JCSI 3 (2017) 28-32
Journal of Computer Sciences Institute

Wysłane: 2016-12-10
Przyjęte: 2016-12-14

Porównanie technologii tworzenia aplikacji internetowych JEE na przykładzie JavaServer Faces i Spring Boot

Michał Marcin Kizeweter*, Beata Pańczyk
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawiono wyniki porównania efektywności wytwarzania aplikacji internetowych na platformie Java Enterprise Edition z zastosowaniem JavaServer Faces i Spring Boot. Analiza porównawcza została przeprowadzona na bazie specjalnie przygotowanych aplikacji testowych, zaimplementowanych w obu technologiach.

Słowa kluczowe: Spring Boot; JavaServer Faces; aplikacje internetowe

*Autor do korespondencji.

Adres e-mail: michalkizeweter@gmail.com

Comparison of JEE platform web applications development using JavaServer Faces and Spring Boot example

Michał Marcin Kizeweter*, Beata Pańczyk
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article presents the results of the web applications development effectiveness on the Java Enterprise Edition platform using JavaServer Faces and Spring Boot. The comparative analysis was performed using the specially prepared test applications, implemented in both technologies.

Keywords: Spring Boot; JavaServer Faces; web application development

*Corresponding author.

E-mail address: michalkizeweter@gmail.com

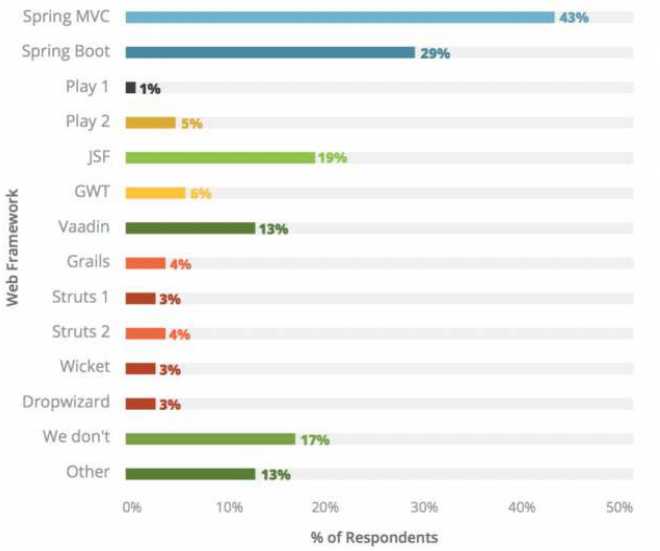
1. Wstęp

Stworzenie nawet prostej aplikacji internetowej w Javie wymaga zwykle wielu plików konfiguracyjnych *xml*, uruchomienia kontenera aplikacji, wdrożenia i wykonania wielu innych powtarzalnych czynności. Wszystko to powoduje, że wytwarzanie aplikacji web na platformie JEE nie jest proste, pomimo od dawna istniejących technologii wspomagających jak Spring czy JavaServer Faces (JSF).

Nowym rozwiązaniem, które ułatwia ten proces jest Spring-Boot. Projekt powstał w celu przyspieszenia i uproszczenia startu z popularnym szkieletem programistycznym Spring (rysunek 1). Spring Boot wprowadził automatyczną konfigurację dla Spring i wyeliminował całkowicie pliki *xml*. Tym samym projekt Spring Boot idealnie nadaje się do projektów studenckich i szybkiego prototypowania aplikacji. Jedynym wymaganiem stawianym przed użytkownikiem jest znajomość struktury projektu Maven, popularnego narzędzia automatyzującego budowę oprogramowania na platformie Java.

Z uwagi na dużą popularność jaką osiągnął Spring Boot w roku 2016 (rysunek 1 - 29% popularności), autorzy przeanalizowali jego konkurencyjność w stosunku do starszego frameworka JSF (rysunek 1 - 19% popularności), który nie jest oparty na założeniach Spring.

Najważniejsze cechy Spring Boot i JSF zostały zestawione w tabeli 1.



Rys. 1. Popularność wybranych frameworków Java w roku 2016 [1]

W artykule przedstawiono wyniki porównania efektywności wytwarzania aplikacji internetowych w oparciu o JSF i Spring Boot. Analiza porównawcza została przeprowadzona na bazie specjalnie przygotowanych aplikacji testowych, zaimplementowanych w obu technologiach.

