



# POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

**Wydział Informatyki**

**Katedra Inżynierii Oprogramowania**

Inżynieria Oprogramowania i Baz Danych

**Artur Pasik**

Nr albumu s17808

**Analiza narzędzi służących do wykonywania testów wydajnościowych  
aplikacji internetowych stworzonych w różnorodnych technologiach**

Praca magisterska napisana  
pod kierunkiem:

dr inż. Mariusz Trzaska

Warszawa, luty 2020

## Streszczenie

Praca dotyczy istotnego problemu, jakim jest wybór adekwatnego narzędzia wspierającego testy wydajnościowe. Ogromna ilość rozwiązań dostępnych na rynku sprawia, że podjęcie odpowiedniej decyzji stało się nie lada wyzwaniem. Wybór utrudniony jest dodatkowo przez dużą złożoność tego typu rozwiązań, na które składają się m.in. symulacja użytkownika za pomocą skryptu, konfiguracja odpowiedniej strategii obciążenia systemu oraz analiza wyników testu.

Praca ma na celu porównanie 5 najpopularniejszych narzędzi, pod kątem ich użyteczności, a także obsługi wymagań, jakie stawiają im testowane aplikacje. Aby zaprezentować ich pełen potencjał wybrano 3 technologie, które znacząco różnią się sposobem ich symulacji.

Analiza narzędzi została przeprowadzona z rozbiciem na dwa główne aspekty procesu: utworzenie skryptów oraz przygotowanie i przeprowadzenie testów wydajnościowych, dla których przygotowano odrębną ocenę i podsumowanie.

**Słowa kluczowe:** testy wydajnościowe, testowanie, wydajność, aplikacja internetowa, LoadRunner, Gatling, JMeter, NeoLoad, TruClient.

# Spis treści

Streszczenie .....	2
Spis treści .....	3
1. Wstęp.....	5
1.1 Cel pracy.....	5
1.2 Rozwiązania przyjęte w pracy.....	5
1.3 Rezultat pracy.....	6
1.4 Organizacja pracy .....	6
2. Testy wydajnościowe .....	8
2.1 Korzyści płynące z testowania wydajności aplikacji .....	8
2.2 Rodzaje testów wydajnościowych.....	8
2.3 Proces przeprowadzania testów wydajnościowych.....	9
3. Wykorzystane narzędzia i technologie.....	10
3.1 Opis narzędzi użytych do testów wydajnościowych .....	10
3.1.1 LoadRunner .....	10
3.1.2 Gatling .....	11
3.1.3 JMeter.....	12
3.1.4 NeoLoad.....	14
3.1.5 TruClient .....	15
3.2 Opis przetestowanych aplikacji.....	16
3.2.1 Web Tours Sample Application .....	16
3.2.2 ASP.NET Web Forms .....	17
3.2.3 Employee Directory (Flex Web Application) .....	19
4. Weryfikacja narzędzi pod kątem tworzenia skryptów dla każdej z aplikacji.....	22
4.1 Skrypty dla aplikacji Web Tours Sample Application .....	22
4.1.1 LoadRunner .....	22
4.1.2 Gatling .....	28
4.1.3 JMeter.....	33
4.1.4 NeoLoad.....	38
4.1.5 TruClient .....	44
4.2 Skrypty dla aplikacji ASP.NET Web Forms .....	49
4.2.1 LoadRunner .....	49
4.2.2 Gatling .....	53
4.2.3 JMeter.....	54

4.2.4	NeoLoad .....	56
4.2.5	TruClient .....	58
4.3	Skrypty dla aplikacji Employee Directory .....	59
4.3.1	LoadRunner .....	59
4.3.2	Gatling .....	63
4.3.3	JMeter .....	64
4.3.4	NeoLoad .....	66
4.3.5	TruClient .....	67
4.4	Analiza porównawcza narzędzi użytych do tworzenia skryptów .....	67
5.	Testy wydajnościowe przeprowadzone w celu porównania narzędzi .....	70
5.1	Przygotowanie i przeprowadzenie testów dla aplikacji Web Tours Sample Application .....	70
5.1.1	LoadRunner .....	70
5.1.2	Gatling .....	74
5.1.3	JMeter .....	77
5.1.4	NeoLoad .....	79
5.1.5	TruClient .....	83
5.1.6	Obciążenie zasobów fizycznych generatora przez każde z narzędzi podczas testów aplikacji Web Tours Sample Application .....	84
5.2	Przygotowanie i przeprowadzenie testów dla aplikacji ASP.NET Web Forms .....	87
5.3	Przygotowanie i przeprowadzenie testów dla aplikacji Employee Directory .....	91
5.4	Analiza porównawcza narzędzi użytych do testów wydajnościowych .....	92
6.	Podsumowanie .....	95
	Prace cytowane .....	96
	Wykaz rysunków .....	98
	Wykaz tabel .....	101
	Wykaz kodu .....	102

# 1. Wstęp

Każda powstająca w dzisiejszych czasach aplikacja internetowa, niezależnie od przyjętej metodyki, na wybranym etapie poddawana jest testom. Jest to bardzo ważny element całego procesu, ponieważ jest on odpowiedzialny za zapewnienie jakości produktu – weryfikowana jest jego zgodność z oczekiwaniami i wymaganiami postawionymi przez zamawiającego.

Testowanie oprogramowania nie ogranicza się jednak jedynie do aspektów funkcjonalnych. Niezwykle istotne są również testy wydajnościowe, które mają na celu zweryfikowanie stabilności działania aplikacji pod zadaniem obciążeniem. Kontrola wydajności aplikacji przed wdrożeniem ma niewątpliwie znaczący wpływ na powodzenia projektu. Pozwala ona z wyprzedzeniem odnaleźć ryzyka produktowe i jest pomocna w znalezieniu drogi do ich minimalizacji.

Aktualnie na rynku obecna jest mnogość zarówno technologii w jakich tworzone są aplikacje internetowe, jak i narzędzi do ich testowania wydajnościowego. Powyższe fakty prowadzą do tego, że wybór narzędzi, które zostają wybrane do realizacji testów często jest dziełem przypadku. Finalnie końcowy wynik daje rezultaty niższej jakości, choć można byłoby uzyskać znacznie lepsze, poprzez wybranie innego narzędzia, bardziej odpowiedniego dla danej technologii.

Selekcja dostępnych narzędzi jest dodatkowo utrudniona przez ich obszerny wachlarz funkcjonalności.

Cały proces otwiera stworzenie skryptu, którego działanie odpowiada pracy fizycznego użytkownika. W zależności od testowanego systemu, wiążą się z tym innego rodzaju wyzwania, których obsługa jest konieczna do uzyskania poprawnie działającej symulacji.

Dopiero wtedy można przystąpić do przygotowania testu. Składa się na nie szereg działań konfiguracyjnych, które mają na celu jak najwierniejsze oddanie spodziewanego ruchu w systemie (liczba użytkowników, intensywność ich działania itp.).

Nawet samo zakończenie testu nie wyczerpuje roli narzędzia, ponieważ kluczowym ich etapem jest ocena wyników. Nie bez znaczenia jest wówczas forma ich prezentacji oraz mechanizmy usprawniające ich analizę.

## 1.1 Cel pracy

Celem pracy jest weryfikacja dostępnych na rynku narzędzi typu open source, jak i komercyjnych pod względem ich użyteczności w testach wydajnościowych aplikacji napisanych w różnych technologiach. Pozwoli to w przyszłości na lepsze ich dopasowanie, a co za tym idzie przeprowadzenie wyższej jakości testów.

Wśród wielu aspektów procesu testowania, największą wagę będą miały:

- Utworzenie skryptów symulujących pracę użytkownika
- Przygotowanie i konfiguracja generatora obciążenia
- Wykonanie testu
- Jego narzut na wydajność fizycznych zasobów maszyny, na której został uruchomiony

## 1.2 Rozwiązania przyjęte w pracy

Warunkiem wykonania testów jest bez wątpienia wybranie aplikacji, którą można by temu procesowi poddać. Z uwagi na charakter weryfikacji wydajnościowej (długotrwałe obciążenie jej

komponentów), wysoce niewskazane było używanie witryn powszechnie dostępnych w sieci, należących do prosperujących firm.

Zdecydowano zatem o uruchomieniu trzech wyizolowanych aplikacji na komputerze lokalnym. Aby zaprezentować pełen potencjał narzędzi testujących, wybrano 3 technologie, które znacząco różnią się sposobem ich symulacji, mianowicie:

- Web Tours Sample Application – pobrana ze strony producenta, polecana jako pierwszy trening testowania wydajnościowego
- ASP.NET Web Forms – stworzona przez autora tej pracy w technologii zgodnej z jej nazwą
- Employee Directory – również wykonana przez autora, jednak w technologii Flex

Spśród ogromnego wachlarza dostępnych na rynku narzędzi, zostało wybranych 5 najbardziej popularnych i prezentujących przekrojowo odrębne podejścia do realizacji testów wydajnościowych:

- LoadRunner
- Gatling
- JMeter
- NeoLoad
- TruClient

### **1.3 Rezultat pracy**

Rezultatem pracy jest szczegółowa analiza wybranych narzędzi do wykonywania testów wydajnościowych, oraz ich tabelaryczna ocena z rozbiciem na poszczególne funkcjonalności.

Opis skryptów oraz wyniki testów poprzedzone zostały ich faktycznym wykonaniem. A zatem dla wszystkich, wymienionych w podrozdziale 1.2 aplikacji, zostały stworzone skrypty – po jednym dla każdego narzędzia, które było w stanie obsłużyć daną technologię. Analogicznie powstały pliki z wynikami testów oraz statystyki obciążenia maszyny.

Produktem pobocznym pracy są dwie aplikacje internetowe stworzone jako baza do testowania, a których nie udało się uzyskać w inny sposób, dającą możliwość ich swobodnego użytkowania.

### **1.4 Organizacja pracy**

Pracę otwiera rozdział opisujący ogólną charakterystykę testów wydajnościowych, ich rodzaje oraz zasadność ich wykonywania.

W dalszej części przybliżone zostały narzędzia poddane analizie, a w następnej – aplikacje, które posłużyły jako materiał do testowania.

Rozdział 4 zawiera bardzo szczegółowy opis utworzenia każdego ze skryptów z podziałem na poszczególne technologie, które zostały obsłużone oraz konkretne aspekty tego procesu. Każdy podrozdział kończy się krótkim podsumowaniem zawierającym ocenę określonych aktywności, a także argumentację, która ją uzasadnia. Na samym końcu znajduje się zestawienie podsumowujące użyteczność wszystkich narzędzi ze względu na tworzenie skryptów wraz komentarzem i ogólną konkluzją.

Kolejny rozdział prezentuje identyczną strukturę, tym razem jednak ocenie poddawane było przygotowanie i przeprowadzenie testu przez każde z narzędzi, a także analiza wyników. Zbadany został również wpływ jaki wywierają one na zasoby fizyczne maszyny, na której zostały uruchomione.

Na zakończenie przedstawiono podsumowanie całej pracy, a także przywołano ogólną ocenę i możliwości badanych narzędzi. Sformułowano również wnioski, wyciągnięte na ich podstawie.

## 2. Testy wydajnościowe

Testy wydajnościowe należą do grupy testów нефункциональных, oznacza to że nie odnoszą się bezpośrednio do funkcjonalności systemu. W tym przypadku głównym celem jest zbadanie jego charakterystyki, czyli zachowania w zadanych okolicznościach pod określonym obciążeniem.

Rozpoznana w ten sposób charakterystyka oprogramowania zestawiana jest następnie z oczekiwanym rezultatem. Tak oto zweryfikowany zostaje stopień spełnienia założonych wymagań.

Cechy systemu, które najczęściej podlegają ocenie to:

- Stabilność działania przy maksymalnej założonej liczbie użytkowników
- Akceptowalna długość czasów odpowiedzi
- Poprawność działania logiki biznesowej
- Odpowiednia reakcja na przeciążenie (co dzieje się z danymi, jak wyglądają procedury awaryjne) [1]

### 2.1 Korzyści płynące z testowania wydajności aplikacji

Testowanie wydajności aplikacji pozwala zidentyfikować problemy i poprawić ogólną efektywność, co ma przełożenie na poprawę komfortu użytkownika i zwiększenie przychodów.

Aplikacje wysłane na rynek ze słabymi wskaźnikami wydajności z powodu braku lub niedostatecznej jakości testów, prawdopodobnie zyskają złą reputację i nie osiągną oczekiwanych celów sprzedażowych. Ponadto systemy o znaczeniu krytycznym, takie jak programy kosmiczne czy ratujące życie sprzęt medyczny, powinny być również zweryfikowane pod kątem pozbawionej odchyleń, stabilnej wydajności przez długi okres [2].

Istnieje wiele typowych problemów, które mogą zostać wykryte podczas tego rodzaju testów, takich jak *wąskie gardła*. Termin ten oznacza przerwanie przepływu danych z powodu ograniczonej przepustowości. Tego typu problemy mogą pojawić się na przykład w wyniku nagłego wzrostu ruchu, którego serwery nie są w stanie obsłużyć.

*Wąskie gardła* to tylko jeden z wielu problemów, które mogą wystąpić, gdy aplikacja nie jest odpowiednio skalowalna. Niski współczynnik skalowalności może też powodować opóźnienia, błędy i wycieki pamięci.

Kolejną możliwą przyczyną występowania problemów wydajnościowych mogą być ograniczenia procesora lub przepustowości. W takim wypadku może okazać się konieczne przydzielenie maszynom większej ilości zasobów, lub zainwestowanie w bardziej niezawodną infrastrukturę [3].

Jak zatem widać, istnieje wiele potencjalnych problemów, które bez wykonania odpowiednich testów mogą objawić się dopiero na środowisku produkcyjnym, sprawiając tym samym mnóstwo kłopotów.

### 2.2 Rodzaje testów wydajnościowych

Testy wydajnościowe dzielą się na wiele różnych rodzajów, w tym podrozdziale zostaną opisane najpopularniejsze z nich. Oprócz polskich nazw zostały podane również angielskie, ponieważ w praktyce są one powszechniej używane.

**Testy obciążeniowe (Load testing)** – mają na celu zbadanie zachowania systemu pod określonym spodziewanym obciążeniem. Obciążenie zwykle rozumiane jest przez równoczesną pracę



określonej liczby użytkowników w aplikacji, wykonujących daną liczbę transakcji w sprecyzowanym czasie. Jest to najczęściej spotykana forma badania wydajności. Testy wykonane w rozdziale 5 to właśnie testy obciążeniowe.

**Testy przeciążeniowe (Stress testing)** – ich przeznaczeniem jest określenie odporności systemu pod względem ekstremalnego obciążenia, co pomaga administratorom aplikacji ustalić, jak system zareaguje, jeśli obecne obciążenie znacznie przekroczy oczekiwane maksimum.

**Testy zanurzeniowe (Soak testing)** – sprawdzają, jak zachowuje się system podczas długotrwałego użytkowania. Monitorowane jest wykorzystanie pamięci w celu wykrycia potencjalnych wycieków. Ważna jest również weryfikacja, czy przepustowość i czasy odpowiedzi po pewnym okresie przedłużonej aktywności są równie dobre lub lepsze niż na początku testu.

**Testy skokowego obciążenia (Spike testing)** - wykonuje się je poprzez nagłe zwiększenie lub zmniejszenie obciążenia generowanego przez bardzo dużą liczbę użytkowników i obserwację zachowania systemu. Celem jest ustalenie, jak system poradzi sobie z drastycznymi zmianami obciążenia.

**Testy wartości granicznych (Breakpoint testing)** – podobne do testów przeciążeniowych. Obciążenie jest stopniowo zwiększane w czasie, podczas gdy system jest monitorowany pod kątem kryteriów akceptacyjnych. Test ten określa maksymalną pojemność, poniżej której system będzie działał zgodnie z wymaganiami.

**Testy konfiguracji (Configuration testing)** - Zamiast testowania wydajności z perspektywy obciążenia, tworzone są testy w celu określenia wpływu zmian konfiguracji komponentów systemu na wydajność i zachowanie systemu.

## 2.3 Proces przeprowadzania testów wydajnościowych

Metodologia przyjęta do testowania wydajności może się znacznie różnić, ale cel testów wydajności pozostaje taki sam. Ten podrozdział prezentuje ogólny proces przeprowadzania testów wydajnościowych:

1. **Identyfikacja środowiska testowego** – rozpoznanie fizycznego środowiska testowego, środowiska produkcyjnego i dostępnych narzędzi testowych, a także zapoznanie się ze szczegółami konfiguracji sprzętu, oprogramowania i sieci używanych podczas testowania.
2. **Określenie kryteriów akceptacji** – kryteria powodzenia projektu obejmujące m.in. cele i ograniczenia dotyczące przepustowości, czasów reakcji i alokacji zasobów.
3. **Zaplanowanie i zaprojektowanie testów** – zdefiniowanie kluczowych scenariuszy, określenie zmienności między reprezentatywnymi użytkownikami, identyfikacja danych testowych, ustalenie metryk do zebrania, a także stworzenie modelu obciążenia.
4. **Konfiguracja środowiska testowego** – przygotowanie środowiska testowego przed uruchomieniem testu oraz konfiguracja narzędzi i zasobów niezbędnych do zrealizowania strategii.
5. **Wdrożenie projektu testu** – utworzenie testów wydajnościowych zgodnie z projektem.
6. **Uruchomienie testu** – uruchomienie, nadzór nad poprawnością przebiegu i jednoczesny monitoring testu.
7. **Analiza, strojenie i retest** – konsolidacja i analiza wyniki testów. Następnie dostrojenie i ponowienie testu, w celu weryfikacji, czy nastąpiła poprawa lub spadek wydajności. [2]

## 3. Wykorzystane narzędzia i technologie

W tym rozdziale przedstawione zostały narzędzia wydajnościowe będące przedmiotem oceny, a także aplikacje internetowe, które zostały za ich pomocą przetestowane.

### 3.1 Opis narzędzi użytych do testów wydajnościowych

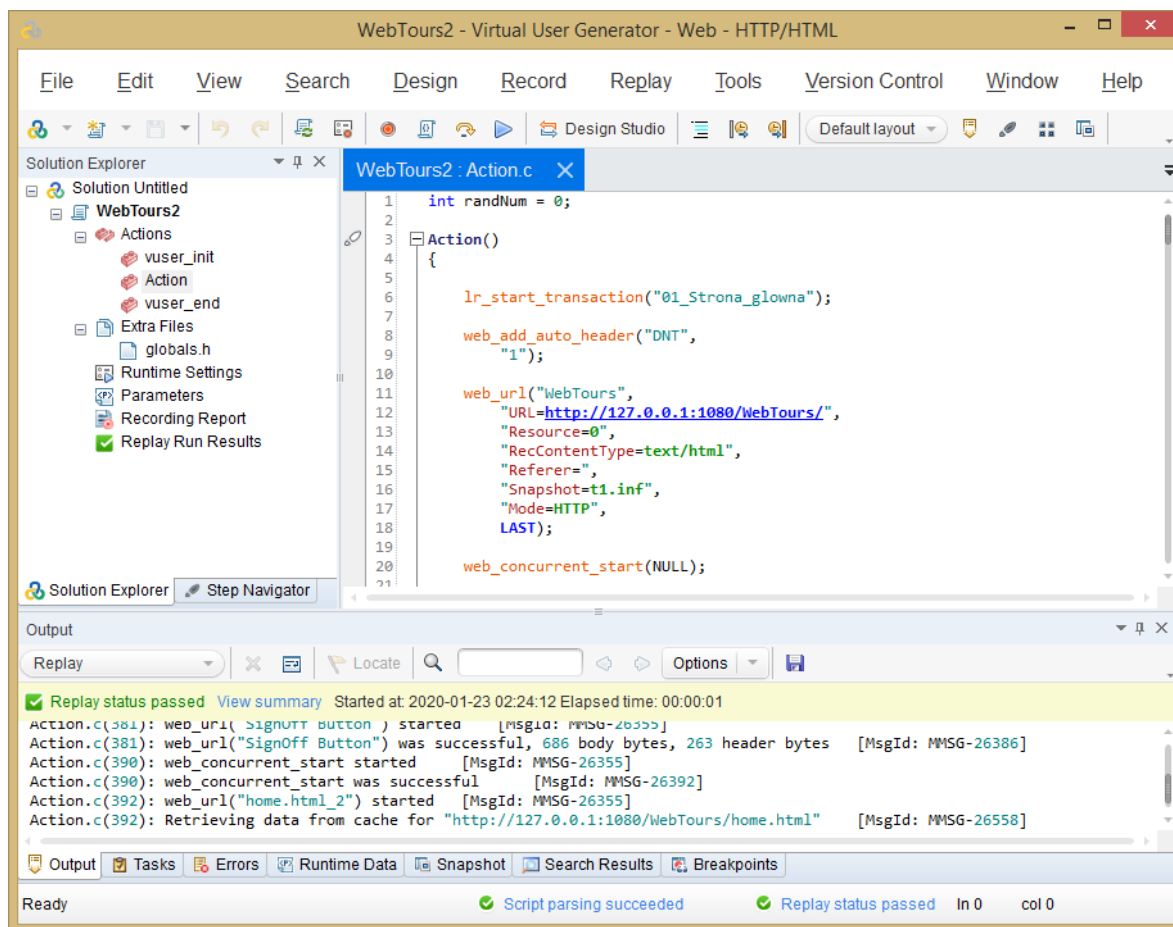
Jak już wspomniano we wstępie, obecnie na rynku dostępnych jest bardzo wiele różnorodnych narzędzi służących do wykonywania testów wydajnościowych. Na potrzeby tej pracy przeanalizowane zostało 5 z nich, które są jednymi z najpopularniejszych i najchętniej używanych. Jednocześnie zostały wybrane również w taki sposób, aby przedstawić różne podejścia do tworzenia skryptów, wykonywania testów oraz analizy wyników.

Podrozdziały 3.1.1 – 3.1.5 stanowią wprowadzenie w tematykę każdego z narzędzi, nie zawierają opisu zasad ich działania oraz szczegółowych funkcjonalności, ponieważ te zostały szeroko omówione w rozdziałach 4 i 5 (przy okazji tworzenia skryptów i testów).

#### 3.1.1 LoadRunner

LoadRunner to narzędzie służące do testowania oprogramowania należące obecnie do firmy Micro Focus. Jest ono wykorzystywane głównie do pomiaru zachowania aplikacji i wydajności pod obciążeniem.

Rysunek 1 przedstawia okno programu z częściowo wyświetlonym, przykładowym skrypcem.



Rysunek 1. Okno narzędzia LoadRunner – moduł VuGen

LoadRunner symuluje aktywność użytkownika, generując komunikaty między komponentami systemu. Wiadomości i interakcje, które mają zostać wysłane, są przechowywane w skryptach. LoadRunner tworzy je na podstawie aktywności w aplikacji, np. żądań HTTP między przeglądarką internetową klienta a serwerem aplikacji. [4]

Hewlett Packard Enterprise nabył LoadRunnera w ramach przejęcia Mercury Interactive w listopadzie 2006 r. [5] W dniu 01 września 2017 roku oprogramowanie zostało sprzedane firmie Micro Focus [6].

Narzędzie składa się z następujących komponentów:

- **Virtual User Generator (VuGen)** - rejestruje procesy biznesowe użytkowników końcowych i tworzy automatyczny skrypt do testowania wydajności.
- **Controller** - organizuje, prowadzi, zarządza i monitoruje test obciążenia.
- **Analysis** - pomaga przeglądać, analizować i porównywać wyniki testów obciążenia.
- **Load Generators** – komputery, na których wykonywana jest praca przez wirtualnych użytkowników<sup>1</sup>, w celu generowania obciążenia w systemie.[7]

Warto również dodać, że komercyjne korzystanie z LoadRunnera wymaga wykupienia odpowiedniej licencji. Jej wycena jest dokonywana indywidualnie na podstawie wielu różnych czynników, są to m.in:

- Typ protokołu
- Czas trwania licencji (dożywotnia, okresowa)
- Liczba użytkowników wirtualnych

Na potrzeby tej pracy wykorzystana została licencja typu *Community*, która daje dostęp do 50 bezpłatnych wirtualnych użytkowników dla wszystkich protokołów. Korzystanie z niej jest dozwolone jedynie do celów niekomercyjnych. [8]

### 3.1.2 Gatling

Zgodnie z [9], Gatling to bardzo sprawne narzędzie do testowania obciążenia. Został zaprojektowany z myślą o łatwości użytkowania, prostocie konserwacji i wysokiej wydajności.

Jest on oparty na zestawie narzędzi o nazwie Akka, który służy do tworzenia wysoce współbieżnych, rozproszonych i odpornych aplikacji bazujących na komunikatach dla języków Java i Scala. Dzięki temu Gatling jest w stanie uruchomić tysiące wirtualnych użytkowników na jednej maszynie. Akka zastępuje ograniczenie JVM dotyczące obsługi wielu wątków. Wirtualni użytkownicy w Gatling są oparci na wysyłaniu wiadomości zamiast tworzenia dedykowanych wątków. [10]

Podstawowe funkcje Gatlinga to:

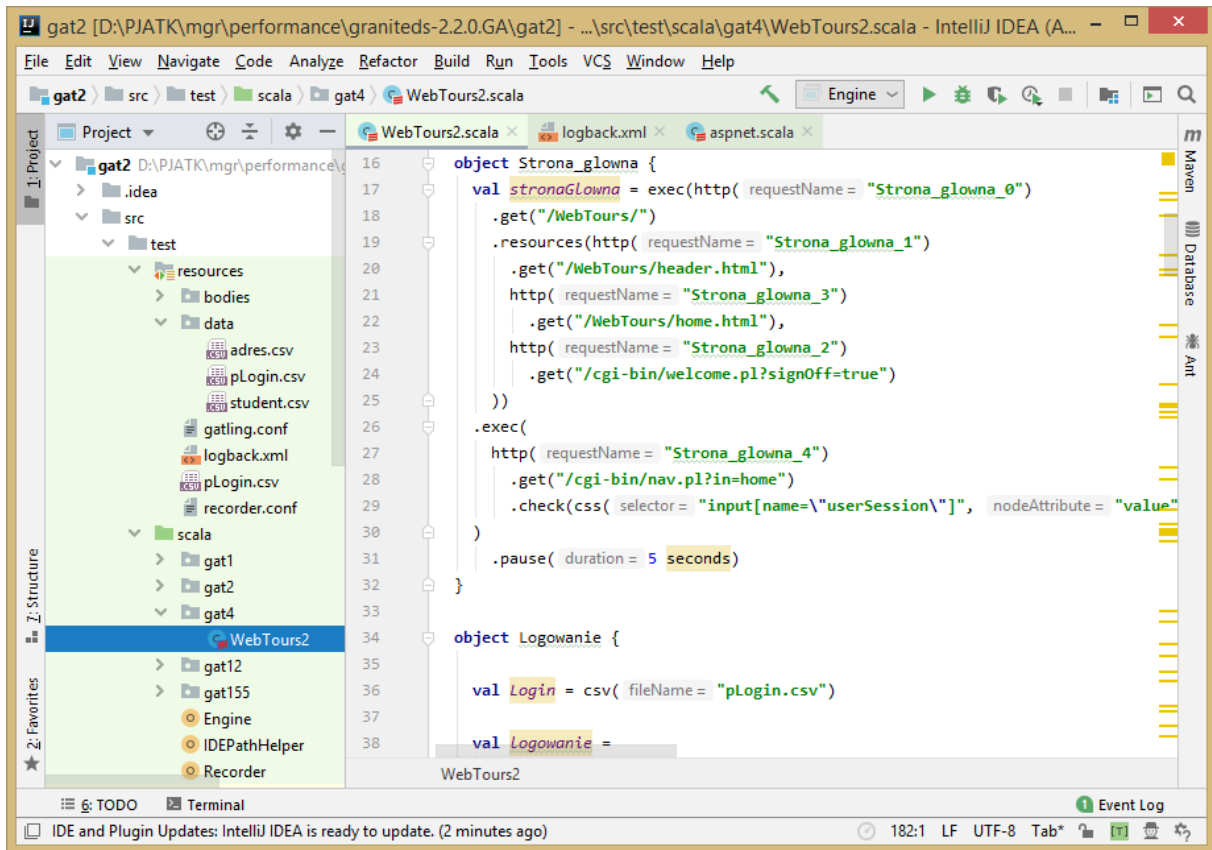
- Przygotowywanie skryptów testowych w języku Scala, nie mniej jednak wystarczy jego podstawowa znajomość, ponieważ narzędzie wykorzystuje łatwy w użyciu DSL (Domain Specific Language)
- Nagrywanie scenariuszy testowych przy pomocy dedykowanego interfejsu graficznego
- Rozszerzenia umożliwiające integrację z systemami kontroli wersji, takimi jak SBT, Maven, Jenkins

---

<sup>1</sup> Użytkownicy wirtualni (inaczej Virtual Users albo VU) - wątki wykonujące kroki scenariusza, emulując w ten sposób działania ludzkich użytkowników pracujących w aplikacji.

- Obsługa protokołów HTTP i JMS, przy czym wsparcie dla innych jest możliwe do zaimplementowania
- Obszerna i czytelna dokumentacja

Kolejną cechą tego oprogramowania jest brak własnego interfejsu graficznego dedykowanego do pisania skryptów. Na szczęście można do tego celu wykorzystać inne narzędzia, jak np. IntelliJ IDEA. Przykład takiego połączenia został zaprezentowany na rysunku 2.



**Rysunek 2. Okno IntelliJ IDEA wyświetlające skrypt stworzony w Gatlingu**

Gatling jest darmowym narzędziem, wydanym w oparciu o Apache License 2.0 w 2011 roku. [11]

### 3.1.3 JMeter

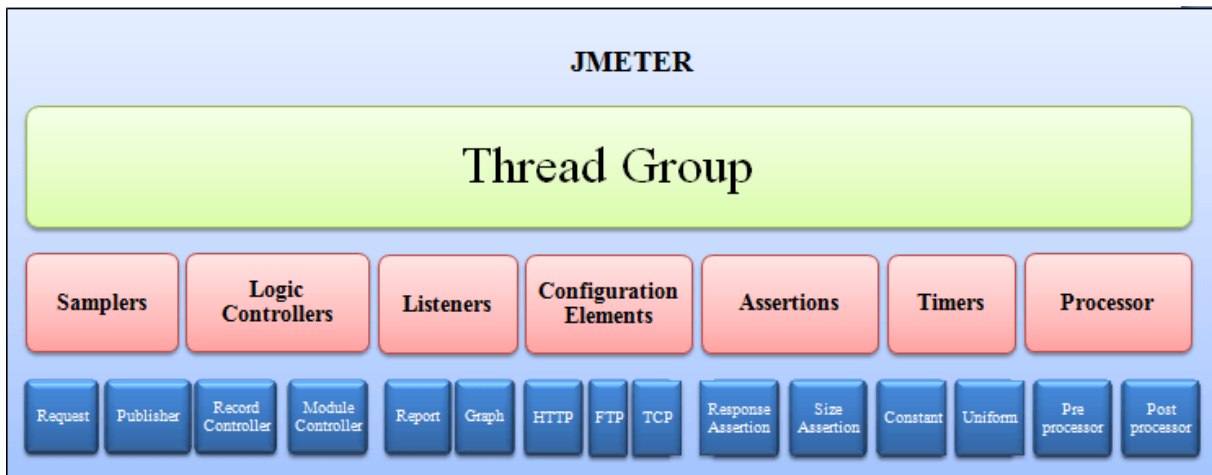
JMeter to oprogramowanie typu open source stworzone w języku Java, przez Apache Software Foundation. Pierwotnie używany był jedynie do testowania wydajnościowego aplikacji WWW lub FTP, obecnie służy także do testów funkcjonalnych, weryfikacji serwerów baz danych itp.

Główne zalety tego narzędzia to:

- Przyjazny interfejs użytkownika, który nie wymaga dużej ilości czasu na zapoznanie się z nim
- Obsługa wszystkich podstawowych protokołów: HTTP, JDBC, LDAP, SOAP, JMS i FTP
- Nagrywanie i odtwarzanie – możliwość rejestracji aktywności użytkownika wykonywanej w przeglądarce i zasymulowanie jej w aplikacji internetowej
- Wysoka rozszerzalność dzięki dużej liczbie zewnętrznych wtyczek zawierających dodatkowe funkcjonalności

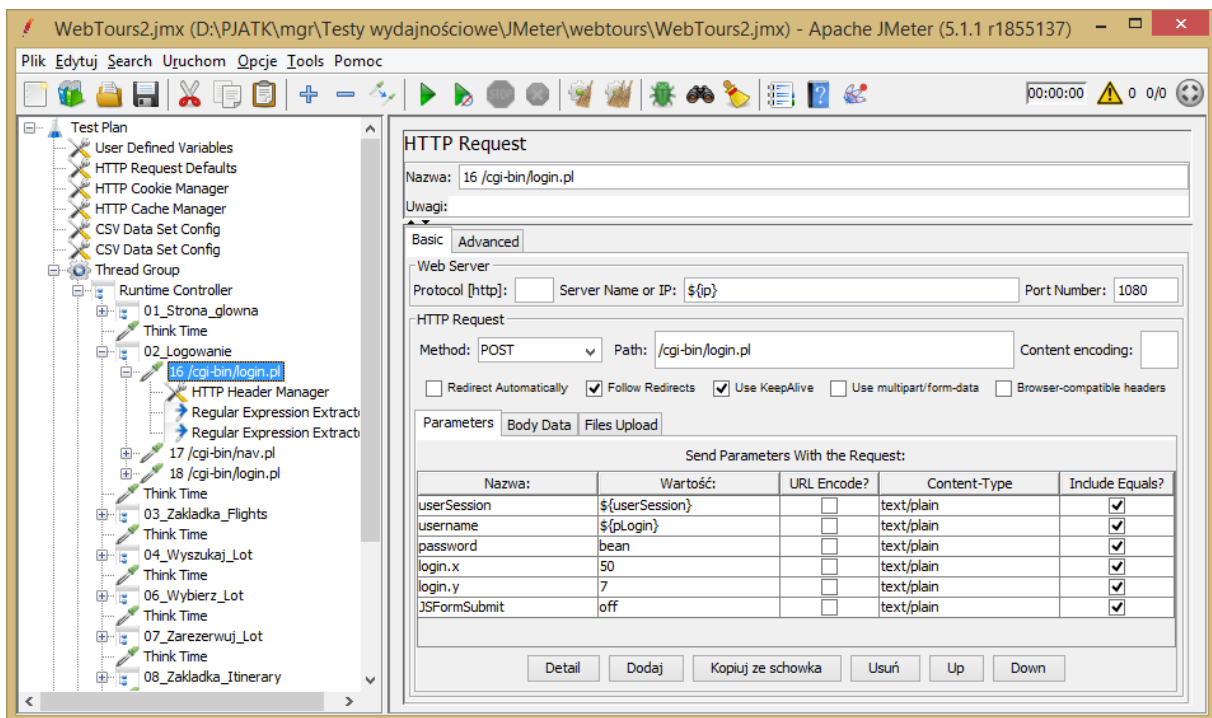
- Wbudowana integracja z narzędziami do raportowania i analizy w czasie rzeczywistym, takimi jak Graphite, InfluxDB i Grafana [12,13]

Skrypt w JMeterze jest zbudowany z graficznych komponentów, które nazywane są elementami. Każdy element jest zaprojektowany do określonego celu. Rysunek 3 przedstawia część najpowszechniej używanych elementów. Ich nazwy nie zostały przetłumaczone na język polski, ponieważ dokładnie w takiej formie są one reprezentowane w aplikacji [12].



**Rysunek 3. Struktura elementów w narzędziu JMeter, Źródło: [12]**

Rysunek 4 prezentuje okno JMetera z gotowym skrypcem, który jest przykładem praktycznego zastosowania wspomnianych elementów.



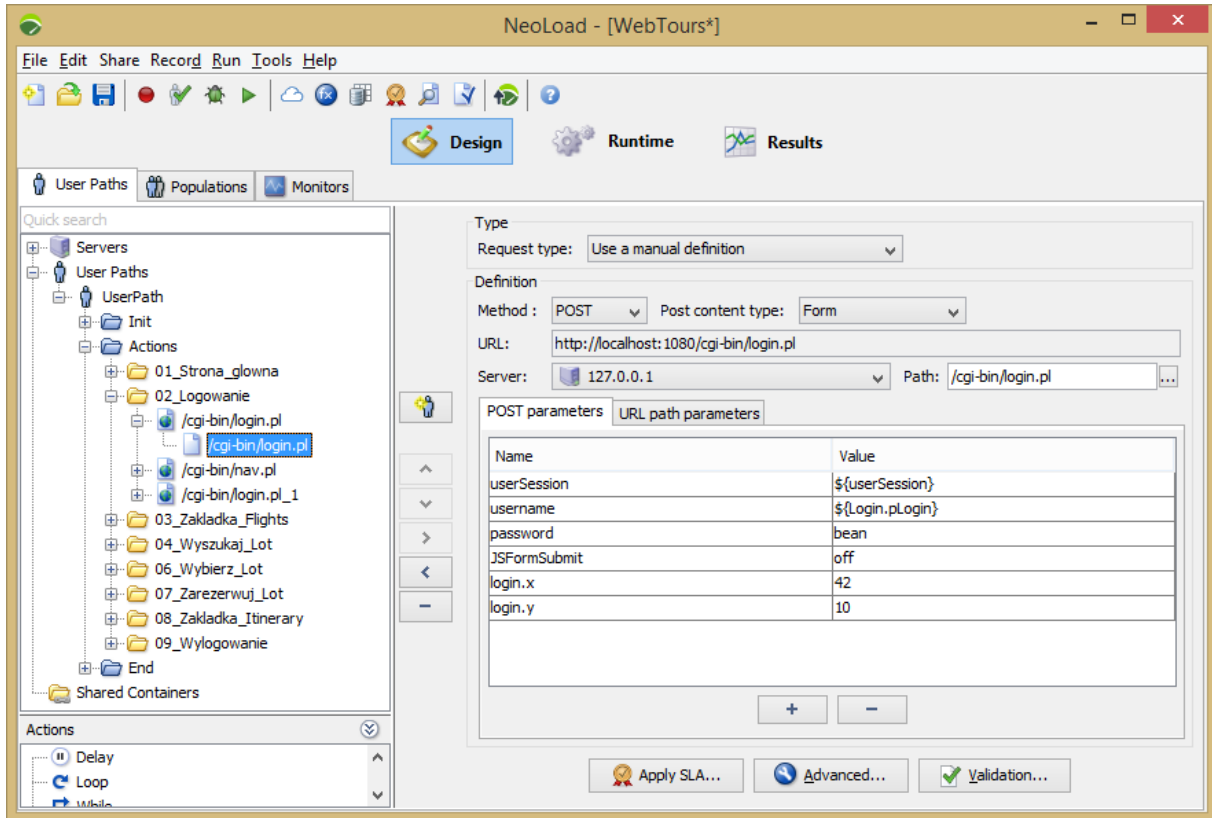
**Rysunek 4. Okno narzędzia JMeter**

Licencja typu open source sprawia, że narzędzie jest całkowicie darmowe, a także pozwala programistom używać kodu źródłowego do dalszego programowania.

### 3.1.4 NeoLoad

NeoLoad to zautomatyzowana platforma testowania wydajności zaprojektowana, opracowana i wprowadzona do obrotu przez Neotys, prywatną firmę z siedzibą we Francji.

Rysunek 5 pokazuje jak wygląda okno tego narzędzia z wyświetlonym przykładowym skrypcem.



