JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 25 (2022) 345–349
Received: 28 June 2022
Accepted: 12 September 2022

Comparative of React and Svelte programming frameworks for creating SPA web applications

Porównanie szkieletów programistycznych React i Svelte do tworzenia aplikacji internetowych typu SPA

Sebastian Dubaj*, Beata Pańczyk

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

Abstract
The aim of the article is to perform a comparative analysis of two programming frameworks on the example of one of the most popular currently React v17 and Svelte, which is gaining considerable popularity. Two applications with the same functionalities in both analyzed frameworks were implemented to conduct the research. The analysis mainly concerns the rendering times of views, but the application structures and their size are also compared. As a result of the research, it was found that the Svelte application is much more efficient compared to the React application.

Keywords: React; Svelte; performance; comparative analysis

Streszczenie
Celem artykułu jest przeprowadzenie analizy porównawczej dwóch szkieletów programistycznych na przykładzie jednego z najpopularniejszych obecnie React v17 oraz Svelte, który zdobywa znaczną popularność. Do przeprowadzenia badań zaimplementowano dwie autorskie aplikacje o takich samych funkcjonalnościach w obu analizowanych szkieletach programistycznych. Analiza dotyczy przede wszystkim czasów renderowania widoków, ale porównywane są także struktury aplikacji oraz ich rozmiar. W wyniku badań stwierdzono, że aplikacja Svelte jest znacznie bardziej wydajna w porównaniu do aplikacji React.

Słowa kluczowe: React; Svelte; wydajność; analiza porównawcza

*Corresponding author
Email address: **sebastian.dubaj@pollub.edu.com** (S. Dubaj)

1. Wstęp

Zapotrzebowanie na strony internetowe ciągle rośnie, wiąże się z tym ciągłe poszukiwanie i tworzenie nowych narzędzi do ich tworzenia. Początkowo portale internetowe przedstawiały jedynie treści, bez możliwości interakcji z użytkownikiem. Były tworzone jedynie aplikacje statyczne za pomocą plików HTML, kaskadowych arkuszy styli oraz języka programowania JavaScript. Wraz ze wzrostem popularności Internetu i coraz większych wymagań użytkowników, powstają nowe rozwiązania zwiększające efektywną interakcję z użytkownikiem.

W przeszłości jedynym rozwiązaniem na tworzenie serwisów internetowych były aplikacje typu MPA (ang. Multi Page Application), za pomocą którego można tworzyć aplikacje wielostronicowe. Tego typu aplikacje działały w trybie żądanie-odpowiedź przesyłane w trybie synchronicznym, co wiąże się z przeładowywaniem od początku każdej strony. To z kolei powoduje wolniejsze ładowanie stron, pobieranie danych oraz wyświetlaniem treści, co wpływa na długi czas oczekiwania i gorsze doświadczenie użytkownika przy korzystaniu z aplikacji MPA [1].

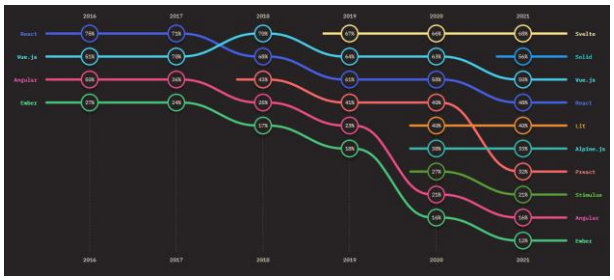
Konkurencyjność stron oraz dbanie o to, aby użytkownik pozostał na stronie jak najdłużej, chętnie na nią wracał oraz miał przyjazne doświadczenia, zmusiła do stworzeniu nowego mechanizmu aplikacji – aplikacji typu SPA [2] (ang. Single Page Application). Obecnie jest to najpopularniejszy sposób na tworzenie aplikacji internetowych. Pozwala na przyjemniejsze korzystanie ze stron, głównie przez to, że strona nie musi się przeładowywać kilka razy, a w związku z tym jej działanie jest znacznie szybsze, ponieważ treść oraz dane są pobierane w tle (asynchronicznie).