Tabela 1. Najważniejsze cechy Spring i JSF

	Spring	JSF
Twórca	Pivotal Software	Oracle
Pierwsza wersja	Czerwiec 2003 (Spring 0.9)	Marzec 2004 (JSF 1.0)
Wersja finalna	Lipiec 2016 (Spring 4.3.2, Spring Boot 1.4.0)	Maj 2013 (JSF 2.2)
Język progr.	Java	
SDK	Java EE 7 SDK	
IDE	NetBeans (Open source), Eclipse (Open source), IntelliJ IDEA (komercyjne)	
Technologia widoku	Brak	Facelet
Inne możliwe tech. widoku	Facelet, JSP, Groovy, AngularJS	JSP
Serwery open-source	GlassFish, Apache Tomcat (wbudowany w Spring Boot)	GlassFish, Apache Tomcat
Wsparcie	Bogate wsparcie i dokumentacja, duża baza użytkowników	
Licencja	Open Source	
Standard stron widoku	Brak	XHTML1.1, HTML5
Niezależność platformy systemowej	Windows, Mac, Linux	
Inne	Struktura aplikacji MVC, Spring nie posiada natywnej technologii widoku, Spring Boot posiada wbudowany kontener aplikacji	Struktura aplikacji MVC, podstawą działania są komponenty zarządzane, generowanie widoków za pomocą facelet

2. Cel, teza i metody badań

Celem badań było porównanie efektywności tworzenia prostej aplikacji internetowej w wybranych frameworkach.

W artykule postawiono następującą tezę:

Framework Spring Boot jest bardziej efektywnym narzędziem wytwarzania aplikacji JEE w porównaniu do JavaServer Faces.

Dla potwierdzenia tej tezy wykorzystano metodę badań opartą na analizie porównawczej obu frameworków.

W tym celu stworzono dwie, funkcjonalnie identyczne, testowe aplikacje internetowe wykorzystując następujące narzędzia i technologie:

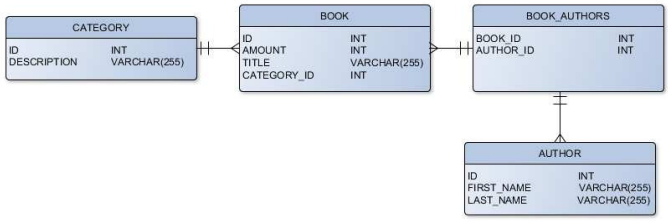
- szkielety programistyczne Spring-Boot oraz JavaServer Faces;
- środowisko developerskie Eclipse [2];
- serwer bazy danych PostgreSQL [3];
- Maven - narzędzie do automatyzacji budowy projektów Java [4];
- AngularJS - framework JavaScript [5];
- Hibernate – framework służący do realizacji warstwy dostępu do danych.

3. Aplikacje testowe

Proste aplikacje testowe (do obsługi biblioteki) zostały zaprojektowane w oparciu o pozycje [6, 7].

Obie aplikacje (Spring Boot i JSF) posiadają identyczne funkcjonalności: dodawanie nowych, edycję, usuwanie istniejących kategorii książek, autorów i książek. Są to typowe funkcjonalności aplikacji typu CRUDS (ang. Create, Read, Update, Delete, Search).

Schemat wykorzystanej bazy danych przedstawiono na rysunku 2. Obie aplikacje korzystają z tej samej bazy danych na serwerze PostgreSQL.



Rys. 2. Schemat bazy danych

Interfejs obu aplikacji umożliwia pełny dostęp do wszystkich, wcześniej opisanych funkcjonalności (Rys. 3).

Rys. 3. Przykładowa strona aplikacji testowej

4. Analiza porównawcza

W tabeli 2 przedstawiono parametry sprzętowe komputera, na którym wykonywano testy.

