JCSI 2 (2016) 114-119
Journal of Computer Sciences Institute

Wysłane: 2016-11-20
Przyjęte: 2016-11-25

Biblioteki AngularJS i ReactJS - analiza wydajnościowa

Karol Kowalczyk*, Małgorzata Plechawska-Wójcik
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Artykuł zawiera porównanie wydajności bibliotek języka JavaScript. Analizie podlegają programy napisane z wykorzystaniem AngularJS oraz ReactJS. Badanie wydajności zostało zrealizowane poprzez implementację aplikacji korzystających z języka JavaScript, z użyciem obu bibliotek. Na każdej z nich wykonano pomiary wydajności, na ich podstawie sporządzono analizę wyników. Analizę wykonano przy użyciu narzędzi developerskich przeglądarek internetowych oraz poprzez zawarty w kodzie mechanizm badawczy.

Słowa kluczowe: Angular; React; wydajność

*Autor do korespondencji.

Adres e-mail: karol.kowalczyk@pollub.edu.pl

AngularJS and ReactJS libraries - performance analysis

Karol Kowalczyk*, Małgorzata Plechawska-Wójcik
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article contains a comparison of the performance of selected JavaScript libraries. The analyzed programs have been written using AngularJS and ReactJS. Performance testing was carried out through the implementation of several applications using JavaScript, with a usage of both libraries. For each of them performance measurement was made and on its basis the results analysis was prepared. The analysis was performed using the development tools and by a mechanism contained in the code.

Keywords: Angular; React; performance

*Corresponding author.

E-mail address: karol.kowalczyk@pollub.edu.pl

1. Wstęp

Aplikacje internetowe z biegiem czasu zyskują coraz większą popularność. Ich twórcy starają się jak najbardziej ułatwić sobie pracę nad nimi. Z tego powodu powstaje wiele różnorodnych frameworków oraz bibliotek. Pisanie aplikacji z ich wykorzystaniem staje się coraz mniej skomplikowane, niemniej ważna jest ich wydajność. Głównym czynnikiem wpływającym na zapewnienie komfortu pracy jest kod wykonywany po stronie przeglądarki internetowej. Wiadomym jest, że aplikacja może dostawać dane z pewnym opóźnieniem. Dobrze napisany interfejs użytkownika (ang. Frontend) może wprowadzić odczucie płynności działania strony, wczytując dane partiami na bieżąco odświeżając wyświetlaną użytkownikowi treść[1].

Analizując współczesne trendy tworzenia aplikacji, można stwierdzić, że największą popularnością cieszą się tak zwane „single-page application", czyli aplikacje zawarte na jednej stronie z dynamicznie zmieniającą się zawartością. W każdej z takich aplikacji najważniejszą technologią od strony użytkownika jest Java Script. Niestety przy nowoczesnych aplikacjach pisanie w czystym JS nie jest już opłacalne. Stosowane są różnego rodzaju frameworki, z których wyróżnić można dwa najbardziej popularne, znacząco różniące się koncepcją. Są to Angular oraz React.

Obie biblioteki oferują programiście automatyczne odświeżanie widoku po zmianie modelu. Takie udogodnienie musi wiązać się z ciągłym obserwowaniem modelu danych oraz przystosowaniem do częstej konieczności odświeżania różnych fragmentów strony internetowej. React wprowadza innowacyjne podejście opierające się na manipulowaniu na wirtualnym DOM (Document Object Model) i odświeżaniu tylko tego fragmentu strony, który rzeczywiście uległ zmianie, podczas gdy Angular wprowadza zmiany bezpośrednio[2]. Czy takie podejście rzeczywiście niesie za sobą wzrost płynności działania aplikacji? Technologia ta z powodzeniem sprawdza się w aplikacjach takich jak Facebook czy Instagram. Natomiast Angular jest wykorzystywany do tworzenia narzędzi dostarczanych przez Google [3].

Celem testów jest zbadanie obydwu bibliotek pod kątem wydajności. W Internecie można znaleźć wiele sprzecznych informacji na ten temat. Przeprowadzając testy postaram się rozwiać wszelkie wątpliwości i odpowiedzieć na pytanie „Korzystanie z której biblioteki jest korzystniejsze pod kątem wydajności?". Dodatkowo artykuł zawiera porównanie bibliotek z wykorzystaniem różnych przeglądarek internetowych.