Rysunek 5. Okno narzędzia NeoLoad

Najważniejsze cechy NeoLoada:

- Obsługa testowania wydajności wszystkich standardowych i bogatych aplikacji (RIA)
- Skrypty są opracowywane za pomocą GUI, które zapewnia warunki, pętle i inne struktury programowania.
- Nagrywanie żądań HTTPS, odtwarzanie i obsługa uwierzytelniania certyfikatów klienta.
- Wsparcie dla uwierzytelniania podstawowego, skrótowego, NTLM i opartego na formularzach.
- Obsługa wszelkiego rodzaju aplikacji internetowych, w tym tych, korzystających z J2EE, .NET, AJAX, Flex, Silverlight, GWT, SOAP, PHP, Push, itd., jeżeli są zgodne z HTTP 1.0 lub 1.1.
- Monitorowanie najnowszych serwerów WWW, baz danych i aplikacji, takich jak serwery aplikacji JBoss, HP-UX 11, Weblogic, WebSphere, Apache Tomcat i bazy danych MySQL.
- Emulacja warunków sieciowych (opóźnienie, utrata pakietów, przepustowość)

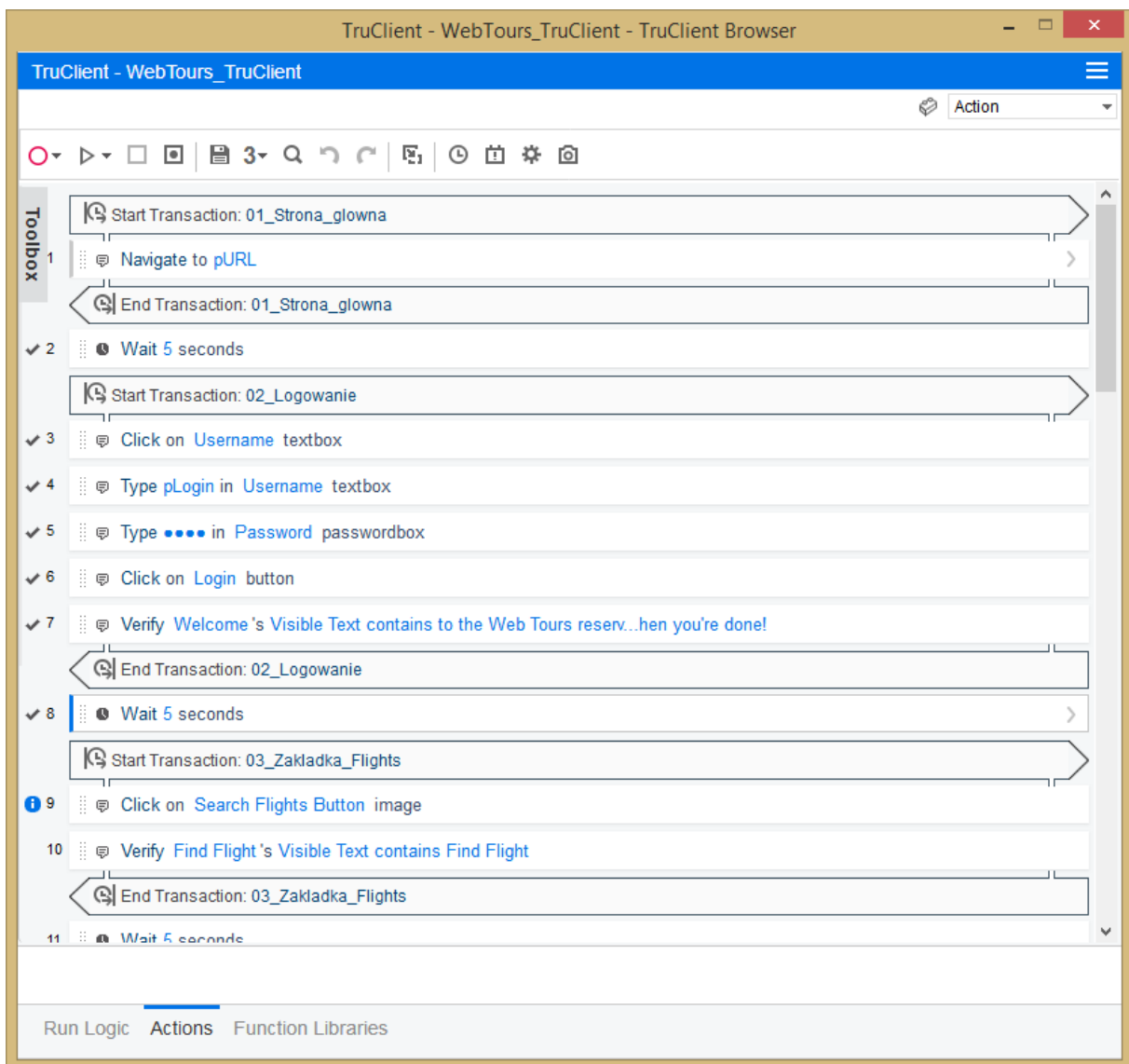
Aby możliwe było przeprowadzenie testów na szeroką skalę, NeoLoad, podobnie jak LoadRunner, wymaga wykupienia odpowiedniej licencji. Na potrzeby tej pracy została wykorzystana darmowa licencja, która umożliwia uruchomienie maksymalnie 50 równoczesnych użytkowników

wirtualnych. Zapewnia ona także obsługę najnowszych technologii internetowych oraz mobilnych, jednak jest aktywna jedynie przez rok. Firma Neotys udostępnia też inne typy licencji (Professional i Enterprise), które oferują większe możliwości (od 50 do 1 mln VU), które są wyceniane indywidualnie. [14]

### 3.1.5 TruClient

TruClient jest jednym z protokołów dostępnych w LoadRunnerze. Został on jednak opisany jako oddzielne narzędzie, ponieważ tego rodzaju skrypt ma niewiele wspólnego z pozostałymi protokołami bazującymi na wysyłaniu zapytań między klientem a serwerem aplikacji. W odróżnieniu od nich, TruClient działa na warstwie przeglądarki internetowej symulując interakcje z interfejsem użytkownika, takie jak np. naciśnięcie klawiszy lub ruchy myszy [4].

Rysunek 6 przedstawia okno TruClienta wyświetlające akcje przykładowego skryptu.



**Rysunek 6. Okno narzędzia TruClient**

Silnik tego narzędzia rejestruje działania użytkownika podczas nawigacji w procesie biznesowym. Tworzy skrypt w czasie rzeczywistym, pozwalając zobaczyć kroki podczas ich wykonywania [15].

## 3.2 Opis przetestowanych aplikacji

Rozdział ten zawiera opis aplikacji stworzonych w przykładowych technologiach, które zostały wybrane jako najbardziej reprezentatywne i zostały poddane testom.

Dla każdej aplikacji utworzono również scenariusz testu, czyli opis działań, które wykonywał skrypt. Każdy scenariusz składa się z nazwy transakcji<sup>2</sup> oraz opisu poszczególnych kroków, które były wykonywane w jej obrębie. Na niektóre z transakcji składa się kilka kroków, ponieważ nie każde działanie użytkownika na interfejsie graficznym powoduje komunikację z serwerem (np. wypełnienie pola tekstowego). Oddzielny pomiar czasu dla takiego kroku miał by się z celem.

### 3.2.1 Web Tours Sample Application

Web Tours Sample Application wybrano jako przykład podstawowej aplikacji internetowej dającej możliwość logowania, odczytywania i zapisywania danych do systemu. Szczegóły kont użytkowników oraz zarezerwowanych przez nich lotów są magazynowane w oddzielnych plikach tekstowych.

Aplikacja ta została wykonana przez twórców LoadRunnery w celu zademonstrowania jego możliwości. Pozwala ona na trening wykonania skryptów oraz przeprowadzenia testów wydajnościowych. Ma postać internetowego biura podróży, w którym użytkownicy łączą się z serwerem sieci Web, wyszukują loty, rezerwują je, a także sprawdzają ich trasy. Na dzień 23.02.2019 jest ona dostępna do pobrania bezpłatnie ze strony producenta. [16]

Rysunek 7 przedstawia wygląd interfejsu graficznego Web Tours Sample Application na jednej z podstron.



The screenshot shows the 'Web Tours' application interface. At the top left is the HP logo and the text 'Web Tours'. On the left side, there is a vertical menu with buttons for 'Flights', 'Itinerary', 'Home', and 'Sign Off'. The main content area is titled 'Find Flight' and contains a search form with the following fields and options:

- Departure City: Denver (dropdown menu)
- Departure Date: 01/13/2020 (text input)
- Arrival City: Los Angeles (dropdown menu)
- Return Date: 01/14/2020 (text input)
- No. of Passengers: 1 (text input)
- Roundtrip ticket:
- Seating Preference: Radio buttons for Aisle, Window, and None (None is selected).
- Type of Seat: Radio buttons for First, Business, and Coach (Coach is selected).
- A 'Continue...' button is located at the bottom of the form.

Rysunek 7. Wyszukiwarka lotów w aplikacji Web Tours

Scenariusz testu dla aplikacji został opracowany w taki sposób, aby odzwierciedlał typowy proces biznesowy, jaki wykonywałby użytkownik końcowy. Tabela 1 przedstawia jego szczegółowy opis.

---

<sup>2</sup> Transakcja to zestaw działań użytkownika końcowego, które reprezentują typową aktywność aplikacji (operacje odczytu i aktualizacji danych) [7].



**Tabela 1. Scenariusz testu dla aplikacji Web Tours Sample Application**

Lp.	Nazwa transakcji	Opis kroku
1	Strona główna	Wprowadź odpowiedni URL do przeglądarki i załaduj stronę główną.
2	Logowanie	W odpowiednich polach wprowadź nazwę użytkownika oraz hasło. Kliknij przycisk <i>Login</i> .
3	Zakładka <i>Flights</i>	W panelu po lewej stronie kliknij w zakładkę <i>Flights</i> . Otwiera się strona <i>Find Flight</i> .
4	Wyszukaj lot	W polu <i>Departure City</i> , wybierz losowe miasto. <i>Departure Date</i> : Losowa data z przyszłości. W polu <i>Arrival City</i> , wybierz kolejne losowe miasto. <i>Return Date</i> : wybierz losową datę, późniejszą niż <i>Departure Date</i> . Pole <i>Seating Preference</i> , wybierz losową wartość. Zachowaj pozostałe domyślne ustawienia i kliknij przycisk <i>Continue</i> . Otwiera się strona z wynikami wyszukiwania.
5	Wybierz lot	Zaznacz losowy wiersz i kliknij przycisk <i>Continue</i> . Otwiera się strona <i>Payment Details</i> .
6	Zarezerwuj lot	W polu <i>Credit Card</i> wpisz 12345678. W polu <i>Exp Date</i> wpisz dowolną datę z przyszłości. Kliknij przycisk <i>Continue</i> . Zostanie otwarta strona <i>Invoice</i> , wyświetlająca fakturę.
7	Zakładka <i>Itinerary</i>	W panelu po lewej stronie kliknij w zakładkę <i>Itinerary</i> . Otwiera się strona <i>Itinerary</i> wyświetlająca plan podróży.
8	Wylogowanie	W panelu po lewej stronie kliknij w zakładkę <i>Sign Off</i> . Wyświetlona zostaje strona logowania.

Potencjalnymi wyzwaniem do obsłużenia przez narzędzia testujące tę aplikację były:

- unikalny identyfikator generowany dla każdej nowej sesji
- randomizacja szczegółów rejestrowanych lotów (m.in. miejsce oraz data przylotu oraz odlotu)
- dynamiczne dane personalne pasażera

### 3.2.2 ASP.NET Web Forms

Aplikacja ASP.NET Web Forms stworzona została specjalnie na potrzeby tej pracy. Dokonano tego na podstawie szablonu dostępnego w Visual Studio 2019 o nazwie *Aplikacja internetowa platformy ASP.NET - Web Forms* z uwierzytelnianiem dla pojedynczych kont użytkowników. Następnie rozszerzono ją o dodatkowe funkcjonalności.

Zakładka *Studenci* została dopisana przez autora pracy w celu dodania funkcjonalności opartych na przetwarzaniu dynamicznych danych przechowywanych w bazie danych SQL Server (odczyt, zapis, edycja, usuwanie). Do jej utworzenia została wykorzystana wiedza własna autora wspomagana literaturą oraz poradnikami dostępnymi w sieci [17, 18, 19, 20]. Wizualny efekt końcowy tej zakładki znajduje się na rysunku 8.

Nazwa aplikacji   Strona główna   Informacje   Kontakt   Studenci   Hello, admin@admin.com | Log off

## Students.

Students

	StudentID	FirstName	LastName	StudentCode
Wybierz1	Piotr	Kowalski	11111	
Wybierz2	Adam	Nowak	11112	
Wybierz3	Anna	Wojcik	11113	

Student ID

First Name

Last Name

Student Code

Insert   Update   Delete   Search by Student Code

© 2020 - Moja aplikacja platformy ASP.NET

**Rysunek 8. Zakładka *Studenci* aplikacji ASP.NET Web Forms**

Scenariusz testu dla aplikacji ASP.NET Web Forms został szczegółowo przedstawiony w Tabeli 2.

**Tabela 2. Scenariusz testu dla aplikacji ASP.NET**

Lp.	Nazwa transakcji biznesowej	Opis kroku
1	Strona główna	Wprowadź odpowiedni URL do przeglądarki i załaduj stronę główną.
2	Zakładka <i>Zarejestruj się</i>	W menu na górze strony kliknij w link <i>Zarejestruj się</i> . Otwiera się formularz tworzenia nowego konta.
3	Rejestracja	W odpowiednich polach wprowadź nazwę użytkownika oraz hasło. Kliknij przycisk <i>Zarejestruj..</i>
4	Zakładka <i>Szczegóły konta</i>	W menu na górze strony kliknij w link z nazwą zalogowanego użytkownika. Otwiera się strona <i>Zarządzanie kontem</i> .
5	Zakładka <i>Zmień hasło</i>	W panelu po prawej stronie kliknij w link <i>Zmień</i> , znajdujący się obok etykiety <i>Hasło</i> . Otwiera się strona <i>Zarządzanie hasłem</i> .
6	Zmiana hasła	W odpowiednich polach wprowadź bieżące oraz nowe hasło. Następuje powrót na stronę <i>Zarządzanie kontem</i> , na której teraz dodatkowo

		widoczny jest napis: <i>Hasło zostało zmienione.</i>
7	Zakładka <i>Studenci</i>	W menu na górze strony kliknij w link <i>Studenci</i> . Otwiera się strona <i>Students</i> .
8	Dodaj studenta	W polu <i>First Name</i> wprowadź imię studenta, w polu <i>Last name</i> wprowadź jego nazwisku, z kolei w polu <i>Student Code</i> wprowadź pięciocyfrowy numer identyfikacyjny (unikalny losowy numer). Kliknij przycisk <i>Insert</i> .
9	Wyszukaj po dodaniu	W polu <i>Student Code</i> wprowadź pięciocyfrowy numer identyfikacyjny – identyczny jak w kroku 8. Kliknij przycisk <i>Search by Student Code</i> . W tabeli <i>Students</i> znajdują się teraz tylko dane dodanego studenta. Te same dane widoczne są w polach formularza poniżej tabeli.
10	Zaktualizuj studenta	W tabeli <i>Students</i> wybierz rekord z dodanym w kroku 8 studentem. Wprowadź nowe wartości dla pól <i>First Name</i> i <i>Last name</i> . Kliknij przycisk <i>Update</i> .
11	Wyszukaj po aktualizacji	W polu <i>Student Code</i> wprowadź pięciocyfrowy numer identyfikacyjny – identyczny jak w kroku 8. Kliknij przycisk <i>Search by Student Code</i> . W tabeli <i>Students</i> znajdują się teraz uaktualnione dane studenta. Te same dane widoczne są w polach formularza poniżej tabeli.
12	Usuń studenta	W tabeli <i>Students</i> wybierz rekord z dodanym w kroku 8 studentem. Kliknij przycisk <i>Delete</i> .
13	Wyszuka po usunięciu	W polu <i>Student Code</i> wprowadź pięciocyfrowy numer identyfikacyjny – identyczny jak w kroku 8. Kliknij przycisk <i>Search by Student Code</i> . W polu <i>Student ID</i> pojawia się komunikat <i>Wrong Student Code</i> . Tabela <i>Students</i> wyświetla pełną listę studentów.
14	Wylogowanie	W menu na górze strony kliknij w link <i>Log Off</i> . Wyświetlona zostaje strona główna.

Głównym wyzwaniem przy tworzeniu skryptów dla tej witryny były tzw. stany widoku (ang. *view state*). Jest to mechanizm służący do przechowywania danych, którą muszą być dostępne pomiędzy następującymi po sobie postbackami [21]. Po nagraniu ma postać obszernego parametru tekstowego, a im więcej informacji jest w nim przechowywana, tym jest dłuższy.

Kolejnymi nieszablonowymi czynnikami były:

- Żądania wysyłane równocześnie, w sposób asynchroniczny
- Dynamiczne generowanie losowego adresu e-mail, który stanowił jednocześnie login do aplikacji
- Dynamiczne generowanie losowej liczby na potrzeby pola *Student Code*

### 3.2.3 Employee Directory (Flex Web Application)

Faktem jest, że w dzisiejszych czasach technologia Flash jest już nieco przestarzała i cieszy się dość niechlubną sławą ze względu na liczne luki bezpieczeństwa wykorzystywane przez złośliwe

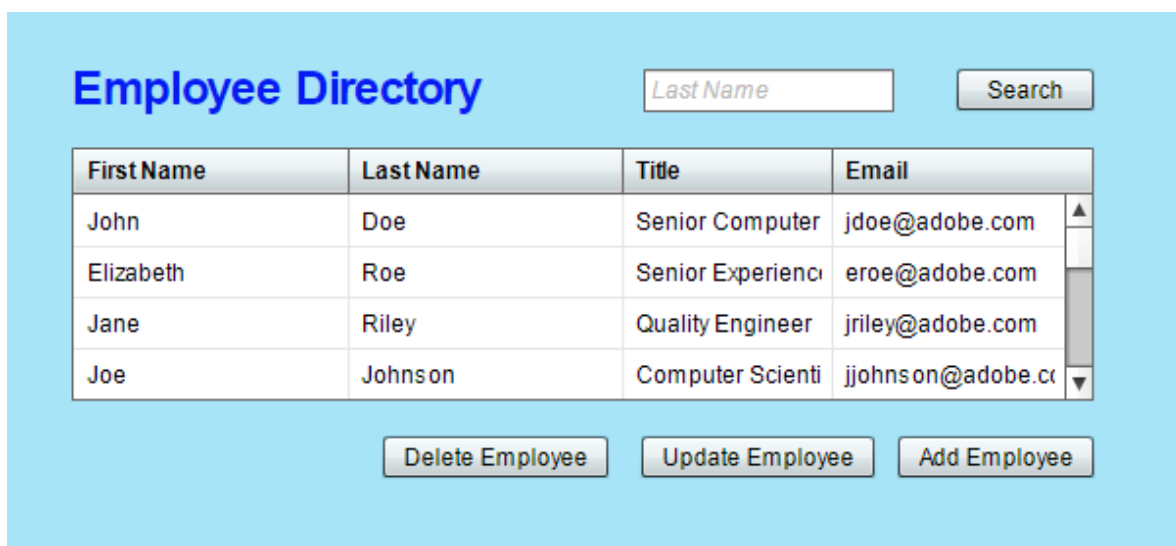
oprogramowanie [22]. Niemniej jednak jest ona ciekawym przykładem aplikacji, która komunikuje się z serwerem przy użyciu niestandardowej serializacji danych.

Aplikacja Flex działa na komputerze klienta we wtyczce Flash Player przeglądarki internetowej. Kiedy kod bajtu (plik SWF) dociera do Flash Playera, ten ostatni stosuje tak zwany kompilator Just-in-Time, aby przekształcić go w kod binarny specyficzny dla platformy. W ten właśnie sposób Employee Directory komunikuje się z serwerem używając protokołu AMF (Action Message Format). Bardzo istotną jego cechą jest serializacja obiektów Action Script<sup>3</sup>, ponieważ każde z narzędzi testowych musi zinterpretować komunikację w formacie binarnym, której standardowy użytkownik nie może odczytywać i modyfikować. [23]

Flex został użyty zatem jedynie jako przykład, reprezentując grupę technologii wymagającej niestandardowej obsługi żądań i odpowiedzi.

Aplikacja Employee Directory została stworzona specjalnie na potrzeby tej pracy przez jej autora. Posłużył do tego poradnik znajdujący się na stronie [24]. Aplikacja ma postać systemu typu CMS (Content Management System) służącego do zarządzania danymi pracowników firmy. Umożliwia on dodanie pracownika, jego aktualizację oraz usunięcie. Możliwe jest również jego wyszukanie oraz wyświetlenie aktualnych informacji o nim. Baza danych jaka została wykorzystana w tej aplikacji to ColdFusion.

Rysunek 9 przedstawia tabelę z podstawowymi danymi pracowników, oraz przyciski służące do wykonywania na nich różnego rodzaju operacji.



**Rysunek 9. Strona główna aplikacji Employee Directory**

Analogicznie do poprzednich aplikacji, dla Employee Directory również został przygotowany scenariusz testowy. Jego opisowa forma znajduje się w Tabeli 3.

<sup>3</sup> Action Script to skryptowy język programowania używany do sterowania filmami i aplikacjami Flash. Jego składnia jest podobna do JavaScript [23].

**Tabela 3. Scenariusz testu dla aplikacji Employee Directory**

Lp.	Nazwa transakcji biznesowej	Opis kroku
1	Strona główna	Wprowadź odpowiedni URL do przeglądarki i załaduj stronę główną.
2	Dodaj pracownika	Kliknij przycisk <i>Add Employee</i> . Wypełnij wszystkie pola formularza, a następnie naciśnij przycisk <i>Add Employee</i> na samym dole.
3	Wyszukaj po dodaniu	W polu <i>Last Name</i> wprowadź nazwisko dodanego w poprzednim kroku pracownika. Naciśnij przycisk <i>Search</i> .
4	Zaktualizuj pracownika	Kliknij w jeden z rekordów tabeli, w celu wczytania szczegółów danego pracownika, a następnie kliknij przycisk <i>Update Employee</i> . Wprowadź zmiany w kilku wybranych polach formularza, a następnie naciśnij przycisk <i>Update Employee</i> na samym dole.
5	Wyszukaj po aktualizacji	W polu <i>Last Name</i> wprowadź nazwisko zaktualizowanego w poprzednim kroku pracownika. Naciśnij przycisk <i>Search</i> .
6	Usuń pracownika	Kliknij w jeden z rekordów tabeli, w celu wczytania szczegółów danego pracownika, a następnie kliknij przycisk <i>Delete Employee</i> .
7	Wyszukaj po usunięciu	W polu <i>Last Name</i> wprowadź nazwisko usuniętego w poprzednim kroku pracownika. Naciśnij przycisk <i>Search</i> . Tabela wynikowa jest pusta, bądź zawiera dane innych pracowników o tym samym nazwisku.

Największym wyzwaniem dla narzędzi testujących był przede wszystkim sam protokół AMF opisany na wstępie tego podrozdziału.

## 4. Weryfikacja narzędzi pod kątem tworzenia skryptów dla każdej z aplikacji

Utworzenie skryptu przy użyciu każdego z analizowanych narzędzi polegało na zarejestrowaniu działań wykonywanych przez fizycznego użytkownika, a następnie nadaniu im odpowiedniej formy umożliwiającej ich odtworzenie.

W tym rozdziale został opisany proces tworzenia skryptów i obsługiwanie przez dane narzędzia konkretnych czynności z tym związanych. Szczegóły tych aktywności zostały zaprezentowane za pomocą konkretnych przykładów.

Podstawowy proces powstawania skryptu dla wszystkich narzędzi został przedstawiony w podrozdziale 4.1 dotyczącym aplikacji Web Tours Sample Application. Każde z narzędzi wykonuje czynności prowadzące do tego samego celu, jednak w odrębny, specyficzny dla siebie sposób.

Kolejne aplikacje: ASP.NET Web Forms oraz Employee Directory są niejako uzupełnieniem – weryfikują dodatkowe funkcjonalności, których nie obejmuje aplikacja Web Tours. Aktywności, które wyglądają identycznie niezależnie od technologii, jak np. nagranie skryptu, zostały w ich przypadku pominięte.

Na końcu każdego podrozdziału znajduje się tabela podsumowująca czynności dokonane podczas tworzenia skryptów. Każde z narzędzi zostało poddane ocenie w skali 1-10 w zależności od tego, jak dobrze sprawdza się ono w obrębie danego zagadnienia, a także z uwzględnieniem subiektywnych odczuć autora pracy.

### 4.1 Skrypty dla aplikacji Web Tours Sample Application

W tym podrozdziale omówiony został pełny proces powstawania skryptów dla aplikacji Web Tours Sample Application.

#### 4.1.1 LoadRunner

Instalacja oprogramowania była niezwykle prosta i nie sprawiła żadnych problemów. Wystarczyło uruchomić instalator i podążać za jego kolejnymi wskazówkami

Pierwszym krokiem do utworzenia skryptu było wybranie odpowiedniego protokołu dedykowanego dla danej technologii. W tym przypadku był to *Web – HTTP/HTML*. Następnie zainicjowano nagrywanie, co skutkowało uruchomieniem instancji wybranej wcześniej przeglądarki. Przy jej użyciu wykonano manualnie wszystkie wymagane czynności zgodnie ze scenariuszem testowym opisanym w podrozdziale 3.2.1.

Podczas nagrywania cały czas widoczny był widżet z dostępnymi opcjami ułatwiającymi nadanie skryptowi odpowiedniej formy (rysunek 10). Szczególnie pomocne okazały się opcje odpowiedzialne za rozpoczęcie i zakończenie transakcji (*Insert Start Transaction, Insert End Transaction*). Dzięki nim, już na etapie ręcznego poruszania się po aplikacji można było określić, gdzie zaczyna się, a gdzie kończy każda z kolejnych transakcji. Było to znacznym ułatwieniem, ponieważ wstawianie ich pomiędzy już nagranyimi żądaniem http/https zajęłoby zdecydowanie więcej czasu.



**Rysunek 10. Widżet zarządzający nagrywaniem ruchu sieciowego w narzędziu LoadRunner**

Nagrywanie zakończyło się w momencie naciśnięcia przycisku *Stop Recording* na wcześniej wspomnianym widżecie. Jego produkt końcowy stanowił specyficzny dla LoadRunniera kod bazujący na języku C, który reprezentował komunikację wysyłaną przez klienta podczas nagrywania skryptu. Kod 1 przedstawia postać przykładowego żądanie bezpośrednio po jego nagraniu (w tym wypadku jest to wyszukanie odpowiedniego lotu w aplikacji WebTours).

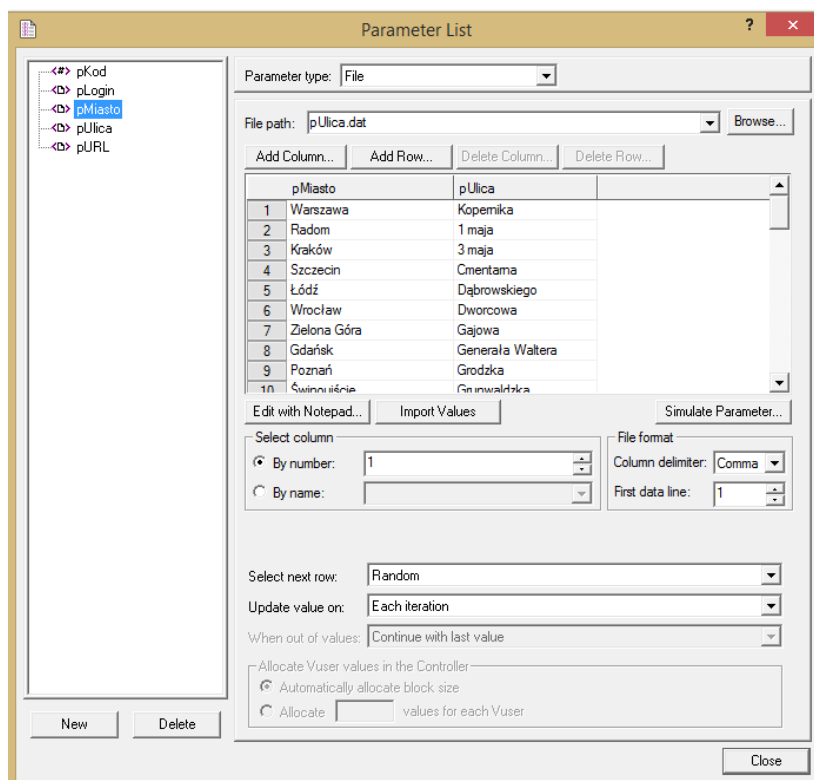
### Kod 1. Przykładowy request nagrany w narzędziu LoadRunner

```
lr_start_transaction("04_Wyszukaj_Lot");

web_submit_data("reservations.pl",
  "Action={pURL}/cgi-bin/reservations.pl",
  "Method=POST",
  "TargetFrame=",
  "RecContentType=text/html",
  "Referer={pURL}/cgi-bin/reservations.pl?page=welcome",
  "Snapshot=t4.inf",
  "Mode=HTML",
  ITEMDATA,
  "Name=advanceDiscount", "Value=0", ENDITEM,
  "Name=depart", "Value=Denver", ENDITEM,
  "Name=departDate", "Value=02/26/2019", ENDITEM,
  "Name=arrive", "Value=London", ENDITEM,
  "Name=returnDate", "Value=03/26/2019", ENDITEM,
  "Name=numPassengers", "Value=1", ENDITEM,
  "Name=seatPref", "Value=Aisle", ENDITEM,
  "Name=seatType", "Value=Coach", ENDITEM,
  "Name=.cgifields", "Value=roundtrip", ENDITEM,
  "Name=.cgifields", "Value=seatType", ENDITEM,
  "Name=.cgifields", "Value=seatPref", ENDITEM,
  "Name=findFlights.x", "Value=38", ENDITEM,
  "Name=findFlights.y", "Value=7", ENDITEM,
  LAST);

lr_end_transaction("04_Wyszukaj_Lot", LR_AUTO);
```

Jak widać na przykładzie Kodu 1, w skrypcie pojawiły się również dynamiczne wartości, które należało sparametryzować (nazwy miast, ulic i pozostałe dane wprowadzone podczas wypełniania formularza). W przeciwnym razie skrypt za każdym razem wysyłałby identyczne parametry lotu, a zgodnie ze scenariuszem powinny one być wybierane losowo. Aby temu zapobiec, dla każdej wartości tekstowej została utworzona kolekcja, z której wybierany był losowy element. Sposób implementacji tego rozwiązania znajduje się na rysunku 11.



Rysunek 11. Plik kolekcji zawierające wartości dla parametru pMiasto

Data wylotu oraz powrotu za każdym razem musiała zostać określona przez użytkownika, zatem obsłużono ją w inny sposób. Do wygenerowanego skryptu dopisano własny kod (kod 2). Posłużono się w tym celu wbudowaną w LoadRunniera funkcją *lr\_save\_datetime()* co pozwoliło na zapisanie wartości daty do parametru, a także kolejną z języka C - *rand()*, w celu jej randomizacji.

### Kod 2. Obsługa dat w narzędziu LoadRunner

```

randNum = rand() % 30 + 1;
lr_save_datetime("%m/%d/%Y", DATE_NOW + ONE_DAY * randNum, "departDate");
randNum = rand() % 30 + 30;
lr_save_datetime("%m/%d/%Y", DATE_NOW + ONE_DAY * randNum, "returnDate");

```

W przypadku pól rozwijalnych, ich wartości za każdym razem pobierane były bezpośrednio z odpowiedzi na wysłane wcześniej żądanie. Pozwalało na to utworzone w tym celu wyrażenie regularne. W ten sposób powstawała lista opcji, z której losowana była ostateczna wartość w celu późniejszego przesłania jej poprzez żądanie. Dzięki takiemu działaniu skrypt był bardziej odporny na potencjalne przyszłe zmiany w aplikacji – zawsze wybierze dostępną wartość, nawet jeśli zawartość pola rozwijalnego ulegnie zmianie. Proces ten nazywany bywa też korelacją<sup>4</sup>. Kod 3 przedstawia sposób w jaki wyciągane były wspomniane wartości, na przykładzie parametru przeznaczonego dla pola *Departure City*.

<sup>4</sup> Korelacja, jak sama nazwa wskazuje, jest mechanizmem definiowania relacji między dwiema zmiennymi lub bytami. Słownik definiuje go jako relację statystyczną między dwiema lub więcej zmiennymi, tak że systematyczne zmiany w drugiej towarzyszą systematycznym zmianom wartości jednej zmiennej. [25]

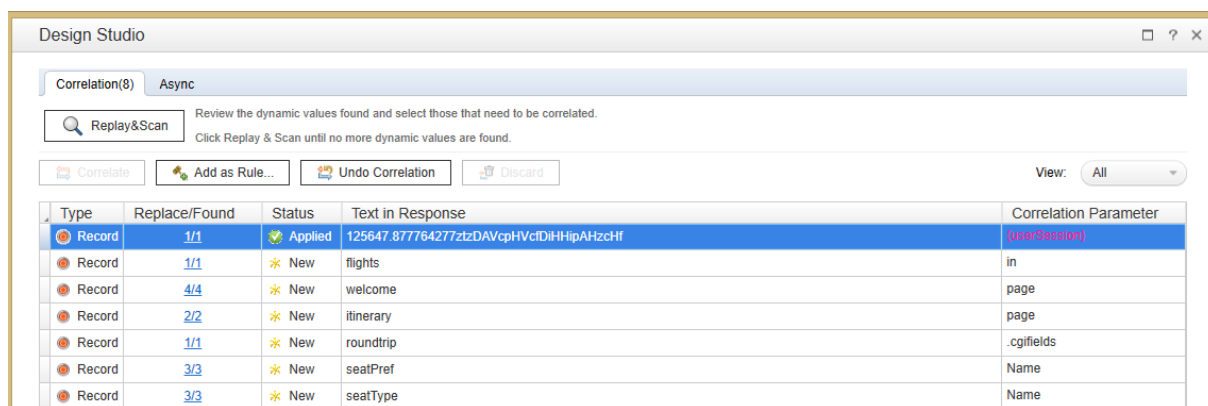


### Kod 3. Mechanizm przechwytywania wartości z odpowiedzi w narzędziu LoadRunner

```
web_reg_save_param_regexp (
    "ParamName=depart",
    "RegExp=\">(.*?)</option>",
    "ordinal=All",
    SEARCH_FILTERS,
    "Scope=Body",
    "IgnoreRedirections=No",
    "RequestUrl=*/reservations.pl*",
    LAST);
lr_save_string(lr_paramarr_random("depart"), "rndDepart");
```

Kolejną cechą aplikacji Web Tours, którą należało obsłużyć było generowanie przez serwer unikalnego identyfikatora dla każdej nowej sesji. Podczas próby odtworzenia nagranych skryptów aplikacja tworzyła nowy identyfikator (podczas gdy skrypt wysyłał ten wcześniej nagrany, już nieaktualny), zatem pomimo że wykonywane były pozornie te same czynności, uruchomienie skryptu kończyło się niepowodzeniem. Ponownie konieczne było utworzenie dynamicznego parametru działającego na podobnej zasadzie, co opisany już mechanizm przechwytywania wartości dla pól rozwijalnych.

Tym razem jednak możliwe było wykorzystanie do tego wbudowanej funkcji LoadRunniera, mianowicie *Design Studio*. Dokonuje ona automatycznego skanu nagranych ruchów w poszukiwaniu dynamicznych wartości, a następnie po zaakceptowaniu przez użytkownika wybranych korelacji, automatycznie tworzy blok z wyrażeniem regularnym (takim jak Kod 2).

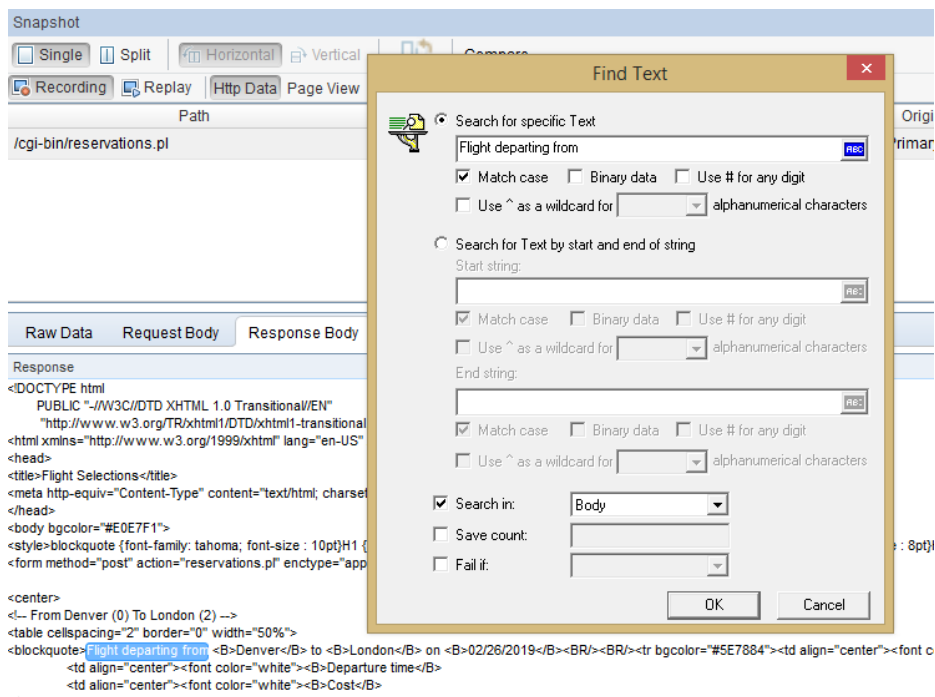


Type	Replace/Found	Status	Text in Response	Correlation Parameter
Record	1/1	Applied	125647.877764277ztzDAVcpHVctDIHIpAHzcHf	(userSession)
Record	1/1	New	flights	in
Record	4/4	New	welcome	page
Record	2/2	New	itinerary	page
Record	1/1	New	roundtrip	.cgifields
Record	3/3	New	seatPref	Name
Record	3/3	New	seatType	Name

Rysunek 12. *Design Studio*

Niestety funkcjonalność ta nie jest doskonała pod każdym względem i wśród wyszukanych atrybutów pojawiły się również te, które faktycznie są statyczne i przy kolejnych odtworzeniach skryptu nie zmieniają swoich wartości. Taką właśnie sytuację można zaobserwować na rysunku 12. Niemniej jednak, przy zachowaniu odpowiedniej czujności, *Design Studio* jest przydatnym mechanizmem i pozytywnie wpływa na szybkość tworzenia skryptu.

Aby mieć całkowitą pewność, że każde z wysłanych żądań otrzymało poprawną odpowiedź, zostały one opatrzone stosownymi asercjami. Wykonano to poprzez zaznaczenie stosownego tekstu w ciele odpowiedzi i wybranie opcji *Add Text Check Step*. Funkcja ta udostępnia również szeroki wachlarz opcji dodatkowych, co zaprezentowane zostało na rysunku 13.



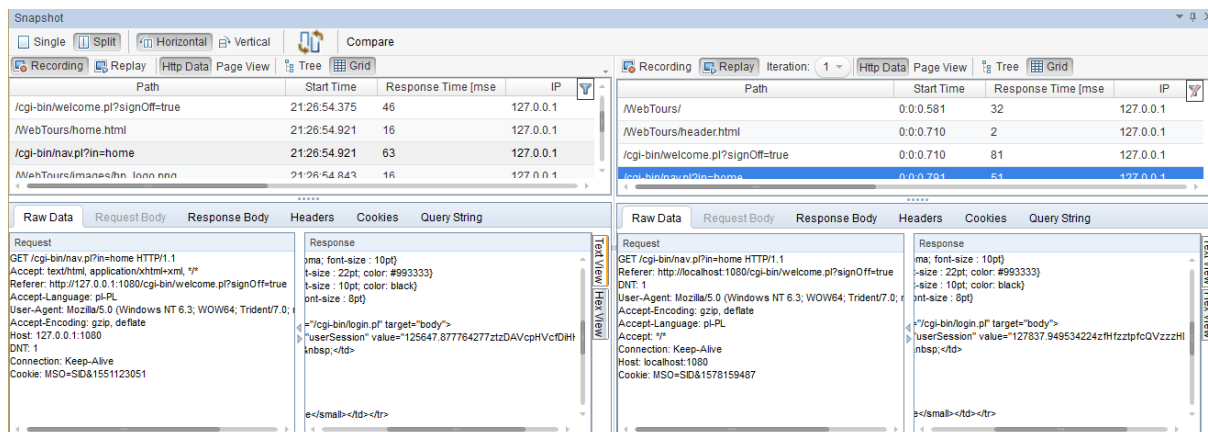
**Rysunek 13. Dodawanie asercji w LoadRunnerze**

Ostatnim elementem skryptu jest tzw. think time<sup>5</sup>. Jest ona niezwykle istotny, ponieważ bez niego skrypt wysyłałby żądania bezpośrednio jedno po drugim generując przy tym nierzeczywisty, znacznie zawyżony ruch w aplikacji. Chcąc symulować realną pracę użytkownika, konieczne było wzięcie po uwagę czynnika ludzkiego, np. wpisanie wartości w kontrolki i kliknięcie przycisku bez wąpienia zajmie kilka sekund, podczas których do serwera nie zostanie wysłany żaden komunikat.

W przypadku LoadRunnera nie było konieczności dodawania go ręcznie. Ustawienia nagrywania pozwoliły na jego automatyczne umieszczenie pomiędzy transakcjami ze zdefiniowaną wartością, już na etapie generowania kodu, bezpośrednio po nagraniu.

Naturalnie, wykonanie powyższych czynności nie było oczywiste i nie od razu przyniosło oczekiwany efekt. Tworzenie skryptu, podobnie jak programowanie wymaga również debugowania. W tym aspekcie LoadRunner również posiada odpowiedni mechanizm wspierający, tzw. *Snapshot*. Zaraz po nagraniu skryptu w sekcji tej, dostępne były wszystkie wysłane żądania i otrzymane odpowiedzi. Po uruchomieniu sparametryzowanego skryptu, analogiczna operacja został wykonana również dla nowej sesji. Dzięki temu możliwe było bardzo szybkie i czytelne porównanie obu odpowiedzi (rysunek 14). Jest to niesamowicie pomocne przy wyszukiwaniu dynamicznych wartości i definiowaniu asercji. Pozwala też w sprawny sposób odnaleźć błędy w skrypcie.

<sup>5</sup> Think time - pauza symulująca czas bezczynności użytkownika końcowego pomiędzy wykonywaniem kolejnych operacji.



**Rysunek 14. Zakładka Snapshot. Porównanie wartości z nagrania i odtworzenia.**

Zgodnie z informacją na wstępie tego rozdziału, wszystkie przedstawione w bieżącym podrozdziale aktywności zostały zebrane i ocenione w formie tabeli. Każdą z ocen opatrzono również krótkim uzasadnieniem.

**Tabela 4. Ocena narzędzia LoadRunner pod kątem tworzenia skryptu**

Cecha	Ocena	Argumentacja
<b>Instalacja narzędzia</b>	10	Prosta, standardowa instalacja z pliku instalacyjnego.
<b>Przejrzystość interfejsu</b>	10	Intuicyjny interfejs graficzny, zapewniający szybki dostęp do wszystkich potrzebnych funkcjonalności.
<b>Nagrywanie requestów</b>	10	Automatycznie uruchamiana instancja przeglądarki, bez konieczności żadnego konfigurowania jej opcji. Wygodny widżet wspierający funkcje nagrywania.
<b>Tworzenie transakcji</b>	9	Dostęp do tworzenia transakcji podczas nagrywania. Możliwość ich późniejszej edycji w kodzie (etykieta, status wykonania itp.).
<b>Obsługa zewnętrznych danych testowych</b>	8	Mechanizm <i>Parameters List</i> umożliwiający utworzenie pliku z danymi i konfigurację zmiennych. Niestety, przy większych plikach, w GUI wyświetlane jest tylko 100 pierwszych rekordów.
<b>Możliwość dodania własnego kodu</b>	7	Skrypt w całości ma postać kodu, więc dopisanie kolejnych struktur nie stanowi problemu. Dużym minusem jest konieczność używania języka C, który jest dość skomplikowany i może sprawiać trudności. Na szczęście zostało to częściowo rozwiązane przez wbudowane w narzędzie funkcje, które

		obsługują większość podstawowych operacji.
<b>Korelacja dynamicznych wartości</b>	10	Możliwość automatycznej korelacji w <i>Design Studio</i> , lub półautomatycznej z poziomu <i>Snapshotu</i> (zaznaczenie tekstu + <i>Create Correlation</i> ). Jednocześnie dostępna jest opcja utworzenie jej ręcznie przy użyciu odpowiedniej składni.
<b>Asercje</b>	10	Możliwość definiowania asercji jednym kliknięciem z poziomu <i>Snapshotu</i> . Kolejną metoda jest tworzenie ich podczas nagrywania skryptu.
<b>Think time</b>	10	Możliwość zdefiniowania stałej wartości w opcjach nagrywania. Wykonalne jest również dodanie go ręcznie za pomocą krótkiej instrukcji w kodzie skryptu.
<b>Debugowanie</b>	9	Debugowanie zoptymalizowane zostało poprzez niezwykle pomocny i czytelny mechanizm <i>Snapshot</i> .
<b>Tempo tworzenia skryptu</b>	10	Wiele usprawnień wykonujących czynności za użytkownika (transakcje, korelacje, Think time, asercje). Wygodny i przejrzysty mechanizm debugowania.