Tabela 2. Parametry sprzętowe zestawu pomiarowego

Element	Stan/Wersja
System operacyjny	Windows 8.1 64-bit
Procesor	Intel Core i7-4700MQ (2,40-3,40 GHz)
Pamięć RAM	8 GB (SO-DIMM DDR3)
PostgreSQL	9.6
ApacheTomcat	8.5
Java	Java EE7

W celu porównania obu aplikacji analizowano 4 kryteria:

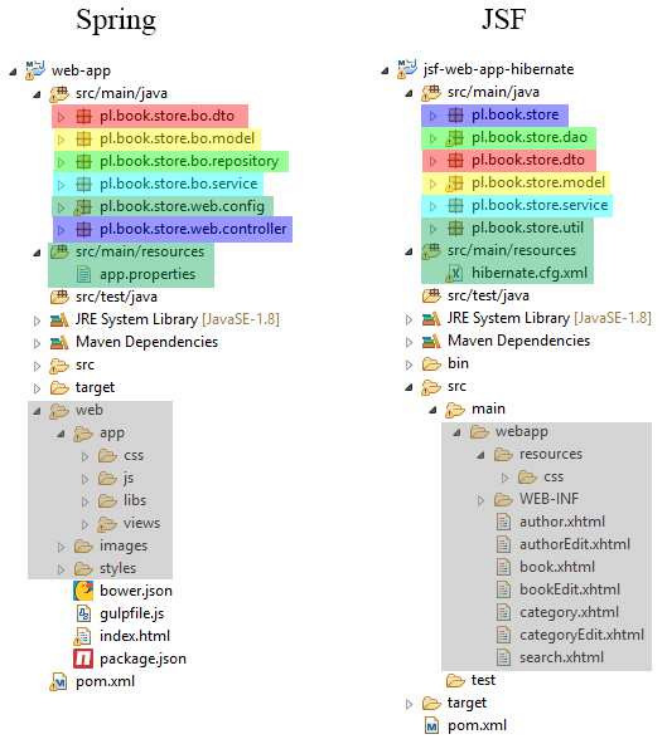
- ogólną strukturę aplikacji;
- efektywność pracy z danymi;
- efektywność wczytywania zasobów aplikacji przez przeglądarkę internetową;
- podstawowe metryki kodu.

4.1. Struktura aplikacji

Rysunek 4 przedstawia strukturę obu aplikacji, opartą na projekcie Maven. Na rysunku wyróżniono kolorem elementy funkcjonalnie wspólne dla obu aplikacji:

- czerwonym zaznaczono pakiet klas **dto**, które mapują encje bazy danych na obiekty w aplikacji;
- żółty wskazuje pakiety dla **modeli**, zawierające klasy mapujące encje z bazy danych za pomocą adnotacji JPA;

- jasno zielony oznacza pakiety klas **repozytoriów** dostępu do danych w bazie, zawierające deklaracje zapytań;
- jasno niebieski oznacza pakiety usług (**service**), zawierające klasy łączące kontrolery z repozytoriami (warstwa logiki biznesowej);
- ciemno zielonym zaznaczono pliki konfiguracyjne obu aplikacji, m.in. połączenia z bazą danych;
- ciemno niebieskim oznaczono **kontrolery** obu aplikacji (klasy odbierające żądania użytkownika);
- szary wskazuje pliki odpowiedzialne za **widoki.**.



Rys. 4. Struktura projektu Spring Boot i JSF

Dzięki zastosowaniu struktury Maven, elementy obu aplikacji są zbliżone. Główne różnice to pliki konfiguracyjne Hibernate (w Spring Hibernate jest głębiej zintegrowany), oraz obecność AngularJS, generującej widoki dla Spring.

4.2. Efektywność pracy z bazą danych

W celu porównania wydajności pracy obu aplikacji, zmierzono czas pracy z danymi. Do testu przygotowano 5 scenariuszy (Tabela 3). Każdy ze scenariuszy powtarzano minimum 10 razy a wynik pomiaru uśredniono.

Tabela 3. Scenariusze testowe

Scenariusz	Opis
1	Zapis 94 autorów do tabeli Author.
2	Zapis 100 książek w tabeli Book.
3	Odczyt wszystkich książek z tabeli Book.
4	Wyszukanie książek w tabeli Book (losowo wybranymi słowami kluczowymi).
5	Zapis wielu książek jednocześnie do tabeli Book.

Testowy zbiór książek i autorów do bazy danych pobrano ze strony [8]. Czasy mierzono w milisekundach [ms]. Operacje na bazie danych realizowano za pomocą Hibernate.

