

WYŻSZA SZKOŁA INFORMATYKI I ZARZĄDZANIA „COPERNICUS” WE WROCŁAWIU

WYDZIAŁ INFORMATYKI, ADMINISTRACJI I FIZJOTERAPII

Kierunek studiów: **Informatyka**

Poziom studiów: **Studia drugiego stopnia-inżynierskie**

Specjalność: **Inżynieria systemów informatycznych**

PRACA DYPLOMOWA INŻYNIERSKA

Daniel Słaby

Analiza porównawcza nowoczesnych rozwiązań technologicznych aplikacji SPA za pomocą jednakowej aplikacji zaimplementowanej w każdym z wybranych rozwiązań

**Comparative analysis of modern technological solutions of SPA
applications using the same application implemented in each of the
selected solutions**

Ocena pracy:
(ocena pracy dyplomowej, data, podpis promotora)

.....
(pieczętka uczelni)

Promotor:

Dr inż. Grzegorz Debita

WROCŁAW 2020

Spis treści

1 Wstęp	3
1.1 Wprowadzenie	3
1.2 Cel pracy	3
1.3 Motywacja	3
1.4 Zakres	4
 Część przeglądowa	 5
2 Część przeglądowa	6
2.1 Słowniczek Pojęć	6
2.2 Przegląd prac o podobnej tematyce	6
2.2.1 Comparison of Single-Page Application Frameworks	6
2.2.2 Lighthouse	9
2.2.3 Porównanie narzędzi do tworzenia aplikacji typu SPA na przykładzie Angular2 i React	10
 Część praktyczna	 12
3 Część praktyczna	13
3.1 Problem pomiaru wydajności aplikacji SPA	15
3.2 Specyfikacja wymagań	16
3.3 Architektura rozwiązania	16
3.3.1 Projekt testu wydajności	16
3.3.2 Budowa aplikacji	16
3.3.3 Przygotowanie aplikacji do badania	16
3.3.4 Konteneryzacja aplikacji	16
3.3.5 Moduł badania	16
3.3.6 Moduł analizy	16
3.4 Analiza wyników	16
3.4.1 Czas załadowania aplikacji	16
3.4.2 Dodanie 1000 wierszy na początku listy	16
3.4.3 Dodanie 1000 wierszy na końcu listy	16
3.4.4 Zamiana wszystkich wierszy na liście	16
3.4.5 Aktualizacja 500 wierszy	16

Spis treści

3.4.6 Zamiana miejscami dwóch wierszy	16
3.4.7 Usunięcie pojedynczego wiersza	16
3.4.8 Usunięcie wszystkich wierszy	16
4 Podsumowanie	31
Bibliografia	32
Spis rysunków	36
Oświadczenie o udostępnianiu pracy dyplomowej	38
Oświadczenie autorskie	39

1. Wstęp

1.1 Wprowadzenie

Technologia front-end ewoluje w zastraszającym tempie. 20 października 2010 rozpoczęła się nowa era technologii webowych wraz z wydaniem biblioteki AngularJS. Był to swoisty początek aplikacji SPA (Single Page Application) i całkowitej zmiany podejścia do tworzenia webowych aplikacji klienckich.

Dzisiaj większość narzędzi jakie używamy w życiu codziennym albo w pełni znajduje się w przeglądarce albo ma już na niej swoje odpowiedniki. Znaczna część aplikacji z których korzystamy na co dzień wykorzystuje te narzędzia jako rdzeń swojego działania. Ze znanych aplikacji z jakich korzysta na co dzień użytkownik możemy wymienić jako przykład odpowiednio: React: Facebook, Instagram, Netflix, WhatsApp Angular 2: GitHub Community Forum, Microsoft Office Home, Google Marketing Platform VueJS: FontAwesome, ChatWoot, Moonitor, Leave Dates

1.2 Cel pracy

W pracy tej, przeanalizowano trzy najpopularniejsze rozwiązania do tworzenia aplikacji SPA. Każde z nich pozwala na stworzenie takiej samej aplikacji z punktu widzenia końcowego użytkownika. Każda z nich jednak w inny sposób rozwiązuje problemy zarządzania zasobami oraz optymalizacji renderowania. W tym celu skonstruuje narzędzie pozwalające na badanie dynamicznych aplikacji webowych i postaram się odpowiedzieć na pytanie, czy różnica wydajności pomiędzy frameworkami jest tak duża, że powinna mieć wpływ na wybór konkretnego rozwiązania w ogólnym przypadku?