#### 4.1.2 Gatling

Gatling jest narzędziem nieco zbliżonym do LoadRunnera pod kątem reprezentacji żądań, zawiera jednak szereg znaczących różnic.

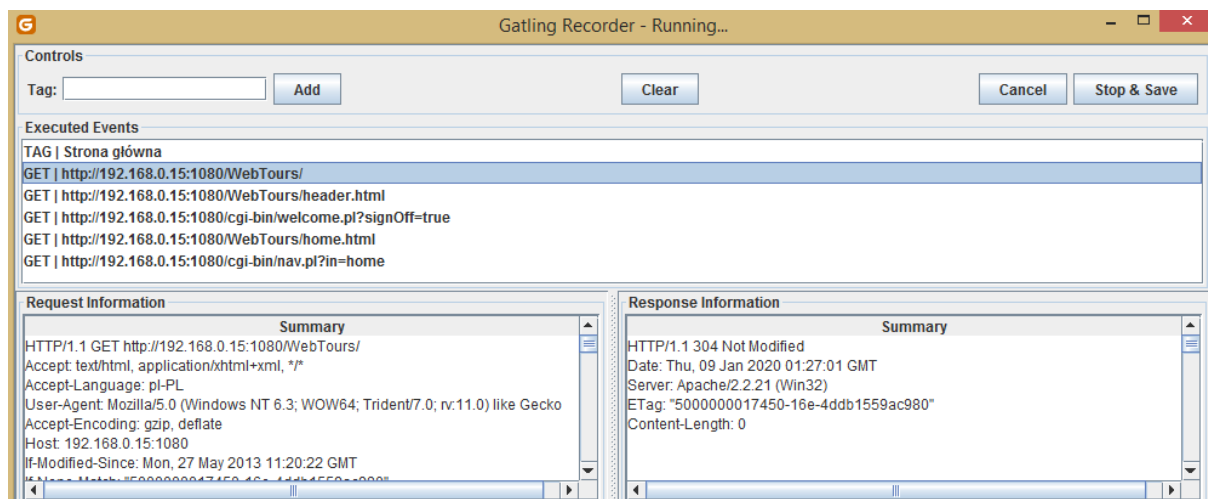
Przed wszystkim nie posiada on własnego interfejsu graficznego służącego do edycji skryptu. Aby to wykonać, jedną z opcji była edycja kodu przy pomocy dowolnego edytora tekstu. Bardziej efektywne okazało się jednak skonfigurowanie wybranego narzędzia deweloperskiego do pracy z Gatlingiem. Operację tę można było wykonać na wiele różnych sposobów, opisanych szerzej w [26].

Nie ma jednak potrzeby przytaczania każdego z nich, na potrzeby bieżącej pracy został on zaimportowany jako maven plugin przy pomocy Intellij IDEA. W celu komfortowej edycji skryptów (podpowiadanie składni, podkreślanie błędów składniowych, itp.) konieczna była zatem instalacja i konfiguracja dodatkowych narzędzi, co znacznie wydłużyło czas całego przedsięwzięcia.

Gdy środowisko do pracy zostało już odpowiednio przygotowane, przystąpiono do nagrywania skryptu. Po uruchomieniu klasy *Recorder*, wyświetliło się okno programu zawierające podstawowe opcje nagrywania.

Na tym etapie najważniejsze było ustawienie spójnego proxy zarówno w parametrach nagrywania, jak i w przeglądarce internetowej, na której wykonywany był scenariusz. Gatling bowiem zbiera ruch, który przechodzi właśnie przez ten adres proxy. Pojawiły się zatem kolejne dodatkowe czynności, które LoadRunner wykonywał automatycznie bez udziału użytkownika.

W tym narzędziu również istnieje możliwość zasygnalizowania rozpoczęcia kolejnej transakcji już podczas nagrywania, jednak funkcjonuje ona nieco inaczej niż w przypadku LoadRunnera. Mianowicie, działa na zasadzie wstawienia komentarza do kodu skryptu, tak aby można było wizualnie oddzielić od siebie poszczególne requesty i np. zgrupować je w metody. Nie dzieje się to jednak automatycznie.



**Rysunek 15. Okno nagrywania programu Gatling**

Po zakończeniu nagrywania domyślnie cała komunikacja została umieszczona w jednym jednej klasie, zwanej symulacją. Są one prezentowane w postaci kodu w języku Scala.

W celu nadania symulacji odpowiedniej formy, pozwalającej na zarządzanie każdą z akcji oddzielnie, konieczne było utworzenie dla nich oddzielnych obiektów. Zadanie to było nieco ułatwione dzięki komentarzom dodanym podczas nagrania. Kod 4 przedstawia obiekt odpowiedzialny za wyszukanie lotu. Warto przy tym wspomnieć, że etykieta żądania (*Wyszukaj Lot*) została wpisana ręcznie, już po nagraniu. Domyślnie każda z nich posiada nazwę wg wzoru: „Request\_” + kolejna liczba całkowita.

#### Kod 4. Przykładowy request nagrany w narzędziu Gatling

```
object Wyszukaj_Lot {
    val wyszukajLot = exec(http("Wyszukaj_Lot")
        .post("/cgi-bin/reservations.pl")
        .headers(headers_5)
        .formParam("advanceDiscount", "0")
        .formParam("depart", "Denver")
        .formParam("departDate", "02/26/2019")
        .formParam("arrive", "London")
        .formParam("returnDate", "03/26/2019")
        .formParam("numPassengers", "1")
        .formParam("seatPref", "Aisle")
        .formParam("seatType", "Coach")
        .formParam(".cgifields", "roundtrip")
        .formParam(".cgifields", "seatType")
        .formParam(".cgifields", "seatPref")
        .formParam("findFlights.x", "61")
        .formParam("findFlights.y", "16")
        .pause(5 seconds)
    )
}
```

Do obsługi dynamicznych wartości (nazwy miast, itp.) zostały użyte tzw. *feedery*. Gatling oferuje ich kilka rodzajów, w tym wypadku najtrafniejszym wyborem było użycie kolekcji zapisanych w plikach csv. Zostały one skonfigurowane w taki sposób, aby przy każdej iteracji wybierały losowy

wiersz z pliku. Składnia tej funkcjonalności jest bardzo zwięzła i przejrzysta, co można zaobserwować w kodzie 5.

### Kod 5. Obsługa kolekcji za pomocą *CSV Feeder*

```
val adres = csv("adres.csv").random
```

Parametry związane z datą ponownie zostały obsłużone poprzez dopisanie własnego kodu (kod 6), tym razem w języku Scala.

### Kod 6. Obsługa dat w narzędziu Gatling

```
val formatDate = new SimpleDateFormat("MM/dd/yyyy")
val cal = Calendar.getInstance()
val departDate = formatDate.format(cal.getTime())
cal.add(Calendar.MONTH, 1)
val returnDate = formatDate.format(cal.getTime())
```

Z kolei do ekstrakcji wartości dla pól rozwijalnych posłużyła funkcja *check* i odpowiednio zredagowane wyrażenie regularne. Jak pokazuje kod 7, instrukcja ta pozwala od razu na implementację losowego wyboru elementu.

### Kod 7. Mechanizm przechwytywania wartości z odpowiedzi w narzędziu Gatling

```
.check(regex("\\">>(.*?)</option>")).findAll.transform(s =>
util.Random.shuffle(s)).saveAs("AllDest"))
```

Za pomocą mechanizmu *check* został również wydobyty token sesji. Niestety Gatling nie posiada żadnego rozwiązania wspierającego korelację, więc konieczne było zdefiniowanie jej manualnie (odnalezienie odpowiedniej odpowiedzi na żądanie, napisanie wyrażenia regularnego, parametryzacja).

Ta sama funkcja posłużyła również do wykonania asercji. Jak widać w kodzie 8, tym razem wywołana została ona z parametrem *substring*, co skutkowało przeszukaniem odpowiedzi pod kątem konkretnego tekstu. W przypadku nieznaledzenia szukanej frazy żądanie przyjmuje status negatywny.

### Kod 8. Dodawanie asercji w Gatlingu

```
.check(substring("Flight departing from"))
```

Think timy w Gatlingu noszą nazwę *pause*. Zostały one wygenerowane w tych samym miejscach i o tych samych wartościach, co wykonane fizyczne pauzy podczas nagrywania. Aby nadać im ustandaryzowaną długość i rozmieszczenie konieczna była edycja kodu symulacji. Przykład pauzy znajduje się w ostatniej linii kodu 4.

Debugowanie skryptu w tym narzędziu było znacznie bardziej skomplikowane, niż miało to miejsce w LoadRunnerze. Odpowiedź na każde z żądań wysłanych podczas nagrywania zapisana została w oddzielnym pliku tekstowym o nazwie odpowiadającej jego pierwotnej etykiecie. Rysunek 16 przedstawia strukturę tych plików, a także zawartość jednego z nich.

WebTours_0000_response.txt	1	<!DOCTYPE html
WebTours_0001_response.txt	2	PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
WebTours_0002_response.txt	3	"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
WebTours_0003_response.txt	4	<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US" xml:lang="en-US">
WebTours_0004_response.txt	5	<head>
WebTours_0005_response.txt	6	<title>Welcome to Web Tours</title>
WebTours_0006_response.txt	7	<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
WebTours_0007_response.txt	8	</head>
WebTours_0008_response.txt	9	<body bgcolor="#E0E7F1">
WebTours_0009_response.txt	10	<p /><style>blockquote {font-family: tahoma; font-size : 10pt}H1 {font-family: tahoma; font-size : 22pt; color:
WebTours_0010_response.txt	11	or review/edit the flights already booked. Don't forget to sign off when
WebTours_0011_response.txt	12	you're done!
WebTours_0012_response.txt	13	</blockquote>
WebTours_0013_response.txt	14	</body>
WebTours_0014_response.txt	15	</html>
WebTours_0015_response.txt		
WebTours_0016_response.txt		
WebTours_0017_response.txt		
WebTours_0018_response.txt		
WebTours_0019_response.txt		
WebTours_0020_response.txt		
WebTours_0021_response.txt		

**Rysunek 16. Przykładowy plik z nagraniem odpowiedzi z serwera**

W sytuacji, gdy wymagany był wgląd w komunikację wysyłaną i odbieraną podczas uruchomienia skryptu, konieczne było przeszukiwanie logów wyświetlanych w konsoli narzędzia IntelliJ IDEA w sposób widoczny na rysunku 17. Z tego też miejsca odczytywano wartości jakie przyjmowały zmienne w kolejnych krokach scenariusza, a także komunikaty błędów, gdy takowe się pojawiały.

```

Engine x
↑ HTTP response:|
↓ status=
  200 OK
↻ headers=
  Date: Sat, 11 Jan 2020 00:43:41 GMT
  Server: Apache/2.2.21 (win32)
  Expires: Fri, 10 Jan 2020 00:43:41 GMT
  Transfer-Encoding: chunked
  Content-Type: text/html; charset=ISO-8859-1

body=
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US" xml:lang="en-US">
<head>
<title>Web Tours Navigation Bar</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body bgcolor="#E0E7F1">

```

**Rysunek 17. Okno konsoli IntelliJ IDEA wyświetlające treść odpowiedzi**

Poziom logowania domyślnie ustawiony był na bardzo niski, więc w celu podejrzenia ciał requestów i responsów, należało uprzednio go zwiększyć. Posłużył do tego plik konfiguracyjny Gatlinga o nazwie logback.xml i instrukcja widoczna w kodzie 9.

## Kod 9. Ustawienie rozszerzonego logowania odpowiedzi w Gatlingu

```
<logger name="io.gatling.http.ahc" level="TRACE" />
<logger name="io.gatling.http.response" level="TRACE" />
```

Tabela 5 zawiera podsumowanie i ocenę poszczególnych cech tworzenia skryptu za pomocą Gatlinga, wraz z krótkim uzasadnieniem.

**Tabela 5. Ocena narzędzia Gatling pod kątem tworzenia skryptu**

Cecha	Ocena	Argumentacja
<b>Instalacja narzędzia</b>	3	Problematiczna i czasochłonna. Konieczna instalacja dodatkowych narzędzi.
<b>Przejrzystość interfejsu</b>	3	Brak interfejsu graficznego (z wyjątkiem nagrywania). Większość parametrów jest konfigurowalna z pozycji kodu, co wymaga znajomości składni Gatlinga w wielu różnych obszarach.
<b>Nagrywanie requestów</b>	6	Przydatny interfejs zawierający wszystkie funkcjonalności potrzebne do nagrania. Niestety proxy w narzędziu i przeglądarce należy ustawić samodzielnie, aby rejestracja ruchu była w ogóle możliwa.
<b>Tworzenie transakcji</b>	0	Narzędzie nie daje możliwości definiowania transakcji.
<b>Obsługa zewnętrznych danych testowych</b>	7	Mechanizm <i>feederów</i> , który spełnia swoją funkcję, ale jest nieco kłopotliwy w konfiguracji.
<b>Możliwość dodania własnego kodu</b>	10	Skrypt w całości ma postać kodu, więc dopisanie kolejnych struktur nie stanowi problemu. Język Scala jest wygodny i intuicyjny, a co za tym idzie, świetnie sprawdza się w tej roli.
<b>Korelacja dynamicznych wartości</b>	6	Brak automatycznej korelacji, konieczne samodzielne redagowanie wyrażeń regularnych na podstawie nagranych odpowiedzi.
<b>Asercje</b>	7	Podobnie jak w przypadku korelacji. Jedyń sposób, to utworzenie ich samodzielnie, co wymaga ręcznego przeszukiwania nagranych odpowiedzi w celu odnalezienia odpowiednich wartości.
<b>Think time</b>	5	Mechanizm <i>pause</i> , z pozoru prosty, jednak bardzo nieintuicyjny w sferze zarządzania. Wprowadzona w każdym z nich wartość nie jest wartością domyślną. Konieczne jest dopisanie kolejnej instrukcji w kodzie



		( <i>constantPauses</i> ), w przeciwnym wypadku zostaną one odtworzone jako 1 sekunda.
<b>Debugowanie</b>	5	Spełnia swoją funkcję, choć jest dość niepraktyczne. Odpowiedzi rejestrowane są w kolejnych plikach tekstowych bez żadnego odniesienia, do którego żądania się odnoszą. Logi z uruchomienia wyświetlane są w konsoli IntelliJ IDEA w postaci tekstu. Wymusza to jego żmudne przewijanie w celu zlokalizowania konkretnych informacji.
<b>Tempo tworzenia skryptu</b>	5	Nagrane żądania domyślnie trafiają do jednego obiektu. Konieczny jest duży nakład pracy, aby skrypt nadawał się do uruchomienia testu. Na szczęście narzędzie posiada kilka usprawnień, które nieco niwelują ten problem. Mimo wszystko, nadal wiele czynności należy wykonać manualnie.

#### 4.1.3 JMeter

JMeter nie wymagał żadnej instalacji, wystarczyło pobranie archiwum, rozpakowanie go i po uruchomieniu odpowiedniego pliku bat, narzędzie było już gotowe do pracy.

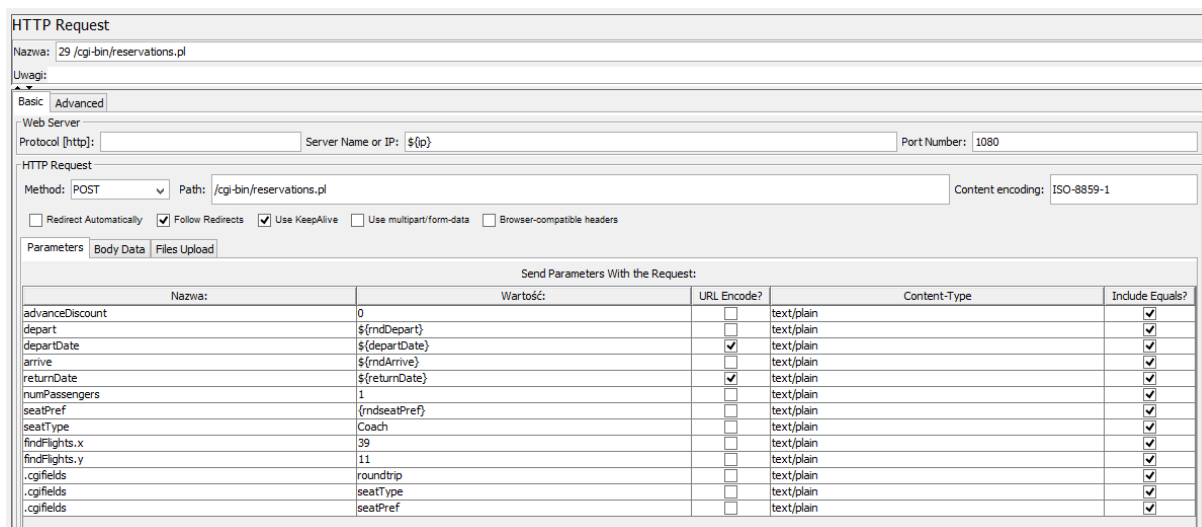
Tworzenie i edycja skryptu odbywają się przy pomocy interfejsu graficznego. Pierwszym krokiem było stworzenie nowego projektu na podstawie odpowiedniego szablonu. W ten sposób automatycznie zostały dodane do niego wszystkie elementy potrzebne do nagrania komunikacji z serwerem.

Na tym etapie najważniejszą funkcjonalną był tzw. *HTTP(S) Test Script Recorder*. Jak sama nazwa wskazuje odpowiedzialny jest on za konfigurację i przeprowadzenie nagrania. Konieczne było ustawienie odpowiedniego proxy (narzędzie + przeglądarka), analogicznie do sytuacji, jaka miała miejsce w Gatlingu.

Kolejną czynnością, którą wykonano jeszcze przed uruchomieniem nagrania było wybranie odpowiedniego mechanizmu grupowania. Bardzo pomocna okazała się opcja *Put each group in a transaction controller*, dzięki której żądania wysyłane przy okazji każdej fizycznej aktywności użytkownika (np. kliknięcie przycisku) rejestrowane były w oddzielnych transakcjach. Tym sposobem, tuż po zakończeniu nagrania otrzymano zestaw gotowych transakcji z odpowiednio przydzieloną zawartością. Wystarczyło zmienić jedynie ich nazwy z domyślnych (utworzonych automatycznie na podstawie ostatniego trzonu URL) na docelowe, zgodne z własnymi preferencjami.

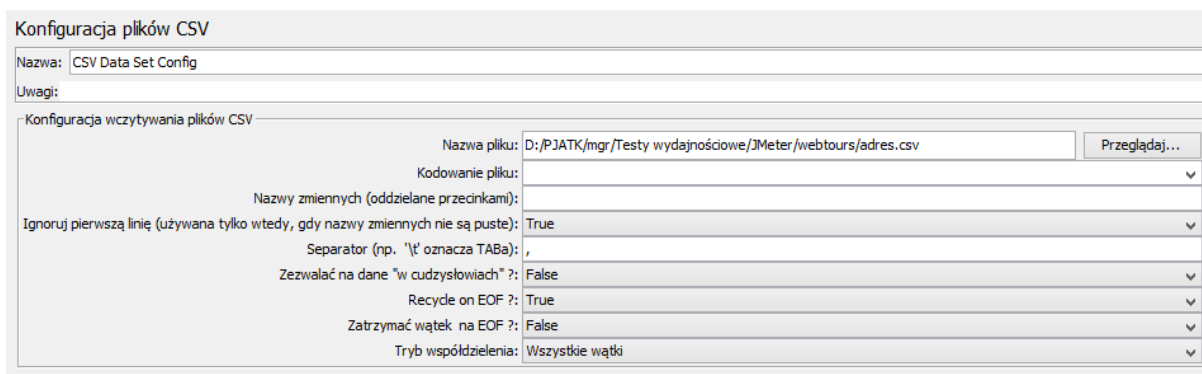
W tym miejscu konieczne było zapoznanie się ze strukturą projektu jaką oferuje JMeter, która szerzej opisana została w podrozdziale 3.1.3.

W odróżnieniu od dwóch poprzednio opisanych narzędzi, JMeter prezentuje z goła odmienne podejście do prezentacji i obsługi żądań, jak i samego skryptu. Nie mamy tutaj do czynienia z wygenerowanym kodem, lecz szeregiem elementów konfiguracyjnych spełniających różnorakie funkcje. Podstawowym z nich jest *HTTP Request*, który jak sama nazwa wskazuje, pełni rolę graficznej reprezentacji żądania http (URL, protokół, metoda, ciało itd.). Rysunek 18 przedstawia interfejs tego elementu, na analogicznym przykładzie, jaki został podany przy poprzednich narzędziach (wyszukanie lotu w aplikacji WebTours).



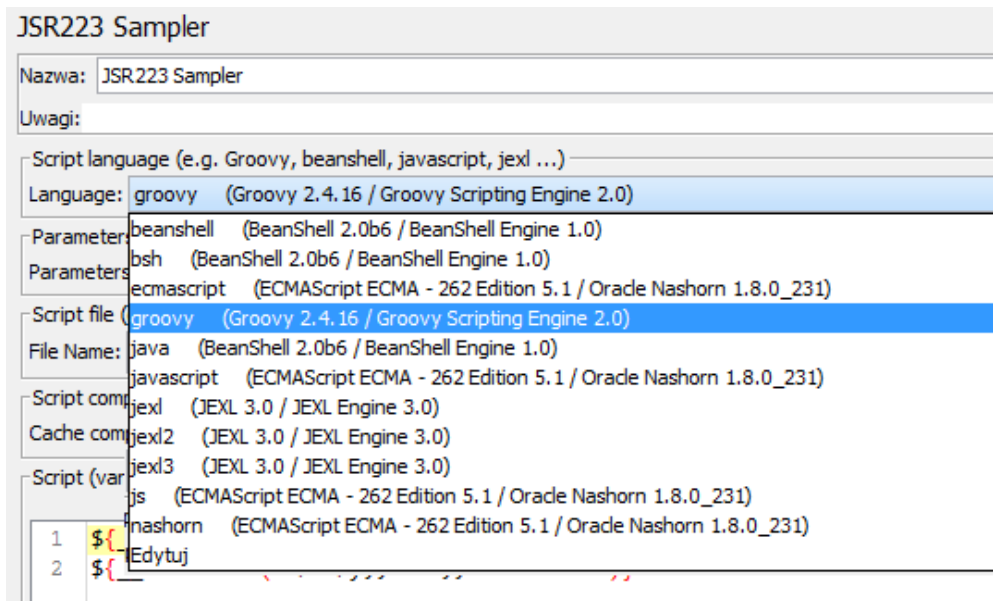
**Rysunek 18. Przykładowy request nagrany w narzędziu JMeter**

Kolekcje zasilające dynamiczne parametry, podobnie jak w Gatlingu, przechowywane były w zewnętrznych plikach csv. Do ich obsługi posłużył tzw. *CSV Data Set Config*, z którego poziomu w prosty i intuicyjny sposób skonfigurowano też obsługę plików z danymi. Przykład tego elementu znajduje się na rysunku 19.



**Rysunek 19. JMeter - CSV Data Set Config zarządzający plikiem z danymi adresowymi**

Tak samo jak to miało miejsce w poprzednich narzędziach, i tym razem obsługa dat została wykonana poprzez dopisanie własnego kodu. Na szczęście, pomimo swojej graficznej natury, JMeter również daje taką możliwość. Służą do tego tzw. Pre Procesory: *BeanShell PreProcessor* oraz *JSR223 Sampler*. W tym przypadku został do tego wykorzystany *JSR223 Sampler*, który pełni funkcję edytora kodu w wybranym języku programowania. Wybór padł na opcję domyślną, jaką jest Groovy, jednak wybór był znacznie szerszy. Pełna lista dostępnych języków programowania znajduje się na rysunku 20.



**Rysunek 20. Dostępne języki programowania w JSR223 Sampler**

Zdefiniowane daty zostały od razu przypisane do parametrów (Kod 10), które od tego momentu mogły zostać użyte w dalszej części skryptu [27].

#### **Kod 10. Obsługa dat w JSR223 Sampler**

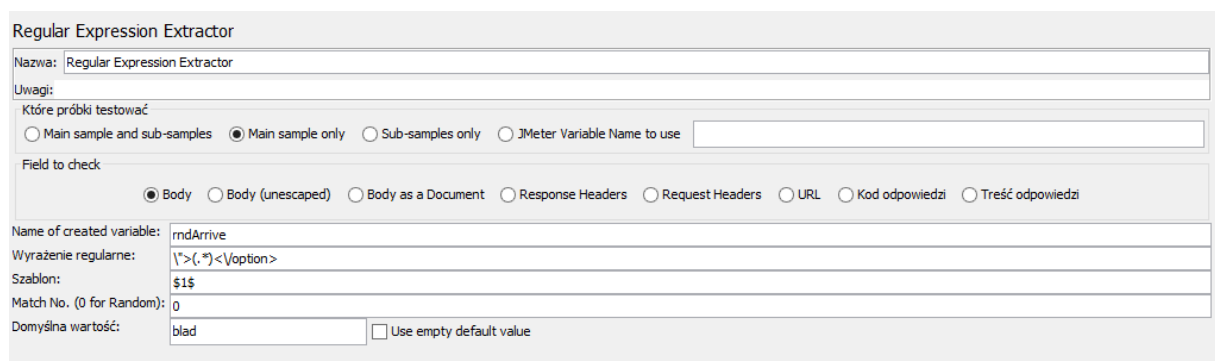
```

${__timeShift(MM/dd/y,,P1D,, departDate)}
${__timeShift(MM/dd/y,,P31D,, returnDate)}

```

Wartości dla pól rozwijalnych wydobyto za pomocą jednego z Post Procesorów – *Regular Expression Extractor*, widocznego na rysunku 21.

Podobnie jak w Gatlingu, obsługa i zredagowanie wyrażenia regularnego leżały po stronie użytkownika. Odpowiednie ustawienie w sekcji *Match No.* pozwoliło na automatyczny wybór losowego elementu z otrzymanej w ten sposób kolekcji.



**Rysunek 21. JMeter - Regular Expression Extractor**

W identyczny sposób został też obsłużony token sesji.

Kolejna grupa elementów, która niezbędna była do stworzenia skryptów to asercje. JMeter oferuje ich wiele rodzajów, jednak w tym przypadku wystarczająca była najbardziej podstawowa z nich – *Response Assertion*. W konfiguracji widocznej na rysunku 17 weryfikuje ona, czy tekst wpisany w polu *Patterns to Test* znajduje się w ciele odpowiedzi na wysłane przez skrypt żądanie.

Znajduje się tam również wiele innych przydatnych funkcji, które dają możliwość zbudowania bardziej złożonych asercji.

The screenshot shows the 'Response Assertion' configuration window in JMeter. It includes fields for 'Nazwa' (Name) set to 'Response Assertion' and 'Uwagi' (Notes). Under 'Które próbki testować' (Which samples to test), 'Main sample only' is selected. The 'Field to Test' section has 'Tekst odpowiedzi' (Response text) selected, along with 'Request Headers', 'Request Data', 'Kod odpowiedzi' (Response code), 'URL Sampled', 'Treść odpowiedzi' (Response content), 'Document (text)', 'Nagłówki odpowiedzi' (Response headers), and 'Ignoruj status' (Ignore status). The 'Pattern Matching Rules' section has 'Substring' selected. The 'Patterns to Test' list contains one entry: '1 Find Flight'. At the bottom, there are buttons for 'Dodaj' (Add), 'Kopiuje ze schowka' (Copy from clipboard), and 'Usuń' (Remove). A 'Custom failure message' field is also present.

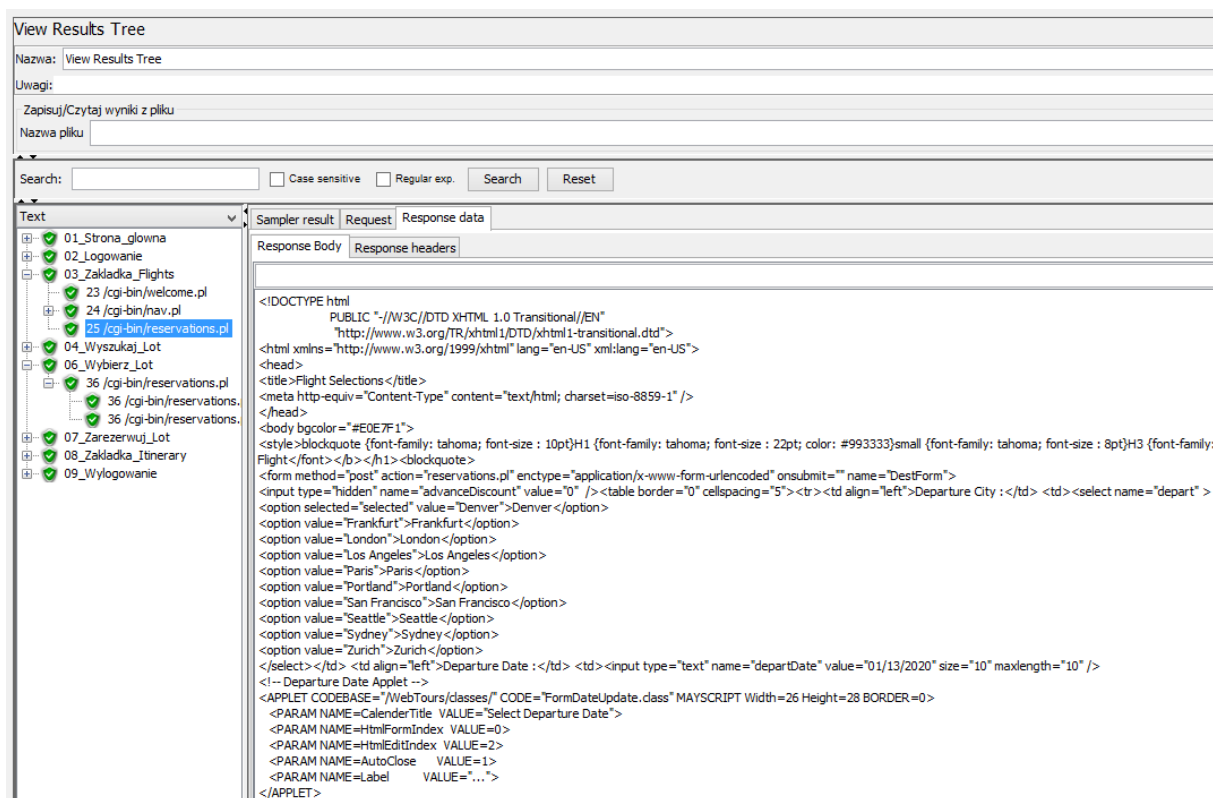
**Rysunek 22. JMeter - Response Assertion**

JMeter nie zarejestrował w żaden sposób think timów podczas nagrywania, zostały one dodane ręcznie. Zakładka *Timer*, grupująca tego typu elementy, oferuje wiele różnych ich odmian. Posłużono się jednak jeszcze inną funkcjonalnością, tzw. *Flow Control Action*, którego przykład znajduje się na rysunku 23. Pozwala on na zdefiniowanie prostego rodzaju pauzy o pożądanej wartości wyrażonej w milisekundach.

The screenshot shows the 'Flow Control Action' configuration window in JMeter. The 'Nazwa' (Name) field is set to 'Think Time'. Under 'Logical Action on Thread', 'Pauza' (Pause) is selected. The 'Czas trwania (milisekund)' (Duration in milliseconds) field is set to '5000'. Other options include 'Start Next Thread Loop', 'Break Current Loop', and 'Go to next iteration of Current Loop'.

**Rysunek 23. Definicja Think timu w narzędziu JMeter**

Podgląd komunikacji zarejestrowanej podczas nagrywania odbywał się przy pomocy jednego ze słuchaczy – *View Results Tree*. Stworzył on plik xml, w którym zapisywane były wszystkie wysłane i odebrane komunikaty. Plik ten został później zinterpretowany przez *View Results Tree* i przedstawiony w czytelnej oraz wygodnej w obsłudze formie, widocznej na rysunku 24.



**Rysunek 24. JMeter - View Results Tree**

Ten sam słuchacz posłużył również do wyświetlania żądań i odpowiedzi wysłanych przez skrypt podczas jego debugowania. Trafiały tam także błędy, jeśli takowe wystąpiły.

Podsumowanie i ocena poszczególnych cech tworzenia skryptu za pomocą narzędzia, jakim jest JMeter znajduje się w tabeli 6.

**Tabela 6. Ocena narzędzia JMeter pod kątem tworzenia skryptu**

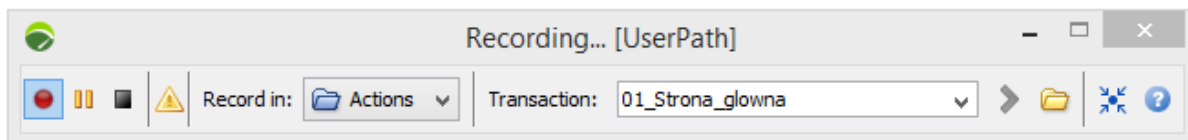
Cecha	Ocena	Argumentacja
<b>Instalacja narzędzia</b>	10	Narzędzie jest gotowe zaraz po pobraniu. Wymagana jest jedynie uprzednia instalacja Javy, jeśli nie została wykonana wcześniej.
<b>Przejrzystość interfejsu</b>	7	W pełni graficzny interfejs, zarządzany za pomocą elementów konfiguracyjnych. Z uwagi na ich dużą liczbę, możliwe jest poczucie zagubienia (zwłaszcza po dodaniu wtyczek), na szczęście są one podzielone na grupy tematyczne, co nieco ułatwia nawigację.
<b>Nagrywanie requestów</b>	6	Dedykowany widżet ułatwiający nagrywanie. Konieczność samodzielnej konfiguracji proxy zarówno w narzędziu, jak i przeglądarce internetowej.
<b>Tworzenie transakcji</b>	8	Automatyczne tworzenie transakcji na

		podstawie aktywności użytkownika. Niestety ich nazwy są dość nieintuicyjne i wysoce wskazana jest definicja własnych etykiet.
<b>Obsługa zewnętrznych danych testowych</b>	9	Możliwość podpięcia zewnętrznych plików z danymi (m.in. csv). Mechanizm konfiguracji parametrów jest dość prosty, ale zarazem w pełni funkcjonalny.
<b>Możliwość dodania własnego kodu</b>	8	Graficzna forma narzędzia wymusza dodawanie kodu w odpowiednich kontenerach, co nie jest zbyt wygodne w obsłudze (szczególnie odwołania do zmiennych globalnych). Dużym plusem jest za to mnogość wyboru języków programowania.
<b>Korelacja dynamicznych wartości</b>	5	Brak automatycznej korelacji, konieczne samodzielne redagowanie wyrażeń regularnych na podstawie nagranych odpowiedzi.
<b>Asercje</b>	8	Proste, ale zarazem dające duże możliwości konfigurowania. Niestety brak jakiegokolwiek mechanizmu automatyzującego, wymagane jest dodanie ich manualnie.
<b>Think time</b>	6	Brak rejestracji think time podczas nagrywania. Konieczne dodanie i zdefiniowanie odpowiedniego elementu. Plusem jest ich kilka różnych rodzajów do wyboru.
<b>Debugowanie</b>	6	Dosyć przejrzysta forma prezentacji. Brak mechanizmów porównania nagranych ruchu z odtworzonym.
<b>Tempo tworzenia skryptu</b>	7	Obsługa skryptu za pomocą elementów graficznych, wymaga wielokrotnego ich dodania oraz konfigurowania. Większość czynności wymaga aktywności ze strony użytkownika, mała liczba automatycznych usprawnień.

#### 4.1.4 NeoLoad

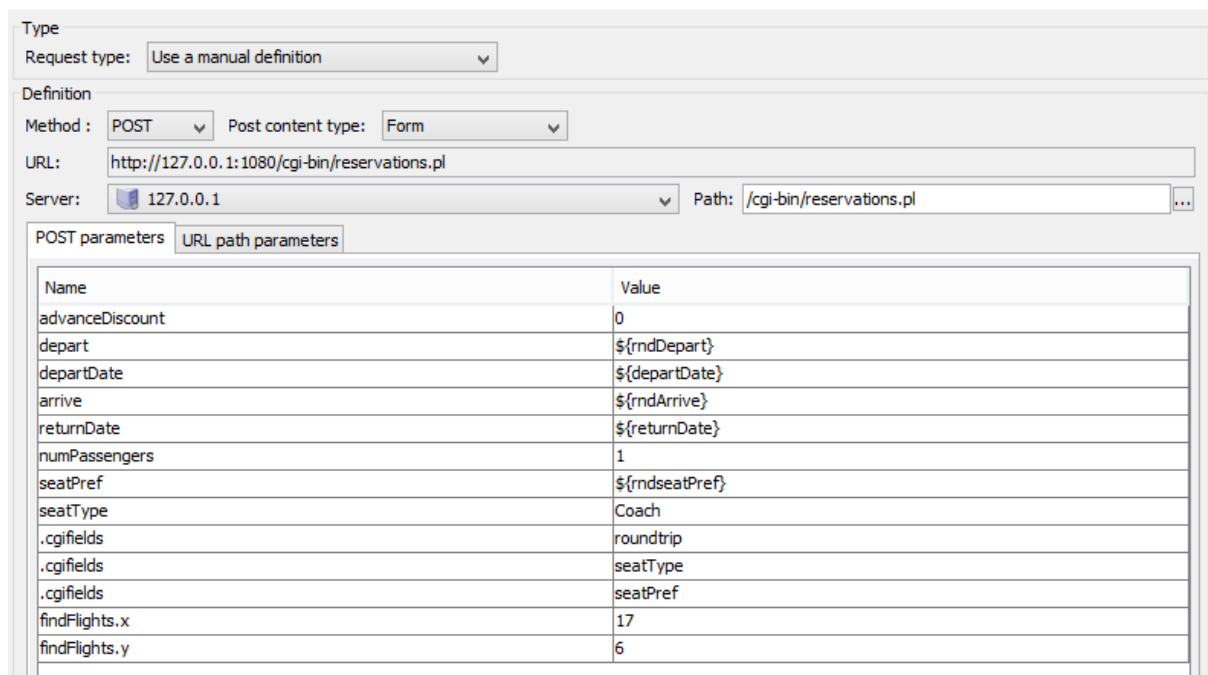
Instalacja oprogramowania przebiegła analogicznie do LoadRunnera. Wystarczyło pobranie pliku instalacyjnego i podążanie za kolejnymi krokami instalatora.

Kolejnym podobieństwem był widżet dostępny podczas nagrywania skryptu, który również umożliwił sprawne utworzenie transakcji (Rysunek 25). Warto zauważyć, że można było przystąpić do tej operacji bezpośrednio po utworzeniu projektu, nie była konieczna żadna dodatkowa konfiguracja.



**Rysunek 25. Widżet zarządzający nagrywaniem ruchu sieciowego w narzędziu NeoLoad**

NeoLoad, podobnie jak JMeter większość aktywności obsługuje poprzez interfejs graficzny. Koncepcja żądania http również prezentuje się bardzo porównywalnie, co można zaobserwować na rysunku 26.



**Rysunek 26. Przykładowy request nagrany w narzędziu NeoLoad**

Obsługa plików csv, które dostarczają wartości dla dynamicznych parametrów, wykonana została przy pomocy mechanizmu *Variables*. Spośród wielu różnego rodzaju typów zmiennych jakie są tam oferowane została wybrana opcja *File*, gdzie wskazano ścieżkę do stworzonego wcześniej pliku z danymi adresowymi. Jak widać na rysunku 27, została też skonfigurowana polityka dystrybucji wartości zmiennych oraz sposób w jaki są one aktualizowane (kolejny rekord losowany na nowo na każdej stronie).

Definition

Variable type: File

Name: Adres

Description:

---

Values

File name: variables/adres.csv ... Import dataset...

Column separator: , Starting from line: 1 Only a fraction of the lines are displayed

Use first line in file as column headings?

pMiasto	pUlica
Warszawa	Kopernika
Radom	1 maja
Krakow	3 maja
Szczecin	Cmentarna
lodz	Dabrowskiego
Wroclaw	Dworcowa
Zielona Gora	Gajowa
Gdansk	Generala W...
Poznan	Grodzka
swinoujscie	Grunwaldzka

Open Refresh Previous Next

---

Value change policy

Choose when the value must change:

On each use  On each request

On each page  On each iteration

For each Virtual User instance

Variable values distribution policy

Choose how the values are distributed:

Scope: Global

Order: Random

When Out Of Values: Cycle values

OK Apply Cancel

**Rysunek 27. NeoLoad - Definicja zmiennej zarządzającej plikiem z danymi adresowymi**

Obsługa dat została zrealizowana za pomocą wbudowanego dedykowanego rozwiązania zaprezentowanego na rysunku 28. Pozwoliło ono na definicję daty w pożądanym formacie, a także jej inkrementację o wybrany interwał bazując na dniu aktualnym. Było to wystarczające na potrzeby tego skryptu i nie było konieczności dopisywania własnego kodu. Gdyby jednak taka potrzeba zaistniała, NeoLoad również udostępnia taką możliwość poprzez wstawienie odpowiedniego elementu (tzw. akcji) obsługującego język JavaScript.