Tworzenie coraz bardziej skomplikowanych aplikacji powodowało przedłużenie procesu ich powstawania, dlatego pojawiły się szkice programistyczne (ang. frameworks). Za pomocą gotowych szkiców, aplikacje można tworzyć znacznie wydajniej co wpływa na szybkość prac nad projektem. Z biegiem lat powstawały nowe szkielety programistyczne (np. Angular, React, Vue.js, Preact, Svelte) [3], które różnią się strukturą i zasadą działania, ale wszystkie bazują na języku JavaScript i żądaniach asynchronicznych. Każdy szkic programistyczny ułatwia tworzenie aplikacji, lecz ich liczba utrudnia podjęcie decyzji, który należy wybrać. Szkice różnią się między innymi szybkością renderowania widoków, wydajnością, strukturą kodu i poziomem trudności samej implementacji.

2. Cel i hipotezy badawcze

Celem badań jest porównanie dwóch szkieletów programistycznych do tworzenia aplikacji internetowych typu SPA. Analiza dotyczy React v17, który obecnie jest najpopularniejszym frameworkiem oraz Svelte, który jest nowoczesnym rozwiązaniem i w ostatnim

czasie zaczął zdobywać popularność i według statystyk znajduje się na 4 miejscu po względem używalności (Rys. 1). Podstawowymi kryteriami porównania narzędzi są: wydajność, metryki kodu, struktura aplikacji, rozmiar aplikacji, łatwość nauki.



Rysunek 1: Popularność frameworków Javascript w roku 2021 [3].

Postawiona została następująca hipoteza badawcza: *„Aplikacja Svelte 3.0 szybciej wykonuje operacje na komponentach graficznego interfejsu użytkownika w porównaniu do analogicznej aplikacji React 17.0”.*

3. Przegląd literatury

Istnieje wiele artykułów porównujących biblioteki JavaScript. Dotyczą one głównie najpopularniejszych rozwiązań takich jak React, Vue czy Angular [4-6]. Natomiast artykułów dotyczących porównania Svelte oraz React jest zaledwie kilka. Jednym z nich jest artykuł [7], w którym autor porównuje narzędzie Angular oraz Svelte. Przeprowadzone badania dotyczą wydajności dwóch takich samych aplikacji napisanych z wykorzystaniem wskazanych narzędzi. Na podstawie wyników autor wywnioskował, że aplikacja Svelte była znacznie wydajniejsza, a w jednym ze scenariuszy Svelte był aż 4-krotnie szybszy w renderowaniu komponetów.

Artykuł [8] dotyczy bibliotek React, Vue, Angular oraz Svelte. Autor powierzchownie opisuje każdy z frameworków, a do badań dotyczących metryki kodu, wydajności, rozmiaru aplikacji, wykorzystuje cztery takie same aplikacje napisane z wykorzystaniem każdego narzędzia. Zgodnie z wnioskami autora tej pracy, Svelte jest najlepszy pod względem liczby linii kodu, natomiast w przypadku wydajności Svelte jest na drugim miejscu, a React na ostatnim.

4. React

React [9] jest najpopularniejszą i często wykorzystywaną biblioteką języka programowania JavaScript, przeznaczoną do tworzenia interfejsów użytkownika. Została stworzona w 2013 roku przez programistów platformy społecznościowej Facebook, w celu usprawnienia budowy serwisów internetowych.

Biblioteka słynie z wielu koncepcji i rozwiązań, między innymi jest to **wirtualne drzewo DOM** (ang. Virtual Document Object Model). Podejście to pozwala na wyszukiwanie różnic zachodzących w aplikacji, porównując zmiany między wirtualnym i prawdziwym DOM, następnie aktualizuje zmiany w komponentach, które tego wymagają, zamiast przeładowywać całą stronę.

Kolejną ze szczególnych cech React jest język skryptowy JSX (JavaScript XML), który jest rozszerzeniem języka JavaScript o możliwość wstawiania znaczników HTML do kodu JavaScript. JSX znacznie poprawia czytelność kodu do poziomu kodu napisanego w HTML.

5. Svelte

Svelte [10-13] jest biblioteką języka JavaScript, która dopiero zdobywa popularność. Została stworzona w 2016 roku przez Richa Harrisa. Svelte w dosłownym tłumaczeniu oznacza elegancki, szczupły, co idealnie opisuje narzędzie, ponieważ aplikacja Svelte zajmuje mało pamięci, a kod źródłowy zawiera stosunkowo mało linii kodu w przeciwieństwie do jego konkurentów.

Główną cechą Sveltepowrót do korzystania z rzeczywistego DOM (ang. Real DOM), który w przeciwieństwie do React nie tworzy wirtualnego DOM. Real DOM jest tworzony podczas kompilacji aplikacji do kodu JavaScript i służy do aktualizowania danego komponentu, gdy został on zmieniony.

