JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 14 (2020) 73-77
Wysłane: 2019-10-14
Przyjęte: 2019-10-23

Porównanie wydajności narzędzi do tworzenia interfejsu aplikacji typu SPA na przykładzie React i Vue.js

Krzysztof Boczkowski*, Beata Pańczyk
Politechnika Lubelska, Katedra Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przeprowadzono analizę wydajności aktualnie stosowanych narzędzi do tworzenia interfejsu aplikacji typu SPA. Badanie przeprowadzono z wykorzystaniem dwóch aplikacji, o takiej samej funkcjonalności, zaimplementowanych w React i Vue.js. Do pomiaru wykorzystano narzędzia deweloperskie dostępne w przeglądarkach internetowych oraz odpowiednie implementacje własnych metod.

Słowa kluczowe: React; Vue.js; wydajność; JavaScript

*Autor do korespondencji.

Adres e-mail: krzysztof.boczkowski@pollub.edu.pl

Comparison of the performance of tools for creating a SPA application interface - React and Vue.js

Krzysztof Boczkowski*, Beata Pańczyk
Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article analyses the performance of currently used tools for creating a SPA application interface. The study was conducted using two applications with the same functionality, implemented in React and Vue.js. The tools available in web browsers and appropriate implementations of own methods were used to measure SPA performance.

Keywords: React; Vue.js; performance; JavaScript

*Corresponding author.

E-mail address: krzysztof.boczkowski@pollub.edu.pl

1. Wstęp

Obecnym trendem w budowaniu stron internetowych jest tworzenie aplikacji typu SPA (Single Page Application). SPA to aplikacja uruchamiana w przeglądarce internetowej, której ideą jest stworzenie interfejsu użytkownika o funkcjonalności zbliżonej do natywnych aplikacji systemowych [1].

Ten typ aplikacji charakteryzuje się wieloma właściwościami, dającymi przewagę nad zwykłą stroną internetową:

- aplikacja typu SPA jest wyświetlana jak zwykła aplikacja, lecz w przeglądarce aplikacje takie aktualizują widok dynamicznie, fragmentami. Każda akcja powoduje natychmiastową zmianę w wyświetlanym interfejsie. Jest to możliwe, ponieważ cała logika oraz elementy interfejsu są realizowane przez kod JavaScript [1].
- dużą zaletą jest szybsza komunikacja z serwerem - wszystkie zapytania do serwera potrzebują mniej czasu na oczekiwanie na odpowiedź. Pobierane są tylko potrzebne dane, zwykle w postaci obiektu JSON (JavaScript Object Notation) [1].

Wydajność, obok funkcjonalności, jest jedną z najważniejszych cech aplikacji. Na jej podstawie budowana jest satysfakcja użytkownika korzystającego z danego oprogramowania. Użytkownicy oczekują, że wczytywana strona wyświetli się nie dłużej niż po 2 sekundach. Jeśli wczytywanie trwa dłużej niż 3 sekundy to prawie 40% użytkowników rezygnuje z dalszego przeglądania witryny [2].

Przeglądarki internetowe podczas budowania i wyświetlania gotowego widoku aplikacji SPA realizują szereg procesów. Pierwszym z nich jest interpretacja języka JavaScript, który jest wykorzystywany do tworzenia interfejsu użytkownika. W kolejnych fazach są analizowane i obliczane style, budowany jest układ strony na bazie znaczników HTML, rysowany interfejs i na końcu ulega on komponowaniu [3].

Istnieje wiele narzędzi online umożliwiających zbadanie wydajności aplikacji internetowych m.in. PageSpeed Insights czy Webpagetest.org. Przeglądarki internetowe również posiadają wbudowane narzędzia do badania wydajności. Są to narzędzia deweloperskie pozwalające na wgląd w każdy element strony i zmierzenie czasów wykonywania poszczególnych procesów [4,5].