Definition

Variable type: Current Date

Name: returnDate

Description:

---

Parameters

Date pattern: MM/dd/yyyy

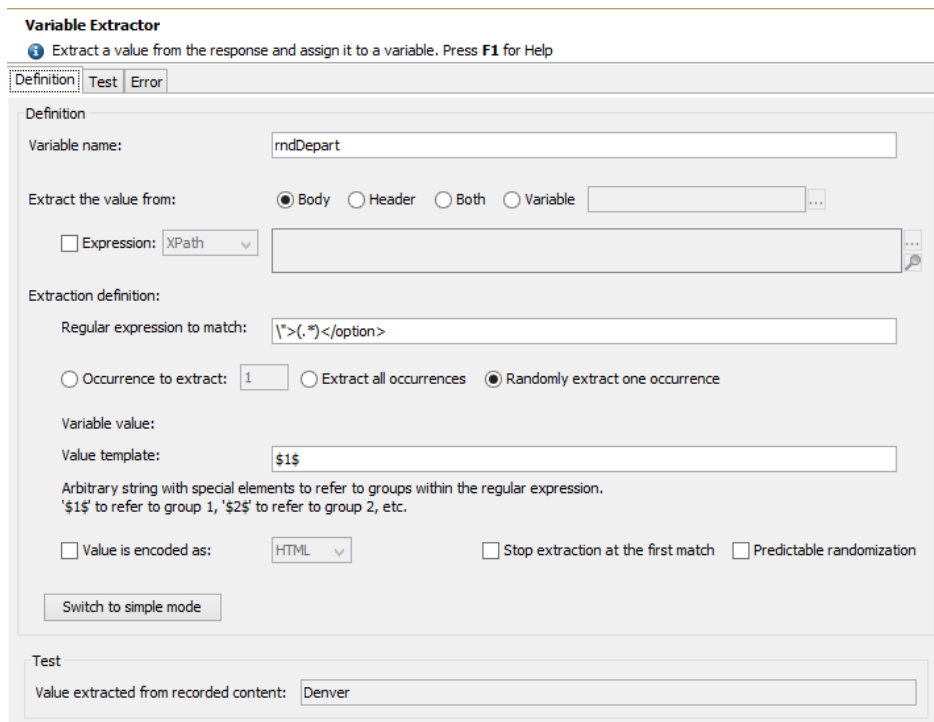
Increment value: 30 Day

**Rysunek 28. Obsługa dat w narzędziu NeoLoad.**

Kolejną niezwykle przydatną funkcjonalnością okazał się *Variable Extractor* dostępny z poziomu żądania (rysunek 29). Wprowadzie wyrażenie regularne ponownie musiało zostać napisane manualnie, jednak wbudowane usprawnienia znacznie ułatwiły to zadanie. W zakładce *Test* znajduje się cała treść odpowiedzi, z kolei po wpisaniu wyrażenia regularnego w odpowiednie pole na zakładce

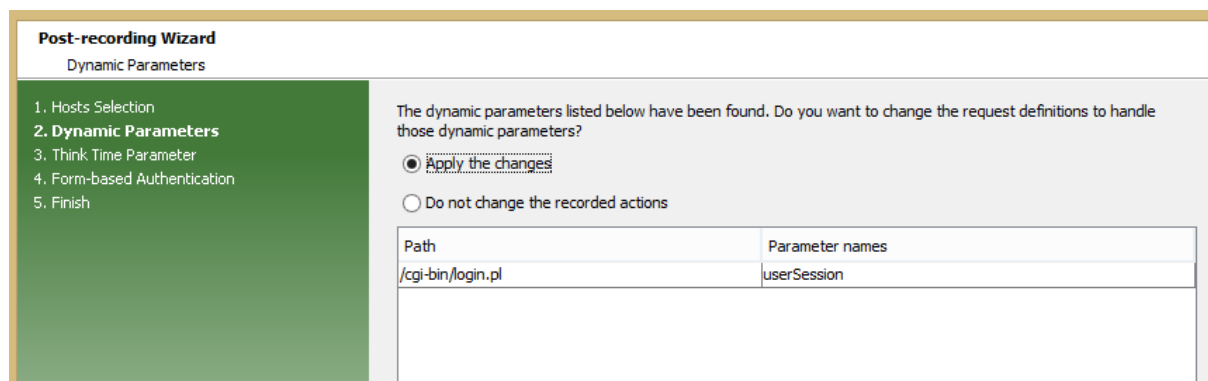


głównej wyświetlana jest jego wynik. Z tej pozycji została również określona randomizacja kolekcji wynikowej. W ten sposób zostały obsłużone dynamiczne wartości dla pól rozwijalnych.



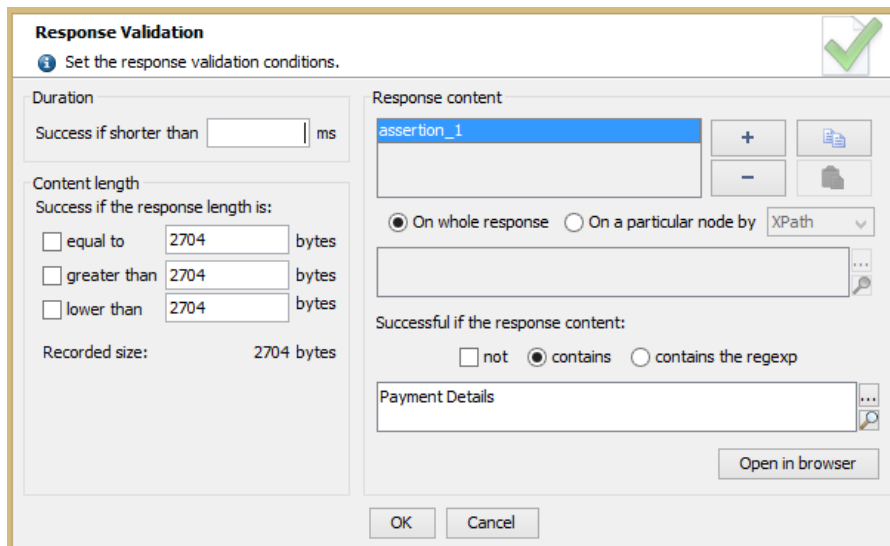
**Rysunek 29. NeoLoad - Variable Extractor**

Korelacja identyfikatora sesji została natomiast wykonana w znacznie prostszy sposób. Użyto w tym celu kolejnego mechanizmu oferowanego przez narzędzie - *Search for Dynamic Parameters* (widoczny na rysunku 30). Działa on w identyczny sposób, co *Design Studio* w LoadRunnerze – przeszukuje całą nagraną komunikację w poszukiwaniu potencjalnych wartości wymagających korelacji. W przypadku ich odnalezienia, automatycznie tworzony jest obiekt *Variable Extractor*, tym razem już z gotowym wyrażeniem regularnym. Z kolei odnaleziona wartość zostaje podmieniona na parametr, we wszystkich żądaniach, w których się pojawia.



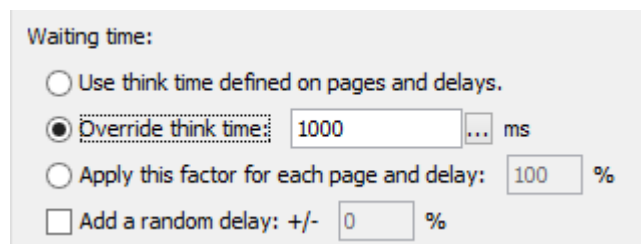
**Rysunek 30. Mechanizm automatycznej korelacji w narzędziu NeoLoad**

Asercje są kolejnym przykładem opcji dostępnej domyślnie z pozycji każdego żądania. Po wybraniu przycisku *Validation* wyświetlone zostało okno pokazane na rysunku 31. Oprócz weryfikacji treści odpowiedzi, możliwe jest również zdefiniowanie asercji na inne aspekty, m.in. czas trwania, objętość itd.



**Rysunek 31. Obsługa asercji w narzędziu NeoLoad**

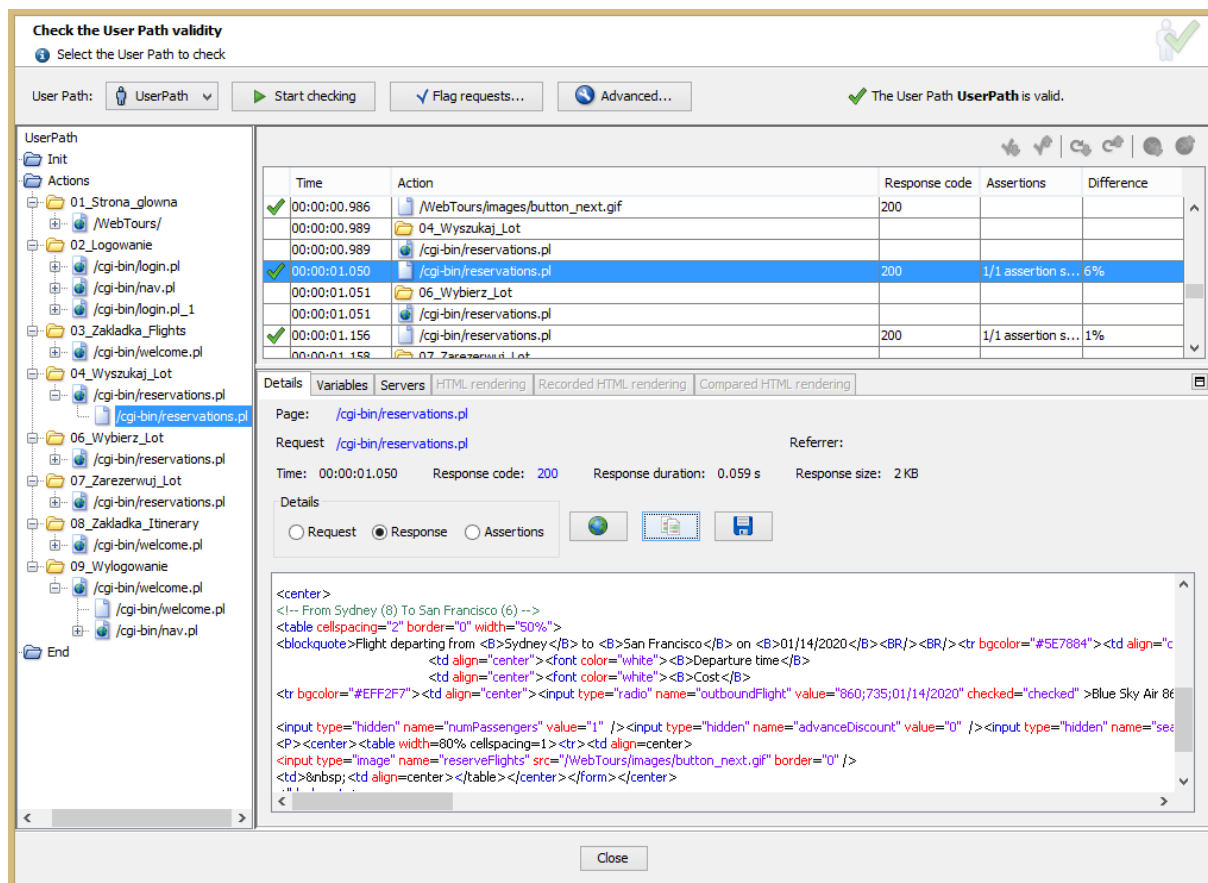
Domyślna wartość think timów, podobnie jak w Gatlingu, była taka, jaka została zarejestrowana podczas nagrywania skryptów. Na szczęście nie było konieczności jej edycji dla każdego z requestów z osobna, ponieważ NeoLoad posiada możliwość sterowania nimi z nadrzędnego poziomu skryptu. Po wybraniu widocznej na rysunku 32 opcji *Override Think time*, wszystkie pauzy przyjęły automatycznie określony w tym miejscu czas trwania.



**Rysunek 32. Zarządzanie Think timami w narzędziu NeoLoad**

Jeżeli zaś chodzi o debugowanie, odbyło się ono bardzo prosto i szybko dzięki szeregowi uprawnień, które posiada NeoLoad. Narzędzie to oferuje odrębny moduł poświęcony tylko i wyłącznie debugowaniu, tzw. *Check User Path* (rysunek 33). Jest on dość podobny do mechanizmu *Snapshot* z LoadRunnera, niemniej jednak bardziej przejrzysty i nieco bardziej rozbudowany.

Oprócz podglądu nagranej i odtworzonej komunikacji, możliwe jest również jej automatyczne porównanie – wszystkie zmiany, dodane bądź usunięte fragmenty zostają podświetlone odpowiednimi kolorami. Na takiej samej zasadzie możemy wyświetlić i porównać każdą stronę załadowaną podczas nagrywania i odtwarzania (w postaci HTML rendering).



Rysunek 33. NeoLoad- mechanizm *Check User Path*

Tabela 7 zawiera podsumowanie i ocenę poszczególnych cech tworzenia skryptu przez NeoLoada, wraz z krótkim uzasadnieniem.

Tabela 7. Ocena narzędzia NeoLoad pod kątem tworzenia skryptu

Cecha	Ocena	Argumentacja
<b>Instalacja narzędzia</b>	10	Prosta, standardowa instalacja z pliku instalacyjnego.
<b>Przejrzystość interfejsu</b>	9	Rozbudowany interfejs graficzny, który daje duże możliwości, choć odnalezienie części potrzebnych funkcji może sprawiać problemy.
<b>Nagrywanie requestów</b>	10	Automatycznie uruchamiana instancja przeglądarki internetowej. Dedykowany widżet wspierający kluczowe funkcje nagrywania.
<b>Tworzenie transakcji</b>	10	Możliwość tworzenia transakcji podczas nagrywania. Wystarczy wpisać na widżecie odpowiednią nazwę i automatycznie tworzony jest kontener transakcji.
<b>Obsługa zewnętrznych danych</b>	8	Narzędzie <i>Variables</i> oferujące dużą liczbę

<b>testowych</b>		różnego rodzaju parametrów. Możliwość podłączenia pliku csv bądź stworzenia własnej kolekcji. Pełna konfiguracja obsługi parametrów. Gotowe przykładowe kolekcje (np. Imiona, Nazwiska, Kraje, Ulice itd.)
<b>Możliwość dodania własnego kodu</b>	6	Obsługa własnego kodu realizowana w obrębie dedykowanego kontenera. Dostępny jedynie język JavaScript, który jest nieco ubogi, niemniej jednak wystarczający do tego zadania.
<b>Korelacja dynamicznych wartości</b>	10	Mechanizm automatycznej korelacji analizujący skrypt pod kątem dynamicznych wartości. Korelacja ręczna wspierana przez szereg ułatwień dostępnych w sekcji <i>Variable Extractor</i> .
<b>Asercje</b>	8	Proste w obsłudze, jednak definiowane manualnie przez użytkownika. Dużo przydatnych opcji konfigurowania.
<b>Think time</b>	7	Domyślnie przyjmuje wartość z nagrania, jednak w bardzo prosty sposób można go dowolnie nadpisywać i konfigurować z nadrzędnego poziomu skryptu.
<b>Debugowanie</b>	10	Odrębny moduł służący tylko i wyłącznie w tym celu. Możliwość podglądu komunikacji na wielu poziomach, dostępne liczne mechanizmy porównywania i analizy.
<b>Tempo tworzenia skryptu</b>	10	Wiele czynności wykonanych zostało automatycznie bądź przy dużym wsparciu ze strony narzędzia. Imponująca ilość dodatkowych narzędzi i usprawnień, zarówno na etapie tworzenia skryptu, jak i jego debugowania.

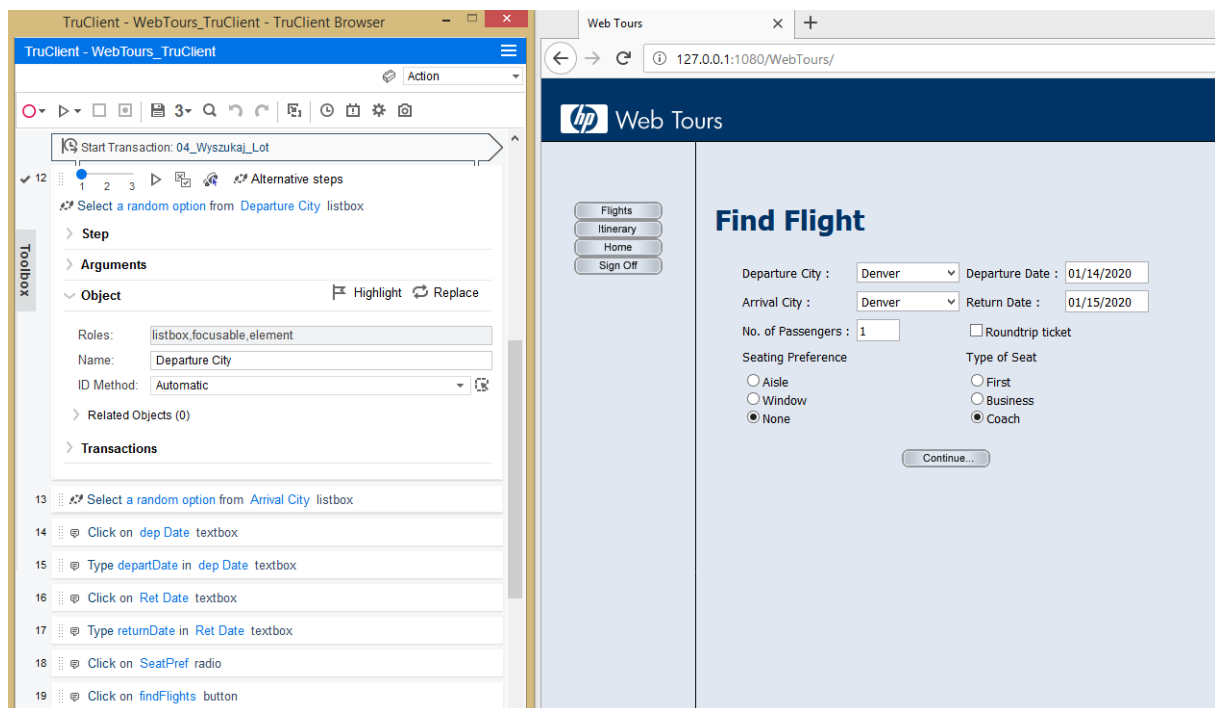
#### 4.1.5 TruClient

Jak już wspomniano w podrozdziale 3.1.5, TruClient jest jednym z protokołów LoadRunniera instalowanym automatycznie razem z całym narzędziem.

W celu sporządzenia skryptu, w pierwszej kolejności utworzono zatem projekt bazujący na tym właśnie protokole.

W odróżnieniu od opisywanych w poprzednich podrozdziałach narzędzi, nie bazuje na komunikatach wysyłanych między klientem i serwerem. Działa on na fizycznej instancji przeglądarki i porusza się w obrębie graficznego interfejsu aplikacji. Przed rozpoczęciem nagrywania należało wybrać pożądaną przeglądarkę internetową oraz jeden z trzech poziomów nagrywania. Poziom 1 oznacza najmniej szczegółowy skrypt, poziom 3 – najbardziej wnikliwy (bardzo wrażliwy na każdą aktywność użytkownika, jak np. najazd kursorem myszy na obiekt).

Samo nagrywanie skryptu wyglądało bardzo podobnie do rejestrowania komunikacji w protokole Web, jednak zamiast żądań i odpowiedzi zapisywane były elementy HTML używane na kolejnych podstronach aplikacji i akcje na nich wykonywane (kliknięcie przycisku, wpisanie tekstu w pole tekstowe itd.). Graficzna reprezentacja skryptu widoczna jest na rysunku 34.



**Rysunek 34. Przykładowy krok nagrany w narzędziu TruClient**

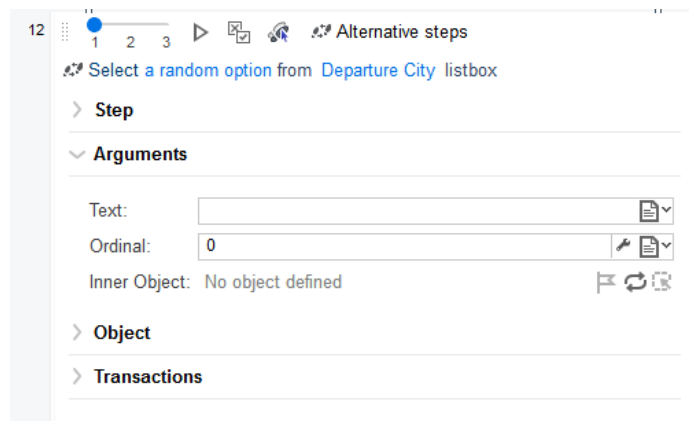
Po zarejestrowaniu wszystkich aktywności konieczne było ich przejrzanie i usunięcie tych niechcianych (np. *Mouse Over events*).

Tym razem tworzenie transakcji nie było możliwe podczas nagrywania. Konieczne było dodanie ich już po jego zakończeniu. Odbyło się to poprzez oznaczenie aktywności początkowej i końcowej. Efekt jest widoczny na rysunku 35.



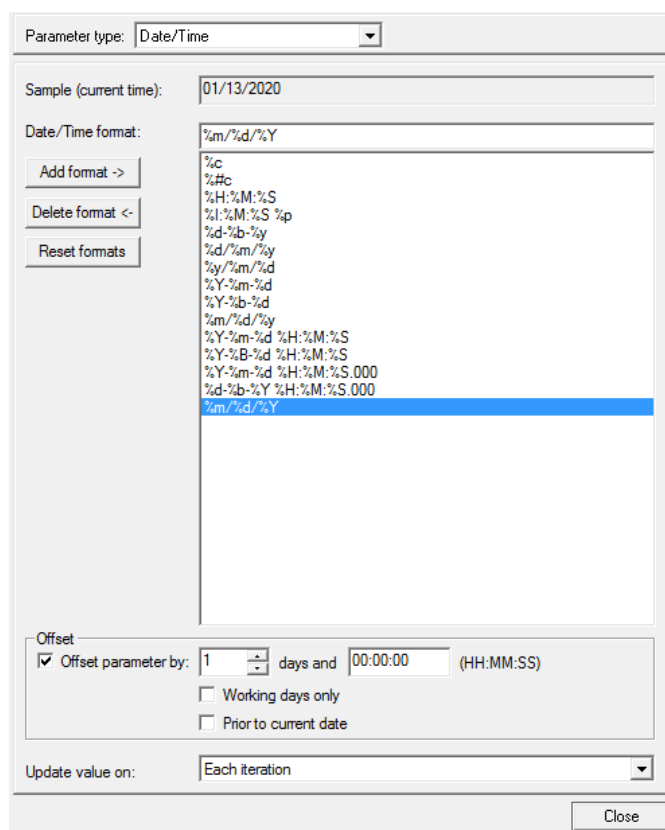
**Rysunek 35. Oznaczenie transakcji w TruClient**

Jako że skrypt poruszał się po GUI, nie było potrzeby korelacji identyfikatora sesji ani ekstrakcji żadnych dynamicznych wartości. Wystarczyło określenie wyboru losowej opcji z pola rozwijalnego, tak jak pokazuje to rysunek 36.



**Rysunek 36. Obsługa pól rozwijalnych w TruClencie**

TruClient umożliwia wstawienie własnego kodu (w języku C lub JavaScript) w odpowiednio do tego stworzonym kontenerze, na podobnej zasadzie jak miało to miejsce w JMeterze i NeoLoadzie. Tym razem jednak daty zostały obsługiwane w inny sposób. Jako, że TruClient jest częścią LoadRunniera, ma on również dostęp do modułu parametrów - takich samych jak protokół Web. Ten z kolei, analogicznie do NeoLoada, posiada możliwość utworzenia i odpowiedniego skonfigurowania parametru typu *Date/Time* (format + inkrementacja daty). Mechanizm ten został przedstawiony na rysunku 37.

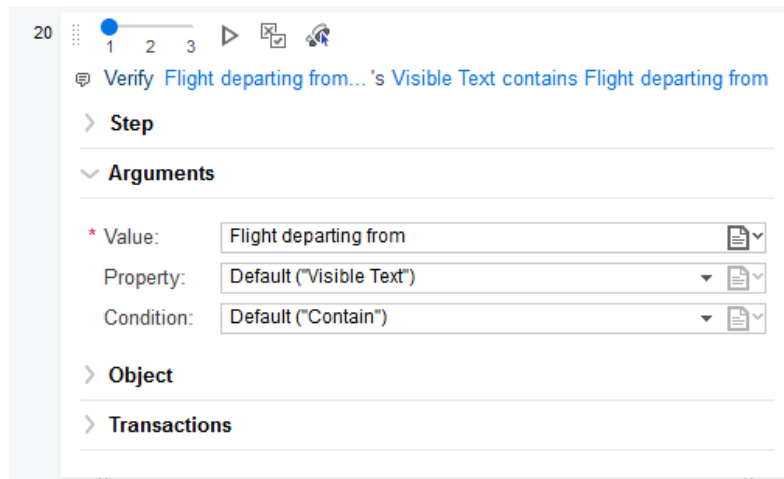


**Rysunek 37. LoadRunner, parametr typu *Date/Time***

Teoretycznie skrypt w obecnej formie powinien już funkcjonować prawidłowo, tak się jednak nie stało. Dodatkową aktywnością niezbędną do jego poprawnego działania, okazała się weryfikacja i ręczna redefinicja deskryptorów, według których odnajdowane były elementy na stronie.

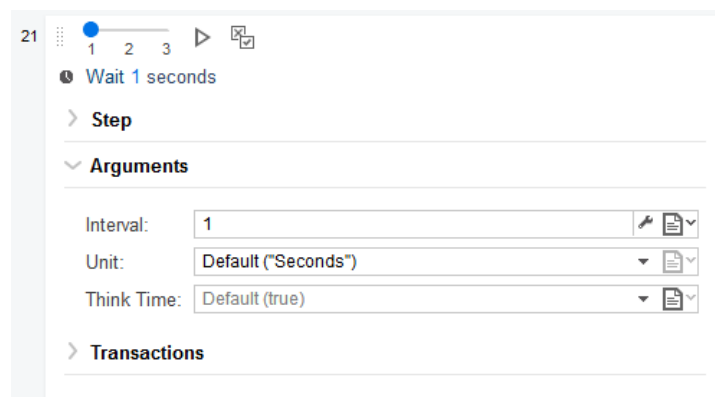
Automatyczna definicja części z nich opierała się na parametrach, których wartość zmieniała się z każdym uruchomieniem, konieczne było zatem podanie bardziej unikalnych atrybutów.

Oczywiście w tego typu skrypcie niezbędne było również dodanie asercji. Wyglądały one jednak zupełnie inaczej niż w przypadku pozostałych narzędzi. W tym wypadku weryfikowana była wartość konkretnego atrybutu obiektu HTML. Posłużył do tego mechanizm *Verify* dostępny w zasobniku (*Toolbox*). Jego obsługa zilustrowana została na rysunku 38.



**Rysunek 38. Obsługa asercji w narzędziu TruClient**

Wspomniany zasobnik oferuje również wiele innych przydatnych funkcjonalności. Kolejną z nich, która została użyta podczas tworzenia skryptu jest *Wait* – odpowiednik think timu. Jego konfiguracja okazała się banalnie prosta – wystarczyło podać wartość i wybrać jednostkę czasu (rysunek 39).



**Rysunek 39. Zarządzanie Think timem w narzędziu TruClient**

Debugowanie sprowadzało się do uruchamiania skryptu i wizualnej weryfikacji, czy wykonywane przez niego kroki są zgodne ze scenariuszem. W przypadku, gdy definicja obiektu była niepoprawna, TruClient wyświetlał stosowny komunikat błędu i przerywał odtwarzanie kolejnych kroków.

W poszukiwaniu deskryptorów posilkowano się wbudowanym w przeglądarkę internetową narzędziem *Zbadaj element*, które wyświetlało m.in. jego listę atrybutów i ich wartości, tak jak na rysunku 40. Następnie aktualizowano obiekt i uruchamiano skrypt ponownie.

```

<table cellpadding="2" border="0" width="50%">
  <tbody>
    <tr bgcolor="#5E7884"></tr>
    <tr bgcolor="#EFF2F7">
      <td align="center">
        <input name="outboundFlight" value="010;386;01/14/2020" checked="checked" type="radio">
          Blue Sky Air 010
        </td>
        <td align="center">8am</td>
        <td align="center">$ 386</td>
      </tr>
    <tr bgcolor="#EFF2F7"></tr>
    <tr bgcolor="#EFF2F7"></tr>
    <tr bgcolor="#EFF2F7"></tr>
  </tbody>
</table>
<input name="numPassengers" value="1" type="hidden">
<input name="advanceDiscount" value="0" type="hidden">
<input name="seatType" value="Coach" type="hidden">
<input name="seatPref" value="None" type="hidden">

```

**Rysunek 40. Podgląd atrybutów obiektu HTML**

Podsumowanie i ocena poszczególnych cech tworzenia skryptu za pomocą TruClienta (wraz z krótkim uzasadnieniem) znajdują się w tabeli 8.

**Tabela 8. Ocena narzędzia TruClient pod kątem tworzenia skryptu**

Cecha	Ocena	Argumentacja
<b>Instalacja narzędzia</b>	10	Analogicznie jak w przypadku LoadRunner Web.
<b>Przejrzystość interfejsu</b>	9	Prosty minimalistyczny interfejs. Wszystkie potrzebne funkcje są dostępne z poziomu danej akcji, albo znajdują się w zasobniku.
<b>Nagrywanie akcji</b>	7	Nagrywanie działa dość sprawnie, jednak nagrywa dużo nadmiarowych akcji, które trzeba później usunąć (nawet na 1 poziomie szczegółowości).
<b>Tworzenie transakcji</b>	5	Transakcje są możliwe do dodania dopiero po nagraniu skryptu. Niezbyt wygodny mechanizm oznaczania początkowej i końcowej akcji.
<b>Obsługa zewnętrznych danych testowych</b>	8	Mechanizm <i>Parameters List</i> , identycznie jak w przypadku LoadRunner Web.
<b>Możliwość dodania własnego kodu</b>	7	Narzędzie udostępnia specjalne obiekty dedykowane do obsługi własnego kodu (do wyboru C i JavaScript). Istnieje również możliwość wpisania kodu JavaScript bezpośrednio w akcji (np. wartość jaka ma zostać wpisana w pole tekstowe).
<b>Korelacja dynamicznych wartości</b>	2	W bardzo wielu przypadkach, obiekt domyślnie definiowany jest na podstawie



		dynamicznych deskrytorach (np. tekst, który zmienia się przy każdej iteracji). Konieczne jest jego manualna redefinicja.
<b>Asercje</b>	7	Funkcjonalny mechanizm <i>Verify</i> umożliwiający dość szeroką konfigurację. Obsługuje on jednak tylko wewnętrzny tekst obiektu lub wewnętrzny HTML.
<b>Think time</b>	3	Dodawany ręcznie, określany poprzez sztywną wartość liczbową wyrażoną w sekundach lub milisekundach. Nie posiada dodatkowych opcji konfiguracyjnych.
<b>Debugowanie</b>	4	Jedyny mechanizm, który można uznać za wspierający to wyświetlenie błędu, gdy skrypt zakończy działanie niepowodzeniem. Podgląd parametrów obiektów realizowany głównie poprzez funkcje wbudowane w przeglądarkę.
<b>Tempo tworzenia skryptu</b>	6	Tego typu skrypt teoretycznie powinien działać od razu po nagraniu. Niestety nadmiarowe akcje i chybiony sposób automatycznego doboru deskrytorów znacznie opóźnia ten proces.

## 4.2 Skrypty dla aplikacji ASP.NET Web Forms

Instalacja oraz nagranie skryptu dla aplikacji ASP.NET Web Forms przez każde z narzędzi wyglądało analogicznie do Web Tours Sample Application. Jednak parametryzacja oraz finalna edycja skryptu wykazały znaczące różnice.

### 4.2.1 LoadRunner

Pierwszą dodatkową czynnością, którą należało wykonać tworząc skrypt dla aplikacji ASP.NET Web Forms była obsługa wspomnianych w podrozdziale 3.2.2. parametrów określających *stan widoku*.

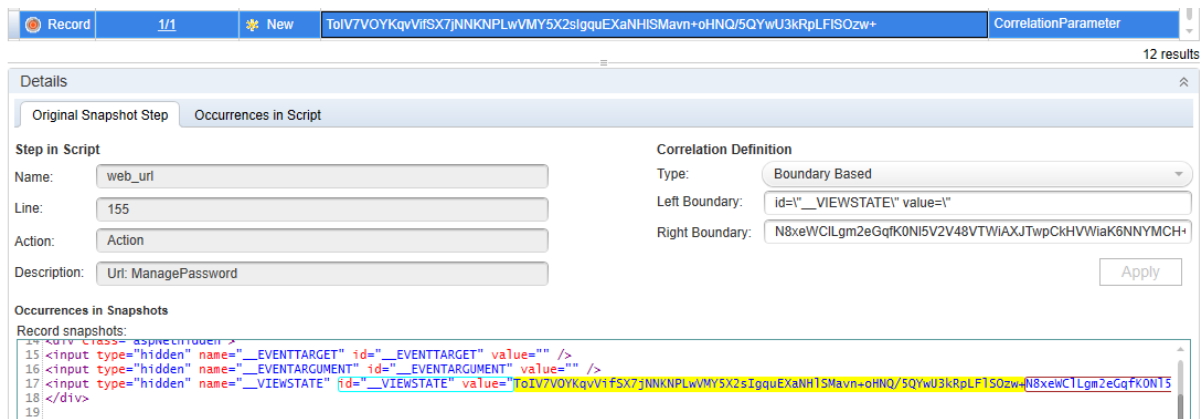
Maksymalna domyślna długość parametru dla narzędzia LoadRunner to 4096 znaków, co okazało się niewystarczające. Skutkiem tego było ignorowanie *stanów widoku* podczas automatycznego poszukiwania dynamicznych wartości, a także zwracanie komunikatu błędu przy próbie ich manualnej ekstrakcji. Aby rozwiązać ten problem konieczne było nadpisanie tego ograniczenia, tak jak zostało to zaprezentowane w kodzie 11.

#### Kod 11. Zwiększenie maksymalnej długości parametru w LoadRunnerze

```
web_set_max_html_param_len("24048");
```

Kolejną opcją dająca podobny efekt byłaby zmiana tej wartości w globalnych parametrach nagrywania, jednak wtedy miałyby to wpływ także na inne skrypty, a nie zawsze jest to pożądane.

Niestety, nawet po zwiększeniu limitu wielkości parametru, *Design Studio* nie poradził sobie z automatyczną korelacją. Z niejasnych przyczyn (prawdopodobnie niedoskonałość algorytmu przeszukiwania dynamicznych parametrów) odnajdywana była jedynie część *stanu widoku* (rysunek 41), zatem stworzona w ten sposób reguła nie przynosiła odpowiedniego skutku.



Rysunek 41. Próba obsługi stanu widoku przez narzędzie Design Studio

Finalnie zostały one obsługane poprzez manualną definicję, tak jak to miało miejsce w przypadku pól rozwijalnych w podrozdziale 4.1.1.

Kolejną funkcjonalnością, którą koniecznie trzeba było obsłużyć, aby odzwierciedlić faktyczną mechanikę aplikacji, było wysyłanie części requestów w sposób asynchroniczny. Na szczęście LoadRunner znajduje tego typu zależności automatycznie i tworzy tzw. *Concurrent group*, której przykład można zaobserwować w kodzie 12. Wszystkie żądania znajdują się wewnątrz grupy, po uruchomieniu skryptu zostają wysłane jednocześnie. Oczywiście, jeśli zaistniałaby taka potrzeba, istnieje również możliwość manualnego dodania *Concurrent group* w kodzie skryptu.

#### Kod 12. Obsługa requestów asynchronicznych przez mechanizm *Concurrent group*

```
web_concurrent_start(NULL);

web_url("modernizr-2.8.3.js",
  "URL={pURL}/Scripts/modernizr-2.8.3.js",
  "Resource=1",
  "RecContentType=application/javascript",
  "Referer={pURL}/",
  "Snapshot=t2.inf",
  LAST);

web_url("bootstrap.css",
  "URL={pURL}/Content/bootstrap.css",
  "Resource=1",
  "RecContentType=text/css",
  "Referer={pURL}/",
  "Snapshot=t3.inf",
  LAST);

web_url("Site.css",
  "URL={pURL}/Content/Site.css",
  "Resource=1",
  "RecContentType=text/css",
  "Referer={pURL}/",
  "Snapshot=t4.inf",
  LAST);

web_url("favicon.ico",
```

```

        "URL={pURL}/favicon.ico",
        "Resource=1",
        "RecContentType=image/x-icon",
        "Referer=",
        "Snapshot=t9.inf",
        LAST);

web_concurrent_end(NULL);

```

Następną kwestią, wartą poruszenia a propos tworzenia skryptu dla aplikacji ASP.NET Web Forms, jest dynamiczne generowanie losowego tekstu. W tym przypadku, na potrzeby rejestracji, konieczne było wpisanie unikalnego adresu e-mail – losowego tekstu w określonym formacie (np. przykład@przykład.com). Oczywiście można by to rozwiązać analogicznie do kolekcji miast i ulic w aplikacji WebTours, czyli poprzez stworzenie kolekcji zawierającej odpowiednią ilość przygotowanych wcześniej adresów e-mail. Byłoby to jednak mało efektywne, ponieważ utworzone w ten sposób dane zużywały by się po każdym uruchomieniu testu – nie można zarejestrować kilku kont używając tego samego loginu. Zostało to zatem rozwiązane przez napisanie odpowiedniej funkcji w języku C (na dość podobnej zasadzie co obsługa daty, jednak na dużo wyższym poziomie złożoności). Funkcja została stworzona na podstawie [28].

### Kod 13. Generowanie losowego adresu e-mail przy wykorzystaniu języka C

```

char bufor[32] = "";
char wynik[32] = "";

random_email()
{
    int r,i;
    char c;
    srand((unsigned int)time(0));

    for (i = 0; i < 20; i++)
    {
        r = rand() % 25 + 65;
        c = (char)r;
        bufor[i] = c;
    }
    bufor[10] = '@';
    strcpy(wynik, bufor);
    strcat(wynik, ".COM");
    lr_save_string(wynik, "pUsername2");
    return 0;
}

```

Jak pokazuje kod 13, zostało to pomyślnie zrealizowane, jednak specyfika języka programowania sprawiła, że funkcja nie byłaby trywialna do napisania przez osobę nieposiadającą odpowiedniego doświadczenia.

Podobne zadanie pojawiło się w dalszej części scenariusza, przy okazji wprowadzania danych nowych studentów w zakładce *Studenti*. Jednym z atrybutów, który każdy z nich musiał posiadać był *Student Code* mający postać losowego numeru. W tym celu skorzystano z parametru typu *Random Number* dostępnego w opcjach *Parameters List*. Pozwala on na określenie dolnego oraz górnego przedziału, spośród którego losowana jest liczba całkowita, a także na wybranie formatu w jakim jest ona prezentowana (rysunek 42).

Parameter type: Random Number

Random range: Min: 10000  
Max: 99999

Sample value: 10000, 99999

Number format: %05lu

- %01lu
- %02lu
- %03lu
- %04lu
- %05lu**
- %06lu
- %07lu
- %08lu
- %lu

**Rysunek 42. LoadRunner, parametr typu *Random Number***

Podobnie jak miało to miejsce w przypadku aplikacji Web Tours Sample Application, opisane w tym podrozdziale czynności zostały poddane ocenie w formie tabelarycznej, wraz z krótkim uzasadnieniem i przedstawione w tabeli 9.

**Tabela 9. Ocena narzędzia LoadRunner pod kątem dodatkowych aktywności podczas tworzenia skryptu do aplikacji ASP.NET Web Forms**

Cecha	Ocena	Argumentacja
<b>Obsługa stanów widoku</b>	3	Błędna automatyczna korelacja w <i>Design Studio</i> , co nie tylko nie pomaga, ale też wprowadza użytkownika w błąd. Konieczność manualnej ekstrakcji wartości stanów widoku.
<b>Obsługa wywołań asynchronicznych</b>	10	Sprawnie działający mechanizm <i>Concurrent grup</i> . Automatyczne grupowanie requestów podczas generowania kodu nagranych skryptu.
<b>Generowanie losowego tekstu</b>	7	Zrealizowane pomyślnie przy pomocy specjalnie stworzonej w tym celu funkcji. Jednak specyfika języka programowania uczyniła zadanie bardziej zawiłym niż powinno to być dla potencjalnego użytkownika niebędącego programistą.
<b>Generowanie losowego numeru</b>	9	Prosty i intuicyjny mechanizm dostępny z pozycji <i>Parameters List</i> pozwala na szybkie i bezproblemowe wygenerowanie losowej liczby całkowitej z wybranego zakresu, oraz w pożądanym formacie.

#### 4.2.2 Gatling

W przypadku narzędzia jakim jest Gatling, korelacja *stanów widoku* odbyła się w identyczny sposób, co obsłużenie wszystkich pozostałych dynamicznych parametrów w aplikacji Web Tours Sample Application. Mianowicie, poprzez użycie mechanizmu *check* oraz odpowiednio zredagowanego wyrażenia regularnego. Zadziałał on bezbłędnie, nie było konieczności żadnej dodatkowej konfiguracji.

#### Kod 14. Korelacja stanu widoku w narzędziu Gatling.

```
object Zakladka_Zarejestruj {
  val zakladkaZarejestruj = exec(http("Zakladka_Zarejestruj_1")
    .get("/Account/Register")
    .headers(headers_0)
    .check(regex("id=\"__VIEWSTATE\"
value=\"(.*)\"").saveAs("viewstate"))
    .check(regex("id=\"__EVENTVALIDATION\"
value=\"(.*)\"").saveAs("eventvalidation"))
    .check(regex("id=\"__VIEWSTATEGENERATOR\"
value=\"(.*)\"").saveAs("viewstategenerator"))
    .check(substring("nowe konto"))
  )
  .pause(5 seconds)
}
```

Styczne zasoby (czcionki, arkusze stylów, itd.) są pobierane automatycznie w sposób asynchroniczny podczas wysłania nadrzędnego żądania. Nie są one w żaden sposób zdefiniowane w skrypcie, jednak, jak pokazuje rysunek 43, możliwe jest ich zaobserwowanie podczas uruchomienia skryptu.



Rysunek 43. Statyczne elementy pobierane automatycznie podczas wczytywania strony głównej aplikacji ASP.NET Web Forms

Niestety, jeśli chodzi o mechanizm faktycznej symulacji wysyłania requestów w sposób asynchroniczny na żądanie, to nie jest on dostępny w tym narzędziu.

Generowanie losowego adresu e-mail zostało obsłużone kolejną odsłoną funkcjonalności *feeder*. Okazało się bowiem, że obsługa plików csv nie jest jego jedynym możliwym wykorzystaniem. W tym przypadku została zaimplementowana wewnątrz niego mapa, która generuje swoje kolejne elementy w czasie rzeczywistym, wraz z każdą kolejną iteracją.

Dzięki specyfice języka Scala, aby wdrożyć tego typu konstrukcję wystarczyła zaledwie jednak linia kodu. Definicja tego *feedera* została przedstawiona w kodzie 15.