Fragment kodu z pomiarem czasu operacji według scenariusza 1 dla Spring i JSF, pokazują odpowiednio przykłady 1 i 2. W pozostałych scenariuszach, pomiar odbywał się analogicznie.

Przykład 1. Pomiar czasu dodania nowego autora - Spring

```
Author newAuthor = new Author();
newAuthor.updateAuthor(dto.getFirstName(),
dto.getLastName());
long startMillis = System.currentTimeMillis();
repository.save(newAuthor);
log.info("Add author {}", (System.currentTimeMillis() -
startMillis));
```

Przykład 2. Pomiar czasu dodania nowego autora- JSF

```
public void insertAuthor(AuthorDto dto) {
Long start = System.currentTimeMillis();
AuthorDao dao = new AuthorDao();
Author author = new Author();
author.setId(dao.getId());
author.setFirstName(dto.getFirstName());
author.setLastName(dto.getLastName());
author.setBooks(Sets.newHashSet());
dao.save(author);
System.out.println("Dodaj autora : " +
(System.currentTimeMillis() - start));
```

W przypadku scenariusza 5 dodawano jednocześnie 10, a następnie 100 książek. Każdą operację wykonano 10 razy a wyniki uśredniono.

W tabelach 4 i 5 przedstawiono rezultaty pomiarów dodawania jednocześnie 10 i 100 książek

Tabela 4. Czasy dodawania jednocześnie 10 książek

	Spring	JSF
Nr. Pomiaru	**Czas [ms]**	
1	89	226
2	84	214
3	85	272
4	70	235
5	72	227
6	84	262
7	71	230
8	75	245
9	71	238
10	69	221
Średni czas	**77**	**237**

Tabela 5. Czasy dodawania jednocześnie 100 książek w obu aplikacjach testowych

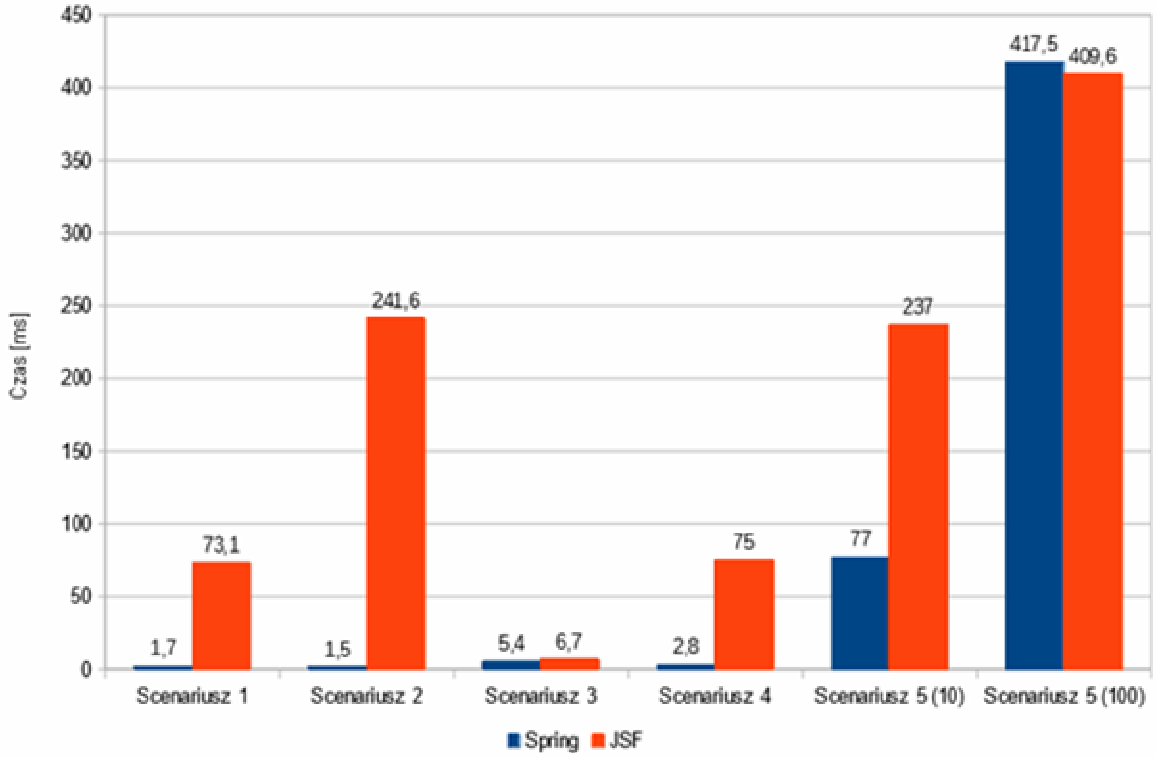
	Spring	JSF
Nr. Pomiaru	**Czas [ms]**	
1	475	397
2	397	448
3	431	399
4	404	395
5	415	420
6	382	416
7	467	390
8	392	392
9	431	437
10	381	402
Średni czas	**417,5**	**409,6**