Dużą zaletą Svelte jest fakt, że posiada niski próg wejścia, to znaczy, że początkujący programiści będą potrafili szybko nauczyć się jego działania.

6. Metoda badań

Do przeprowadzenia badań stworzono dwie proste aplikacje testowe w React oraz Svelte z identycznymi komponentami interfejsu użytkownika, o takiej samej funkcjonalności. Każda aplikacja służy do tworzenia losowych użytkowników, którzy posiadają zdjęcie, imię, nazwisko, wiek, numer telefonu. Wszystkie dane zostały zapisane w pliku JSON, z którego aplikacje pobierają losowe dane. Dane zostały pobrane z ogólnodostępnego i darmowego API [14], a przykładowy obiekt z danymi użytkownika przedstawiono na listingu 1.

Listing 1: Struktura danych pobranych z randomAPI

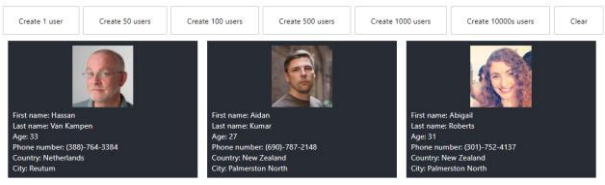
```
{
 "results": [
  {
   "gender": "male",
   "name": {
    "title": "mr",
    "first": "brad",
    "last": "gibson"
   },
   "location": {
    "street": "9278 new road",
    "city": "kilcoole",
    "state": "waterford",
    "postcode": "93027",
    "coordinates": {
     "latitude": "20.9267",
     "longitude": "-7.9310"
    },
    "timezone": {
     "offset": "-3:30",
     "description": "Newfoundland"
    }
```

```
    },
    "email": "brad.gibson@example.com",
    "login": {
      "uuid": "155e77ee-ba6d-486f-95ce-0e0c0fb4b919",
      "username": "silverswan131",
      "password": "firewall",
      "salt": "TQA1Gz7x",
      "md5": "dc523cb313b63dfe5be2140b0c05b3bc",
      "sha1": "7a4aa07d1bedcc6bcf4b7f8856643492c191540d",
      "sha256":
"74364e96174afa7d17ee52dd2c9c7a4651fe1254f471a78bda0190135dcd3480"
    },
    "dob": {
      "date": "1993-07-20T09:44:18.674Z",
      "age": 26
    },
    "registered": {
      "date": "2002-05-21T10:59:49.966Z",
      "age": 17
    },
    "phone": "011-962-7516",
    "cell": "081-454-0666",
    "id": {
      "name": "PPS",
      "value": "0390511T"
    },
    "picture": {
      "large": "https://randomuser.me/api/portraits/men/75.jpg",
      "medium":
"https://randomuser.me/api/portraits/med/men/75.jpg",
      "thumbnail":
"https://randomuser.me/api/portraits/thumb/men/75.jpg"
    },
    "nat": "IE"
  }
 ],
 "info": {
  "seed": "fea8be3e64777240",
  "results": 1,
  "page": 1,
  "version": "1.3"
 }
}
```


Obie aplikacje na początkowej stronie zawierają elementy:

- przycisk do tworzenia 1 użytkownika,
- przycisk do tworzenia 50 użytkowników,
- przycisk do tworzenia 100 użytkowników,
- przycisk do tworzenia 500 użytkowników,
- przycisk do tworzenia 1000 użytkowników,
- przycisk do tworzenia 10000 użytkowników.

Widok aplikacji po wygenerowaniu losowych użytkowników przedstawiono na Rysunku 3.



Rysunek 2: Widok aplikacji z wygenerowanymi użytkownikami.

Aplikacje uruchomiono na przeglądarce Google Chrome w wersji: 97.0.4692.71 (64-bitowa).

Badanie wydajności polegało na sprawdzeniu czasów renderowania żądanej liczby komponentów. Jeden jeden komponent odpowiada jednemu wygenerowanemu użytkownikowi.

Do badań wykorzystano stanowisko badawcze o następującej specyfikacji:

- procesor: Intel Core i5-7200U 2.5Ghz,
- pamięć: 8GB RAM,
- dysk: 240GB SSD,
- karta graficzna: GeForce 940MX 2GB,
- system operacyjny: Windows 10 Home,
- model: Acer Aspire F5-573G.