Wnioski z analizy wyników testów Angular oraz React z pewnością przyczynią się do lepszego doboru technologii. Zbadane zostanie, która z nich lepiej się sprawdzi przy wykonaniu małych aplikacji, a która przy większych. Jaką technologię należy wybrać chcąc rozpocząć tworzenie bardziej statycznej strony, a jaką dla dynamicznej. Dzięki tej wiedzy będzie można tworzyć nową aplikację w jednej

z badanych bibliotek, bez wątpliwości czy ta druga nie byłaby lepsza [4, 5].

2. Realizacja testów

2.1. Sposób testowania

Testy wydajnościowe przeprowadzone zostały na podstawie trzech aplikacji, z których każda bada inny aspekt wykorzystania bibliotek. Podejście to daje możliwość przetestowania obu technologii w różnych, niezależnych sytuacjach. W pierwszej kolejności zbadano aplikację prezentującą na ekranie losowy ciąg wyrazów. Aplikacja została wykonana w obu technologiach. Zbadano szybkość ładowania oraz zużycie pamięci. Następnie wykonane zostały testy podstawowych funkcji Java Script, a na koniec testom poddana została aplikacja prezentująca tabelę danych o różnych rozmiarach. Zasymulowano duże obciążenie danymi, które w kolejnych testach poddano dynamicznej zmianie [6]. Poniżej przedstawiono konfigurację sprzętową maszyny testującej, na której wykonane zostały testy:

- procesor: Intel Core i7-4710HQ (4 rdzenie, od 2.50 GHz do 3.50 GHz, 6 MB cache),
- pamięć: 8 GB (SO-DIMM DDR3, 1600 MHz)
- dysk: 1000 GB SATA 7200 obrotów,
- grafika: + Intel HD Graphics 4600, NVIDIA GeForce GTX 860M,
- system operacyjny : Windows 8.1 Pro 64-bit (6.3, Build 9600) .

Wszystkie przypadki rozpatrzone zostały przy użyciu trzech najbardziej popularnych przeglądarek:

- Google Chrome wersja 54.0.2840.71,
- Mozilla Firefox wersja 49.0.2,
- Internet Explorer wersja 11.0.9600.16384.

Pomiary wykonane zostały przy użyciu narzędzi deweloperskich oraz biblioteki JSLitmus.

Narzędzia developerskie – to zestaw narzędzi dostarczany wraz z przeglądarką internetową, pozwalające programistom na testowanie oraz debugowanie elementów strony internetowej. Niektóre z nich pozwalają na dynamiczną zmianę kodu HTML, JS czy też CSS. Aktualnie każda znana przeglądarka jest w nie wyposażona.

JSLitmus - jest to narzędzie typu open source dające możliwość nie tylko badania wydajności funkcji, ale też tworzenia na tej podstawie wykresów. Jego zaletą jest niewielki rozmiar kodu źródłowego, co pozwala na instalację jedynie poprzez dodanie jednego pliku do aplikacji. Testy wykonane tą biblioteką pokazują ile razy na sekundę przeglądarka może wykonać badany kod. Takie podejście bardzo dobrze sprawdza się w charakteryzowaniu wydajności funkcji.

2.2. Badane parametry

Czas odpowiedzi – jest to główny czynnik odpowiadający za wydajność. Znacząco wpływa na komfort korzystania z aplikacji. Na czas odpowiedzi strony internetowej składają się między innymi czasy wczytywania, wykonywania skryptów oraz czasy rysowania czy też renderowania strony. Do badania tych parametrów zastosowane zostanie narzędzie „timeline" oraz opisany wcześniej JSLitmus[7].

Zużycie zasobów – zbadano pamięć, jaką zajmuje aplikacja, z wyróżnieniem pamięci potrzebnej na przechowanie obiektów Java Script oraz elementów DOM. Parametr ten jest mniej odczuwalny dla użytkownika, jednakże nie pozostaje bez znaczenia, szczególnie w przypadku dużych aplikacji. Do badania tego parametru wykorzystana została zakładka „profiles".