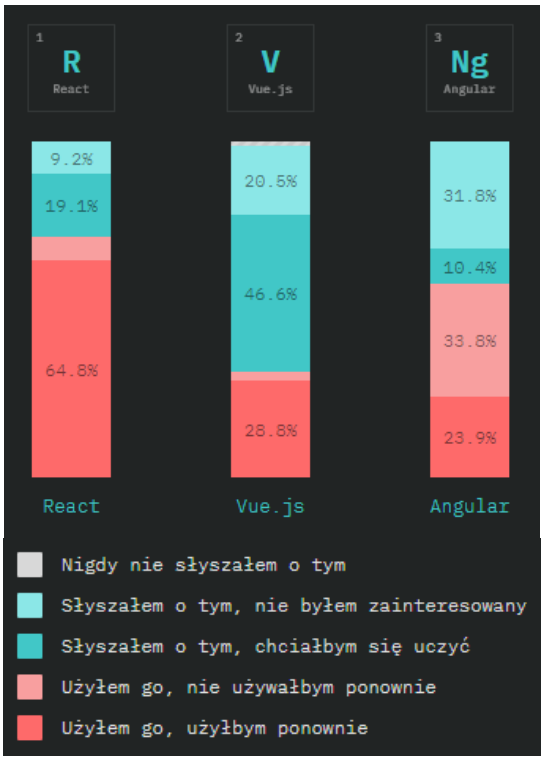
Język JavaScript posiada wbudowane narzędzia do badania wydajności kodu. Do nich zaliczamy m.in. interfejs ***performance*** dostarczający dostęp do informacji wydajnościowych wyświetlanej strony, czy interfejs ***console,*** który oprócz standardowej funkcji zapisu tekstu do konsoli

narzędzi deweloperskich umożliwia odczyt czasu pomiędzy wywołaniami metod ***time*** oraz ***timeEnd*** [6].

Celem badań przeprowadzonych w niniejszym artykule jest porównanie czasów wczytywania interfejsu użytkownika z poziomu kodu JavaScript aplikacji SPA tworzonej za pomocą biblioteki React i Vue.js, oraz wskazanie efektywniejszego rozwiązania.

2. Aktualne narzędzia do tworzenia aplikacji typu SPA

React i Vue.js należą do grona najpopularniejszych obecnie narzędzi do tworzenia interfejsów graficznych aplikacji typu SPA. Według badań przeprowadzonych dla potrzeb serwisu StateOfJs na grupie 20000 programistów z całego świata, w 2018 roku porównywane narzędzia zajęły dwa pierwsze miejsca (rys. 1) [7]. 64.8% respondentów chciałoby ponownie użyć React do budowy swoich aplikacji, w przypadku Vue było to 28.8% badanych.



Rys. 1. Wyniki badań satysfakcji programistów z 2018 roku [3]

2.1. React

React jest biblioteką do tworzenia interfejsów graficznych dynamicznych aplikacji internetowych, opartą o wirtualne drzewo DOM (Document Object Model). Za jego powstanie odpowiada Facebook, a dokładniej pracujący w nim programista Jordan Walke, który w 2012 napisał bibliotekę FaxJS - podwalinę pod React [8].

W React budowanie interfejsu opiera się o tworzenie komponentów wielokrotnego użytku, które są opisywane za pomocą JSX (JavaScript XML). To mieszanina kodu JavaScript oraz HTML (HyperText Markup Language), która pozwala na wykorzystywanie w kodzie znaczników HTML wraz z logiką. Każdy z komponentów posiada swój cykl życia składający się z etapu montowania, aktualizacji oraz usuwania. Każdy etap posiada odzwierciedlenie w odpowiednio nazwanych metodach, które umożliwiają reakcję na zmiany związane z cyklem życia komponentu [9].

2.2. Vue.js

Vue.js, podobnie jak React, jest narzędziem opartym o wirtualny DOM, do budowy warstwy wizualnej aplikacji. Jego twórcą jest Evan You, który jako pracownik Google wykorzystywał w pracy inne narzędzie - Angular. Zapragnął on stworzyć taki framework, który będzie wykorzystywał to co najlepsze w Angular oraz będzie jednocześnie łatwiejsze do wykorzystania i lżejsze. Konsekwencją tego było powstanie narzędzia Vue, opublikowanego w 2014 roku na GitHub [10].

Interfejs budowany jest tutaj z komponentów, które w swojej implementacji wykorzystują szablony HTML i logikę napisaną w JavaScript. Do przypisywania funkcjonalności i danych służą dyrektywy (podobnie jak w Angular). Analogicznie jak w React, każdy z komponentów posiada swój cykl życia i możliwa jest reakcja na jego zmianę [11].