#### Kod 15. Generowanie losowego adresu e-mail przy wykorzystaniu języka Scala

```
val random_email = Iterator.continually(Map("email" ->
(Random.alphanumeric.take(10).mkString + "@" +
Random.alphanumeric.take(5).mkString + ".com")))
```

W ten sam sposób obsłużono wartość parametru *Student Code*, z tą różnicą, że zamiast tekstu, losowane były liczby całkowite ze zdefiniowanego zakresu.

#### Kod 16. Generowanie losowego numeru *Student Code* przy wykorzystaniu języka Scala

```
val StudentCode = Iterator.continually(Map("pStudentCode" -> (10000
+ scala.util.Random.nextInt(( 99999 - 10000) + 1))))
```

Wszystkie opisane w tym podrozdziale czynności zostały poddane ocenie i przedstawione w tabeli 10.

**Tabela 10. Ocena narzędzia Gatling pod kątem dodatkowych aktywności podczas tworzenia skryptu do aplikacji ASP.NET Web Forms**

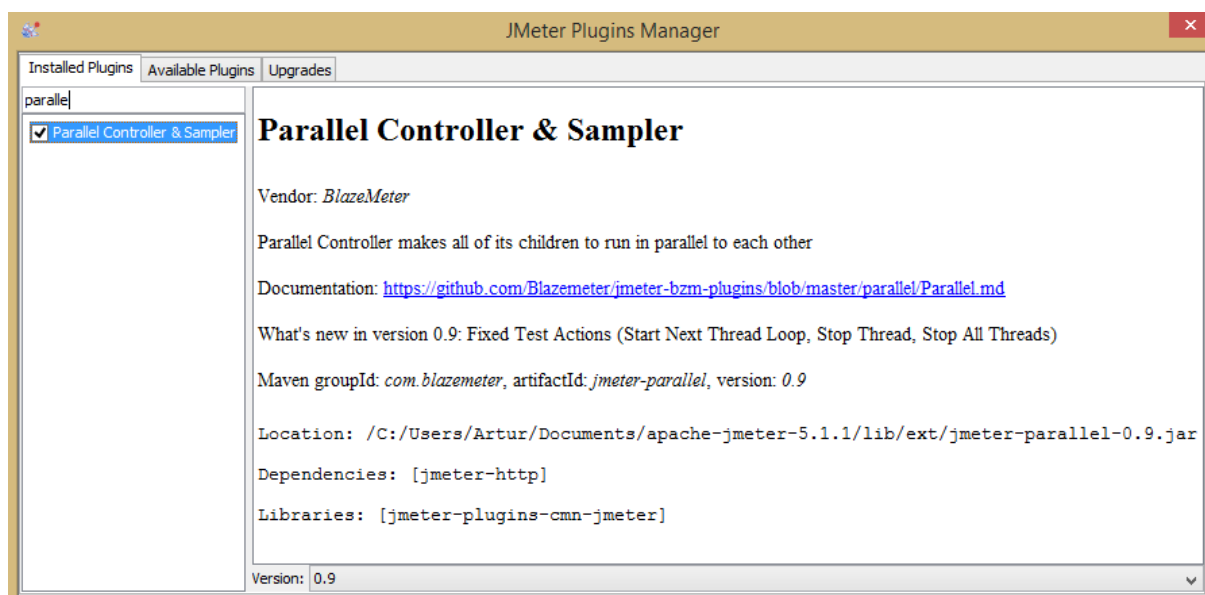
Cecha	Ocena	Argumentacja
Obsługa stanów widoku	6	Brak narzędzi wspomagających korelację, jednak obsługując ją manualnie nie pojawiły się żadne dodatkowe trudności spowodowane stanami widoku.
Obsługa wywołań asynchronicznych	2	Jedynie dla statycznych zasobów. Brak asynchronicznego wysyłania requestów na żądanie użytkownika.
Generowanie losowego tekstu	10	W pełni funkcjonalna, a zarazem zwięzła implementacja wykorzystująca mechanizm <i>feeder</i> .
Generowanie losowego numeru	10	Analogicznie, jak w przypadku generowania losowego tekstu.

#### 4.2.3 JMeter

Podobnie jak Gatling, JMeter obsłużył *stany widoku* bez najmniejszych problemów, w identyczny sposób jak wszystkie pozostałe dynamiczne parametry. Zostało to wykonane przy pomocy opisanego w podrozdziale 4.1.3 Post Procesora – *Regular Expression Extractor*. Wystarczyło odpowiednio zredagowane na tę okoliczność wyrażenie regularne.

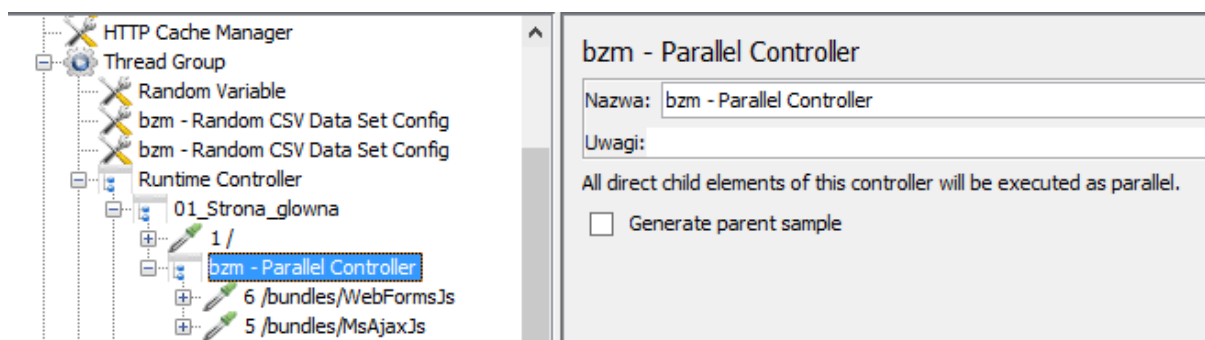
Podstawowy zestaw elementów konfiguracyjnych, jakie oferuje JMeter nie zawiera domyślnie żadnego mechanizmu pozwalającego na jednoczesne wysyłanie requestów.

Na szczęście, w sieci dostępnych jest mnóstwo dodatkowych wtyczek, które rozszerzają możliwości tego narzędzia. Poszukiwanie odpowiedniej wtyczki okazało się niezwykle proste i szybkie, dzięki wbudowanemu modułowi o nazwie *Jmeter Plugins Manager* widoczny na rysunku 44. Po wprowadzeniu odpowiedniego słowa kluczowego, pojawiła się lista dostępnych rozszerzeń. Zostały one zainstalowane od razu z tej pozycji, nie było konieczności przeszukiwania stron internetowych, w celu ich pobrania i instalacji.



**Rysunek 44. Jmeter Plugins Manager**

Po restarcie narzędzia, w zakładce *Logic Controller* dostępny był już nowy element do wyboru - *bzm - Parallel Controller*. Jego obsługa okazała się niezwykle prosta. Wystarczyło dodanie tego elementu do odpowiedniej transakcji i przeniesienie wybranych żądań (tych które będą wysyłane równocześnie) tak, aby stanowiły jego elementy podrzędne. Gotowa konfiguracja została przedstawiona na rysunku 45.



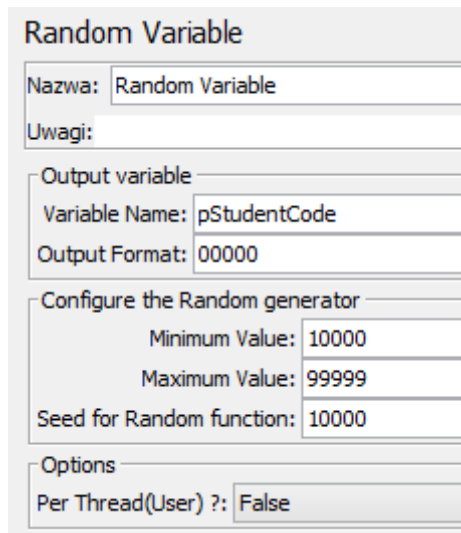
**Rysunek 45. JMeter - Parallel Controller**

Do wygenerowania losowego adresu e-mail, podobnie jak w przypadku losowej daty, został użyty element *JSR223 Sampler*. Ponownie podjęto decyzję o utworzeniu kodu w języku Groovy, co pozwoliło na napisanie zwięzłej, nieskomplikowanej funkcji (kod 17), która w pełni wykonywała swoje zadanie.

#### **Kod 17. Generowanie losowego adresu e-mail przy wykorzystaniu języka Groovy**

```
vars.put("pUsername", org.apache.commons.lang3.RandomStringUtils.random(9, true, true)+'@'+org.apache.commons.lang3.RandomStringUtils.random(5, true, true)+'com')
```

Parametr *Student Code* został z kolei obsługiwany przez inny z elementów konfiguracyjnych JMetera, mianowicie tzw. *Random Variable*. Działa on bardzo podobnie jak analogiczny mechanizm LoadRunniera - parametr typu *Random Number* (opisany w podrozdziale 4.2.1.), czyli zwraca losową liczbę w wybranym formacie ze zdefiniowanego przez użytkownika zakresu. Jego w pełni skonfigurowaną postać można zaobserwować na rysunku 46.



**Random Variable**

Nazwa: Random Variable

Uwagi:

Output variable

Variable Name: pStudentCode

Output Format: 00000

Configure the Random generator

Minimum Value: 10000

Maximum Value: 99999

Seed for Random function: 10000

Options

Per Thread(User)?: False

**Rysunek 46. Jmeter, *Random Variable***

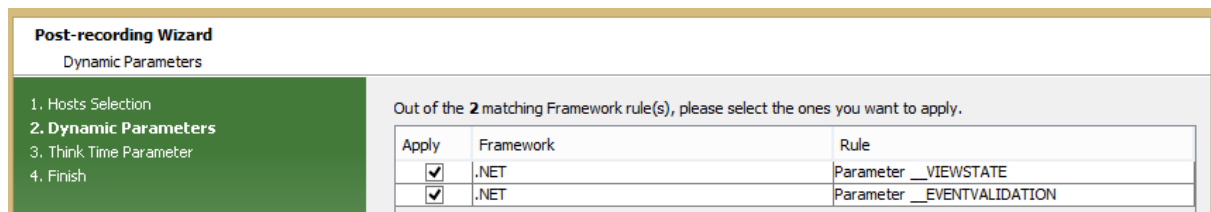
Ocena opisanych w tym podrozdziale czynności wykonanych w narzędziu JMeter znajduje się w tabeli 11.

**Tabela 11. Ocena narzędzia JMeter pod kątem dodatkowych aktywności podczas tworzenia skryptu do aplikacji ASP.NET Web Forms**

Cecha	Ocena	Argumentacja
<b>Obsługa stanów widoku</b>	5	Analogiczna jak w przypadku korelacji pozostałych dynamicznych parametrów. Nie jest wymagana żadna dodatkowa konfiguracja.
<b>Obsługa wywołań asynchronicznych</b>	7	Możliwa po instalacji dodatkowej wtyczki.
<b>Generowanie losowego tekstu</b>	8	Zwięzła i prosta funkcja napisana wewnątrz dedykowanego elementu konfiguracyjnego.
<b>Generowanie losowego numeru</b>	9	Intuicyjny i funkcjonalny kontroler, który pozwala w prosty sposób wygenerować losową liczbę całkowitą w wybranym formacie ze zdefiniowanego zakresu.

#### 4.2.4 NeoLoad

NeoLoad jako jedyny z opisywanych narzędzi dokonał automatycznej korelacji *stanów widoku* w poprawny sposób i to bez konieczności dodatkowej konfiguracji (rysunek 47). Do tego celu został użyty mechanizm *Search for Dynamic Parameters* opisany wcześniej w podrozdziale 4.1.4.



**Post-recording Wizard**  
Dynamic Parameters

1. Hosts Selection  
2. **Dynamic Parameters**  
3. Think Time Parameter  
4. Finish

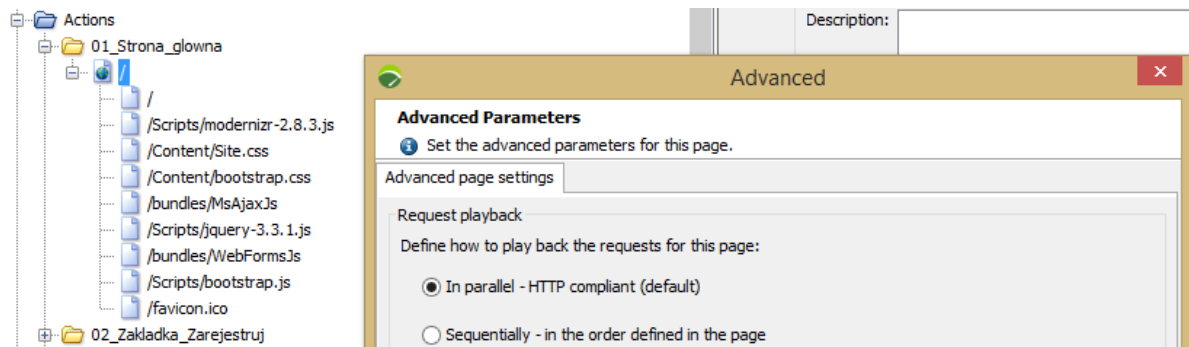
Out of the 2 matching Framework rule(s), please select the ones you want to apply.

Apply	Framework	Rule
<input checked="" type="checkbox"/>	.NET	Parameter __VIEWSTATE
<input checked="" type="checkbox"/>	.NET	Parameter __EVENTVALIDATION

**Rysunek 47. Automatyczna korelacja *stanów widoku* w narzędziu NeoLoad**

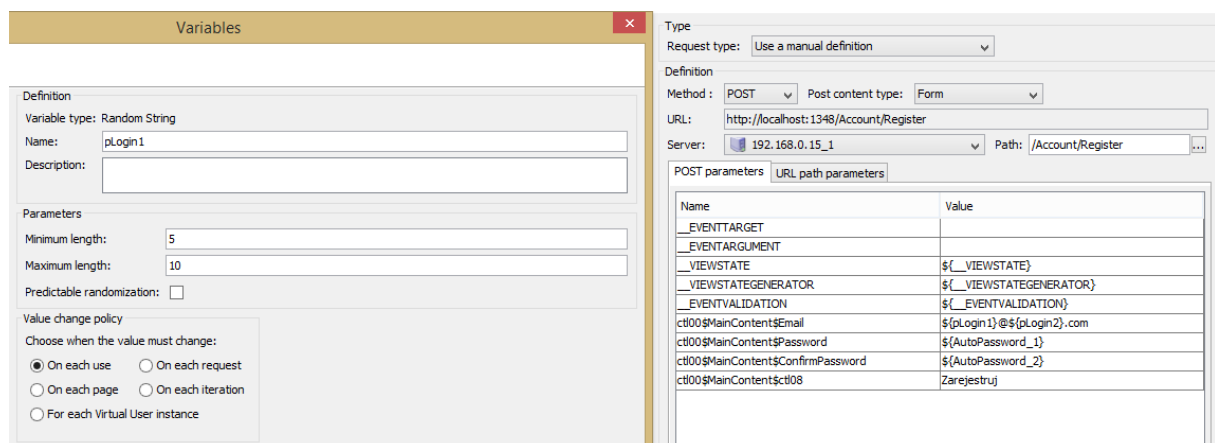


Również asynchroniczne wysłanie requestów zostało obsługiwane automatycznie już na podstawie nagrania. Jednocześnie wysłane żądania znalazły się w jednej grupie, która określa porządek ich wywoływania. Rysunek 48 prezentuje okno odpowiedzialne za to ustawienie. Oczywiście w dalszym ciągu istniała możliwość ręcznego przestawienia tego parametru na sekwencyjny.



**Rysunek 48 Obsługa requestów asynchronicznych w narzędziu NeoLoad**

Analogicznie, jak w przypadku obsługi dat w porozdziale 4.1.4, NeoLoad posiada mechanizm do generowania losowego ciągu znaków, tzw. *Random String*. Aby otrzymać tego typu tekst, wystarczyło podać jego minimalną oraz maksymalną długość. Nadanie mu formy adresu e-mail zostało obsługiwane w sposób pokazany na rysunku 49.



**Rysunek 49. NeoLoad, parametr typu *Random String***

Wśród automatycznie obsługiwanych parametrów w zakładce *Variables*, znajduje się również tzw. *Random Integer*, który jest dedykowany do generowania losowych liczb całkowitych. Niestety nie posiada definicji formatu, zatem aby za każdym razem otrzymać liczbę pięciocyfrową zakres losowania został zdefiniowany jako 10000-99999 (rysunek 50).

**Definition**

Variable type: Random Integer

Name:

Description:

---

**Parameters**

Minimum value:

Maximum value:

Predictable randomization:

---

**Value change policy**

Choose when the value must change:

On each use   
 On each request  
 On each page   
 On each iteration  
 For each Virtual User instance

**Rysunek 50. NeoLoad, parametr typu *Random Integer***

Wszystkie opisane w tym podrozdziale czynności zostały poddane ocenie i przedstawione w tabeli 12.

**Tabela 12. Ocena narzędzia NeoLoad pod kątem dodatkowych aktywności podczas tworzenia skryptu do aplikacji ASP.NET Web Forms**

Cecha	Ocena	Argumentacja
<b>Obsługa stanów widoku</b>	10	Poprawnie obsłużone za pomocą mechanizmu automatycznej korelacji. Nie była wymagana żadna dodatkowa konfiguracja.
<b>Obsługa wywołań asynchronicznych</b>	10	Skonfigurowane automatycznie na podstawie nagrania.
<b>Generowanie losowego tekstu</b>	8	Automatyczny mechanizm generowania losowego tekstu, niestety bez możliwości określenia jego formatu.
<b>Generowanie losowego numeru</b>	8	Jw. Liczby losowane automatycznie z podanego zakresu, jednak bez opcji podania pożądanego formatu.

#### 4.2.5 TruClient

Z uwagi na specyfikę skryptu, TruClient nie obsługuje *stanów widoku*, czy asynchronicznych requestów w sposób zauważalny dla użytkownika. Nie oznacza to oczywiście, że nie robi tego wcale, wręcz przeciwnie. Z uwagi na fakt, że skrypt porusza się w obrębie interfejsu graficznego na fizycznej instancji przeglądarki, cała komunikacja z serwerem jest realizowana w dokładnie taki sam sposób, w jaki ma to miejsce podczas faktycznej pracy realnego użytkownika.

Tak jak zostało to opisane w podrozdziale 4.1.5, jedną z dostępnych funkcjonalności TruClienta jest specjalny kontener umożliwiający wpisanie własnego kodu w wybranym języku programowania. Dokładnie w taki sposób zostało obsłużone generowanie losowego adresu e-mail. Do tego celu posłużył element o nazwie *Evaluate JavaScript* oraz funkcja napisana w tym właśnie języku.

#### Kod 18. Generowanie losowego adresu e-mail przy wykorzystaniu języka JavaScript

```
var pUsername = Math.random().toString(36).substring(2, 15) + '@' +
Math.random().toString(36).substring(2, 8) + '.com';
```

Jak pokazuje kod 18, wygenerowana w ten sposób wartość została od razu przypisana do zmiennej, co pozwoliło wykorzystać ją później w polu tekstowym.

Obsługa losowego numeru została wykonana w identyczny sposób jak miało to miejsce w LoadRunnerze w protokole WEB (podrozdział 4.2.1), czyli przy pomocy parametru typu *Random Number*.

**Tabela 13. Ocena narzędzia TruClient pod kątem dodatkowych aktywności podczas tworzenia skryptu do aplikacji ASP.NET Web Forms**

Cecha	Ocena	Argumentacja
Obsługa stanów widoku	10	Brak konieczności ręcznej obsługi, wszystkie niuanse komunikacyjne obsługiwane są przez Front End oraz przeglądarkę internetową.
Obsługa wywołań asynchronicznych	10	jw.
Generowanie losowego tekstu	8	Prosty, zwięzły kod napisany w języku JavaScript dodany do skryptu przy pomocy odpowiedniego kontenera.
Generowanie losowego numeru	9	Tak jak w przypadku protokołu Web, został użyty mechanizm <i>Random Number</i> dostępny z pozycji <i>Parameters List</i> .

### 4.3 Skrypty dla aplikacji Employee Directory

Z uwagi na fakt, że tylko 2 z 5 narzędzi były w stanie obsłużyć aplikację tego typu, tym razem nie zostały stworzone tabele oceniające poszczególne aspekty tworzenia skryptu. Obsługę Flex potraktowano całościowo, zatem ocena i jej uzasadnienie znajduje się na końcu każdego podrozdziału w formie tekstowej.

#### 4.3.1 LoadRunner

W pierwszej kolejności podjęto próbę nagrania skryptu przy pomocy protokołu Web, tak jak w przypadku pozostałych aplikacji. Niestety zakończyło się to niepowodzeniem, ponieważ binarne ciało żądania zostało zarejestrowane w postaci czystego tekstu. Wynik tego nagrywania przedstawia kod 19.

#### Kod 19. Przykładowy request aplikacji Employee Directory nagrany protokołem Web

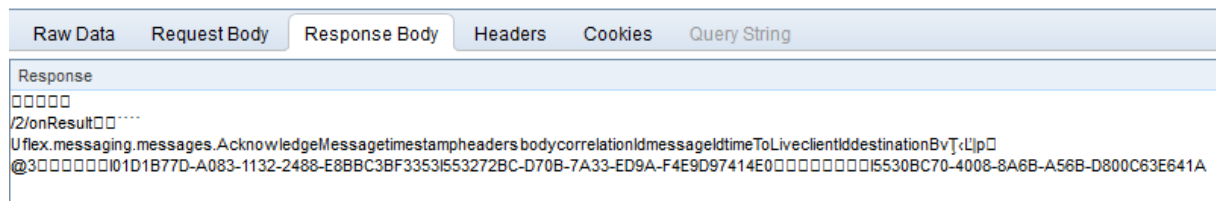
```
web_custom_request(";jsessionid=8430d2cac66bb75980c6fe325c42671d2e3f_2", "URL=http://192.168.0.15:8500/flex2gateway/;jsessionid=8430d2cac66bb75980c6fe325c42671d2e3f",
```

```

"Method=POST",
"TargetFrame=",
"Resource=0",
"RecContentType=application/x-amf",
"Referer=http://192.168.0.15:8500/Flex_CMS-
debug/Flex_CMS.swf/[ [DYNAMIC]]/4",
"Snapshot=t13.inf",
"Mode=HTML",
"EncType=application/x-amf",
"BodyBinary=\\x00\\x03\\x00\\x00\\x00\\x01\\x00\\x04null\\x0
0\\x02/2\\x00\\x00\\x02\\x07\\n\\x00\\x00\\x00\\x01\\x11\\n\\x81\\x130
flex.messaging.messages.RemotingMessage\\x13operation\\rsource\\x13ti
mestamp\\tbody\\x11clientId\\x0Fheaders\\x17destination\\x13messageId
\\x15timeToLive\\x06\\x1DcreateEmployee\\x06ETestDrive.services.Empl
oyeeService\\x04\\x00\\t\\x05\\x01\\n\\x81c\\x01\\x0Bstate\\x05id\\x13
photofile\\roffice\\x19departmentid\\rstreet\\t"
"city\\x13firstname\\x0Btitle\\x11lastname\\x0Fzipcode\\x0Be
mail\\x13cellphone\\x17officephone\\x06\\x01\\x04\\x00\\x06\\x01\\x0
6\\x01\\x04\\x00\\x06\\x01\\x06\\x01\\x06\\x11imieeeee\\x06\\x01\\x0
6)nazwiskoooooooooooooooo\\x06\\x01\\x06\\x01\\x06\\x01\\x06\\x01\\x01
\\x06I5530BC70-4008-8A6B-A56B-
D800C63E641A\\n\\x0B\\x01\\tDSId\\x06I5530BB63-B21F-5490-2251-
4A907A820D09\\x15DSEndpoint\\x06\\x11my-
cfamf\\x01\\x06\\x15ColdFusion\\x06I01D1B77D-A083-1132-2488-
E8BBC3BF3353\\x04\\x00",
LAST);

```

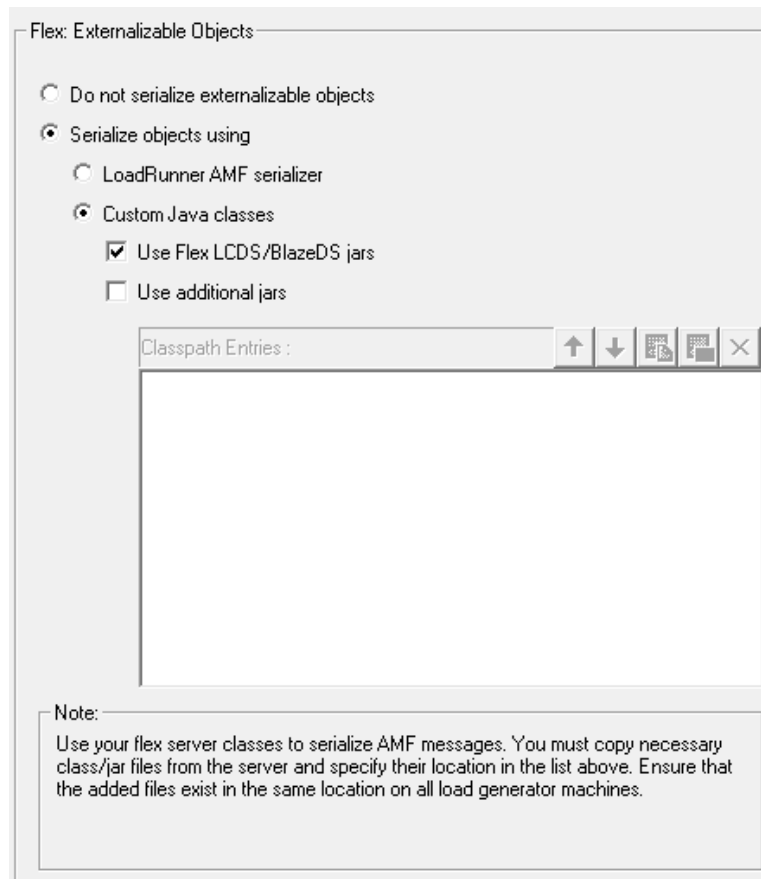
Co prawda w dalszym ciągu możliwe było wysłanie żądania w takiej formie, jednak jego edycja stawała się już niewykonalna. Podobny problem dotyczył zarejestrowanych odpowiedzi. Format ich prezentacji był nieczytelny, co skutecznie uniemożliwiło była ekstrakcję dynamicznych parametrów. Przykładowa odpowiedź widoczna jest na rysunku 51.



**Rysunek 51. Przykładowa odpowiedź aplikacji Employee Directory nagrana protokołem Web**

Aby skrypt otrzymał czytelną formę, która pozwoli na obsługę dynamicznych parametrów konieczna była odpowiednia serializacja obiektów. W tym celu utworzono nowy projekt, dla którego wybrano tym razem inny protokół – *Flex*.

Dawał on możliwość określenia czy i w jaki sposób ma zostać wykonana wspomniana serializacja. W tym celu można było skorzystać z klas oferowanych przez LoadRunniera bądź wskazać ścieżki do własnych, jeśli zaistniała by taka potrzeba. W tym wypadku wystarczyły wbudowane biblioteki BlazeDS. Sam proces nagrywania wyglądał identycznie jak w przypadku pozostałych opisywanych aplikacji.



**Rysunek 52. Opcje nagrywania dla protokołu Flex**

Po nagraniu skryptu przy użyciu ustawień widocznych na rysunku 52, zauważalna była znaczna różnica w wygenerowanym kodzie. Tym razem ciało żądania otrzymało format xml i bez problemu można było odczytać i sparametryzować każde z wysyłanych pól, co zostało zaprezentowane w kodzie 20.

**Kod 20. Przykładowy request aplikacji Employee Directory nagrany protokołem Flex**

```
flex_amf_call(
"AMF3_call_2",
"Gateway={pURL}/flex2gateway/;jsessionid={jsessionid}",
"Snapshot=t15.inf",
MESSAGE,
"Method=null",
"TargetObjectId=/2",
BEGIN_ARGUMENTS,
"<AMF3>"
"<object-externalizable-custom>"
"<flex.messaging.messages.RemotingMessage>"
"<clientId class=\"string\">{DSId_1}</clientId>"
"<destination>ColdFusion</destination>"
"<messageId>FDE0B75A-ED1B-9404-7D57-3178C8C0EA1C</messageId>"
"<timestamp>0</timestamp>"
"<timeToLive>0</timeToLive>"
"<headers>"
"<entry>"
"<string>DSEndpoint</string>"
```

```

"<string>my-cfamf</string>"
"</entry>"
"<entry>"
"<string>DSId</string>"
"<string>{DSId}</string>"
"</entry>"
"</headers>"
"<source>TestDrive.services.EmployeeService</source>"
"<operation>createEmployee</operation>"
"<parameters>"
"<flex.messaging.io.amf.ASObject serialization=\"custom\">"
"<unserializable-parents></unserializable-parents>"
"<map>"
"<default>"
"<loadFactor>0.75</loadFactor>"
"<threshold>24</threshold>"
"</default>"
"<int>32</int>"
"<int>14</int>"
"<string>firstname</string>"
"<string>{pImie}</string>"
"<string>city</string>"
"<string>{pMiasto}</string>"
"<string>departmentid</string>"
"<int>0</int>"
"<string>office</string>"
"<string>{pMiasto}</string>"
"<string>title</string>"
"<string>{pStanowisko}</string>"
"<string>lastname</string>"
"<string>{pNazwisko}</string>"
"<string>zipcode</string>"
"<string>{pKod}</string>"
"<string>photofile</string>"
"<string>{pImie}{pNazwisko}.jpg</string>"
"<string>street</string>"
"<string>{pUlica}</string>"
"<string>cellphone</string>"
"<string>{pNrKom}</string>"
"<string>state</string>"
"<string></string>"
"<string>id</string>"
"<int>0</int>"
"<string>officephone</string>"
"<string>0048{pNrTel}</string>"
"<string>email</string>"
"<string>{pImie}{pNazwisko}@test.com</string>"
"</map>"
"<flex.messaging.io.amf.ASObject>"
"<default>"
"<inHashCode>>false</inHashCode>"
"<inToString>>false</inToString>"
"</default>"
"</flex.messaging.io.amf.ASObject>"
"</flex.messaging.io.amf.ASObject>"

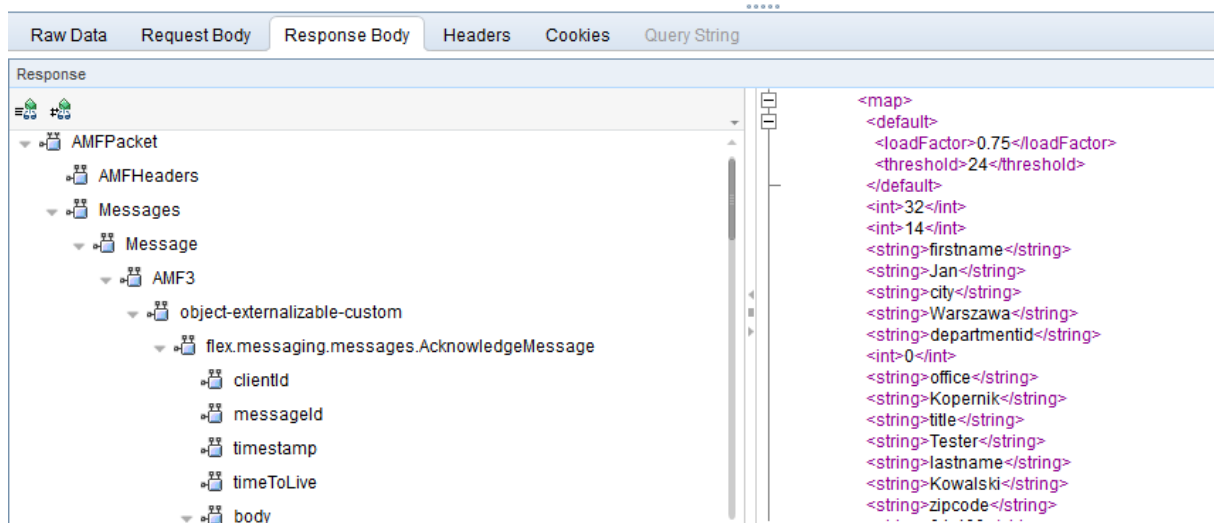
```

```

"<null></null>"
"</parameters>"
"</flex.messaging.messages.RemotingMessage>"
"</object-externalizable-custom>"
"</AMF3>",
END_ARGUMENTS,
LAST);

```

Proces serializacja obiektów został również przeprowadzony na odpowiedziach otrzymanych z serwera podczas nagrywania, jak również później przy okazji uruchamiania skryptu. Fragment ciała przykładowej odpowiedzi został przedstawiony na rysunku 53.



**Rysunek 53. Przykładowa odpowiedź aplikacji Employee Directory nagrana protokołem Flex**

Dzięki tym zabiegom wykonalne stało się przeprowadzenie dalszych czynności prowadzących do powstania w pełni funkcjonalnego skryptu. Analogicznie do podrozdziału 4.1.1 były to m.in. korelacja dynamicznych wartości, randomizacja wprowadzanych parametrów, weryfikacja poprawności odpowiedzi za pomocą asercji itd.

Podsumowując, stworzenie skryptu dla aplikacji Employee Directory zakończyło się sukcesem, jednak, aby stało się to możliwe konieczne było wybranie odpowiedniego protokołu oraz właściwa konfiguracja opcji nagrywania. Ostateczna ocena dla tego narzędzia, pod kątem bieżącego podrozdziału to 8.

### 4.3.2 Gatling

W przypadku Gatlinga, nagranie komunikacji dla aplikacji *Employee Directory* poskutkowało wygenerowaniem na pozór standardowego kodu. Widoczna była jednak jedna znacząca różnica, mianowicie binarne ciało żądania zostało przeniesione do zewnętrznego pliku, w skrypcie natomiast zapisana była jedynie jego referencja, tak jak pokazuje to kod 21.

#### Kod 21. Przykładowy request aplikacji Employee Directory nagrany w narzędziu Gatling

```

.exec (http ("request_20")
  .post ("/flex2gateway/")
  .headers (headers_20)
  .body (RawFileBody ("flex_0020_request.txt"))
  .resources (http ("request_21"))

.post ("/flex2gateway/;jsessionid=84304d82e60548f6b2373c4323e686d74773")

```

```
.headers(headers_20)
.body(RawRequestBody("flex_0021_request.txt"),
      http("request_22"))
.get(uri1 + "")
.headers(headers_22))
```

Niestety, po otwarciu wspomnianego pliku okazało się, że ma on nie do końca czytelną postać pełną nieprawidłowo obsłużonych znaków (kod 22).

### Kod 22. Przykładowe ciało żądania aplikacji Employee Directory nagrane w narzędziu Gatling

```
null /1 ❖
❖ Mflex.messaging.messages.CommandMessage operation correlationId de
stination headers
body clientId messageId timestamp timeToLive

%DSMessagingVersion      DSId nil
ICCED99E7-ED0B-1FA0-FDA1-3F8C1B02C95A
```

Nie było to jednak głównym problemem, treść parametrów wyświetlona została czystym tekstem i drogą dedukcji, przy wsparciu innymi narzędziami, możliwe byłoby ich sparametryzowanie. Niemniej jednak treść odpowiedzi odbieranych z serwera prezentowały analogiczną postać. Tym samym wykluczało to użycie mechanizmów służących do ekstrakcji dynamicznych wartości, co z kolei uniemożliwiało stworzenie w pełni działającego skryptu.

Podjęto szereg prób zapisania wyrażeń regularnych przy użyciu wyświetlanych w kodzie 23 niestandardowych symboli, jednak wszystkie zakończyły się niepowodzeniem. Oczekiwana wartość nie była znajdowana.

### Kod 23. Przykładowe ciało odpowiedzi aplikacji Employee Directory nagrane w narzędziu Gatling

```
/7/onResult ❖❖❖❖
❖ Uflex.messaging.messages.AcknowledgeMessage timestamp headers
body correlationId messageId timeToLive clientId destination Bv❖❖
❖`
I5DDE765F-B498-F4A7-E2E7-BF609B1FF98D ID23ADAE-E-D60F-D1AA-C56F-
84C63F12B4AD ID230D41B-E70C-73B9-894F-26DB8596D92C
```

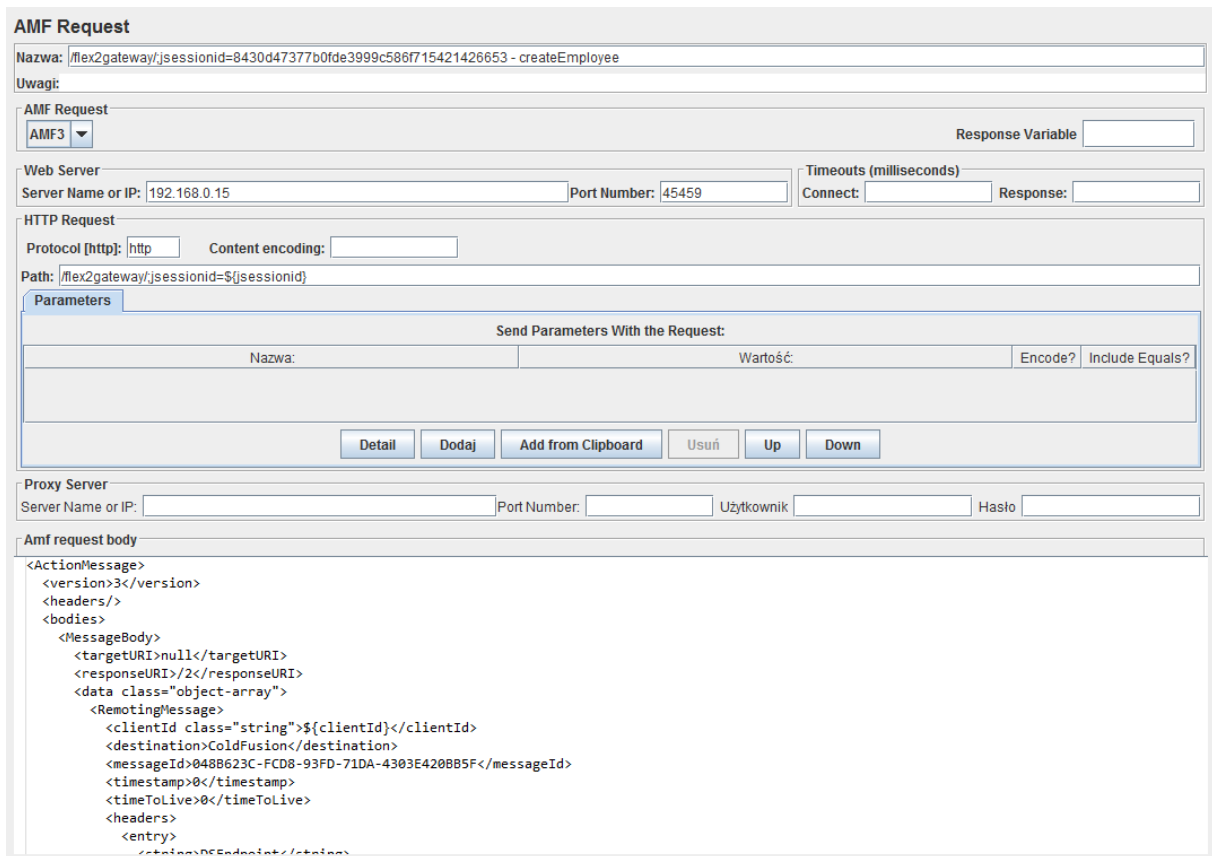
Podsumowując, Gatling nie był w stanie obsłużyć komunikacji generowanej przez aplikację *Employee Directory* w sposób pozwalający na wykonanie skryptu. Możliwe do odczytania były jedynie pojedyncze wartości parametru. Ostateczna ocena tego narzędzia pod kątem bieżącego podrozdziału to 2.

#### 4.3.3 JMeter

Próba nagrania skryptu tradycyjną metodą, podobnie jak w przypadku LoadRunnera i Gatlinga zakończyła się niepowodzeniem. Zarówno żądania jak i odpowiedzi wyświetlane były w postaci binarnej, co uniemożliwiało ich poprawne odczytanie.

Kolejne podejście zostało wykonane przy użyciu specjalnej wtyczki, tzw. *JMeter AMF*, pobranej z [30]. Pozwoliła ona na poprawne zarejestrowanie żądań oraz ich serializację (wyświetlenie ich ciał w formacie xml). Do ich obsługi posłużył nowy, dołączony do wtyczki element konfiguracyjny – tzw. AMF Request widoczny na rysunku 54.

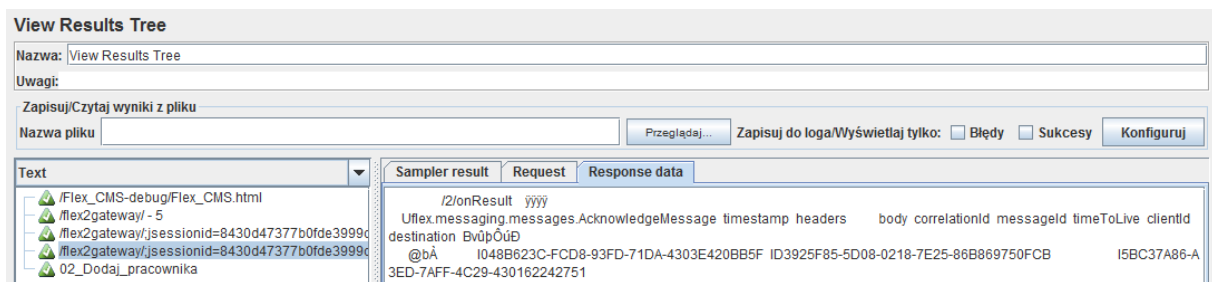




**Rysunek 54. Przykładowe żądanie aplikacji Employee Directory nagrany za pomocą wtyczki *JMeter AMF***

Niestety opiswane rozszerzenie nie działa na najnowszej na tą chwilę wersji JMetera 5.1 i konieczne było użycie jednego ze starszych, już niekatulanych wydań tego narzędzia (2.13) [29].

Kolejnym dużym minusem okazał się fakt, że serializacja przeprowadzana jest tylko i wyłącznie dla żądań. Ciało odpowiedzi nadal wyświetlane były w binarnej postaci (rysunek 55). W tym momencie pojawiła się identyczna przeszkoda jaka miała miejsce w Gatlingu – brak możliwości ekstrakcji wartości dynamicznych parametrów z odbieranych odpowiedzi. Z tego powodu, stworzenie w pełni działającego skryptu okazało się finalnie niemożliwe.