Rozmiar skryptu – to liczba linii kodu. Parametr bardzo ważny z punktu widzenia twórcy, ponieważ mniej linii kodu zwykle niesie za sobą mniejszy wkład programisty w implementację. Przy badaniu tego parametru pomijane są puste linie.

Płynność działania przy dużym obciążeniu danymi – z użyciem wspomnianej wcześniej aplikacji, zbadano jak radzą sobie one z prezentacją dużej liczby danych tekstowych oraz wizualnych na ekranie. Wykonane zostaną testy dla tabeli danych o rozmiarach 10x10, 20x20, 30x30 oraz 40x40. Tabela zawierać będzie losowe dane numeryczne wraz z losowym kolorem wypełnienia. Uwzględniona zostanie ich modyfikacja. Umożliwi to określenie komfortu pracy z taką aplikacją.

Działanie przy dynamicznie zmieniających się danych – dane z poprzedniego punktu zostały przegenerowane z losowym opóźnieniem o maksymalnej wartości 1000ms. Zasymulowało to pracę aplikacji przy dużej liczbie zmieniających się danych[8].

3. Badania

Pierwsza zaprezentowana tabela przedstawia analizę liczby linii kodu potrzebnych na wykonanie poszczególnych aplikacji.

1) Aplikacja wyświetlająca losowy ciąg wyrazów o wybranej przez użytkownika liczbie.
2) Aplikacja pozwalająca na sortowanie oraz wyszukiwanie elementów w tablicy o wybranym rozmiarze.
3) Aplikacja prezentująca tablicę wypełnioną losowymi danymi, umożliwiająca ich dynamiczną zmianę wraz ze zmianą koloru tła komórki.

Tabela 1. Zestawienie liczby linii kody potrzebnych na implementację

Aplikacja	Angular	React
1	36	41
2	69	77
3	134	151

Pod tym względem zdecydowanie lepszy okazał się AngularJS. Jego przewaga zwiększa się wraz ze wzrostem złożoności aplikacji.

3.1. Testy aplikacji do wyświetlania tekstu

W pierwszej kolejności wykonano testy aplikacji wyświetlającej losowy tekst na ekranie. Czas jaki przedstawia tabela, to czas jaki upłynął od momentu przeładowania strony, do czasu pojawienia się na niej tekstu.

Tabela 2. Wyniki testów aplikacji do wyświetlania tekstu – analiza czasu odpowiedzi

	Liczba danych	Google Chrome	Mozilla Firefox	Internet Explorer
Angular	100	149,8ms	83,5ms	87,3ms
	10000	185,3ms	130,5ms	96,2ms
	1000000	1930,2ms	1800,0ms	960,4ms
React	100	280,4ms	350,0ms	245,0ms
	10000	305,4ms	358,0ms	247,0ms
	1000000	2475,4ms	2320,0ms	1480,0ms

Z wyników pierwszego testu wyraźnie widać, że aplikacje napisane w Angular uruchamiają się o wiele szybciej niż React. Przyczyną wolniejszego wczytywania się strony w React jest czas wykonywania skryptów. Jak wiadomo, podczas inicjalizacji musi on bowiem stworzyć i przegenerować swój wirtualny DOM. Dodatkowym środkiem spowalniającym jest wykorzystanie typu dokumentu „text/babel". Jest to typ pozwalający na używanie HTML w plikach Java Script. Stwarza to konieczność dodatkowego parsowania kodu HTML zawartego w JS, co dodatkowo obciąża przeglądarkę podczas wczytywania.

Następnym badanym czynnikiem jest wielkość pamięci zajmowanej przez aplikację. Ponieważ w tym przypadku wielkość zajmowanej pamięci wskazana przez przeglądarkę po załadowaniu strony nie ulegała zmianie, wykonano jeden pomiar dla każdego z parametrów.

Tabela 3. Wyniki testów aplikacji do wyświetlania tekstu – analiza zużycia pamięci

	Liczba danych	Google Chrome	Mozilla Firefox	Internet Explorer
Angular	100	4,20MB	1,22MB	0,45MB
	10000	4,25MB	1,29MB	0,93MB
	1000000	5,80MB	7,95MB	5,16MB
React	100	7,72MB	6,97MB	2,16MB
	10000	8,40MB	7,03MB	2,63MB
	1000000	9,80MB	13,71MB	6,86MB

Wielkość pamięci jaką zajmują obydwie aplikacje na poszczególnych przeglądarkach różni się znacząco. Ma na to wpływ fakt, iż różne przeglądarki biorą pod uwagę różne elementy, prezentując ile pamięci zajmuje strona. Jednakże z porównania jednoznacznie widać, że Angular zajmuje zdecydowanie mniej pamięci przeglądarki dla każdego z testowanych przypadków.