3. Metoda badań

Do przeprowadzenia badań stworzono dwie aplikacje w języku z takim samym interfejsem użytkownika i funkcjonalnością, z wykorzystaniem opisanych w rozdziale 2 narzędzi języka JavaScript. W obu przypadkach zaimplementowano tabelę z możliwością zmiany liczby wyświetlanych wierszy. Dane do tabeli w postaci pliku JSON zostały wygenerowane za pomocą biblioteki faker.js jako tablica elementów (Przykład 1).

Przykład 1. Skrypt generujący dane testowe

```
const faker = require('faker');
const fs = require('fs');
const mockData = [];
for(let i=0; i<10000; i++){
  mockData.push({
    no: i + 1,
    firstName: faker.name.firstName(),
    lastName: faker.name.lastName(),
    hours: faker.random.number(24) + 160,
    jobTitle: faker.name.jobTitle(),
  });
}
fs.writeFile(
  `./public/report-data-mock.json`,
  JSON.stringify(mockData, null, 2),
  `utf-8`,
  (error) => {
    error
      ? console.log(error)
      : console.log(`generating finished`);
  }
);
```

Obie aplikacje uruchamiano w dwóch przeglądarkach internetowych:

- Google Chrome - v. 74.0.3729,
- Mozilla Firefox - v. 67.0.1.

Każda z przeglądarek posiada własny silnik renderowania, co może mieć wpływ na różnice w wydajności.

Testy wykonano według dwóch scenariuszy testowych:

- **Scenariusz 1**: czas renderowania komponentu tabeli dla 100, 500, 2500 i 10000 wierszy z wykonaniem pomiarów przy pomocy narzędzi deweloperskich przeglądarek. Mierzony jest czas od momentu wykrycia akcji kliknięcia myszy na przycisku nawigującym do ekranu z tabelą, do momentu zamontowania komponentu w drzewie DOM.
- **Scenariusz 2:** czas renderowania komponentu tabeli dla 100, 500, 2500 i 10000 wierszy z wykonaniem pomiarów za pomocą implementacji w metodzie cyklu życia komponentu. Ten scenariusz zakłada zbadanie wydajności przy pomocy obiektu ***console*** i jego metod: ***time*** oraz ***timeEnd*** (przykład 2 i 3).

Przykład 2. Implementacja testu wydajności w Vue

```
export default class Report extends Vue {
  constructor(props: any) {
    super(props);
    console.time("report screen render");
  }
  mounted() {
    console.timeEnd("report screen render");
  }
```

Przykład 3. Implementacja testu wydajności w komponencie React

```
class Report extends React.Component<any> {
  constructor(props: any) {
    super(props);
    console.time("report screen render");
  }
  componentDidMount() {
    console.timeEnd("report screen render");
  }
```

Podane metody obiektu ***console*** umożliwiają obliczenie czasu pomiędzy ich wywołaniami. Pozwalają na wprowadzenie identyfikatora do argumentu funkcji, który służy do wyznaczenia bloku kodu jaki będzie poddany pomiarom [12].

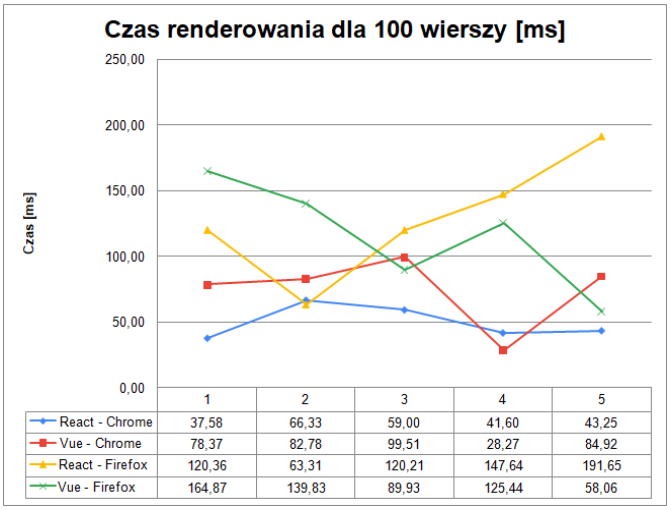
4. Platforma testowa

Badania wykonano na sprzęcie o specyfikacji:

- Procesor: Intel Core i7-7500u @ 3.5GHz, 4 rdzenie, 4 wątki;
- Pamięć: 8192MB LPDDR3 1866MHz;
- Dysk: SSD 256GB NVMe;
- System operacyjny: Antergos Linux (kernel wer. 5.1.6-arch1-1-ARCH);
- Ekran: 1920x1080;
- Układ graficzny: Intel Iris HD Graphics 620;
- Model: Lenovo Ideapad 710s-13ikb.

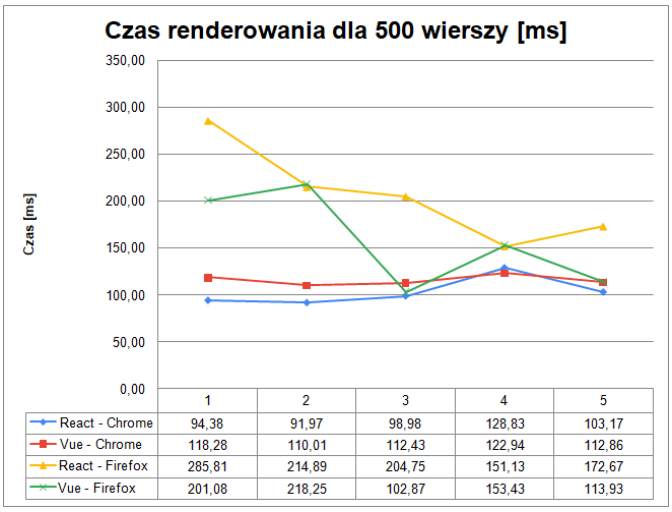
5. Wyniki badań

Wyniki testów wykonanych zgodnie z pierwszym scenariuszem przedstawiają rysunki 2-4. Rysunki 5-7 prezentują wyniki testów przeprowadzonych w drugim scenariuszu.



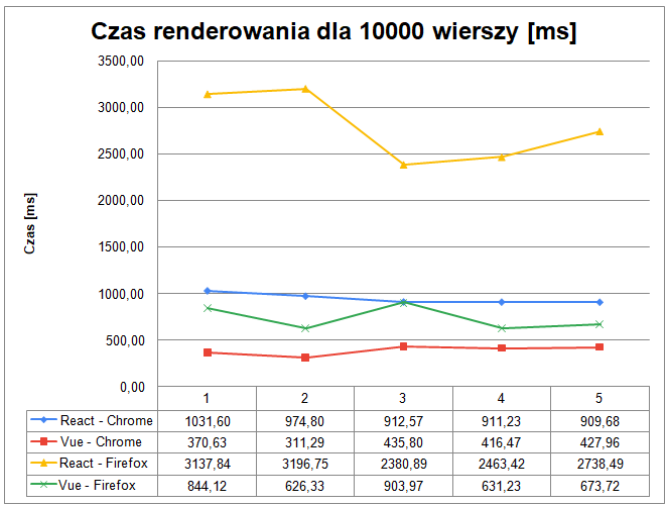
	1	2	3	4	5
React - Chrome	37,58	66,33	59,00	41,60	43,25
Vue - Chrome	78,37	82,78	99,51	28,27	84,92
React - Firefox	120,36	63,31	120,21	147,64	191,65
Vue - Firefox	164,87	139,83	89,93	125,44	58,06

Rys. 2. Wyniki pomiarów dla 100 wierszy (Scenariusz 1)