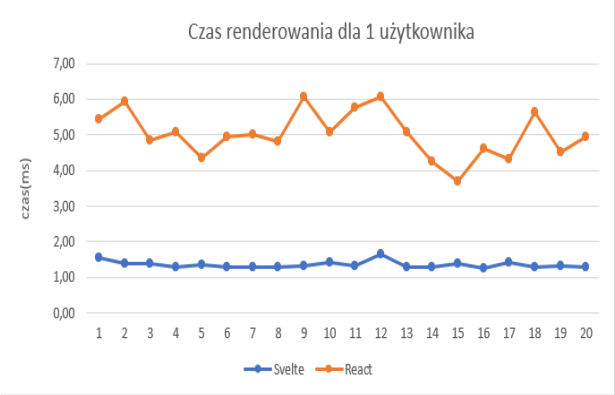
Testy wykonano według scenariuszy:

- Scenariusz 1 – pomiar czasu renderowania komponentów dla 1, 50, 100, 500, 1000 oraz 10000 użytkowników z wykorzystaniem metod console.time oraz console.timeEnd.
- Scenariusz 2 – czas wyszukiwania jednego użytkownika spośród 50, 100, 500, 1000 oraz 5000.

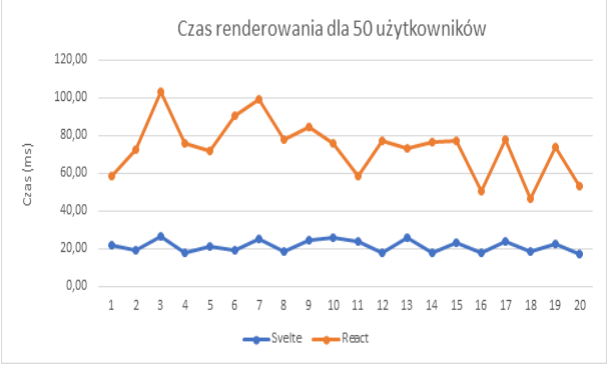
Ponadto porównano liczbę linii kodu, liczby folderów i plików oraz rozmiar gotowych aplikacjach.

7. Wyniki badań

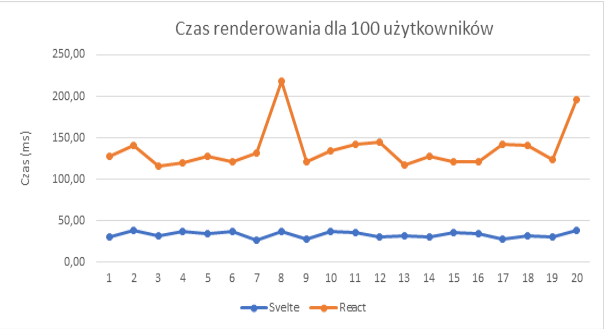
Wyniki dla scenariusza 1 przedstawione zostały na Rysunkach 3-8.



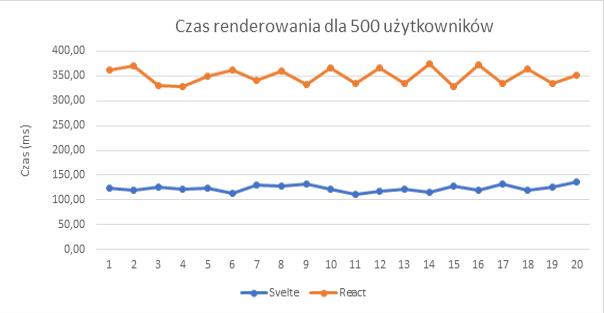
Rysunek 3: Wykres z czasami renderowania dla 1 użytkownika.



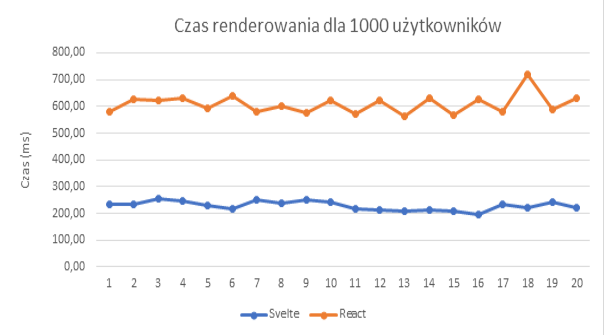
Rysunek 4: Wykres z czasami renderowania dla 50 użytkowników.



Rysunek 5: Wykres z czasami renderowania dla 100 użytkowników.