Na rysunku 5 zestawiono średnie czasy pomiarów dla scenariuszy 1-5. Porównując obie aplikacje, w kategorii dodawania nowych rekordów (scenariusz 1 i 2) , Spring-Boot jest zdecydowanie szybszy od JSF. W przypadku pobierania rekordów (scenariusz 3) - Spring jest nieznacznie szybszy. Dla scenariusza 4 (wyszukiwanie po słowach kluczowych), Spring jest szybszy średnio o 70 milisekund od JSF.

Przy dodaniu większej liczby rekordów jednocześnie (scenariusz 5), Spring jest szybszy przy 10 rekordach, jednak dla 100 rekordów nieznacznie efektywniejszy okazuje się JSF.

4.3. Efektywność wczytywania zasobów aplikacji w przeglądarce internetowej

Kolejnym ważnym pomiarem wydajności aplikacji webowych jest czas mierzony od momentu wystąpienia żądania użytkownika, do momentu załadowania wszystkich potrzebnych zasobów. Czas ten został zmierzony za pomocą konsoli deweloperskiej Google Chrome w odpowiedzi na 20 żądań użytkownika. Wyniki pomiarów przedstawia tabela 6.



Rys. 5. Średnie czasy operacji dla wszystkich scenariuszy testowych dla Spring Boot i JSF

Tabela 6. Czas pobierania zasobów przez przeglądarkę Chrome

	Spring	JSF			Spring	JSF
Nr. Pomiaru	**Czas [ms]**			**Nr. Pomiaru**	**Czas [ms]**	
1	386	261		**11**	30	268
2	35	253		**12**	34	250
3	32	248		**13**	30	273
4	33	228		**14**	29	259
5	30	256		**15**	33	251
6	30	226		**16**	31	277
7	31	248		**17**	27	260
8	32	231		**18**	30	264
9	35	265		**19**	27	240
10	29	227		**20**	31	247
Średni czas:	48,75	251,6				

Pierwszy pomiar w aplikacji Spring, jest niewspółmiernie długi w porównaniu do reszty pomiarów. Jest to spowodowane tym iż przeglądarka pierwszy raz otwierając aplikację musi pobrać wszystkie skrypty Java Script i zasoby AngularJS potrzebne do jej prawidłowego działania. Ponieważ jest to Single Page Application (SPA), przy kolejnej nawigacji pobierane jest tylko minimum danych.

Patrząc na wyniki pomiarów aplikacji JSF, trzeba wziąć pod uwagę zaobserwowane w poprzednim teście długie czasy dostępu do bazy danych. Hipotetycznie, gdyby można je skrócić, prędkość pobierania stron aplikacji JSF byłaby bardziej zbliżona do czasów aplikacji Spring.

4.4. Porównanie metryk aplikacji

Ostatnim elementem porównania obu aplikacji jest analiza metryk kodu. Do tego pomiaru wykorzystano aplikację LocMetrics [9]. Zlicza ona linie kodu z pominięciem linii pustych oraz komentarzy. Wyniki pomiarów przedstawiono w tabeli 7.

Liczba linii kodu dla Spring Boot jest mniejsza od aplikacji JSF, ponieważ Spring automatycznie konfiguruje oraz samodzielnie zarządza wieloma aspektami aplikacji (np. mapowaniem, Hibernate, aspektami MVC). W JSF należy ręcznie to wszystko skonfigurować oraz samodzielnie implementować metody do pracy z danymi.. Widać to w tabeli 7 na przykładzie liczby bibliotek używanych przez obie

aplikacje i różnicy długości kodu kontrolerów. Kod kontrolerów jest zawarty w tej samej liczbie plików w obu aplikacjach testowych.