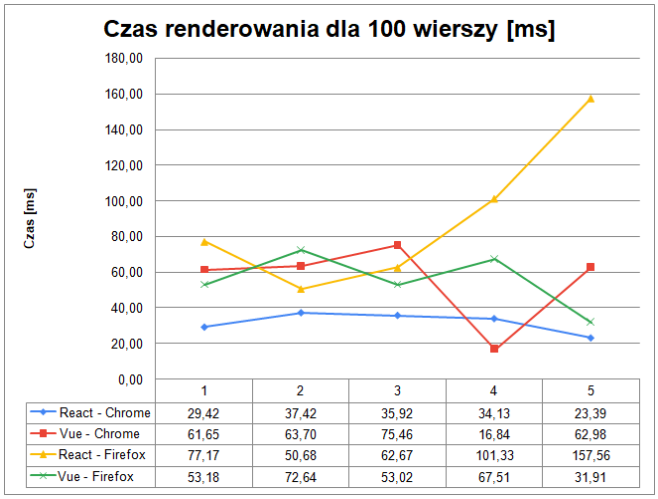
	1	2	3	4	5
React - Chrome	94,38	91,97	98,98	128,83	103,17
Vue - Chrome	118,28	110,01	112,43	122,94	112,86
React - Firefox	285,81	214,89	204,75	151,13	172,67
Vue - Firefox	201,08	218,25	102,87	153,43	113,93

Rys. 3. Wyniki pomiarów dla 500 wierszy (Scenariusz 1)



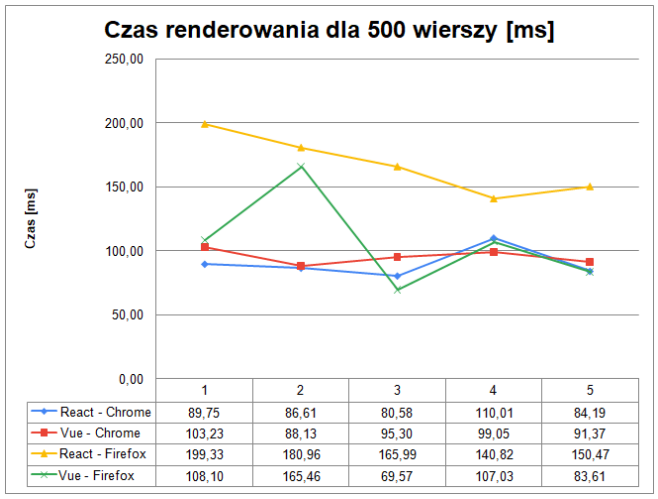
	1	2	3	4	5
React - Chrome	1031,60	974,80	912,57	911,23	909,68
Vue - Chrome	370,63	311,29	435,80	416,47	427,96
React - Firefox	3137,84	3196,75	2380,89	2463,42	2738,49
Vue - Firefox	844,12	626,33	903,97	631,23	673,72

Rys. 4. Wyniki pomiarów dla 100000 wierszy (Scenariusz 1)



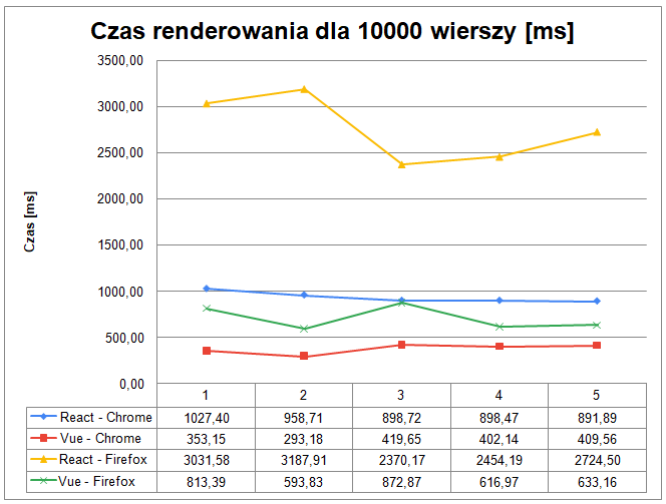
	1	2	3	4	5
React - Chrome	29,42	37,42	35,92	34,13	23,39
Vue - Chrome	61,65	63,70	75,46	16,84	62,98
React - Firefox	77,17	50,68	62,67	101,33	157,56
Vue - Firefox	53,18	72,64	53,02	67,51	31,91

Rys. 5. Wyniki pomiarów dla 100 wierszy (Scenariusz 2).