Rysunek 6: Wykres z czasami renderowania dla 500 użytkowników.

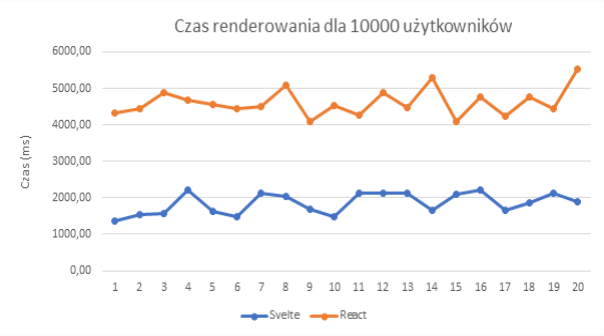


Rysunek 7: Wykres z czasami renderowania dla 1000 użytkowników.

Średnie wyniki przedstawia Tabela 1.

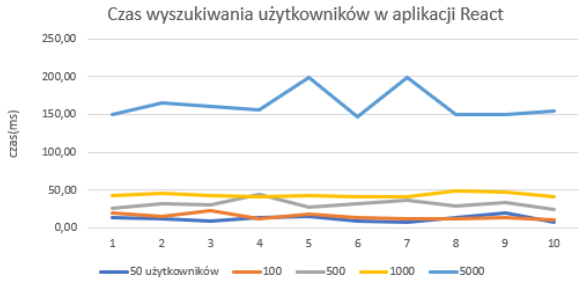
Tabela 1: Średnie czasy renderowania

Średnie czasy renderowania [ms]						
Liczba użytkowników	1	50	100	500	1000	10000
Svelte	1.36	21	33	123	228	1851
React	5	73	136	350	608	4617

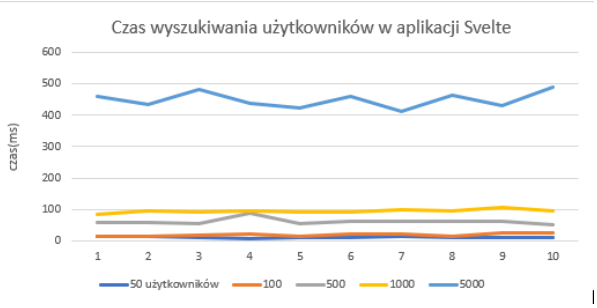


Rysunek 8: Wykres z czasami renderowania dla 10000 użytkowników.

Wyniki pomiarów dla scenariusza 2 przedstawiono na Rysunkach 9-10.



Rysunek 9: Czasy wyszukiwania użytkowników w aplikacji React.



Rysunek 10: Czasy wyszukiwania użytkowników w aplikacji Svelte.

Średnie czasy wyszukiwania użytkowników w obu aplikacjach przedstawiono w Tabeli 2.

Tabela 2: Średnie czasy wyszukiwania

Średnie czasy wyszukiwania [ms]					
Liczba użytkowników	50	100	500	1000	5000
Svelte	12	15	32	44	163
React	12	20	62	95	448

Metryki kodu i rozmiar obu aplikacji przedstawiono w Tabeli 3.

Tabela 3: Metryka kodu dla aplikacji React oraz Svelte

	React	Svelte
Liczba linii kodu w folderze src	381	317
Liczba linii kodu we wszystkich plikach	39652	7708
Liczba plików JavaScript	9	4
Liczba plików z rozszerzeniem .svelte	0	5
Liczba plików we wszystkich folderach	18	16
Rozmiar aplikacji	616 kB	352 kB

8. Analiza wyników

Analizując wyniki przedstawione na Rysunkach 3-8 można stwierdzić, że wydajność w generowaniu losowych użytkowników jest znacznie lepsza w przypadku aplikacji Svelte. Dla każdego przypadku ze scenariusza wykonano po 20 pomiarów i w żadnym z nich React nie był lepszy. Na wykresach można zauważyć, że w przypadku React pomiary nie były jednakowe, pojawiały się znaczne odchylenia pomiędzy testami, co oznacza, ze framework nie działa zbyt stabilnie. W przypadku Svelte nie było tak wielkich odchyleń, jedynie przy generowaniu 10000 użytkowników, pojawiały się różnice

odchyleń o maksymalnie 1 sekundę, natomiast odchylenia dla React wynosiły ponad 1,5 sekundy.