3.2. Testy sortowania oraz wyszukiwania

Kolejny program posłużył do wykonania pomiarów funkcji języka Java Script. Testowanymi algorytmami były sortowanie oraz wyszukiwanie danych w tablicy. Pierwszym badanym parametrem był czas wykonywania skryptów. Oddzielnie wykonano testy sortowania oraz wyszukiwania. Pomiary wykonano z wykorzystaniem JSLitmus.

Tabela 4. Wyniki testów aplikacji do wyszukiwania – analiza liczba wykonań na sekundę

	Liczba danych	Google Chrome	Mozilla Firefox	Internet Explorer
Angular	100	31503	26402	35199
	1000	9312	11940	8809
	10000	1122	1089	550
	100000	47	98	35
React	100	30386	76420	28854
	1000	9252	28100	8782
	10000	1292	1033	545
	100000	46	78	34

Wyniki testów przedstawione w tabeli 4 pokazują, że lekką przewagę w czasie wyszukiwania danych zyskał AngularJS. Wynik ten można zaobserwować na każdej z przeglądarek oraz dla każdego badanego rozmiaru tablicy wejściowej. Można też zauważyć, że przeglądarka Internet Explorer wykonała obliczenia najwolniej dla obydwu technologii.

Tabela 5. Wyniki testów aplikacji do sortowania – analiza liczba wykonań na sekundę

	Liczba danych	Google Chrome	Mozilla Firefox	Internet Explorer
Angular	100	6209	6980	7448
	1000	725	689	1210
	10000	64	59	85
	100000	4	3	5
React	100	16359	21409	26709
	1000	1885	3650	4058
	10000	168	250	317
	100000	35	59	23

Test sortowania danych przedstawiony w tabeli 5 prezentuje nieco odmienne wyniki. Okazało się, że z tym testem znacznie lepiej poradził sobie ReactJS. Uwagę w wynikach testu przykuwa duża rozbieżność w czasach wykonania na przeglądarce Mozilla Firefox. Osiągnęła ona najgorszy wśród wszystkich przeglądarek wynik z biblioteką Angular, ale okazała się być najszybsza z React. Bardzo dobry wynik z biblioteką Angular osiągnęła przeglądarka Internet Explorer.

Zastanawiać może co spowodowało, że przy sortowaniu znacząco lepszy okazał się React, natomiast przy wyszukiwaniu lekką przewagę zyskał Angular. Stało się tak ponieważ podczas implementacji sortowania w Angular wykorzystano gotowy filtr dostarczony wraz z biblioteką, natomiast do wyszukiwania wykorzystano możliwość implementacji własnego. Gotowy filtr do sortowania dostarczony przez bibliotekę pozwala na zaawansowane sortowanie wielu rodzajów danych, nie tylko tablic. Jest on więc bardziej uniwersalny, co wpływa negatywnie na jego wydajność. Aby korzystając z AngularJS, osiągnąć wysoką wydajność aplikacji, warto używać własnych filtrów. Zwykle jednak nie sortuje się tak dużych ilości danych, aby mogło to znacząco obniżyć wydajność strony, a stosowanie uniwersalnego filtra jest o wiele prostsze.

W następnej kolejności zbadano pamięć, jaką przeglądarka przeznaczyła na aplikację w każdej z technologii.

Tabela 6. Wyniki testów aplikacji do sortowania danych – analiza zużycia pamięci

	Google Chrome	**Mozilla Firefox**	**Internet Explorer**
Angular	6,30MB	1,03MB	0,87MB
React	6,20MB	2,61MB	1,26MB

Dla Google Chrome można zaobserwować porównywalne wartości tego parametru. W przypadku pozostałych przeglądarek nieco więcej pamięci potrzebuje ReactJS. Porównując wartości zajmowanej pamięci przez aplikację z pierwszego testu z aplikacją z testu numer dwa widać, że druga aplikacja napisana w ReactJS zajmuje o połowę mniej niż pierwsza. Jest tak, ponieważ do wykonania aplikacji drugiej nie było potrzeby wykorzystania kodu html w Java Script, czyli type="text/babel". Jak widać, jego wykorzystanie znacząco wpływa na wielkość potrzebnej pamięci. Mimo to obydwie aplikacje zajmują więcej niż te pisane w AngularJS.