	1	2	3	4	5
React - Chrome	89,75	86,61	80,58	110,01	84,19
Vue - Chrome	103,23	88,13	95,30	99,05	91,37
React - Firefox	199,33	180,96	165,99	140,82	150,47
Vue - Firefox	108,10	165,46	69,57	107,03	83,61

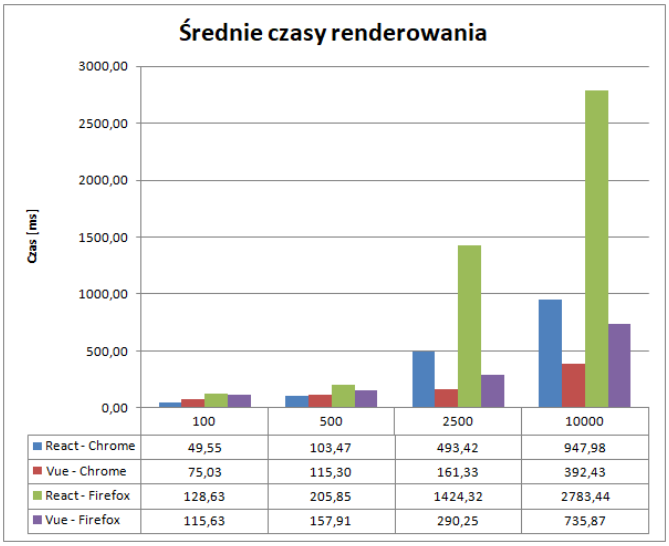
Rys. 6. Wyniki pomiarów dla 500 wierszy (Scenariusz 2).



	1	2	3	4	5
React - Chrome	1027,40	958,71	898,72	898,47	891,89
Vue - Chrome	353,15	293,18	419,65	402,14	409,56
React - Firefox	3031,58	3187,91	2370,17	2454,19	2724,50
Vue - Firefox	813,39	593,83	872,87	616,97	633,16

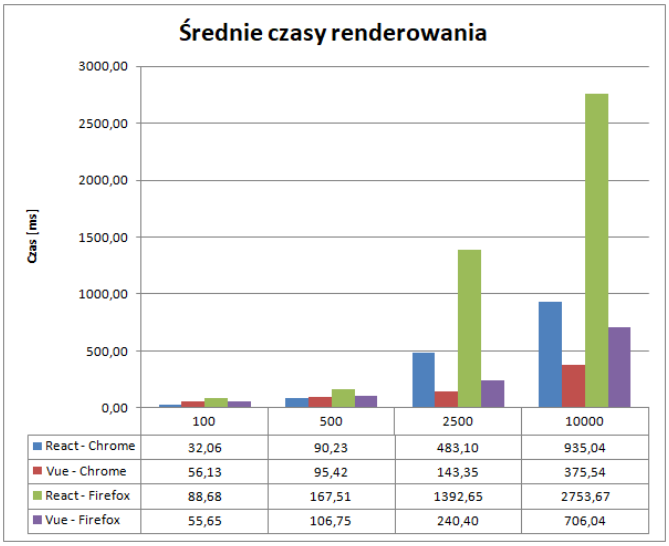
Rys. 7. Wyniki pomiarów dla 10000 wierszy (Scenariusz 2).

Średnie czasy uzyskane w wyniku pomiarów zestawiono na rysunkach 8 i 9.



	100	500	2500	10000
React - Chrome	49,55	103,47	493,42	947,98
Vue - Chrome	75,03	115,30	161,33	392,43
React - Firefox	128,63	205,85	1424,32	2783,44
Vue - Firefox	115,63	157,91	290,25	735,87

Rys. 8. Średnie czasy renderowania komponentu dla scenariusza 1.



	100	500	2500	10000
React - Chrome	32,06	90,23	483,10	935,04
Vue - Chrome	56,13	95,42	143,35	375,54
React - Firefox	88,68	167,51	1392,65	2753,67
Vue - Firefox	55,65	106,75	240,40	706,04

Rys. 9. Średnie czasy renderowania komponentu dla scenariusza 2.