W przypadku Spring Boot liczba linii kodu widoku oraz liczba plików, w których jest on zawarty, jest znacznie większa od tej dla widoku JSF. Spring nie posiada własnego mechanizmu tworzenia widoku a tylko gotowe interfejsy, które należy podłączyć. W tym przypadku zastosowano bibliotekę AngularJS.

Liczba linii kodu modelu jest zbliżona, ponieważ obie aplikacje mają taką samą funkcjonalność.

Aplikacja Spring Boot zajmuje znacznie większą przestrzeń dyskową ponieważ posiada wbudowany kontener aplikacji Tomcat oraz bibliotekę AngularJS.

Tabela 7. Wybrane metryki kodu obu aplikacji

	Liczba linii kodu	
	Spring	**JSF**
Kod Java	599	731
Kod html	203	-
Kod JavaScript/Angular	471	-
Kod xhtml	-	378
Inne pliki konfiguracyjne	13	59
Razem linijek kodu	1286	1168
Liczba bibliotek	90	36
. Kontroler – linie kodu	97	235
Kontroler – liczba plików	3	3
Widok – linie kodu	674	378
Widok – liczba plików	23	8
Model – linie kodu	502	496
Model – liczba plików	19	13
Waga projektu (KB)	**2524**	**96**

5. Wnioski

Na podstawie przeprowadzonych badań można sformułować następujące wnioski:

- zastosowanie zarówno Spring Boot jak i JSF wymaga znajomości podstaw technologii Java oraz JEE;
- Spring jest stabilną i stale rozwijającą się platformą developerską, oferującą dość bogate możliwości przystosowania do potrzeb własnych projektów oraz predefiniowanych konfiguracji, które ułatwiają start początkującym programistom;
- JSF jest zdefiniowanym standardem platformy JEE i cieszy się oficjalnym wsparciem ze strony firmy Oracle, ale wymaga aby programista samodzielnie zadbał o każdy aspekt aplikacji (konfiguracja serwletu, konfiguracja bazy danych);
- wokół obu frameworków istnieje bogata społeczność developerska, zrzeszająca zarówno amatorów jak i profesjonalnych developerów, obie posiadają bogatą dokumentację techniczną;
- na podstawie. przeprowadzonych badań, Spring Boot, dzięki gotowym wbudowanym metodom CRUD, znacznie szybciej przeprowadza operacje na bazach danych;
- aplikacja JSF jest lżejsza, ale oferuje ograniczony potencjał rozbudowy;
- do obsługi widoków w aplikacji Spring Boot, należy wykorzystać dodatkowe technologie widoków (np. AngularJS);
- Spring Boot dostarcza gotowy kontener aplikacji w postaci wbudowanej dystrybucji Apache Tomcat.

Wyniki badań pozwalają potwierdzić postawioną tezę - Spring Boot jest korzystniejszym wyborem zarówno dla początkujących, jak i zaawansowanych programistów. Daje developerom gotowe elementy kodu, które należy zastosować do tworzonego projektu, a nie budować je od podstaw. Aplikację zbudowaną na bazie Spring Boot można dość swobodnie łączyć z innymi rozwiązaniami opartymi nie tylko o język Java.

Literatura

[1] https://zeroturnaround.com/rebellabs/most-popular-java-frameworks-tools-and-libraries-2016/ [10.11.2016]

[2] https://www.eclipse.org/ [07.12.2016]

[3] https://www.postgresql.org/ [07.12.2016]

[4] https://maven.apache.org/ [07.12.2016]

[5] https://angularjs.org/ [07.12.2016]

[6] Vishal Layka, *Java Projektowanie aplikacji WWW*, przeł. Lech Lachowski, Gliwice, Helion, 2015

[7] Vishal Layka, *Learn Java for Web Development*, Apress, 2014

[8] http://booklips.pl/zestawienia/100-ksiazek-ktore-trzeba-przeczytac-wg-polskich-dziennikarzy-i-pisarzy/ [08.11.2016]

[9] http://www.locmetrics.com/ [08.11.2016]