3.3. Testy dyrektyw

Bindowanie zmiennych jest chyba największym udogodnieniem dostarczanym przez obie biblioteki. Niegdyś programista musiał sam martwić się, aby dane przetwarzane przez kod Java Script trafiły do widoku, a widok został odświeżony. Teraz to wszystko za programistę robi biblioteka. Niestety jak wynikło z analizy, bindowanie najmocniej wpływa na wydajność aplikacji. Dodatkowo obydwie biblioteki mają różne sposoby na obsługę odświeżania widoku. Testy przeprowadzone w tym punkcie dadzą odpowiedź, która z nich lepiej sobie z tym radzi pod względem wydajności. Początkowo zostało sprawdzone działanie aplikacji obarczonej dużą liczbą powiązanych z widokiem zmiennych, następnie sprawdzono jak każda z nich zareaguje na dużą liczbę zapytań typu AJAX, z których każde zechce odświeżyć fragment widoku.

Pierwszym testem było uruchomienie aplikacji z tablicą danych 20x20, 30x30, 40x40. Zbadany został czas odpowiedzi na zmianę jednego elementu, rozmiar skryptu oraz ilość zajmowanej pamięci. Początkowo opóźnienie serwera ustawiono na 1s.

Tabela 7. Wyniki testów aplikacji do testowanie dyrektyw – analiza czasu odpowiedzi dla 1 elementu

	Rozmiar tablicy	**Google Chrome**	**Mozilla Firefox**	**Internet Explorer**
Angular	10x10	14,6ms	51,1ms	35,3ms
	20x20	48,1ms	103,1ms	64,2ms
	30x30	136,3ms	147,5ms	122,5ms
	40x40	171,6ms	193,4ms	182.0ms
React	10x10	12,9ms	12,6ms	32.3ms
	20x20	37,3ms	27,3ms	59.1ms
	30x30	80,0ms	39,3ms	87.5ms
	40x40	142,3ms	57,6ms	123,4ms

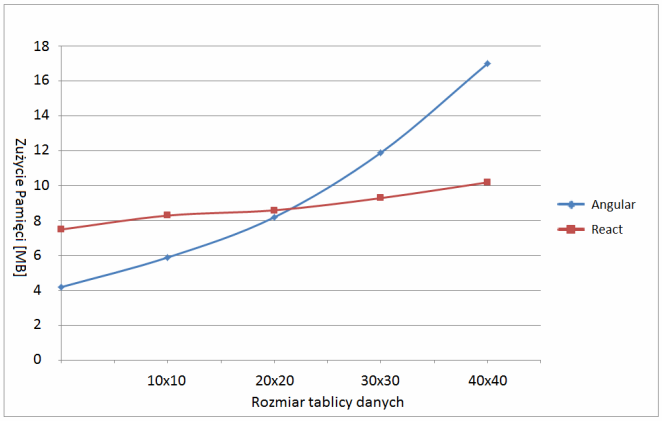
Podczas testowania zauważono, że Angular znacznie więcej czasu poświęca na wykonywanie skryptów, natomiast React na renderowanie strony. Wynika to ze wcześniej opisanych różnic w koncepcji odświeżania widoku. ReactJS najpierw operuje na wirtualnym DOM w celu obliczenia zmian w widoku, a czas poświęcony na te operacje zalicza się do czasu renderowania. Zauważyć jednak można, iż dłuższy czas renderowania komponentów przez React jest rekompensowany z nawiązką o wiele krótszym czasem wykonywania skryptów. Fakt ten potwierdza Tabela 7 przedstawiająca wyniki pomiarów czasu modyfikacji danych jednej komórki tabeli. Wyraźnie widać, że React jest pod tym względem o wiele szybszy od Angular. Można zaobserwować też jak dobrze React działa z przeglądarką Mozilla Firefox. Uzyskał on wynik ponad dwukrotnie lepszy od pozostałych przeglądarek. Potwierdza to poprzednie testy, które również stawiały tą przeglądarkę na czele szybkości działania z tą technologią. Podobnie jest ze słabszym działaniem AngularJS na tej przeglądarce. Uzyskała ona najgorszy wynik czasu odpowiedzi. Angular w tym przypadku najlepiej działał na Google Chrome. Podczas tego testu pojawił się problem z przeglądarką Internet Explorer. Aplikacja napisana w AngularJS nie uruchamiała się, co wymagało przeprowadzenia specjalnych modyfikacji kodu dla tej przeglądarki.