6. Analiza wyników

Na podstawie uzyskanych wyników renderowania 100 wierszy nie można jednoznacznie stwierdzić, które z narzędzi w tym przypadku jest efektywniejsze. Zarówno pomiary za pomocą narzędzi deweloperskich w przeglądarce (Rys. 2), jak i te z wykorzystaniem metod cyklu życia komponentu (Rys. 5), posiadają duże odchylenia. Najlepiej widać to w przypadku przeglądarki Firefox, gdzie pierwsze pomiary dla aplikacji React są większe, niż w przypadku Vue. Bardziej wyrównane wyniki uzyskano w przeglądarce Chrome, w której lepszy okazuje się React. Należy jednak zauważyć, że na dość długi czas tworzenia widoku nie składa się jedynie montowanie i renderowanie testowanego komponentu. Różnica w czasach pomiędzy narzędziami deweloperskimi (gdzie mierzono czas od momentu wykrycia akcji myszki), a pomiarem na podstawie cyklu życia komponentu, jest zauważalna i stanowi dużą część procesu.

W przypadku tworzenia widoku z 500 wierszami, pomiary w Chrome są zbliżone (rysunek 3 i 6). W Firefox widać

większe różnice - Vue okazuje się wydajniejszy. Średni czas renderowania dla scenariusza 1 dla Vue wynosi 157,91 ms, a React potrzebuje już 205.85 ms na obsługę akcji kliknięcia myszą i zbudowanie widoku (Rys. 8). Czas renderowania komponentu w React, stanowi ponad 130% czasu w Vue. Stosunek ten zaczyna wzrastać w przypadku renderowania samego komponentu (Scenariusz 2) do 160% (Rys. 9).

Podczas renderowania 10000 wierszy jest już zauważalna znacząca różnica pomiędzy React i Vue. Widać też różnice pomiędzy przeglądarkami. Chrome wydaje się być stabilniejsza i lepiej zoptymalizowana. Pomiary dla 10000 wierszy charakteryzują się mniejszym procentowym błędem pomiarowym (Rys. 3,4,7,8) i wyraźniej widać różnicę w pomiarach.

7. Wnioski

Wyniki badań pokazują różnice w wydajności pomiędzy narzędziami w wyświetlaniu bardzo rozbudowanych interfejsów. React przewyższa nieznacznie wydajnością Vue w przypadku komponentów z małą liczbą wierszy. Ulega to jednak zmianie jeśli budowany widok ma większą liczbę elementów. Wtedy Vue.js jest wydajniejsze.

Różnice między przeglądarkami są wyraźne. Na podstawie wyników pomiarów aplikacji testowych, w przypadku Chrome silnik uruchomieniowy języka JavaScript jest wydajniejszy i jego kod jest bardziej optymalny w stosunku do jego odpowiednika w Firefox.

Aplikacje powstałe na potrzeby badań, dobrze pokazują, że duży wpływ na wydajność ma liczba prezentowanych danych na stronie, ale również przeglądarka, z której korzysta końcowy użytkownik. Dobrym zwyczajem programistów front-endu jest tworzenie widoków, które nie wyświetlają zbyt wielu informacji jednorazowo. Wówczas różnice w czasie renderowania komponentów nie będą zbyt wielkie w różnych przeglądarkach, niezależnie od tego, czy aplikacja SPA korzysta z React czy Vue.js.

Literatura

[1] E. Scott, SPA Design and Architecture: Understanding Single Page Web Applications, Manning Publications, 2015.

[2] J. Wagner, Web Performance in Action: Building Faster Web Pages, Manning Publications, 2017.

[3] https://developers.google.com/web/fundamentals/performance/rendering [20.10.2019]

[4] https://crystallize.com/blog/frontend-performance-measuring-and-kpis [20.10.2019]

[5] https://developers.google.com/web/tools/chrome-devtools/evaluate-performance/ [20.10.2019]

[6] https://ourcodeworld.com/articles/read/144/measuring-the-performance-of-a-function-with-javascript-using-browser-tools-or-creating-your-own-benchmark [20.10.2019]

[7] http://2018.stateofjs.com/front-end-frameworks/overview/ [08.10.2019]

[8] http://www.education-ecosystem.com/guides/programming/react-js/history [08.10.2019]

[9] http://pl.reactjs.org/docs/getting-started.html [08.10.2019]

[10] http://www.freecodecamp.org/news/between-the-wires-an-interview-with-vue-js-creator-evan-you-e383cbf57cc4/ [08.10.2019]

[11] http://vuejs.org/v2/guide/ [08.10.2019]

[12] Ch. Adams, Mastering JavaScript High Performance, Packt Publishing, 2015.