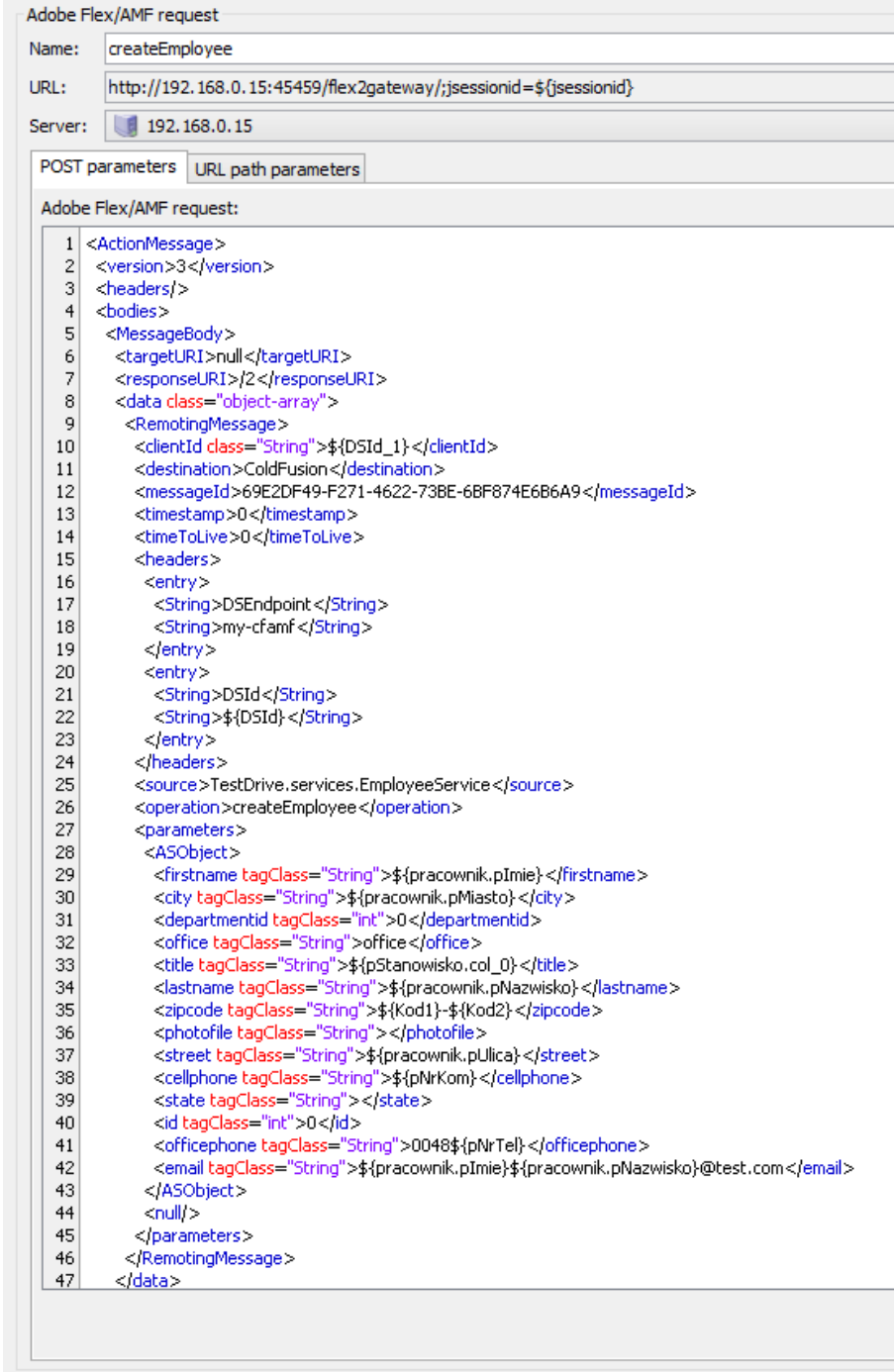
**Rysunek 55. Przykładowe ciało odpowiedzi aplikacji Employee Directory nagrane w narzędziu *JMeter***

Podsumowując, dodatkowa wtyczka pozwala w poprawny sposób odczytywać i parametryzować żądania. Niestety, mechanizm ten nie działał na odbierane odpowiedzi, przez co utworzenie skryptu stało się niewykonalne. Biorąc pod uwagę te czynniki, ostateczna ocena JMetera, pod kątem bieżącego podrozdziału to 4.

#### 4.3.4 NeoLoad

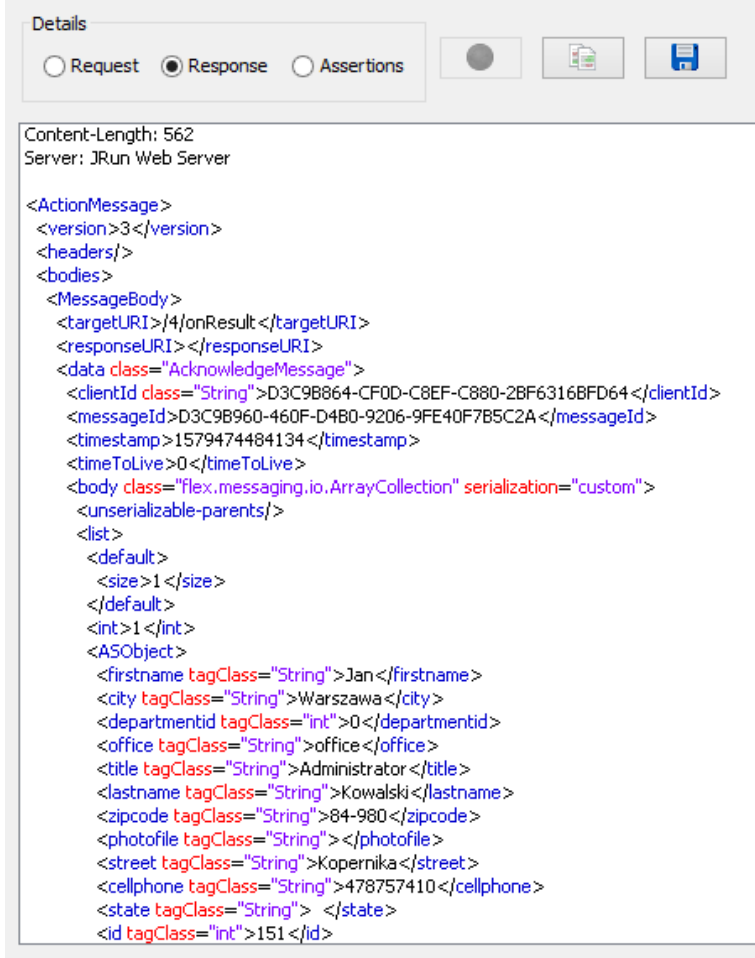
NeoLoad, jako jedyne z opisywanych narzędzi, poprawnie zarejestrował i dokonał serializacji całej nagranej komunikacji przy użyciu domyślnych ustawień, bez konieczności żadnej dodatkowej konfiguracji.

Po stworzeniu nowego projektu i nagraniu skryptu, NeoLoad automatycznie rozpoznał jakiego rodzaju komunikacja została zarejestrowana i bezpośrednio po zakończeniu nagrania każde żądanie i odpowiedź widoczne było w postaci czystego tekstu w formacie xml (rysunek 56 i 57).



```
1 <ActionMessage>
2 <version>3</version>
3 <headers/>
4 <bodies>
5 <MessageBody>
6 <targetURI>null</targetURI>
7 <responseURI>/2</responseURI>
8 <data class="object-array">
9 <RemotingMessage>
10 <clientId class="String">${DSId_1}</clientId>
11 <destination>ColdFusion</destination>
12 <messageId>69E2DF49-F271-4622-73BE-6BF874E6B6A9</messageId>
13 <timestamp>0</timestamp>
14 <timeToLive>0</timeToLive>
15 <headers>
16 <entry>
17 <String>DSEndpoint</String>
18 <String>my-cfamf</String>
19 </entry>
20 <entry>
21 <String>DSId</String>
22 <String>${DSId}</String>
23 </entry>
24 </headers>
25 <source>TestDrive.services.EmployeeService</source>
26 <operation>createEmployee</operation>
27 <parameters>
28 <ASObject>
29 <firstname tagClass="String">${pracownik.pImie}</firstname>
30 <city tagClass="String">${pracownik.pMiasto}</city>
31 <departmentid tagClass="int">0</departmentid>
32 <office tagClass="String">office</office>
33 <title tagClass="String">${pStanowisko.col_0}</title>
34 <lastname tagClass="String">${pracownik.pNazwisko}</lastname>
35 <zipcode tagClass="String">${Kod1}-${Kod2}</zipcode>
36 <photofile tagClass="String"></photofile>
37 <street tagClass="String">${pracownik.pUlica}</street>
38 <cellphone tagClass="String">${pNrKom}</cellphone>
39 <state tagClass="String"></state>
40 <id tagClass="int">0</id>
41 <officephone tagClass="String">0048${pNrTel}</officephone>
42 <email tagClass="String">${pracownik.pImie}${pracownik.pNazwisko}@test.com</email>
43 </ASObject>
44 <null/>
45 </parameters>
46 </RemotingMessage>
47 </data>
```

Rysunek 56. Przykładowy request aplikacji Employee Directory nagrany za pomocą narzędzia NeoLoad



```
Details
Request Response Assertions
Content-Length: 562
Server: JRun Web Server

<ActionMessage>
  <version>3</version>
  <headers/>
  <bodies>
    <MessageBody>
      <targetURI>/4/onResult</targetURI>
      <responseURI></responseURI>
      <data class="AcknowledgeMessage">
        <clientId class="String">D3C9B864-CF0D-C8EF-C880-2BF6316BFD64</clientId>
        <messageId>D3C9B960-460F-D4B0-9206-9FE40F7B5C2A</messageId>
        <timestamp>1579474484134</timestamp>
        <timeToLive>0</timeToLive>
        <body class="flex.messaging.io.ArrayCollection" serialization="custom">
          <unserializable-parents/>
          <list>
            <default>
              <size>1</size>
            </default>
            <int>1</int>
            <ASObject>
              <firstname tagClass="String">Jan</firstname>
              <city tagClass="String">Warszawa</city>
              <departmentid tagClass="int">0</departmentid>
              <office tagClass="String">office</office>
              <title tagClass="String">Administrator</title>
              <lastname tagClass="String">Kowalski</lastname>
              <zipcode tagClass="String">84-980</zipcode>
              <photofile tagClass="String"></photofile>
              <street tagClass="String">Kopernika</street>
              <cellphone tagClass="String">478757410</cellphone>
              <state tagClass="String"></state>
              <id tagClass="int">151</id>
            </ASObject>
          </list>
        </body>
      </data>
    </MessageBody>
  </bodies>
</ActionMessage>
```

**Rysunek 57. Przykładowa odpowiedź aplikacji Employee Directory nagrana za pomocą narzędzia NeoLoad**

Powyższe fakty pozwoliły na podjęcie dalszych kroków (analogicznie jak w podrozdziale 4.1.4) prowadzących do stworzenia finalnego, w pełni działającego skryptu.

Podsumowując, NeoLoad umożliwił nagranie i edycję skryptu w sposób analogiczny do standardowej aplikacji internetowej. Użytkownik nie musiał podejmować żadnych dodatkowych działań spowodowanych odmiennością technologii w jakiej powstała aplikacja. Narzędzie to otrzymuje zatem maksymalną ocenę dla bieżącego podrozdziału, czyli 10.

#### 4.3.5 TruClient

Bez względu na użyte ustawienia nagrywania, TruClient nie rejestrował żadnych akcji wykonywanych na aplikacji *Employee Directory*. Wygląda na to, że narzędzie to nie jest w stanie obsłużyć technologii w jakiej napisana została aplikacja. Jego ocena pod kątem bieżącego podrozdziału wynosi zatem 0.

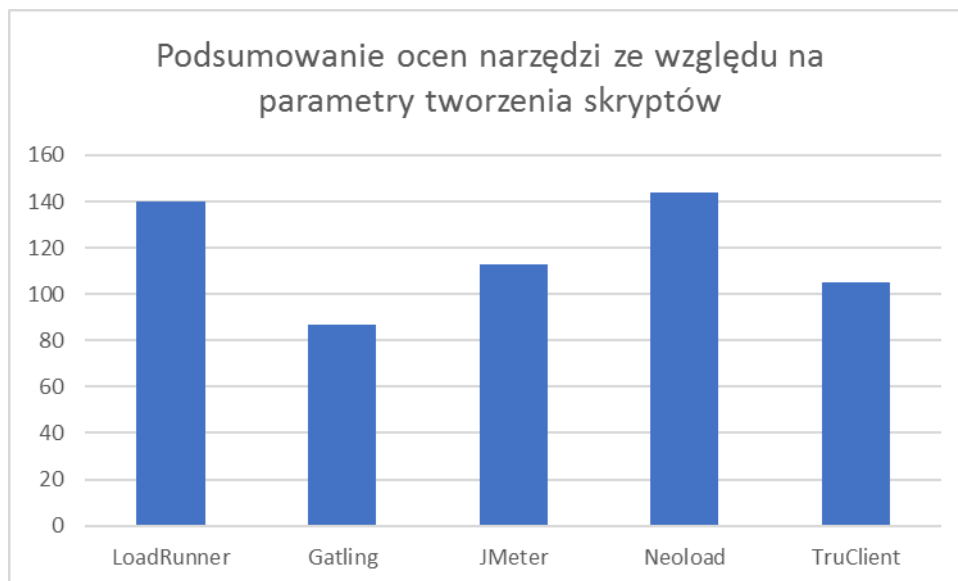
## 4.4 Analiza porównawcza narzędzi użytych do tworzenia skryptów

W tym podrozdziale przedstawione zostało podsumowanie ocen dla każdego z narzędzi opisanych w podrozdziale 3.1. Noty dla każdej z aktywności zostały zagregowane i zsumowane w tabeli 14.

Tabela 14. Podsumowanie ocen każdego z narzędzi pod kątem tworzenia skryptów

Cecha	LoadRunner	Gatling	JMeter	NeoLoad	TruClient
Instalacja narzędzia	10	3	10	10	10
Przejrzystość interfejsu	10	3	7	9	9
Nagrywanie requestów	10	6	6	10	7
Tworzenie transakcji	9	0	8	10	5
Obsługa zewnętrznych danych testowych	8	7	9	8	8
Możliwość dodania własnego kodu	7	10	8	6	7
Korelacja dynamicznych wartości	10	6	5	10	2
Asercje	10	7	8	8	7
Think time	10	5	6	7	3
Debugowanie	9	5	6	10	4
Tempo tworzenia skryptu	10	5	7	10	6
Obsługa stanów widoku	3	6	5	10	10
Obsługa wywołań asynchronicznych	10	2	7	10	10
Generowanie losowego tekstu	7	10	8	8	8
Generowanie losowego numeru	9	10	9	8	9
Obsługa niestandardowej serializacji obiektów (binarne body)	8	2	4	10	0
<b>SUMA:</b>	<b>140</b>	<b>87</b>	<b>113</b>	<b>144</b>	<b>105</b>

Rysunek 58 pokazuje dodatkowo proporcję zdobytych punktów przez poszczególne narzędzia.



Rysunek 58. Wykres przedstawiający sumę zdobytych punktów przez każde z narzędzi

Najbardziej wszechstronnym i funkcjonalnym narzędziem ze względu na parametry tworzenia skryptów okazał się NeoLoad, tuż za nim jest LoadRunner.

Najwyższe oceny otrzymały zatem narzędzia komercyjne, których użytkowanie wymagała zakupienia płatnych licencji. O wysokich notach zadecydowały w dużej mierze szereg usprawnień i dodatkowych mechanizmów usprawniających wykonywane czynności. Narzędzia te w wielu kwestiach wręcz wyręczyły użytkownika, przez co w znaczny sposób przyspieszyły proces tworzenia skryptów testowych, a co za tym idzie testów wydajnościowych samych w sobie.

Ich darmowi konkurenci odstają od nich w znaczącej mierze, jednak należy pamiętać o tym, że w tym porównaniu nie były brane pod uwagę koszty danego przedsięwzięcia. W przypadku projektów o niskim budżecie ten czynnik może okazać się decydujący.

Najlepszym darmowym narzędziem okazał się JMeter. Zakres jego funkcjonalności jest imponujący, jednak większość czynności wymagało zaangażowania ze strony użytkownika, co wydłużyło czas wykonania skryptów, a to z kolei opóźniło testy i ich wyniki.

## 5. Testy wydajnościowe przeprowadzone w celu porównania narzędzi

W odróżnieniu do tworzenia skryptów, konfiguracja i uruchomienie testu w danym narzędziu wykonywana jest identycznie, bez względu na technologię w jakiej została stworzona aplikacja. Z tego powodu, w celu porównania potencjału poszczególnych narzędzi, wystarczyło szczegółowe opisanie jednej reprezentacyjnej aplikacji, na którą wybrano Web Tours Sample Application.

Ponieważ niezwykle ważne było zweryfikowanie, czy pozostałe stworzone skrypty sprawdzą się również podczas faktycznych testów, zostały one także uruchomione dla dwóch pozostałych aplikacji: ASP.NET Web Forms oraz Employee Directory. Jednak w ich przypadku opisano jedynie wyniki testów (proces ich przeprowadzenia był identyczny).

### 5.1 Przygotowanie i przeprowadzenie testów dla aplikacji Web Tours Sample Application

W celu zbadania, jaki wpływ na zasoby fizyczne maszyny miało wykonanie testu przez każde z narzędzi, skonfigurowany został monitoring w postaci *Performance Monitora*, który zbierał dane podczas każdego z uruchomień. Posłużono się przy tym wskazówkami z [31].

Każdy z testów miał też zapewnione identyczne warunki początkowe. W tym celu, pomiędzy kolejnymi testami, wszyscy użytkownicy mieli kasowaną historię lotów, które zarezerwowali podczas poprzedniego uruchomienia.

Jak już wspomniano w podrozdziale 3.2.1., dane użytkowników w aplikacji Web Tours Sample Application (w tym zamówione loty) przechowywane są w stworzonych w tym celu plikach tekstowych, które są aktualizowane na bieżąco z poziomu aplikacji. Aby nie logować się oddzielnie na każde konto z osobną, został stworzony specjalny skrypt w języku Visual Basic Script, który automatycznie usuwał wszystkie zamówione loty z pliku, pozostawiając w nim jedynie dane personalne. Podczas jego tworzenia wspomagano się poradnikiem [32].

Kolejne podrozdziały opisują szczegółowo jak przebiegał proces konfiguracji oraz uruchomienia testu, jego monitorowania oraz zebrania wyników, odpowiednio dla każdego z narzędzi.

#### 5.1.1 LoadRunner

Konfiguracja i uruchomienie testu w narzędziu LoadRunner odbyła się za pomocą jego kolejnego modułu – *Controllera*. Po utworzeniu w nim nowego scenariusza konieczne było wskazanie skryptów, które miały wziąć udział w teście. W tym wypadku został stworzony tylko jeden, opisany w podrozdziale 4.1.1.

Na tym etapie wybrany został również typ scenariusza. LoadRunner oferuje 2 różne typy:

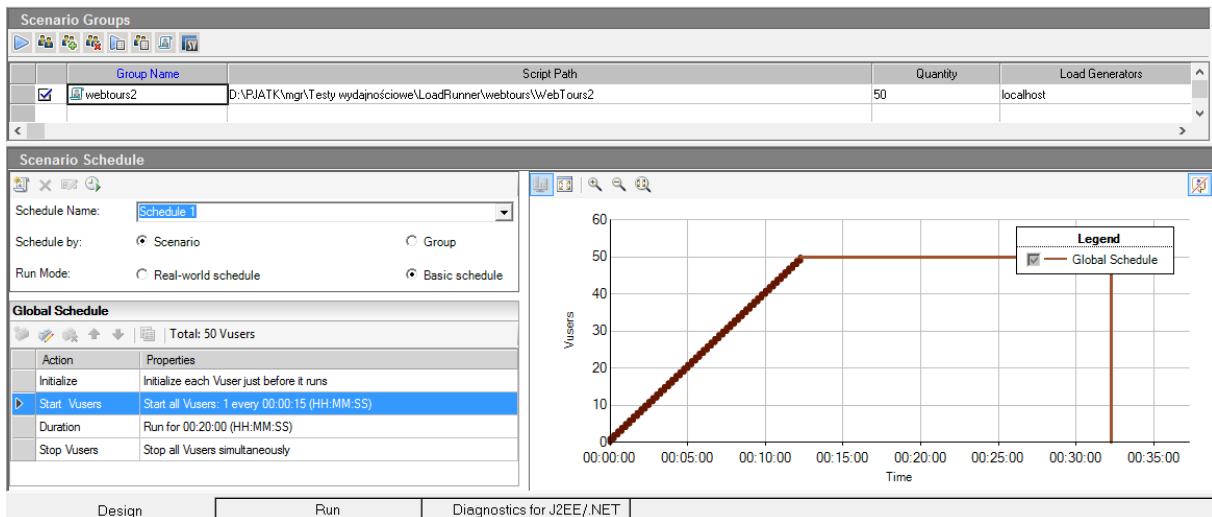
- Manual Scenario – scenariusz manualny, pozwala przetestować, jak wielu wirtualnych użytkowników jest w stanie obsłużyć dana aplikacja. Daje również możliwość zdefiniowania ich liczby oraz czasu w jakim są inicjowani. Rozdzielenie użytkowników może zostać określone procentowo lub za pomocą konkretnych wartości liczbowych.
- Goal-Oriented Scenario – scenariusz zorientowany na cel, służy do ustalenia, czy aplikacja może osiągnąć określony punkt docelowy. Jest on definiowany na podstawie konkretnego czynnika, np. czasu odpowiedzi transakcji lub liczby requestów/transakcji na sekundę. LoadRunner automatycznie tworzy scenariusz na podstawie sprecyzowanego celu.

W tym przypadku bardziej adekwatny był scenariusz manualny i taki też został wybrany.

Kolejnym krokiem było ustawienie modelu obciążenia. Składało się na niego kilka aspektów:

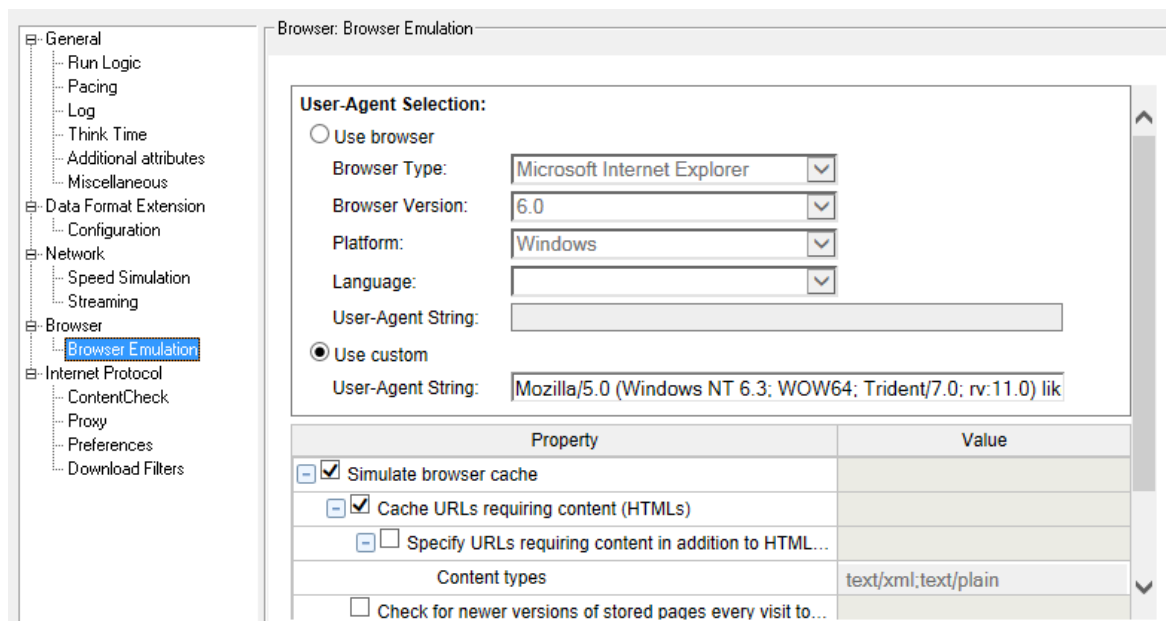
- Liczba wirtualnych użytkowników (50)
- Czas i sposób ich inicjalizacji, tzw. *ramp up* (1 użytkownik dodawany co 15 sekund)
- Czas trwania testu (20 minut + *ramp up*)

Po wprowadzaniu tych danych do *Controllera* wygenerowana została graficzna reprezentacja harmonogramu testów, widoczna na rysunku 59.



**Rysunek 59. Model obciążenia ustawiony w narzędziu LoadRunner**

W dalszej kolejności wykonano bardziej szczegółową konfigurację testu. Odbывается ona za pomocą wygodnego interfejsu graficznego, dającego dostęp do szerokiej gamy funkcji i parametrów. W ten sposób ustawione zostały docelowe think timy (z wielokrotnionie pięć razy), a także emulacja przeglądarki (ustawienia agenta, cache'u, statycznych zasobów itd.). Dodatkowo z tego poziomu można sprecyzować, jak wirtualny użytkownik ma się zachować w przypadku napotkania błędu, zdefiniować poziom logowania podczas testu, zasymulować przepustowość łącza, określić ustawienia proxy, timeoutów oraz wiele innych parametrów (rysunek 60).



**Rysunek 60. Emulacja przeglądarki w narzędziu LoadRunner**

Po uruchomieniu testu jego przebieg nadzorowano z pozycji panelu dostępnego w zakładce *Run* (rysunek 61). Okazała się ona bardzo czytelna i pomocna.

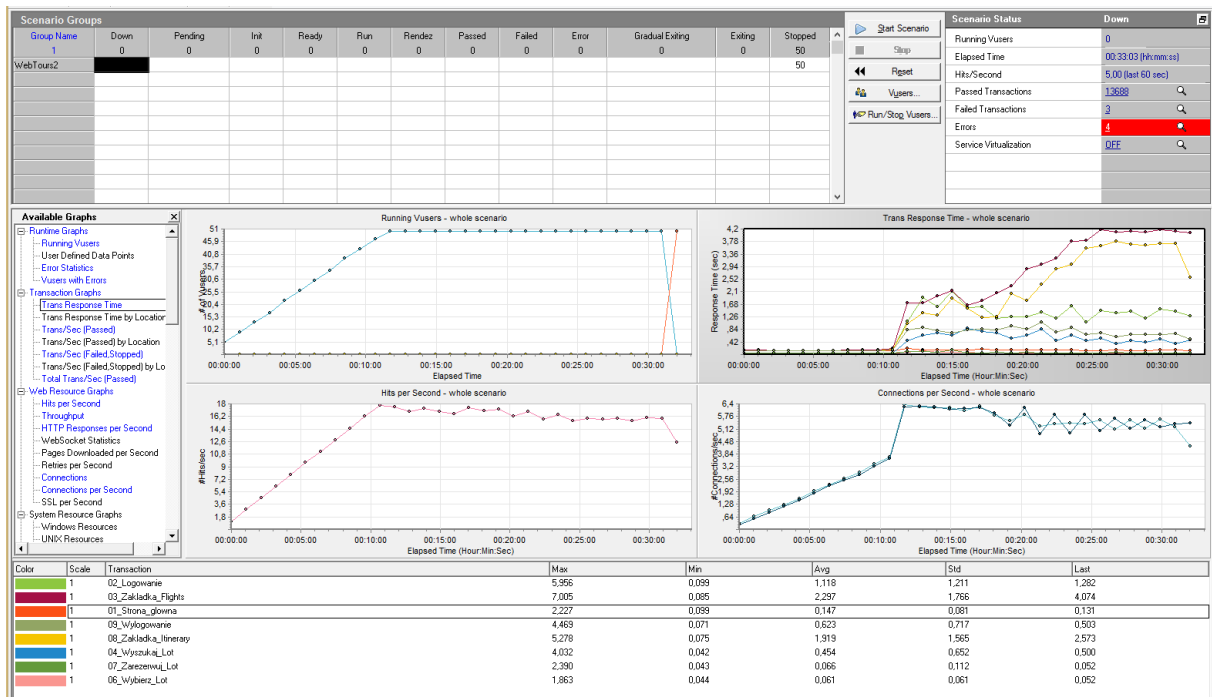
Najważniejszymi funkcjonalnościami dostępnymi na tym etapie testu bez wątplenia były te, związane z odczytem bieżących metryk, m. in.:

- Aktualna liczba aktywnych wirtualnych użytkowników
- Średnia liczba żądań na sekundę z ostatniej minuty (tzw. *Hits per second*)
- Pełna lista i całkowita liczba wykonanych transakcji (z podziałem na zakończone pomyślnie oraz negatywnie)
- Lista błędów napotkanych od początku testu

Aby podgląd był pełny, dane prezentowane były również w postaci szeregu różnego rodzaju wykresów, które pozwalały ocenić ogólną tendencję na osi czasu. Domyślnie wyświetlane to:

- Liczba użytkowników
- Średnie czasy odpowiedzi
- Liczba żądań na sekundę

Liczbę i rodzaje wykresów można definiować według własnych preferencji.



**Rysunek 61. Test uruchomiony w narzędziu LoadRunner**

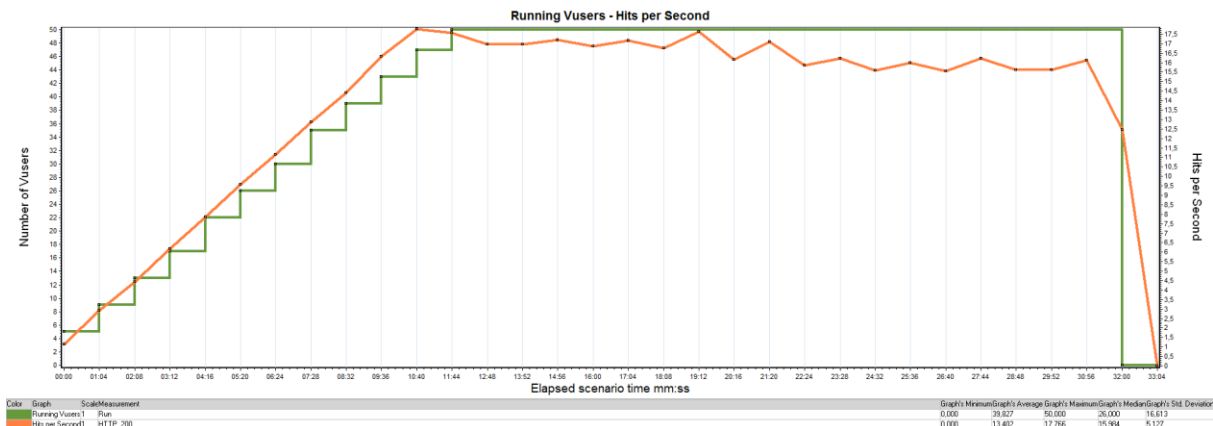
Na szczególną uwagę zasługuje również panel zarządzania wirtualnymi użytkownikami. Nawet w trakcie testu możemy zwiększyć ich liczbę, a także zarządzać tymi, którzy już biorą udział w teście (pojedynczo, lub całą grupę), jak również wstrzymać czasowo ich działanie lub zatrzymać całkowicie.

Po zakończonym teście zebrane wyniki analizowane były w kolejnym z modułów LoadRunniera – *Analysis*. Jego możliwości są ogromne, pozwala m. in. na:

- Filtrowanie wyników pod wieloma różnymi kryteriami
- Ustawienie SLA



- Tworzenie i konfigurowanie szeregu wykresów spośród kilkudziesięciu ich rodzajów (ustawienia granulacji, aspektów wizualnych, filtrowania, grupowania danych, itp.), jak np. na rysunku 62
- Tworzenie raportów
- Łączenie i porównywanie wyników z kilku odrębnych uruchomień testów



**Rysunek 62. Przykładowy wykres prezentujący liczbę żądań na sekundę podczas całego testu na tle liczby aktywnych wirtualnych użytkowników**

Podsumowanie i ocena poszczególnych aktywności, związanych z wykonaniem testu, przy użyciu LoadRunnera znajduje się w tabeli 15.

**Tabela 15. Ocena narzędzia LoadRunner pod kątem wykonania testu**

Cecha	Ocena	Argumentacja
<b>Zarządzanie strategią</b>	10	Wygodny interfejs graficzny dający kontrolę nad wszystkimi potrzebnymi w tym celu funkcjonalnościami.
<b>Zarządzanie wirtualnymi użytkownikami</b>	9	Prosta i intuicyjna sekcja pozwalająca na szczegółowe określenie zachowania VU podczas testu.
<b>Konfigurowalność</b>	10	Dostęp do praktycznie każdego parametru pracy testu.
<b>Emulacja przeglądarki</b>	10	Możliwość wyboru spośród wielu przeglądarek dostępnych na rynku, a także zdefiniowania własnego agenta. Dostęp do ustawień cache, ściągania statycznych zasobów itd.
<b>Zarządzanie testem w trakcie uruchomienia</b>	9	Możliwość zarządzania VU podczas trwania testu. Wykonalne jest również dodawanie kolejnych skryptów nawet już po uruchomieniu testu.  Ciekawa jest także opcja wstrzymania toku testu ( <i>pause scheduler</i> ), tym samym wydłużenia jego trwania.

<b>Podgląd metryk w czasie rzeczywistym</b>	10	Przejrzysty i konfigurowalny podgląd w postaci szeregu różnorodnych tabel i wykresów.
<b>Analiza wyników</b>	10	Oddzielny moduł stworzony tylko w tym celu, dający ogromne możliwości, a zarazem czytelny i intuicyjny w użytkowaniu.

### 5.1.2 Gatling

Wprawdzie utworzenie skryptów w Gatlingu miało wiele odniesień do LoadRunnera, jednak przygotowanie i uruchomienie testu okazało się już zupełnie inne. Moduł inicjujący działanie symulacji niestety nie posiada żadnego interfejsu graficznego i jest w całości konfigurowalny z poziomu kodu – identycznie jak w przypadku tworzenia skryptu.

Pierwszym krokiem w konfiguracji testu było wybranie typu scenariusza, spośród 2 dostępnych:

- *Open* – otwarty, który opiera się na wskaźniku przyrostu wirtualnych użytkowników. W tym modelu istnieje możliwość swobodnego ustawienia szybkości przybywania użytkowników na początku testu.
- *Closed* – zamknięty, oparty na utrzymywaniu stałej liczby jednocześnie pracujących użytkowników. Po jej osiągnięciu kolejny użytkownik może wejść do systemu, tylko i wyłącznie, jeśli poprzedni go opuści. Model bardziej odpowiedni dla aplikacji kolejujących użytkowników przed połączeniem (np. callcenter).

W tym przypadku wybrany został typ otwarty, który był analogiczny do testu wykonanego w LoadRunnerze.

Odpowiednie ustawienie tak podstawowych parametrów (jak np. czas testu, liczba iteracji, sposób inicjowania użytkowników, czy też zdefiniowanie, w jaki miały zostać odtworzone think timy) okazało się niesamowicie nieintuicyjne. Stworzenie modelu obciążenia referencyjnego do tego, które prezentują pozostałe narzędzia okupione było żmudnym studiowaniem dokumentacji, for branżowych, a także dużą ilością prób i błędów. Finalnie zakończyło się to sukcesem, czego wynikiem był kod 24.

#### Kod 24. Model obciążenia zdefiniowany w narzędziu Gatling

```
val users = scenario("Users").during(33 minutes){
  exec(Strona_glowna.stronaGlowna, Logowanie.logowanie,
  Zakladka_Flights.zakladkaFlights, Wyszukaj_Lot.wyszukajLot,
  Wybierz_Lot.wybierzLot, Zarezerwuj_Lot.zarezerwujLot,
  Zakladka_Itinerary.zakladkaItinerary, Wylogowanie.wylogowanie)
}
setUp(users.inject(rampUsers(50) over (750
seconds))).constantPauses.protocols(httpProtocol)
```

Składa się on z 2 części. Pierwszą z nich jest definicja scenariusza, w której określony został czas testu, a także kroki które były wykonywane w jego obrębie.

Druga część, to tzw. *setUp*, odpowiedzialny za konfigurację wirtualnych użytkowników (ich liczbę, a także czas i sposób ich inicjowania), jak również obsługę think timów (ustawiona została wartość *constantPauses*, co oznacza, że zostały odtworzone identycznie jak w kodzie – 5 sekund).

Konfiguracja oraz emulacja przeglądarki internetowej możliwa była do spersonalizowania jedynie w dość podstawowym stopniu:

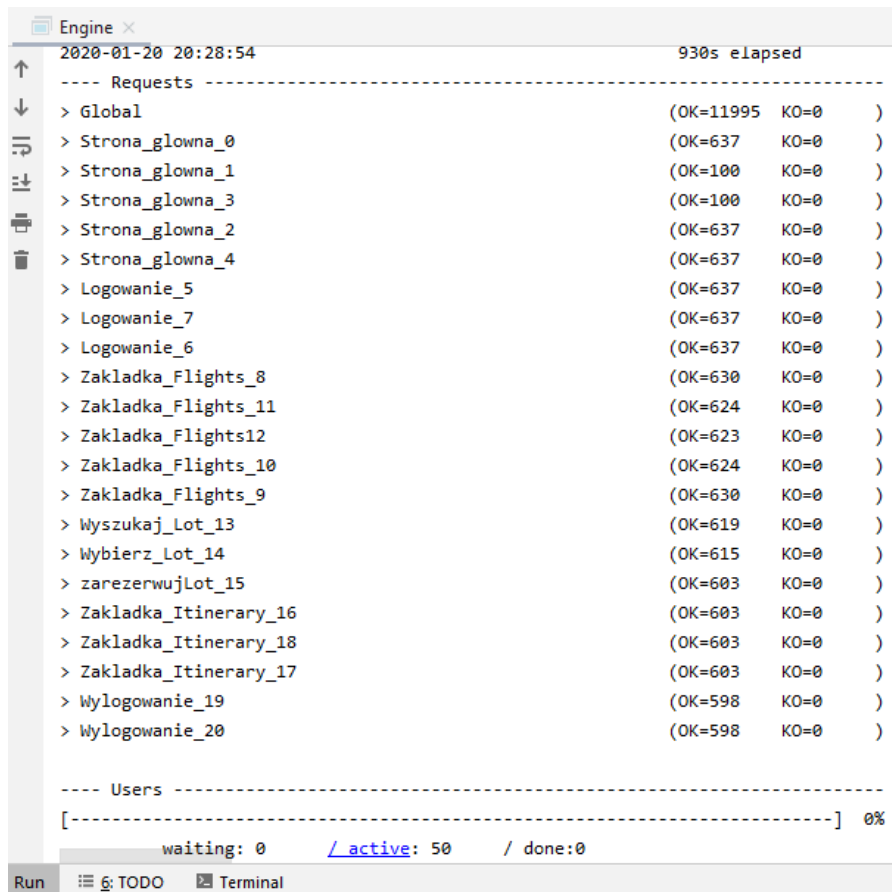
- Agenta przeglądarki należało zdefiniować ręcznie poprzez edycję nagłówka żądania, bądź pozostawić wartość z nagrania.
- Pobieranie statycznych zasobów można było nadzorować poprzez zdefiniowaną czarną i białą listę (kod 25).
- Cache i pliki cookie obsługiwane były automatycznie (osobno dla każdego VU), jedyną opcją ingerencji w ten mechanizm mogło być ich całkowite wyczyszczenie w wybranym momencie (dopisane odpowiedniego kodu w danym kroku scenariusza). Istniała również możliwość dodania pliku cookie z pozycji kodu, w przypadku pojawienia się takiej potrzeby.

### Kod 25. Konfiguracja atrybutów protokołu w narzędziu Gatling

```
val httpProtocol = http
  .baseUrl("http://127.0.0.1:1080")
  .inferHtmlResources(BlackList(""".*\.js""", """.*\.css""",
    """.*\.gif""", """.*\.jpeg""", """.*\.jpg""", """.*\.ico""",
    """.*\.woff""", """.*\.(\t|o)tf""", """.*\.png""", """.*\.txt"""),
  WhiteList())
  .acceptHeader("text/html, application/xhtml+xml, */*")
  .acceptEncodingHeader("gzip, deflate")
  .acceptLanguageHeader("pl-PL")
  .doNotTrackHeader("1")
  .userAgentHeader("Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0;
rv:11.0) like Gecko")
```

Po uruchomieniu testu możliwe było śledzenie jego przebiegu na podstawie logów wyświetlanych na bieżąco w oknie konsoli. Zawierały one bardzo podstawowe informacje, takie jak:

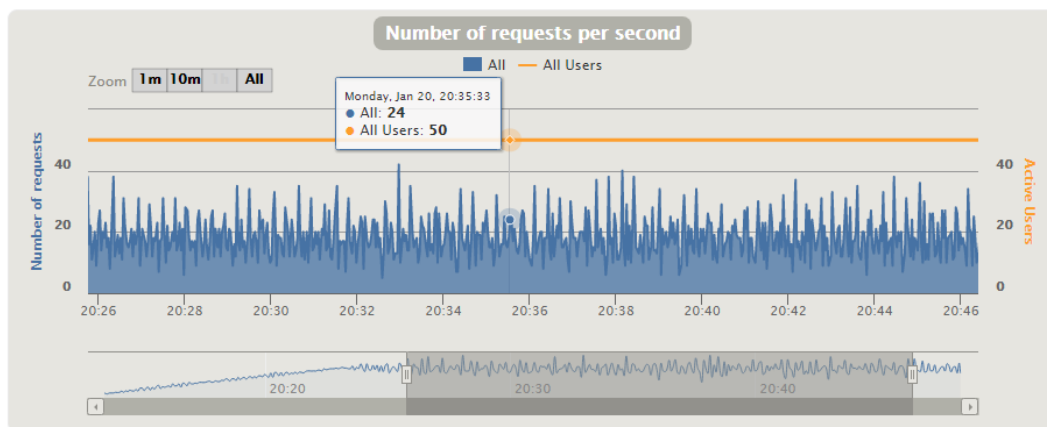
- Czas testu, który na daną chwilę upłynął
- Liczba aktywnych użytkowników wirtualnych
- Lista wysyłanych żądań wraz z liczbą pozytywnych i błędnych próbek (rysunek 63).