W przypadku scenariusza 2 badano czas wywoływania funkcji wyszukiwania użytkowników z wybranym imieniem. Wyniki były nieco inne niż w przypadku scenariusza 1, ponieważ to aplikacja React była wydajniejsza. W pomiarach wyszukiwania wśród 50, 100, 500 lub 1000 użytkowników były niewielkie, natomiast w przypadku wyszukiwania dla 5000 użytkowników, różnica czasów wynosiła średnio 284 ms, co daje wynik 3 razy lepszy dla aplikacji React.

Z Tabeli 3 wynika, że lepsze wyniki uzyskał Svelte, ponieważ w każdym przypadku zawiera mniej linii kodu, nie licząc przypadków z liczbą plików z rozszerzeniem .svelte. Aplikacja React korzysta tylko z plików JavaScript. Liczba linii kodu miała wpływ też na rozmiar aplikacji Svelte rozmiar aplikacji. Aplikacja React jest prawie 2 razy większa niż Svelte.

9. Wnioski

Na podstawie uzyskanych wyników z wykorzystaniem prostych aplikacji testowych, można stwierdzić, iż Svelte jest znacznie wydajniejszy od React dla przypadku renderowania komponentów. Dla liczby komponentów (100), aplikacja Svelte była nawet o 400% wydajniejsza niż aplikacja React. Przy większej liczbie komponentów (10000), aplikacja Svelte była szybsza o ponad 2 sekundy, czyli aż o 155%. Porównując odchylenia standardowe wyników można również stwierdzić lepszą stabilność aplikacji Svelte, ponieważ w wynikach dla aplikacji React wartości były bardziej rozproszone.

Aplikacja React była wydajniejsza dla przypadku wyszukiwania a największa różnica czasowa wynosiła średnio 300ms.

Liczba linii kodu oraz rozmiar aplikacji były znacznie mniejsze dla Svelte.

W artykule [4] autor analizował wydajność bibliotek Angular oraz Svelte porównując między innymi czasy renderowania komponentów w obu aplikacjach. We wnioskach wskazano, że aplikacja Svelte była 4-krotnie szybsza od aplikacji Angular. W niniejszym artykule porównywano React i Svelte a wnioski także wskazują na lepszą wydajność renderowania komponentów w przypadku Svelte.

React oraz Svelte są wydajnymi narzędziami, jednak w kwestii renderowania komponentów, Svelte jest wyraźnie wydajniejsze, co potwierdza założoną hipotezę.

Literatura

[1] Single Page Application (SPA) vs Multi Page Application (MPA): Pros and Cons, http://merehead.com/blog/single-page-application-vs-multi-page-application/, [23.05.2022].

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

[3] Popularność frameworków JavaScript w 2021 roku, https://2021.stateofjs.com/en-US/libraries/front-end-frameworks, [07.05.2022].

[4] R. Nowacki, M. Plechawska-Wójcik, Analiza porównawcza narzędzi do budowania aplikacji Single Page Application – AngularJS, ReactJS, Ember.js, Journal of Computer Sciences Institute 2 (2016) 98-103.

[5] J. Wróbel, Porównanie narzędzi do tworzenia aplikacji typu SPA na przykładzie Ember i React, Journal of Computer Sciences Institute 11 (2019) 145-148.

[6] S. Aggarwal, Modern Web-Development Using ReactJS, International Journal of Recent Research Aspects, 5(1) (2018) 133-137.

[7] G. Białecki, B. Pańczyk, Performance analysis of Svelte and Angular applications, Journal of Computer Sciences Institute 19 (2021) 139-143.

[8] Xu Wenquing, Benchmark Comparison of JavaScript Frameworks, M.Sc. Computer Science Interactive Digital Media, 2021.

[9] Dokumentacja React, https://reactjs.org/, [05.11.2021].

[10] Dokumentacja Svelte, https://svelte.dev/docs, [05.11.2021].

[11] O. Therox, Svelte i TypeScript, https://svelte.dev/blog/svelte-and-typescript, [05.11.2021].

[12] S. Kołodziejczak, Svelte – wszystko co powinieneś wiedzieć o nowej wersji tego narzędzia, https://geek.justjoin.it/svelte-frontend, [05.11.2021].

[13] T. Tolliday, Getting Acquainted With Svelte, the New Framework on the Block, https://css-tricks.com/getting-acquainted-with-svelte-the-new-framework-on-the-block/, [05.11.2021].

[14] Generator losowych użytkowników, https://randomuser.me/, [18.05.2022].