W kolejnym teście zbadano pamięć, jaką potrzebuje przeglądarka na wyświetlenie strony. Została ona zbadana, w momencie kiedy tablica została w pełni wypełniona danymi[9].

Tabela 8. Wyniki testów aplikacji do testowanie dyrektyw – analiza zużycia pamięci

	Rozmiar tablicy	Google Chrome	Mozilla Firefox	Internet Explorer
Angular	10x10	5,9MB	2,6MB	2,5MB
	20x20	8,2MB	5,6MB	4,4MB
	30x30	11,9MB	11,7MB	8,5MB
	40x40	17,0MB	18,7MB	14,3MB
React	10x10	8,3MB	6,5MB	2,9MB
	20x20	8,6MB	7,7MB	3,9MB
	30x30	9,3MB	9,4MB	6,0MB
	40x40	10,2MB	11,7MB	7,5MB

Analiza wyników tych pomiarów daje bardzo ciekawe rezultaty. Można zauważyć, że dla tabeli danych 10x10 wyraźnie mniej pamięci wymaga AngularJS, natomiast wraz ze wzrostem liczby elementów na stronie traci on swoją przewagę. Dla tablicy 20x20 zaobserwowano bardziej zbliżone wyniki. Sytuacja odwróciła się dla tablicy 30x30. Aplikacja w Angular dla takich danych potrzebuje nieco więcej zasobów. Podobnie dla największej badanej tablicy. Tutaj wynik zużycia pamięci przez React jest prawie dwa razy mniejszy. Dzieje się tak na każdej z przeglądarek.



Rys. 1. Wykres zużycia pamięci przez biblioteki dla przeglądarki Google Chrome

Na Rys. 1 widać jak zmienia się zapotrzebowanie na pamięć przy wzroście liczby prezentowanych elementów dla obydwu bibliotek. React odnotowuje wyraźnie mniejszą tendencję wzrostową przy zwiększaniu liczby komponentów.

Ponieważ w kolejnym etapie testowania zmienianych było wiele elementów widoku jednocześnie, zmodyfikowany został parametr odpowiedzialny za symulowanie odpowiedzi serwera. Ustawiono go na losową wartość z przedziału 0-1000ms, następnie uruchomiony został mechanizm odświeżający wszystkie elementy. Zbadano czas, jaki aplikacja potrzebowała na ukończenie tego procesu.

Tabela 9. Wyniki testów aplikacji do testowanie dyrektyw – analiza czasu modyfikacji wszystkich elementów tabeli

	Rozmiar tablicy	Google Chrome	Mozilla Firefox	Internet Explorer
Angular	10x10	1014,3ms	1006,2ms	1014,5ms
	20x20	4290,4ms	2742,5ms	2694,5ms
	30x30	19780,2ms	13457,3ms	12802,5ms
	40x40	62371,6ms	38931,4ms	37972.6ms
React	10x10	1034,9ms	1019,3ms	1039.3ms
	20x20	1292,2ms	1121,2ms	1159.2ms
	30x30	4869,0ms	1199,5ms	1485.4ms
	40x40	10642,4ms	1622,1ms	1787,6ms

Ogromną przewagę w tym teście osiągnął ReactJS. Angular kompletnie nie poradził sobie z tablicami większymi od 20x20. Na Google Chrome przy największej badanej tablicy trzeba było czekać około minuty, w przypadku gdy React wykonał to samo w 10s. W przypadku testów na innych przeglądarkach zaobserwowano brak płynności w odświeżaniu widoku. Jedyną przeglądarką, która prawidłowo prezentowała zmianę danych przy tablicach większych od 10x10, było Google Chrome. Czasy przeglądarki Firefox oraz IE są odpowiednio niższe, ponieważ dla dużej liczby danych nie odświeżały one widoku po zmianie każdego elementu. Wiele danych zmieniających się niemal jednocześnie grupowane były w jedno zdarzenie odświeżenia, co odpowiednio zmniejszyło czasy. Pomimo tego w każdej z przeglądarek znacząco lepszy okazał się React[10].