Rysunek 63. Test uruchomiony w narzędziu Gatling

Gdy test dobiegł końca automatycznie wygenerowany został raport w postaci pliku HTML. Jego zawartość to tabela podsumowująca aktywności każdego z żądań, kilka wykresów oraz lista błędów jakie pojawiły się w trakcie uruchomienia. Dostępne były również szczegółowe statystyki dla każdego z żądań z osobna.

Teoretycznie było to wystarczające do ogólnej oceny wydajności aplikacji, jednak brakowało jakichkolwiek narzędzi umożliwiających filtrowanie danych bądź konfigurację wykresów. Możliwe było jedynie zawężenie obszaru wykresu za pomocą JavaScriptu, z którego korzystał plik raportu (rysunek 64).



Rysunek 64. Wykres requestów na sekundę na tle liczby użytkowników zawężony do czasu pełnego obciążenia

Tabela 16 przedstawia zestawienie i ocenę aspektów dotyczących wykonania testu przy pomocy Gatlinga.

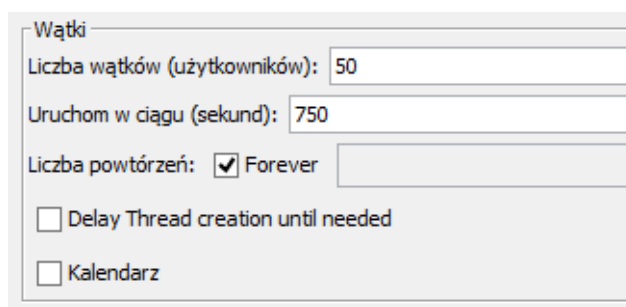
**Tabela 16. Ocena narzędzia Gatling pod kątem wykonania testu**

Cecha	Ocena	Argumentacja
Zarządzanie strategią	5	Możliwe z pozycji kodu, jednak bardzo skomplikowane i nieintuicyjne.
Zarządzanie wirtualnymi użytkownikami	5	Jw.
Konfigurowalność	6	Wykonalna, jednak wymagająca znajomości składni Gatlinga. Dość podstawowe możliwości manipulacji.
Emulacja przeglądarki	4	Dostępna w bardzo ograniczonym zakresie.
Zarządzanie testem w trakcie uruchomienia	3	W trakcie testu nie ma możliwości ingerencji w jego przebieg. Można go jedynie zatrzymać.
Podgląd metryk w czasie rzeczywistym	4	Widoczne jedynie bardzo podstawowe informacje w postaci logu wyświetlanego w konsoli.
Analiza wyników	5	Wyniki dostępne w formie czytelnego raportu. Praktycznie brak możliwości dalszej obróbki wyników.

### 5.1.3 JMeter

W przypadku JMetera ustawienie parametrów testu odbyło się podobnie jak w przypadku tworzenia skryptu – za pomocą graficznych kontrolerów.

Standardowa *Grupa Wątków* dostępna w podstawowej wersji narzędzia, daje możliwość utworzenia tylko jednego typu scenariusza, takiego jak na rysunku 65. Do dyspozycji są również wtyczki rozszerzające tę funkcjonalność, jednak w tym przypadku nie było konieczności ich instalowania, ponieważ dostępne rozwiązanie było wystarczające. W opcjach tego elementu definiowany jest również model obciążenia.

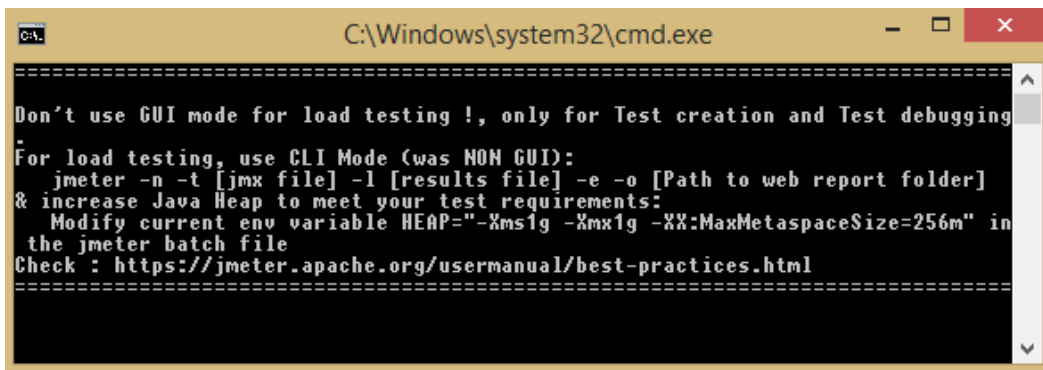


**Rysunek 65. Kształtowanie modelu obciążenia na poziomie Grupy Wątków w JMeterze**

Zarządzanie think timami obsługiwane jest poprzez umieszczenie w tych elementach odwołania do zmiennej, zamiast konkretnej wartości liczbowej. Wówczas w przypadku konieczności zmiany czasu paazy, wystarczy zrobić to w jednym miejscu, bez potrzeby uaktualniania każdego z obiektów z osobna.

Dalsza konfiguracja odbyła się przy pomocy kolejnych kontrolerów. Pliki cookie zarządzane były z poziomu tzw. *Cookie Managera*, gdzie istniała możliwość ustawienia ich polityki oraz czyszczenia z każdą iteracją. Analogicznie, do modelowania Cache'u posłużył tzw. *Cache Manager*. Agent przeglądarki, podobnie jak w Gatlingu wpisywany był tekstowo do odpowiedniego nagłówka żądania.

Samo uruchomienie testu wprawdzie można było wykonać z poziomu interfejsu graficznego, jednak jest to wysoce odradzane przez twórcę narzędzia. Już przy jego uruchomieniu wyświetlany jest komunikat widoczny na rysunku 66, który przed tym przestrzega. Z tego też powodu test został uruchomiony z wiersza poleceń przy pomocy sugerowanej komendy.



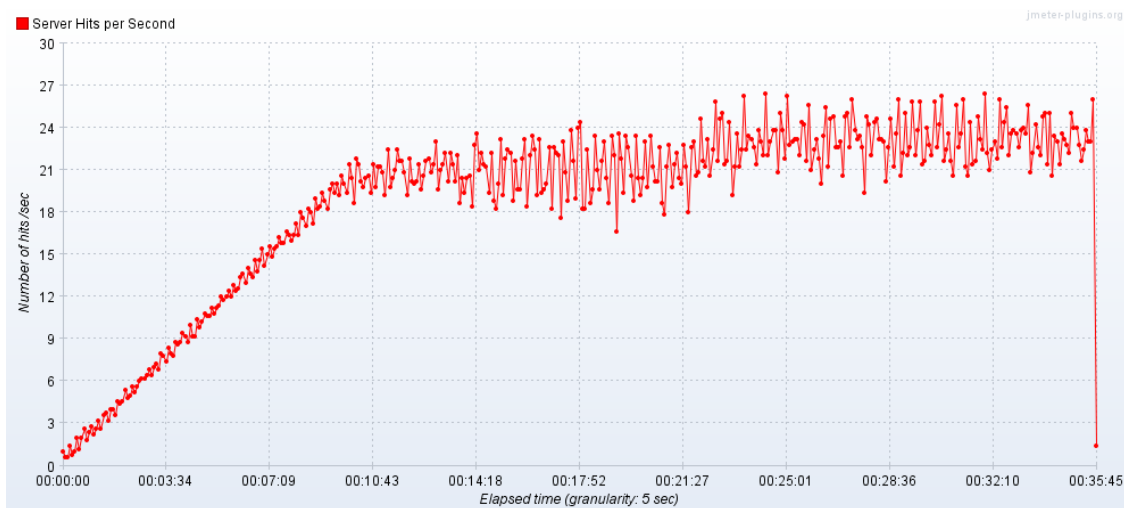
```
C:\Windows\system32\cmd.exe

=====
Don't use GUI mode for load testing !, only for Test creation and Test debugging
=====
For load testing, use CLI Mode (was NON GUI):
  jmeter -n -t [jmx file] -l [results file] -e -o [Path to web report folder]
& increase Java Heap to meet your test requirements:
  Modify current env variable HEAP="-Xms1g -Xmx1g -XX:MaxMetaspaceSize=256m" in
the jmeter batch file
Check : https://jmeter.apache.org/usermanual/best-practices.html
=====
```

**Rysunek 66. Komunikat widoczny zaraz po uruchomieniu JMetera**

Podczas testu, w oknie poleceń wypisywany był cyklicznie aktualny stan testu: liczba aktywnych użytkowników, liczba błędów, a także średni i maksymalny czas z ostatniej próbki.

Efektem końcowym testu był plik wynikowy w postaci mało czytelnego dokumentu csv. Na szczęście był on w pełni kompatybilny ze słuchaczami dostępnymi w interfejsie graficznym i po załadowaniu go do jednego z nich w prosty sposób można było otrzymać potrzebną w danej sytuacji tabelę bądź wykres. Rysunek 67 przedstawia graficzną prezentację żądań na sekundę stworzoną przy pomocy słuchacza o nazwie *jp@gc - Hits per Second*.



**Rysunek 67. Wykres przedstawiający liczbę requestów na sekundę podczas testu aplikacji Web Tours Sample Application**

Podsumowanie i ocena poszczególnych aktywności, związanych z wykonaniem testu, przy użyciu narzędzia jakim jest JMeter, została zaprezentowana w tabeli 17.

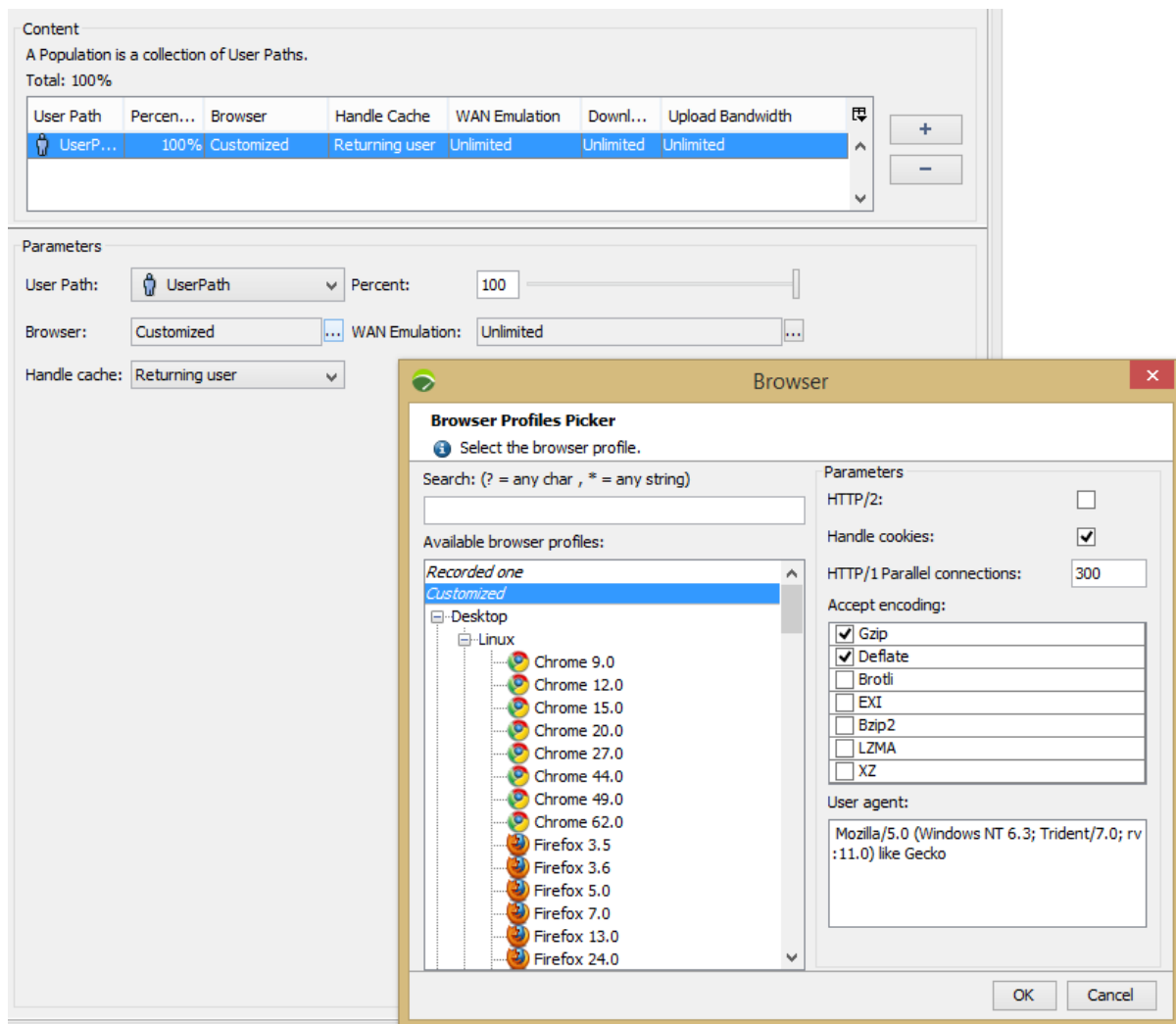
**Tabela 17. Ocena narzędzia JMeter pod kątem wykonania testu**

Cecha	Ocena	Argumentacja
Zarządzanie strategią	7	Wykonywane z poziomu interfejsu graficznego, jednak zobrazowane jedynie wartościami liczbowymi.
Zarządzanie wirtualnymi użytkownikami	6	Jw.
Konfigurowalność	8	Możliwość zarządzania wieloma parametrami z poziomu wygodnych i intuicyjnych kontrolerów.
Emulacja przeglądarki	8	Dosyć szerokie spektrum działania. Dostęp do ustawień plików cookie, cache, ściągania statycznych zasobów itp.
Zarządzanie testem w trakcie uruchomienia	3	W trakcie testu nie ma możliwości ingerencji w jego przebieg. Można go jedynie zatrzymać.
Podgląd metryk w czasie rzeczywistym	4	Wyświetlane są jedynie podstawowe informacje w niezbyt wygodnej formie (okno wiersza poleceń).
Analiza wyników	8	Plik wynikowy testu jest kompatybilny z większością słuchaczy i daje duże możliwości prezentacji danych w najróżniejszych formach. Ich konfigurowalność jest zależna od konkretnego kontrolera, do którego zaczytana zostaną dane.

#### 5.1.4 NeoLoad

Pierwszym krokiem do wykonania testu w narzędziu NeoLoad było utworzenie tzw. Populacji (rysunek 68). Służy ona do wybrania jednego z skryptów (tzw. *UserPath*) i ustawienia dla niego parametrów jakie będą używane podczas jego uruchomienia, czyli:

- Emulacja przeglądarki –do wyboru dostępnych było kilkadziesiąt ich rodzajów dla różnych systemów operacyjnych oraz platform (Desktop, Mobile, Tablet)
- Obsługa cache
- Możliwe było również ograniczenie przepustowości sieci poprzez manualna definicję bądź wybraniec jednego z predefiniowanych profili (symulacja Wifi, Ethernet, GPRS, 5G itd.)

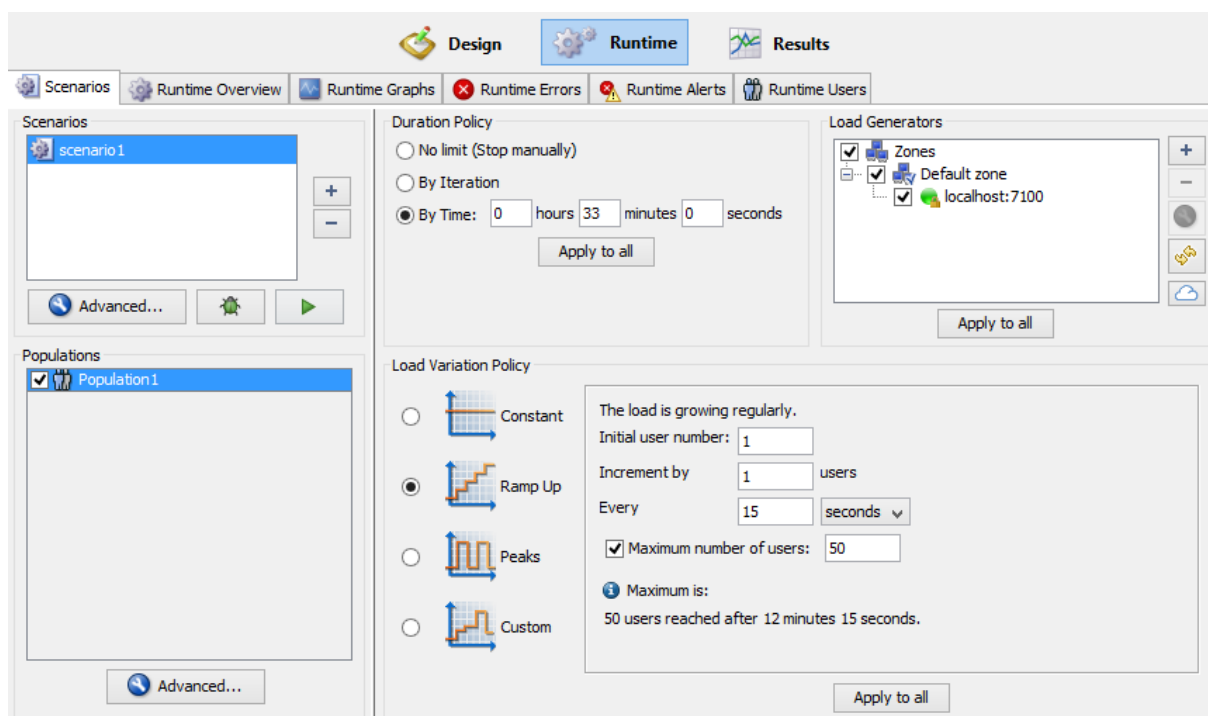


**Rysunek 68. Konfiguracja Populacji w narzędziu NeoLoad**

Następnie na podstawie skonfigurowanej już Populacji stworzony został scenariusz. Spośród dostępnych typów wybrano *Ramp Up*, który był najbardziej referencyjny w stosunku do testów wykonanych w pozostałych narzędziach. W tym panelu usawiony został również model obciążenia. Wszystkie jego aspekty opisane były w jasny i intuicyjny sposób, co widać na rysunku 69.

Ciekawą opcją, zasługującą na uwagę był scenariusz typu *Custom*. Pozwalał on na stworzenie modelu obciążenia w formie graficznej – poprzez narysowanie wykresu przyrostu użytkowników bezpośrednio na osi czasu. Daje to bardzo duże możliwości przy minimalizacji potrzebnego nakładu pracy.



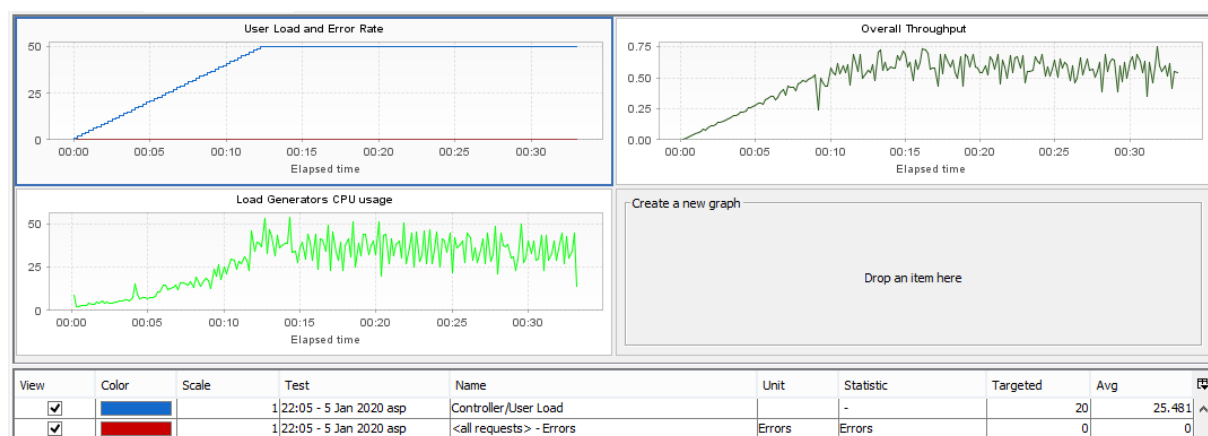


**Rysunek 69. Przygotowanie scenariusza oraz modelu obciążenia w narzędziu NeoLoad**

Podobnie jak w przypadku LoadRunnera, w trakcie pracy testu możliwy jest podgląd wykonywanych przez niego akcji oraz szereg różnego rodzaju innych statystyk (rysunek 70). Domyślnie wyświetlane wykresy to:

- Liczba użytkowników oraz liczba błędów
- Ogólna przepustowość
- Wykorzystanie procesora maszyny, na której uruchomiony został test

Dostępnych jest także mnóstwo innego rodzaju wykresów przedstawiających informacje dotyczące nie tylko ruchu generowanego przez narzędzie, ale także fizycznych zasobów generatora, statystyk karty sieciowej itp.



**Rysunek 70. Test uruchomiony w narzędziu NeoLoad**

NeoLoad pozwalał również na kontrolę nad wirtualnymi użytkownikami w trakcie testu – możliwe było dodanie oraz zatrzymanie części z nich. Ciekawą funkcjonalnością był także pełen podgląd ścieżki jaką przechodzi wybrany z listy użytkownik.

Po zakończeniu testu udostępniona została zakładka *Results*, w której automatycznie wygenerowany był obszerny raport. Oprócz karty z poglądowym podsumowaniem całego testu, zostały stworzone także takie, które zawierały informacje dotyczące poszczególnych jego aspektów, np.: populacji, scenariusza, serwerów, wszystkich transakcji, żądań, błędów jw. W oddzielnej sekcji znalazły się wykresy dla wielu przydatnych metryk, np. czasów odpowiedzi, requestów na sekundę, przepustowości itp.

Zakładka *Results* daje również znacznie szersze możliwości, np.:

- Filtrowanie danych w istniejących tabelach i wykresach
- Tworzenie własnych
- Konfiguracja SLA
- Generowanie raportów
- Porównywania wyników z poszczególnych uruchomień

Tabela 18 przedstawia zestawienie i ocenę parametrów dotyczących przygotowania i uruchomienia testu przy pomocy narzędzia, jakim jest NeoLoad.

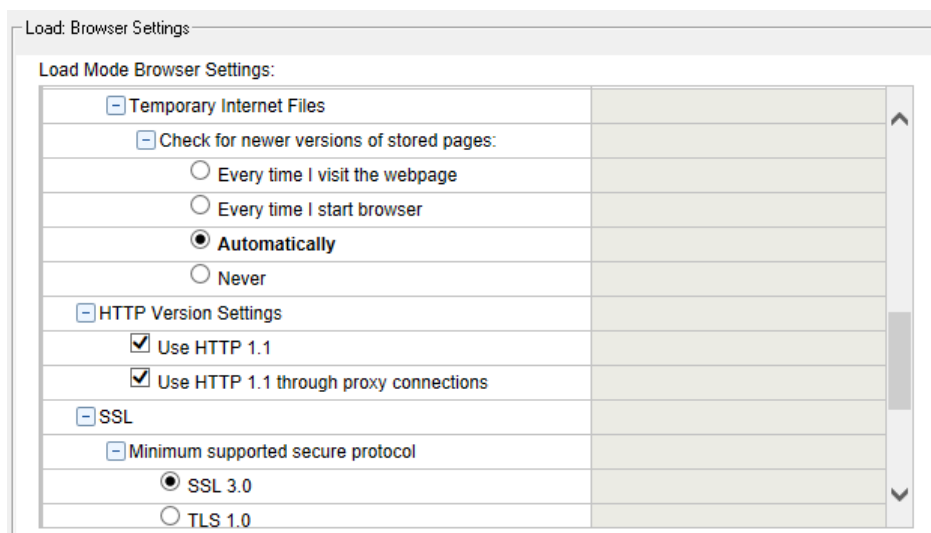
**Tabela 18. Ocena narzędzia NeoLoad pod kątem wykonania testu**

Cecha	Ocena	Argumentacja
<b>Zarządzanie strategią</b>	10	Przejrzysty i intuicyjny interfejs graficzny, dający zarazem duże pole do personalizacji (szczególnie tryb <i>Custom</i> ).
<b>Zarządzanie wirtualnymi użytkownikami</b>	10	Prosty i wygodny mechanizm w pełni wyczerpujący tę funkcjonalność.
<b>Konfigurowalność</b>	10	Dostęp do bardzo wielu aspektów zarówno samego scenariusza jak i symulacji komponentów pobocznych.
<b>Emulacja przeglądarki</b>	10	Dostęp do kilkudziesięciu rodzajów przeglądarek dla różnych systemów operacyjnych oraz platform (Desktop, Mobile, Tablet). Możliwość dalszego konfigurowania każdej z nich.
<b>Zarządzanie testem w trakcie uruchomienia</b>	9	Zarządzanie VU podczas trwania testu. Pełny podgląd do ścieżki, którą przechodzą poszczególni z nich.
<b>Podgląd metryk w czasie rzeczywistym</b>	10	Przejrzysty i konfigurowalny podgląd w postaci szeregu różnorodnych tabel i wykresów.
<b>Analiza wyników</b>	9	Oddzielna zakładka służąca do kompleksowej analizy i wizualizacji wyników. Mnóstwo opcji konfigurowania i personalizacji.

### 5.1.5 TruClient

W przypadku TruClienta uruchomienie testu odbywało się w identyczny sposób jak to miało miejsce w protokole Web w podrozdziale 5.1.1. Posłużył do tego moduł *Controller* i identycznie skonfigurowany scenariusz oraz model obciążenia.

Jedyną różnicą były możliwości konfiguracyjne, które okazały się nieco uboższe, głównie ze względu na specyfikę skryptu GUI. Dostępne były jedynie 3 przeglądarki internetowe, analogicznie jak w przypadku tworzenia skryptu. Dostępna była możliwość konfiguracji ich podstawowych parametrów, jednak również w mniejszym stopniu niż w protokole Web (rysunek 71).



**Rysunek 71. Emulacja przeglądarki w narzędziu TruClient**

Podsumowanie i ocena wykonania testu przy użyciu TruClienta została zaprezentowana w tabeli 19.

**Tabela 19. Ocena narzędzia TruClient pod kątem wykonania testu**

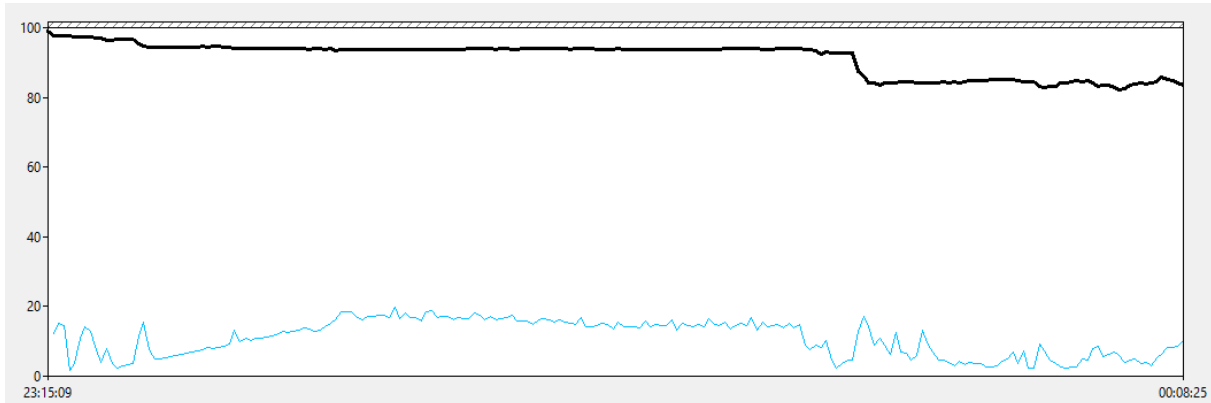
Cecha	Ocena	Argumentacja
Zarządzanie strategią	10	Identycznie jak w przypadku protokołu Web w LoadRunnerze
Zarządzanie wirtualnymi użytkownikami	9	Jw.
Konfigurowalność	8	Nieco uboższa niż w przypadku protokołu Web w LoadRunnerze
Emulacja przeglądarki	8	Jw.
Zarządzanie testem w trakcie uruchomienia	9	Identycznie jak w przypadku protokołu Web w LoadRunnerze
Podgląd metryk w czasie rzeczywistym	10	Jw.
Analiza wyników	10	Jw.

### 5.1.6 Obciążenie zasobów fizycznych generatora przez każde z narzędzi podczas testów aplikacji Web Tours Sample Application

Bardzo ważnym aspektem testów wydajnościowych stanowi również obciążenie maszyny, z której generowany jest ruch przez narzędzie za to odpowiedzialne. Przeciążony generator fałszuje wyniki testu, ponieważ wykazywane przez niego opóźnienia nie są spowodowane problemami wydajnościowymi badanej aplikacji, tylko jego samego. Może to też doprowadzić do zawieszenia narzędzia i utraty wszystkich wyników z danego uruchomienia.

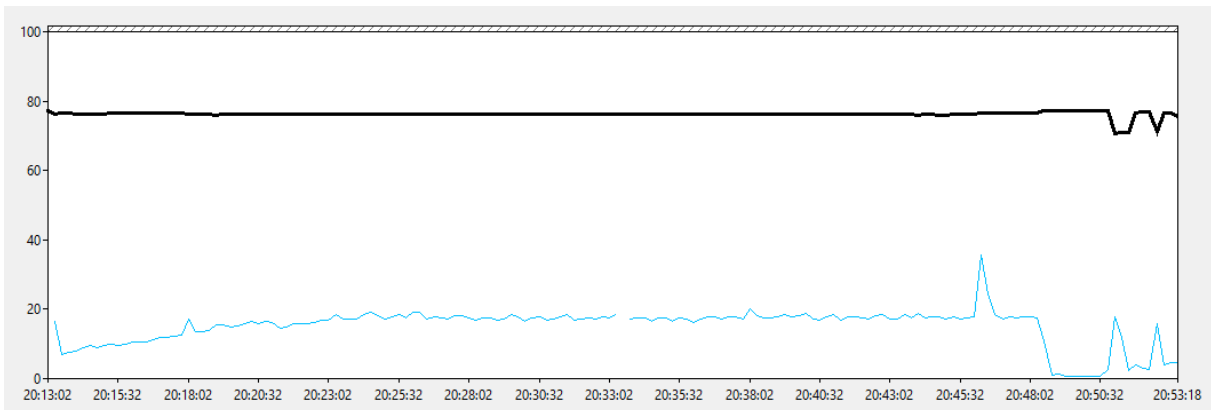
Charakterystyka obciążenia maszyny przez poszczególne narzędzia została przedstawiona na rysunkach 72-76 (ciemna linia oznacza dostępną pamięć RAM, jasna – procent wykorzystania CPU):

- LoadRunner



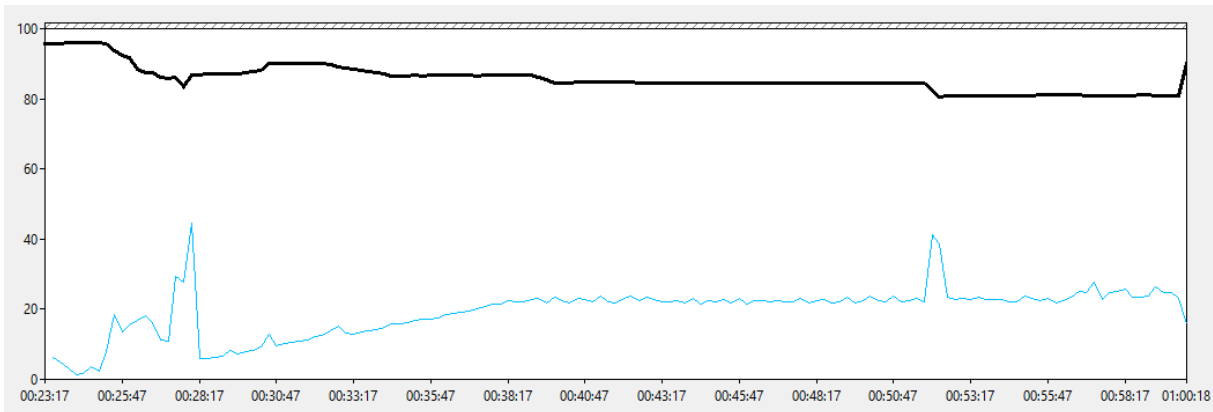
**Rysunek 72. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia LoadRunner**

- Gatling



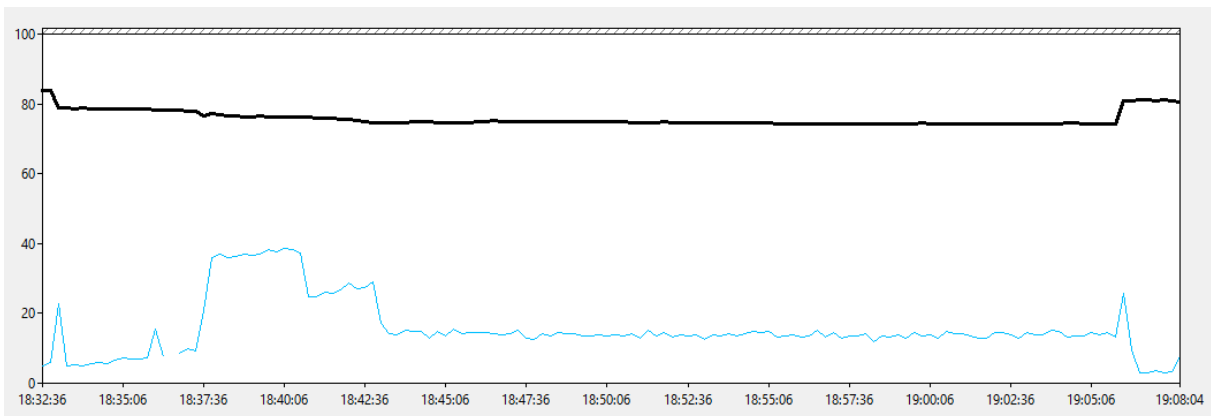
**Rysunek 73. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia Gatling**

- JMeter



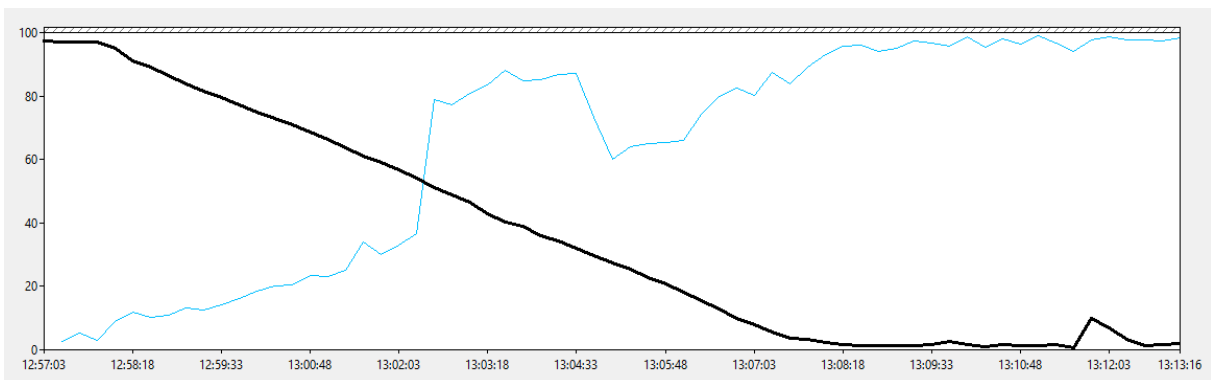
**Rysunek 74. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia JMeter**

- NeoLoad



**Rysunek 75. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia NeoLoad**

- TruClient

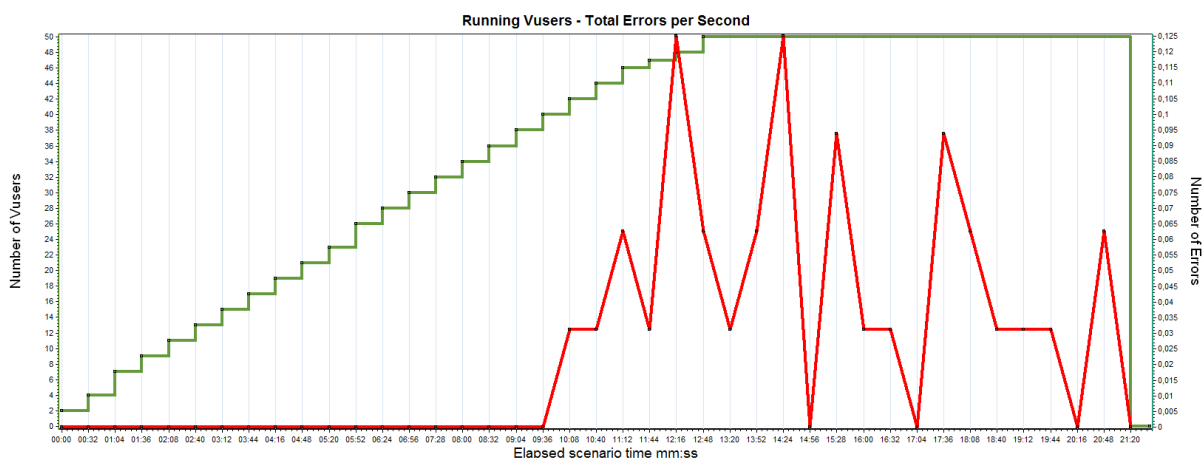


**Rysunek 76. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia TruClient**

Podczas testów przy użyciu TruClienta, po załadowaniu 40 z 50 wirtualnych użytkowników, w aplikacji zaczęły pojawiać się błędy (rysunek 77). W tym samym czasie wykorzystanie procesora

sięgnęło blisko 100%, jak również dostępna pamięć RAM spadła prawie do zera, co można zaobserwować na rysunku 76.

Z uwagi na dużą ilość błędów funkcjonalnych i brak stabilności aplikacji spowodowany przeciążeniem maszyny, test został zatrzymany przed jego planowanym zakończeniem.



**Rysunek 77. Liczba błędów na sekundę na tle liczby użytkowników podczas testu przy użyciu narzędzia TruClient**

Podwyższone wykorzystanie zasobów maszyny było oczekiwane, ponieważ każdy wirtualny użytkownik podnosił w tle swoją własną instancję przeglądarki internetowej. Niemniej jednak trudna do przewidzenia była skala tego zjawiska, a jak się okazało była ona na tyle duża, że uniemożliwiła wykonanie stabilnego testu.

Tabela 20 przedstawia podsumowanie obciążenia maszyny dla każdego testu. Jest to składowa ogólnej oceny w podrozdziale 5.4.

**Tabela 20. Średnie zużycie zasobów fizycznych generatora podczas testów aplikacji ASP.NET Web Forms**

	LoadRunner	Gatling	JMeter	NeoLoad
<b>Wykorzystanie CPU [%]</b>	14	16	20	17
<b>Dostępna pamięć [MB]</b>	9 406	7 623	8 474	7 525
<b>Zadeklarowane bajty w użyciu – RAM [%]</b>	19	38	33	39
<b>Bajty odczytu dysku / s (średnia)</b>	172 659	37 633	356 305	101 775
<b>Bajty zapisu dysku / s (średnia)</b>	58 756	65 564	119 772	95 410
<b>Długość kolejki dysku (średnia)</b>	0,023	0,031	0,071	0,000

Dowodem na to, że poszczególne narzędzia wygenerowały porównywalny ruch jest Tabela 21, która prezentuje porównanie wykonanych przez nie akcji. Z uwagi na fakt, że każde z narzędzi generuje ruch w nieco inny sposób, za pomocą własnego, niepowtarzalnego mechanizmu, otrzymane wyniki nieznacznie się od siebie różnią.

**Tabela 21. Ruch wygenerowany podczas testów aplikacji Web Tours Sample Application**

	LoadRunner	Gatling	JMeter	NeoLoad
<b>Requesty na sekundę</b>	13,9	15,9	18,3	14,6
<b>Ogólny średni czas odpowiedzi [s]</b>	0,8	0,8	0,8	0,8
<b>Odsetek błędów</b>	0,002	0,001	0,04	0,3
<b>Liczba aktywnych wątków (użytkowników)</b>	50	50	50	50

## 5.2 Przygotowanie i przeprowadzenie testów dla aplikacji ASP.NET Web Forms

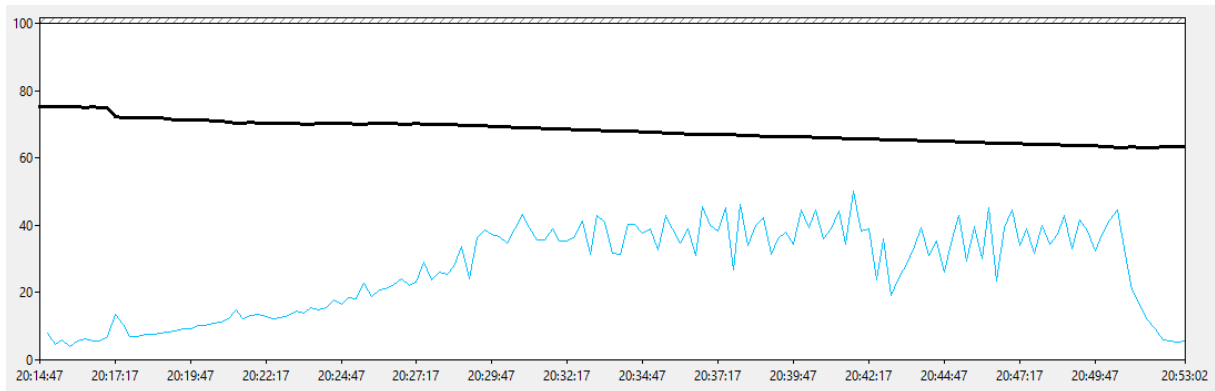
Podczas testów aplikacji ASP.NET Web Forms również zbierane były statystyki obciążenia fizycznej maszyny. Ponownie dokonano tego z pomocą narzędzia Performance Monitor.

