracano uwagę w pozycji [1]), gdyż składnia języka jest podobna do Javy.

Podsumowując:

- Apache JMeter najbardziej nadaje się dla dużych projektów, w których zachodzi konieczność wykorzystania najpopularniejszych, sprawdzonych rozwiązań posiadających bogatą dokumentację i wsparcie techniczne.
- Gatling można polecić nowym projektom, w których nacisk kładziony jest na dalsze rozwijanie czystego kodu, a przygotowanie testów wydajności leży w domenie zespołu programistów.
- LoadNinja oferuje przyjazny interfejs, godny polecenia dla np. analityków biznesowych. Rozwiązanie sprawdzi się w dużych, rentownych projektach, w których wymagania biznesowe są jasno zdefiniowane.

Literatura

[1] B. Erinle, Performance Testing with JMeter, Packt Publishing (2015) 7-14.

[2] A. Rodrigues, B. Demion, P. Mouawad, Master Apache JMeter - From Load Testing to DevOps: Master performance testing with JMeter, Packt Publishing (2019) 23-29.

[3] When Should I use Gatling vs. JMeter? A Tale of Two Tools, https://www.flood.io/blog/when-should-i-use-gatling-vs-jmeter-a-tale-of-two-tools, [29.07.2020].

[4] LoadNinja vs. JMeter: When to Use Each of Them, https://loadninja.com/resources/articles/performance-testing/loadninja-vs-jmeter-when-to-use-each-of-them/, [30.07.2020].

[5] 13 BEST Performance Testing Tools. Load Testing Tool (2020), https://www.guru99.com/performance-testing-tools.html, [23.04.2020].

[6] N. Hastings, Physical Asset Management, Springer (2010) 239.

[7] Performance Testing Tutorial: What is, Types, Metrics & Example. Common Performance Problems, https://www.guru99.com/performance-testing.html#3, [03.04.2020].

[8] D. Bryant, A. Marín-Pérez, Continuos Delivery in Java: Essentials Tools and Best Practises for Deploying Code to Production, O'Reilly Media (2018), 340-345.

[9] M. Bernardino, A.F. Zorzo, E. Rodrigues, FM de Oliveira, A Domain-Specific Language for Modeling Performance Testing (2014).

[10] L. Molyneaux, The Art of Application Performance Testing, O'Reilly Media (2009) 11-50.

[11] Apache JMeter Documentation, https://jmeter.apache.org/usermanual/, [05.04.2020].

[12] Getting started with JMeter – A basic Tutorial, https://www.blazemeter.com/blog/getting-started-jmeter-basic-tutorial, [05.04.2020].

[13] LoadNinja Documentation, https://support.smartbear.com/loadninja/docs/, [21.07.2020].

[14] Gatling Documentation, https://gatling.io/docs/current/, [03.05.2020].

[15] Gatling Feeders, https://gatling.io/docs/current/session/feeder, [03.05.2020].