4. Wnioski

W artykule zostały porównane biblioteki Angular oraz React pod kątem wydajności. Wykonane zostały trzy aplikacje w każdej z technologii tak, aby możliwe było zbadanie bibliotek przy różnych zastosowaniach. Testy przeprowadzone zostały z wykorzystaniem trzech najpopularniejszych przeglądarek internetowych.

Ze sporządzonych analiz można wywnioskować, że odpowiedni dobór biblioteki może okazać się kluczowy jeśli chce się aby aplikacja działała możliwie najlepiej. Ciężko jednak jednoznacznie stwierdzić, która technologia jest wydajniejsza. Można natomiast wyróżnić kilka grup aplikacji.

Chcąc wykonać małą aplikację z niewielką ilością zmieniających się danych, zdecydowanie wydajniejszy okaże się Angular. Jest to biblioteka, która według testów zajmuje mniej pamięci po uruchomieniu mniejszych aplikacji. Czas

uruchamiania się stron również przemawia za tą biblioteką. Dodatkowo kody źródłowe są mniejsze, a komfort pisania aplikacji o wiele lepszy.

W przypadku średniej wielkości aplikacji, obydwie biblioteki wydają się być odpowiednie. Zużycie pamięci będzie zbliżone. Czasy odpowiedzi obydwu aplikacji dla niedużej liczby bindowanych danych również są porównywalne. Różnica zaczyna być widoczna dopiero przy ich znaczącym wzroście.

ReactJS najlepiej sprawdzi się w przypadku aplikacji z dużą liczbą danych dynamicznie zmieniających się. Jeśli chcemy napisać aplikację, w której znajdzie się powyżej 100 zmieniających się elementów, AngularJS może sobie z nią nie poradzić i wtedy zdecydowanie lepszy okaże się React. Są to jednak bardzo specyficzne przypadki.

Jeśli chodzi o przeglądarki, to można zauważyć skrajne wyniki testów przeglądarki Mozilla Firefox, która najlepiej radzi sobie z React, natomiast słabo z Angular, co również może mieć wpływ na decyzję w wyborze biblioteki.

Biorąc pod uwagę całokształt testów, można by stwierdzić, że obie biblioteki są wydajne jeśli używa się ich adekwatnie do potrzeb.

Literatura

[1] Chugh Ravi, i inni. Staged information flow for JavaScript. (2009), Conference on Programming Language Design and Implementation, 50-62.

[2] Valente Marco Tulio, Terra Ricardo i Santos Gustavo. AngularJS in the Wild: A Survey with 460 Developers. The 7th International Workshop. Amsterdam : ACM, 2016.

[3] Fedosejev Artemij. React.js Essentials. Birmingham : Packt Publishing, 2015.

[4] Green Brad i Seshadri Shyam. AngularJS. [tłum.] Robert Górczyński. Gliwice : Helion, 2014.

[5] Chansuwath Wutthichai i Senivongse Twittie. A model-driven development of web applications using AngularJS framework. 2016. 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS).

[6] Hunt Pete. React: Facebook's Functional Turn on Writing JavaScript. New York : ACM, (2016), Queue - Web, 40-57.

[7] Weyuker Elaine J. Experience with performance testing of software systems: issues, an approach, and case study. IEEE Transactions on Software Engineering 26. 2001.

[8] Khan Rizwan Performance testing (load) of webapplications based on test casemanagement. Perspectives in Science. 8, (2016), 355–357.

[9] Stępniak Wojciech i Nowak Ziemowit. Performance Analysis of SPA Web Systems. [aut. książki] Borzemski Leszek, i inni. Information Systems Architecture and Technology: Proceedings of 37th International Conference on Information Systems Architecture and Technology – ISAT 2016 – Part I. (2016), 235-247.

[10] Antonio Cássio de Sousa. Architecting Applications with Components. Pro React. (2015), 51-89.