Aby zapewnić każdemu z testów identyczne warunki początkowe, pomiędzy poszczególnymi uruchomieniami, z bazy danych usuwane były rekordy stworzone podczas poprzedniego uruchomienia. Wykonywane było to przez uruchomienie odpowiednich instrukcji SQL, bezpośrednio na bazach danych:

- DELETE FROM AspNetUsers; - usunięcie wszystkich użytkowników zarejestrowanych w aplikacji
- DELETE FROM Students where StudentID>3; - usunięcie studentów dodanych do systemu podczas testu, przy pozostawieniu 3 przykładowych rekordów dostępnych wejściowo w tabeli

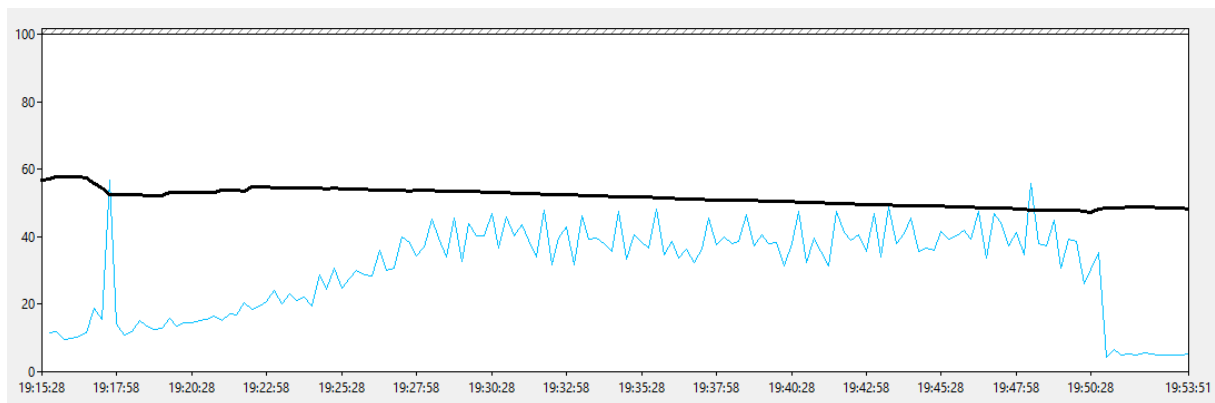
Analogicznie, jak w przypadku testów aplikacji Web Tours Sample Application, stworzone zostały wykresy przedstawiające charakterystykę obciążenia maszyny generującej ruch przez poszczególne narzędzia (ciemna linia oznacza dostępną pamięć RAM, jasna – procent wykorzystania CPU). Można je zaobserwować na obrazkach 78-82:

- LoadRunner



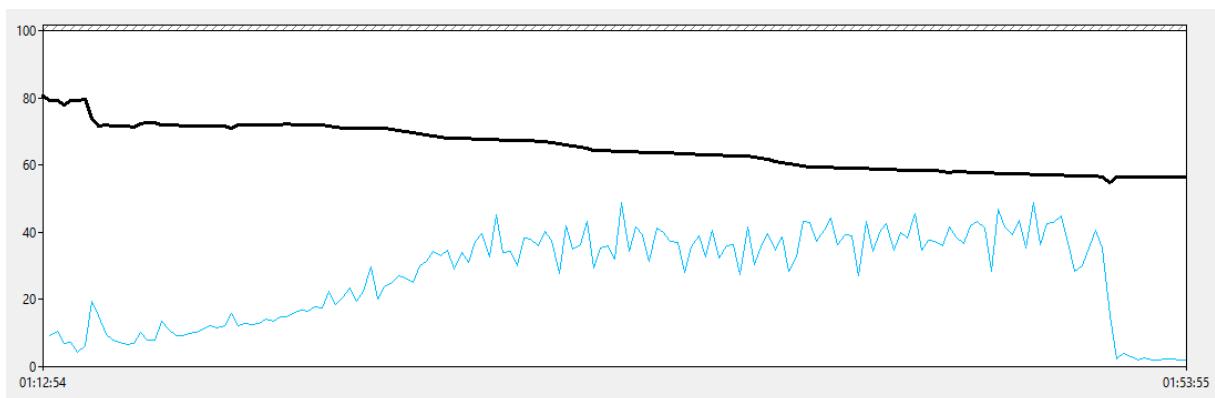
**Rysunek 78. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia LoadRunner**

- Gatling



**Rysunek 79. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia Gatling**

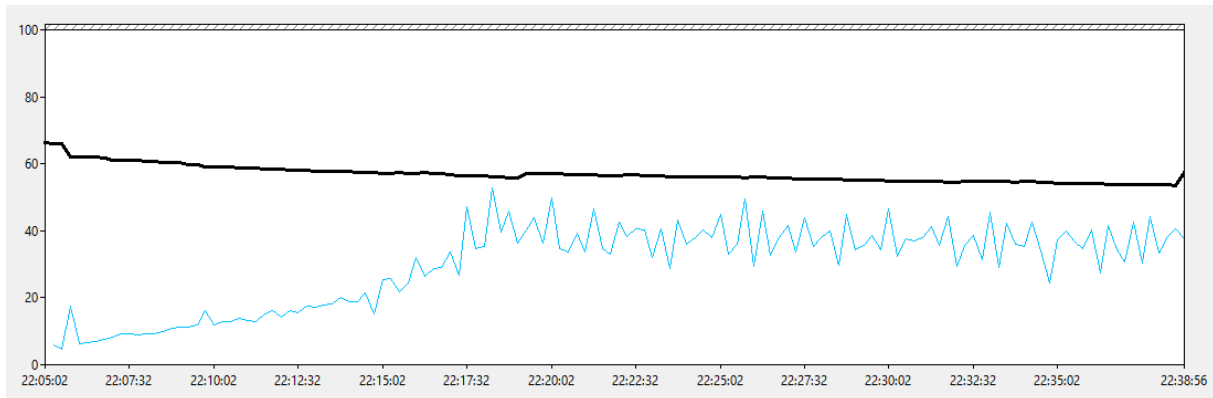
- JMeter



**Rysunek 80. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia JMeter**

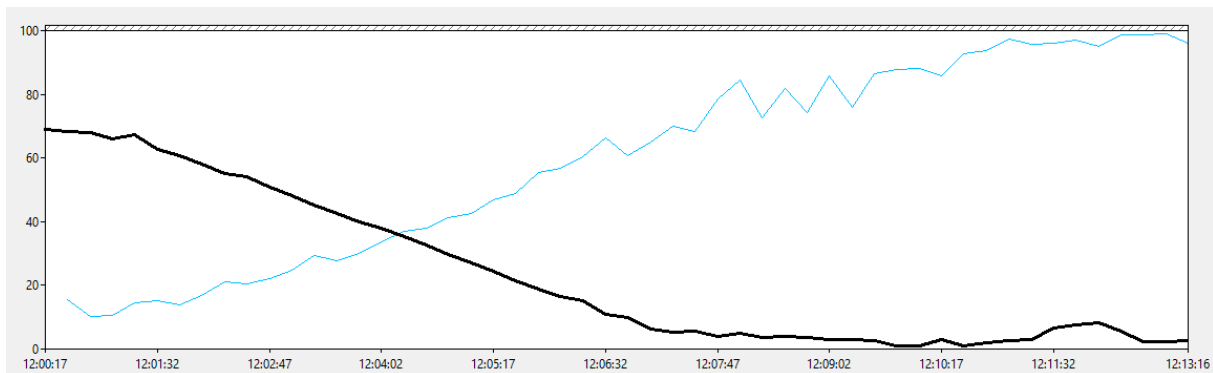


- NeoLoad



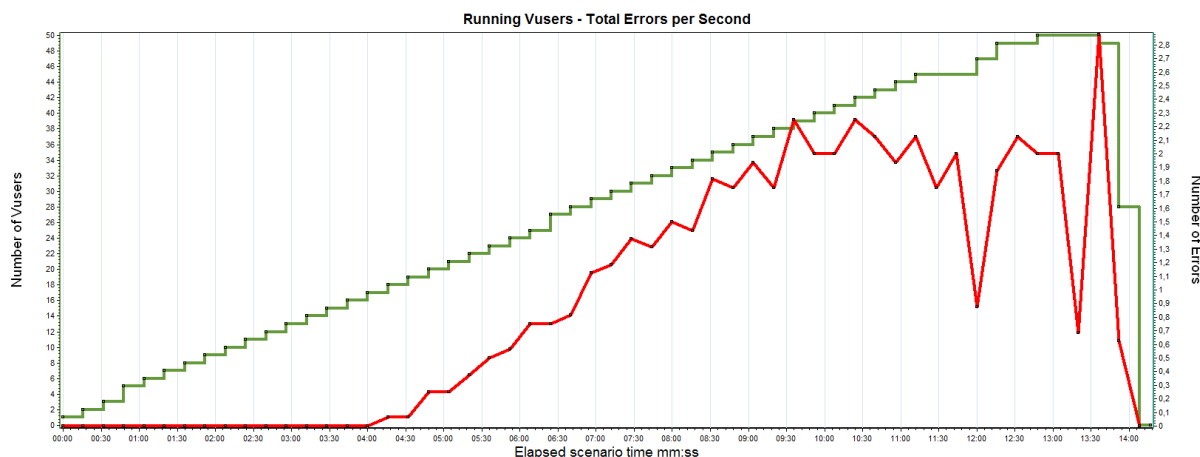
**Rysunek 81. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia NeoLoad**

- TruClient



**Rysunek 82. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia TruClient**

Podczas testu wykonanego przy pomocy narzędzia TruClient ponownie pojawiły się komplikacje. Po załadowaniu około 20 wirtualnych użytkowników obciążenie fizycznych zasobów osiągnęło już nader wysoki poziom (w szczytach do 100% wykorzystania CPU) i w aplikacji zaczęły pojawiać się błędy (rysunek 83). Po osiągnięciu pełnej liczby VU, użycie procesora sięgało stale blisko 100%, a także dostępna pamięć RAM oscylowała w okolicach pełnego wykorzystania (co widać na rysunku 82). W następstwie tych faktów, błędy zaczęły pojawiać się coraz częściej, a czasy odpowiedzi rosnąć. Test został zatem przerwany.



**Rysunek 83. Liczba błędów na sekundę na tle liczby użytkowników podczas testu przy użyciu narzędzia TruClient**

Tabela 22 przedstawia średnie zużycie zasobów fizycznych maszyny podczas każdego z testów.

**Tabela 22. Średnie zużycie zasobów fizycznych generatora podczas testów aplikacji ASP.NET Web Forms**

	LoadRunner	Gatling	JMeter	NeoLoad
Wykorzystanie CPU [%]	29	33	30	30
Dostępna pamięć [MB]	6 806	5 221	6 545	5 664
Zadeklarowane bajty w użyciu – RAM [%]	42	63	53	55
Bajty odczytu dysku / s (średnia)	71 292	408 940	203 479	55 245
Bajty zapisu dysku / s (średnia)	89 905	166 395	63 534	181 654
Długość kolejki dysku (średnia)	0,039	0,078	0,016	0,008

Dowodem na to, że każda z aplikacji wygenerowała porównywalny ruch jest Tabela 23, która przedstawia podsumowanie wykonanych akcji.

**Tabela 23. Ruch wygenerowany podczas testów aplikacji ASP.NET Web Forms**

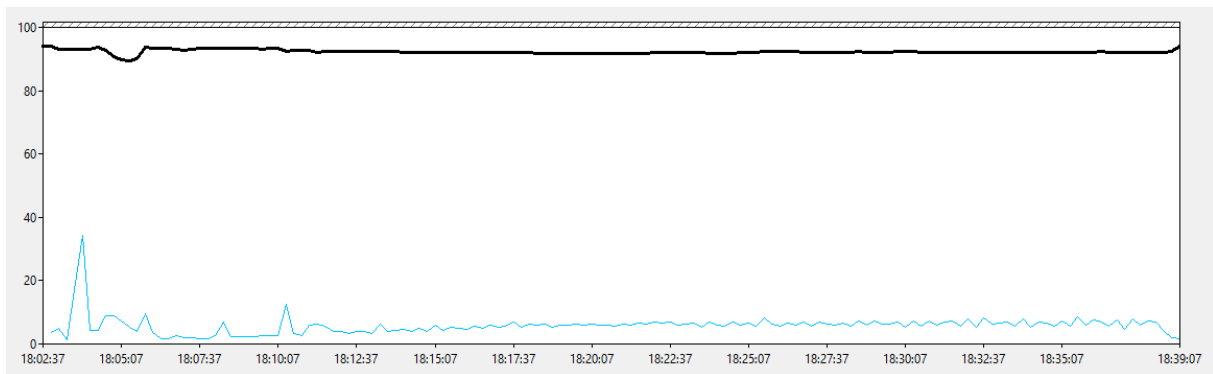
	LoadRunner	Gatling	JMeter	NeoLoad
Requesty na sekundę	7,6	7,7	7,3	10,6
Ogólny średni	1,2	1,2	1,3	1,4

<b>czas odpowiedzi [s]</b>				
<b>Odsetek błędów</b>	0	0	0	0
<b>Liczba aktywnych wątków (użytkowników)</b>	50	50	50	50

### 5.3 Przygotowanie i przeprowadzenie testów dla aplikacji Employee Directory

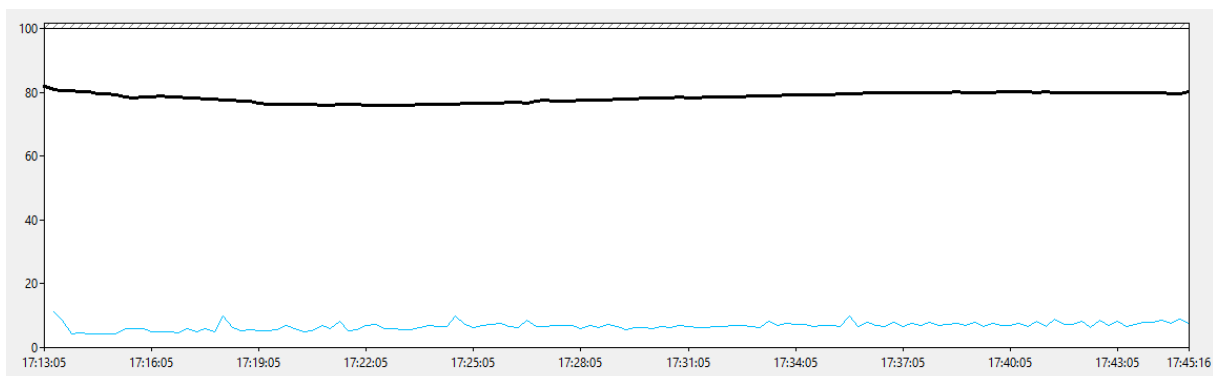
Ponownie, tak jak w przypadku testów poprzednich dwóch aplikacji, wykonane zostały wykresy przedstawiające charakterystykę obciążenia maszyny generującej ruch przez poszczególne narzędzia (ciemna linia oznacza dostępną pamięć RAM, jasna – procent wykorzystania CPU). Szczegóły widoczne są na obrazkach 84 i 85:

- LoadRunner



**Rysunek 84. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia LoadRunner**

- NeoLoad



**Rysunek 85. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia NeoLoad**

Tabela 24 przedstawia ich średnie zużycie fizycznych zasobów maszyny podczas każdego z testów.

**Tabela 24. Średnie zużycie zasobów fizycznych generatora podczas testów aplikacji Employee Directory**

	LoadRunner	NeoLoad
Wykorzystanie CPU [%]	5	7
Dostępna pamięć [MB]	9 227	7 841
Zadeklarowane bajty w użyciu – RAM [%]	24	38
Bajty odczytu dysku / s (średnia)	61 355	164 862
Bajty zapisu dysku / s (średnia)	72 712	87 999
Długość kolejki dysku (średnia)	0,000	0,016

Dowodem na to, że każda z aplikacji wygenerowała porównywalny ruch jest Tabela 25, która przedstawia podsumowanie wykonanych akcji.

**Tabela 25. Ruch wygenerowany podczas testów aplikacji Employee Directory**

	LoadRunner	NeoLoad
Requesty na sekundę	14	22
Ogólny średni czas odpowiedzi [s]	0,007	0,006
Odsetek błędów	0,5	0,3
Liczba aktywnych wątków (użytkowników)	50	50

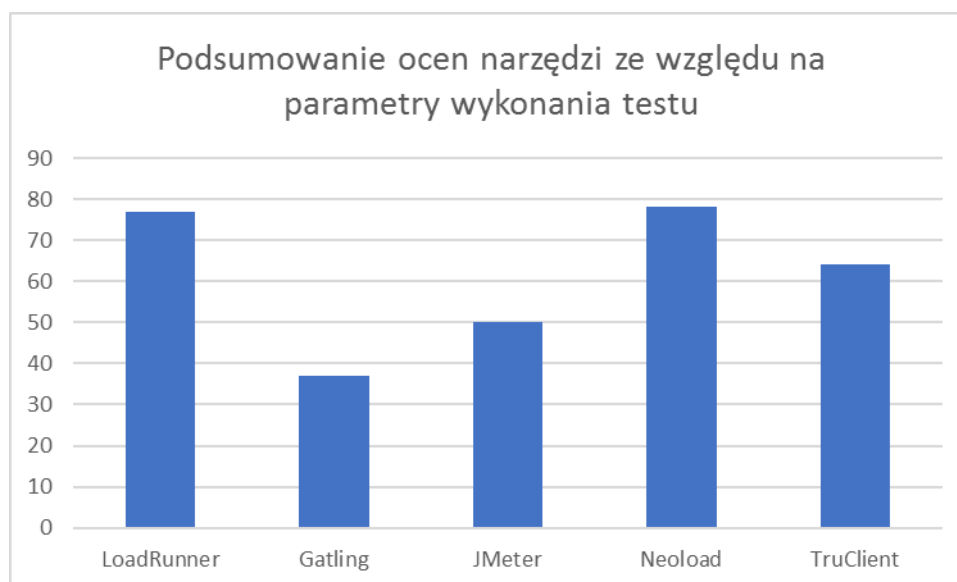
## 5.4 Analiza porównawcza narzędzi użytych do testów wydajnościowych

Bieżący podrozdział podsumowuje aktywności dokonane podczas tworzenia skryptów. Każde z narzędzi zostało poddane ocenie w skali 1-10 w zależności od tego, jak dobrze sprawdza się ono w obrębie danego zagadnienia oraz subiektywnych odczuć autora pracy. Efekt wspomnianej oceny został przedstawiony w Tabeli 26.

Tabela 26. Ocena narzędzi pod kątem wykonania testów

	LoadRunner	Gatling	JMeter	NeoLoad	TruClient
Zarządzanie strategią	10	5	7	10	10
Zarządzanie wirtualnymi użytkownikami	9	5	6	10	9
Konfigurowalność	10	6	8	10	8
Emulacja przeglądarki	10	4	8	10	8
Zarządzanie testem w trakcie uruchomienia	9	3	3	9	9
Podgląd metryk w czasie rzeczywistym	10	4	4	10	10
Analiza wyników	10	5	8	9	10
Wykorzystanie zasobów generatora	9	5	6	10	0
<b>SUMA:</b>	<b>77</b>	<b>37</b>	<b>50</b>	<b>78</b>	<b>64</b>

Rysunek 86 pokazuje dodatkowo proporcję zdobytych punktów przez poszczególne narzędzia.



Rysunek 86. Wykres przedstawiający sumę zdobytych punktów przez każde z narzędzi

Podobnie, jak w przypadku oceny tworzenia skryptów, w dziedzinie wykonania testów najbardziej wszechstronnym i funkcjonalnym narzędziem okazał się NeoLoad i ponownie, tuż za nim znalazł się LoadRunner.

Po raz kolejny zatem najwyższe oceny zdobyły narzędzia komercyjne. Tym razem jednak o wysokich notach zaważyły przede wszystkim możliwości konfigurowania parametrów testu, podgląd metryk w czasie jego trwania oraz szerokie wsparcie dla analizy otrzymanych wyników.

Darmowe narzędzia udostępniają w tej sferze mniejszy zakres funkcjonalności, niemniej jednak dla części projektów z pewnością mogą okazać się one wystarczające.

Warto też wziąć pod uwagę fakt, że płatne narzędzia wyceniają licencje m.in. pod kątem liczby wirtualnych użytkowników. W przypadku bardzo dużych wolumenów może okazać się, że cena narzędzia wywrze znaczący wpływ na budżet całego projektu. Bardziej opłacalne może być wtedy użycie bezpłatnych rozwiązań, nawet kosztem dłuższego przygotowania testu i analizy wyników.

## 6. Podsumowanie

Efektem pracy jest kompleksowa analiza 5 najpopularniejszych narzędzi służących do wykonywania testów wydajnościowych. Sformułowane wnioski stanowią obszerne źródło wiedzy, które pozwala w adekwatny sposób dobrać konkretne rozwiązanie do potrzeb projektu i testowanej aplikacji.

Na potrzeby przebadania wybranych narzędzi, utworzono 12 skryptów – po 5 dla aplikacji Web Tours Sample Application oraz ASP.NET Web Forms, a także 2 skrypty dla Employee Directory. Przy użyciu każdego ze skryptów zostały wykonane oddzielne testy wydajnościowe, które są potwierdzone plikami wynikowymi oraz obszernym opisem zawartym w rozdziale 5.

Wyniki wykazały, że zarówno pod kątem utworzenia skryptów, jak i wykonania testów większe możliwości dają narzędzia komercyjne, których użytkowanie wymaga zakupu płatnych licencji: NeoLoad i LoadRunner.

Darmowe narzędzia, pomimo uzyskania niższych ocen, również mogą znaleźć zastosowanie w części projektów informatycznych (np. poprzez oszczędności w budżecie, bądź po prostu mniejsze oczekiwania wobec funkcjonalności).

Dużą zaletą pracy jest ocena każdego z narzędzi w formie tabelarycznej z rozbiem na poszczególne aspekty tworzenia skryptów oraz przeprowadzania testów. Dzięki temu potencjalny tester, bądź inna osoba planująca zbadanie wydajności danego systemu, jest w stanie wybrać rozwiązanie najbardziej spełniające pożądane kryteria. W przypadku mniej skonkretyzowanych oczekiwań istnieje możliwość oparcia się o ogólną ocenę danego narzędzia.

Niestety, spośród ogromnej liczby dostępnych na rynku rozwiązań udało się zweryfikować jedynie 5 z nich, co znacznie ogranicza wybór. Jest to niewątpliwie wadą tej pracy. Podobnie jest z technologiami, które zostały przetestowane. Prezentowały one szeroki przekrój różnego rodzaju aspektów wymagających obsługi, jednak z pewnością można będzie znaleźć kolejne, które nie zostały tutaj ujęte.

Pracę można rozwinąć w przyszłości poprzez analizę kolejnych narzędzi testowych oraz sprawdzenie większej liczby aplikacji.

## Prace cytowane

- [1]. R. Pawlak. *Testowanie oprogramowania. Podręcznik dla początkujących*, Helion, strony 74-89, 2014, ISBN: 9788328301399
- [2]. *Performance Testing Tutorial: What is, Types, Metrics & Example*, <https://www.guru99.com/performance-testing.html>, Dostęp 31.01.2020
- [3]. M. Williams, *Performance Testing - Tools, Steps, and Best Practices*, <https://www.keycdn.com/blog/performance-testing>, Dostęp 31.01.2020
- [4]. W. Mar, *LoadRunner architecture*, <http://www.wilsonmar.com/1loadrun.htm>, Dostęp 01.02.2020
- [5]. T. Krazit, *HP snaps up Mercury Interactive*, <https://www.cnet.com/news/hp-snaps-up-mercury-interactive/>, Dostęp 01.02.2020
- [6]. *Mercury Interactive (aka LoadRunner) sold to Micro Focus*, <https://www.appvance.ai/loadrunner-is-sold>, Dostęp 01.02.2020
- [7]. *LoadRunner Tutorial*, [https://admhelp.microfocus.com/lr/en/12.56-12.57/help/WebHelp/Content/Tutorial/Introducing\\_LoadRunner.htm](https://admhelp.microfocus.com/lr/en/12.56-12.57/help/WebHelp/Content/Tutorial/Introducing_LoadRunner.htm), Dostęp 08.01.2020
- [8]. *Understanding LoadRunner licensing*, <https://www.jds.net.au/loadrunner-licensing/>, Dostęp 01.02.2020
- [9]. *Gatling*, <https://gatling.io/docs/3.3/>, Dostęp 01.02.2020
- [10]. J. Willett, *Gating Introduction – High Level Overview of the Stress Test Tool*, <https://www.james-willett.com/gatling-introduction>, Dostęp 01.02.2020
- [11]. M. Herber, *Gatling. Testy wydajności w innej formie. Część 1.*, <https://testerzy.pl/baza-wiedzy/gatling-testy-wydajnosci-w-innej-formie-czesc-1>, Dostęp 01.02.2020
- [12]. *JMeter Tutorial for Beginners: Learn in 7 Days*, <https://www.guru99.com/jmeter-tutorials.html>, Dostęp 01.02.2020
- [13]. B. Erinle. *Performance Testing with JMeter 3*, Packt Publishing, strona 17, 2017, ISBN: 9781787285774
- [14]. *Load Testing with NeoLoad – the Most Powerful Performance Testing Solution, Flexible Pricing to Fit Any Team*, <https://www.neotys.com>, Dostęp 01.02.2020
- [15]. *Introducing TruClient*, [https://admhelp.microfocus.com/tc/en/2020/Content/TruClient/\\_tc\\_c\\_truclient\\_overview.htm](https://admhelp.microfocus.com/tc/en/2020/Content/TruClient/_tc_c_truclient_overview.htm), Dostęp 02.02.2020
- [16]. *Web Tours Sample Application*, <https://marketplace.microfocus.com/appdelivery/content/web-tours-sample-application>, Dostęp 23.02.2019
- [17]. Matulewski J., Grabek M., Pakulski M., Borycki D., *ASP.NET Web Forms. Kompletny przewodnik dla programistów interaktywnych aplikacji internetowych w Visual Studio*, Helion, strony 152-160; 209-213, 2014, ISBN: 978-83-246-9164-7
- [18]. Evjen B., Hanselman S., Rader D., *ASP.NET 4 z wykorzystaniem C# i VB. Zaawansowane programowanie*, Helion, strony 86-90; 328-331, 2011, ISBN: 978-83-246-8352-9
- [19]. S. Bageri, *Create and Implement 3-Tier Architecture in ASP.Net*, <https://www.c-sharpcorner.com/UploadFile/4d9083/create-and-implement-3-tier-architecture-in-Asp-Net/>, Dostęp 02.11.2019



- [20]. G. Dogi, *Three tier architecture in asp.net using c# example*, [https://www.onlinebuff.com/article\\_step-by-step-3-tier-architecture-in-aspnet-using-c-example\\_45.html](https://www.onlinebuff.com/article_step-by-step-3-tier-architecture-in-aspnet-using-c-example_45.html), Dostęp 02.11.2019
- [21]. M. Urbański, *ASP.NET: View state – podstawy*, <https://michalurbanski.wordpress.com/2010/10/03/asp-net-view-state-podstawy/>, Dostęp 15.01.2020
- [22]. A. Prusator, *Dlaczego nie powinieneś korzystać z systemów używających Flash?*, <https://fordata.pl/dlaczego-nie-powinienes-korzystac-z-systemow-uzywajacych-flash/>, Dostęp 02.02.2020
- [23]. A. Rahim, *Performance Testing Flex Applications using LoadRunner*, <https://softwaretesttips.com/2011/11/21/performance-testing-flex-applications-using-loadrunner/>, Dostęp 02.02.2020
- [24]. *CREATING A FLEX CONTENT MANAGEMENT SYSTEM (CMS)*, [http://www.richmediacs.com/user\\_manuals/Flash%20Builder%20and%20Flex%20/Creating%20a%20Flex%20CMS/CreatingAFlexCMS.html](http://www.richmediacs.com/user_manuals/Flash%20Builder%20and%20Flex%20/Creating%20a%20Flex%20CMS/CreatingAFlexCMS.html), Dostęp 04.12.2019
- [25]. *Correlation in LoadRunner with Web\_Reg\_Save\_Param Example*, <https://www.guru99.com/correlation-in-loadrunner-ultimate-guide.html>, Dostęp 03.02.2020
- [26]. *INSTALLATION*, <https://gatling.io/docs/current/installation#installation>, , Dostęp 03.02.2020
- [27]. *Creating Dates in JMeter Using the TimeShift Function*, <https://www.blazemeter.com/blog/creating-dates-in-JMeter-using-the-timeshift-function/>, Dostęp 09.11.2019
- [28]. *Random String Generator with no consecutive duplicate characters*, <http://usefulfunctionsinloadrunner.blogspot.com/2013/10/random-string-generator-with-no.html>, Dostęp 16.11.2019
- [29]. *How to Load Test Flex and AMF Protocol Apps with JMeter*, <https://www.blazemeter.com/blog/how-to-load-test-flex-and-amf-protocol-apps-with-JMeter/> Dostęp 26.12.2019
- [30]. *AMF Plugin for JMeter*, <https://github.com/maciejfrank/JMeter-amf>, Dostęp 26.12.2019
- [31]. *Collect Data with Windows Performance Monitor*, [https://help.tableau.com/current/server/en-us/perf\\_collect\\_perfmon.htm](https://help.tableau.com/current/server/en-us/perf_collect_perfmon.htm), 04.01.2020
- [32]. *How to delete (remove) a specific line of a text file based on the line number?*, <https://stackoverflow.com/questions/38494940/how-to-delete-remove-a-specific-line-of-a-text-file-based-on-the-line-number>, 05.01.2020

## Wykaz rysunków

Rysunek 1. Okno narzędzia LoadRunner – moduł VuGen .....	10
Rysunek 2. Okno IntelliJ IDEA wyświetlające skrypt stworzony w Gatlingu.....	12
Rysunek 3. Struktura elementów w narzędziu JMeter, Źródło: [12].....	13
Rysunek 4. Okno narzędzia JMeter.....	13
Rysunek 5. Okno narzędzia NeoLoad .....	14
Rysunek 6. Okno narzędzia TruClient .....	15
Rysunek 7. Wyszukiwarka lotów w aplikacji Web Tours.....	16
Rysunek 8. Zakładka <i>Studenci</i> aplikacji ASP.NET Web Forms .....	18
Rysunek 9. Strona główna aplikacji Employee Directory.....	20
Rysunek 10. Widżet zarządzający nagrywaniem ruchu sieciowego w narzędziu LoadRunner.....	23
Rysunek 11. Plik kolekcji zawierające wartości dla parametru pMiasto .....	24
Rysunek 12. <i>Design Studio</i> .....	25
Rysunek 13. Dodawanie asercji w LoadRunnerze .....	26
Rysunek 14. Zakładka <i>Snapshot</i> . Porównanie wartości z nagrania i odtworzenia.....	27
Rysunek 15. Okno nagrywania programu Gatling .....	29
Rysunek 16. Przykładowy plik z nagraniem odpowiedzi z serwera.....	31
Rysunek 17. Okno konsoli IntelliJ IDEA wyświetlające treść odpowiedzi.....	31
Rysunek 18. Przykładowy request nagrany w narzędziu JMeter .....	34
Rysunek 19. JMeter - <i>CSV Data Set Config</i> zarządzający plikiem z danymi adresowymi.....	34
Rysunek 20. Dostępne języki programowania w <i>JSR223 Sampler</i> .....	35
Rysunek 21. JMeter - <i>Regular Expression Extractor</i> .....	35
Rysunek 22. JMeter - Response Assertion.....	36
Rysunek 23. Definicja Think timu w narzędziu JMeter.....	36
Rysunek 24. JMeter - <i>View Results Tree</i> .....	37
Rysunek 25. Widżet zarządzający nagrywaniem ruchu sieciowego w narzędziu NeoLoad .....	39
Rysunek 26. Przykładowy request nagrany w narzędziu NeoLoad .....	39
Rysunek 27. NeoLoad - Definicja zmiennej zarządzającej plikiem z danymi adresowymi .....	40
Rysunek 28. Obsługa dat w narzędziu NeoLoad.....	40
Rysunek 29. NeoLoad - <i>Variable Extractor</i> .....	41
Rysunek 30. Mechanizm automatycznej korelacji w narzędziu NeoLoad.....	41
Rysunek 31. Obsługa asercji w narzędziu NeoLoad.....	42
Rysunek 32. Zarządzanie Think timami w narzędziu NeoLoad .....	42
Rysunek 33. NeoLoad- mechanizm <i>Check User Path</i> .....	43
Rysunek 34. Przykładowy krok nagrany w narzędziu TruClient.....	45

Rysunek 35. Oznaczenie transakcji w TruClient .....	45
Rysunek 36. Obsługa pól rozwijalnych w TruClientcie .....	46
Rysunek 37. LoadRunner, parametr typu <i>Date/Time</i> .....	46
Rysunek 38. Obsługa asercji w narzędziu TruClient .....	47
Rysunek 39. Zarządzanie Think timem w narzędziu TruClient .....	47
Rysunek 40. Podgląd atrybutów obiektu HTML .....	48
Rysunek 41. Próba obsługi <i>stanu widoku</i> przez narzędzie <i>Design Studio</i> .....	50
Rysunek 42. LoadRunner, parametr typu <i>Random Number</i> .....	52
Rysunek 43. Statyczne elementy pobierane automatycznie podczas wczytywania strony głównej aplikacji ASP.NET Web Forms .....	53
Rysunek 44. Jmeter Plugins Manager .....	55
Rysunek 45. JMeter - <i>Parallel Controller</i> .....	55
Rysunek 46. Jmeter, <i>Random Variable</i> .....	56
Rysunek 47. Automatyczna korelacja <i>stanów widoku</i> w narzędziu NeoLoad .....	56
Rysunek 48. Obsługa requestów asynchronicznych w narzędziu NeoLoad .....	57
Rysunek 49. NeoLoad, parametr typu <i>Random String</i> .....	57
Rysunek 50. NeoLoad, parametr typu <i>Random Integer</i> .....	58
Rysunek 51. Przykładowa odpowiedź aplikacji Employee Directory nagrana protokołem Web .....	60
Rysunek 52. Opcje nagrywania dla protokołu Flex .....	61
Rysunek 53. Przykładowa odpowiedź aplikacji Employee Directory nagrana protokołem Flex .....	63
Rysunek 54. Przykładowe żądanie aplikacji Employee Directory nagrany za pomocą wtyczki <i>JMeter AMF</i> .....	65
Rysunek 55. Przykładowe ciało odpowiedzi aplikacji Employee Directory nagrane w narzędziu JMeter .....	65
Rysunek 56. Przykładowy request aplikacji Employee Directory nagrany za pomocą narzędzia NeoLoad.....	66
Rysunek 57. Przykładowa odpowiedź aplikacji Employee Directory nagrana za pomocą narzędzia NeoLoad.....	67
Rysunek 58. Wykres przedstawiający sumę zdobytych punktów przez każde z narzędzi.....	68
Rysunek 59. Model obciążenia ustawiony w narzędziu LoadRunner.....	71
Rysunek 60. Emulacja przeglądarki w narzędziu LoadRunner.....	71
Rysunek 61. Test uruchomiony w narzędziu LoadRunner.....	72
Rysunek 62. Przykładowy wykres prezentujący liczbę żądań na sekundę podczas całego testu na tle liczby aktywnych wirtualnych użytkowników .....	73
Rysunek 63. Test uruchomiony w narzędziu Gatling.....	76
Rysunek 64. Wykres requestów na sekundę na tle liczby użytkowników zawężony do czasu pełnego obciążenia.....	76
Rysunek 65. Kształtowanie modelu obciążenia na poziomie <i>Grupy Wątków</i> w JMeterze .....	77
Rysunek 66. Komunikat widoczny zaraz po uruchomieniu JMetera .....	78

Rysunek 67. Wykres przedstawiający liczbę requestów na sekundę podczas testu aplikacji Web Tours Sample Application .....	78
Rysunek 68. Konfiguracja Populacji w narzędziu NeoLoad.....	80
Rysunek 69. Przygotowanie scenariusza oraz modelu obciążenia w narzędziu NeoLoad.....	81
Rysunek 70. Test uruchomiony w narzędziu NeoLoad.....	81
Rysunek 71. Emulacja przeglądarki w narzędziu TruClient .....	83
Rysunek 72. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia LoadRunner .....	84
Rysunek 73. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia Gatling.....	84
Rysunek 74. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia JMeter.....	85
Rysunek 75. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia NeoLoad .....	85
Rysunek 76. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia TruClient .....	85
Rysunek 77. Liczba błędów na sekundę na tle liczby użytkowników podczas testu przy użyciu narzędzia TruClient .....	86
Rysunek 78. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia LoadRunner .....	88
Rysunek 79. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia Gatling.....	88
Rysunek 80. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia JMeter.....	88
Rysunek 81. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia NeoLoad .....	89
Rysunek 82. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia TruClient .....	89
Rysunek 83. Liczba błędów na sekundę na tle liczby użytkowników podczas testu przy użyciu narzędzia TruClient .....	90
Rysunek 84. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia LoadRunner .....	91
Rysunek 85. Charakterystyka obciążenia CPU i RAM podczas testu z wykorzystaniem narzędzia NeoLoad .....	91
Rysunek 86. Wykres przedstawiający sumę zdobytych punktów przez każde z narzędzi.....	93

## Wykaz tabel

Tabela 1. Scenariusz testu dla aplikacji Web Tours Sample Application .....	17
Tabela 2. Scenariusz testu dla aplikacji ASP.NET.....	18
Tabela 3. Scenariusz testu dla aplikacji Employee Directory .....	21
Tabela 4. Ocena narzędzia LoadRunner pod kątem tworzenia skryptu .....	27
Tabela 5. Ocena narzędzia Gatling pod kątem tworzenia skryptu .....	32
Tabela 6. Ocena narzędzia JMeter pod kątem tworzenia skryptu .....	37
Tabela 7. Ocena narzędzia NeoLoad pod kątem tworzenia skryptu .....	43
Tabela 8. Ocena narzędzia TruClient pod kątem tworzenia skryptu.....	48
Tabela 9. Ocena narzędzia LoadRunner pod kątem dodatkowych aktywności podczas tworzenia skryptu do aplikacji ASP.NET Web Forms .....	52
Tabela 10. Ocena narzędzia Gatling pod kątem dodatkowych aktywności podczas tworzenia skryptu do aplikacji ASP.NET Web Forms .....	54
Tabela 11. Ocena narzędzia JMeter pod kątem dodatkowych aktywności podczas tworzenia skryptu do aplikacji ASP.NET Web Forms .....	56
Tabela 12. Ocena narzędzia NeoLoad pod kątem dodatkowych aktywności podczas tworzenia skryptu do aplikacji ASP.NET Web Forms .....	58
Tabela 13. Ocena narzędzia TruClient pod kątem dodatkowych aktywności podczas tworzenia skryptu do aplikacji ASP.NET Web Forms .....	59
Tabela 14. Podsumowanie ocen każdego z narzędzi pod kątem tworzenia skryptów .....	68
Tabela 15. Ocena narzędzia LoadRunner pod kątem wykonania testu .....	73
Tabela 16. Ocena narzędzia Gatling pod kątem wykonania testu .....	77
Tabela 17. Ocena narzędzia JMeter pod kątem wykonania testu .....	79
Tabela 18. Ocena narzędzia NeoLoad pod kątem wykonania testu .....	82
Tabela 19. Ocena narzędzia TruClient pod kątem wykonania testu .....	83
Tabela 20. Średnie zużycie zasobów fizycznych generatora podczas testów aplikacji ASP.NET Web Forms.....	86
Tabela 21. Ruch wygenerowany podczas testów aplikacji Web Tours Sample Application.....	87
Tabela 22. Średnie zużycie zasobów fizycznych generatora podczas testów aplikacji ASP.NET Web Forms.....	90
Tabela 23. Ruch wygenerowany podczas testów aplikacji ASP.NET Web Forms.....	90
Tabela 24. Średnie zużycie zasobów fizycznych generatora podczas testów aplikacji Employee Directory.....	92
Tabela 25. Ruch wygenerowany podczas testów aplikacji Employee Directory .....	92
Tabela 26. Ocena narzędzi pod kątem wykonania testów .....	93

## Wykaz kodu

Kod 1. Przykładowy request nagrany w narzędziu LoadRunner .....	23
Kod 2. Obsługa dat w narzędziu LoadRunner.....	24
Kod 3. Mechanizm przechwytywania wartości z odpowiedzi w narzędziu LoadRunner .....	25
Kod 4. Przykładowy request nagrany w narzędziu Gatling .....	29
Kod 5. Obsługa kolekcji za pomocą <i>CSV Feeder</i> .....	30
Kod 6. Obsługa dat w narzędziu Gatling.....	30
Kod 7. Mechanizm przechwytywania wartości z odpowiedzi w narzędziu Gatling.....	30
Kod 8. Dodawanie asercji w Gatlingu.....	30
Kod 9. Ustawienie rozszerzonego logowania odpowiedzi w Gatlingu .....	32
Kod 10. Obsługa dat w <i>JSR223 Sampler</i> .....	35
Kod 11. Zwiększenie maksymalnej długości parametru w LoadRunnerze.....	49
Kod 12. Obsługa requestów asynchronicznych przez mechanizm <i>Concurrent group</i> .....	50
Kod 13. Generowanie losowego adresu e-mail przy wykorzystaniu języka C .....	51
Kod 14. Korelacja <i>stanu widoku</i> w narzędziu Gatling. ....	53
Kod 15. Generowanie losowego adresu e-mail przy wykorzystaniu języka Scala .....	53
Kod 16. Generowanie losowego numeru <i>Student Code</i> przy wykorzystaniu języka Scala .....	54
Kod 17. Generowanie losowego adresu e-mail przy wykorzystaniu języka Groovy .....	55
Kod 18. Generowanie losowego adresu e-mail przy wykorzystaniu języka JavaScript .....	59
Kod 19. Przykładowy request aplikacji Employee Directory nagrany protokołem Web.....	59
Kod 20. Przykładowy request aplikacji Employee Directory nagrany protokołem Flex .....	61
Kod 21. Przykładowy request aplikacji Employee Directory nagrany w narzędziu Gatling .....	63
Kod 22. Przykładowe ciało żądania aplikacji Employee Directory nagrane w narzędziu Gatling .....	64
Kod 23. Przykładowe ciało odpowiedzi aplikacji Employee Directory nagrane w narzędziu Gatling. ....	64
Kod 24. Model obciążenia zdefiniowany w narzędziu Gatling.....	74
Kod 25. Konfiguracja atrybutów protokołu w narzędziu Gatling .....	75