1.3 Motywacja

Za motywacją do stworzenia tejże pracy stały dwa główne powody. Pierwszym z nich jest chęć porównania trzech najczęściej wybieranych narzędzi w środowisku front-end. Każde z nich rozwiązuje ten sam problem, jednak nie jest do jasne, jak wygląda ich wydajność w przypadku użytkowania aplikacji. Jest to bowiem problem który trudno zbadać, co prowadzi nas do drugiego powodu. Badanie aplikacji typu SPA w momencie ich działania jest niezwykle trudnym zadaniem, dlatego też, celem praktycznym pracy jest stworzenie narzędzia do rozwiązania tego problemu.

1.4 Zakres

Zadanie będzie polegać na zaprojektowaniu odpowiedniej aplikacji która zawiera kilka znanych problemów wydajnościowych występujących w świecie aplikacji SPA. Następnie implementacja takiej aplikacji w każdym z wybranych rozwiązań. Kolejno na specjalnie stworzonej maszynie wirtualnej z ograniczonymi zasobami, wykonane zostaną testy wydajnościowe za pomocą wbudowanego profilera.

Mając już konkretne dane, wyniki zostaną przeanalizowane porównując które rozwiązanie jest najwydajniejsze w konkretnym przypadku co pozwoli na odpowiedź na pytanie: czy różnica wydajności pomiędzy narzędziami jest tak duża, że powinna mieć wpływ na wybór konkretnego rozwiązania w ogólnym przypadku?

Część przeglądowa

2. Część przeglądowa

- JSON [1] (JavaScript Object Notation) - jest to tekstowy format wymiany danych, bazującym na podzbiorze języka JavaScript. JSON przechowuje dane określone przez standard ECMA-262 3-cia edycja - Grudzień 1999.
- Transpilacja [2] - Transpilatory lub kompilatory typu źródło-źródło to narzędzia, które odczytują kod źródłowy napisany w jednym języku programowania i wytwarzają równoważny kod w innym języku. Języki, które piszesz, które są transpilowane na JavaScript, są często nazywane językami komplikacji do JS i mówi się, że są ukierunkowane na JavaScript.
- SPA [3] (także Single Page Application) - jednostronna aplikacja internetowa, czyli taka, która posiada tylko jeden plik HTML. Taka aplikacja nie przeładowuje strony w trakcie użytkowania. Może w tym celu korzystać z technologii AJAX lub innych dostępnych w przeglądarkach internetowych. Logika aplikacji SPA napisana jest w JavaScript lub w języku transpilowanym do języka JavaScript Framework.

2.1 Słowniczek Pojęć

2.2 Przegląd prac o podobnej tematyce

W części tej zajmiemy się przeglądem dostępnych rozwiązań oraz prac dotyczących pomiaru wydajności aplikacji internetowych z naciskiem na aplikacje typu SPA.

2.2.1 Comparison of Single-Page Application Frameworks

Pierwszą pozycją jest praca autorstwa Pan Eric'a Molina z instytutu KTH Royal Institute of Technology w Sztokholmie [4]. Autor stara się porównać frameworki na pełnej płazczyźnie ich złożoności. W jego pracy znaleźć możemy pełen spis kryteriów oceny frameworków wraz z wagami przypisanymi do cech określonymi za pomocą wywiadu środowiskowego jaki autor przeprowadził na grupie 10 profesjonalnych deweloperów mających doświadczenie w tych zagadnieniach przedstawionych na rysunku 2.1.

W pracy tej, porównywany frameworkami są AngularJS, Angular2 oraz React. I tutaj wyróżnić możemy już pierwsze różnice pomiędzy pracami. Praca Pana Molina została wykonana w 2016 roku. JavaScript jest jednym z najszybciej rozwijających się oraz najczęściej spotykanych technologii webowych na świecie [5]. Przez okres 4 lat, firmy masowo porzuciły technologie AngularJS na rzecz Angular 2 oraz Reacta. Dodatkowo, w międzyczasie do wyścigu dołączył Vue.js [6]. Przez ten okres, także implementacja tychże rozwiązań znacznie dojrzała. Angular z wersji 2 ewoluował już do wersji 9 [7]. Także React ewoluował z wersji

Criteria	Frequency
Cache performance	2
Compatibility	4
Developer guidelines	3
Documentation	6
Does “in-house” experience exists?	2
Simple to use	6
Maintainability	5
Maturity	4
Modularity	3
Performance	5
Popularity	8
Portability	4
Scalability	3
Security	2
Framework’s size	1
Testability	3
Who is the developer?	4

Rysunek 2.1. Grafika przedstawiająca ocenę wpływu wybranego elementu na atrakcyjność danego narzędzia [4]

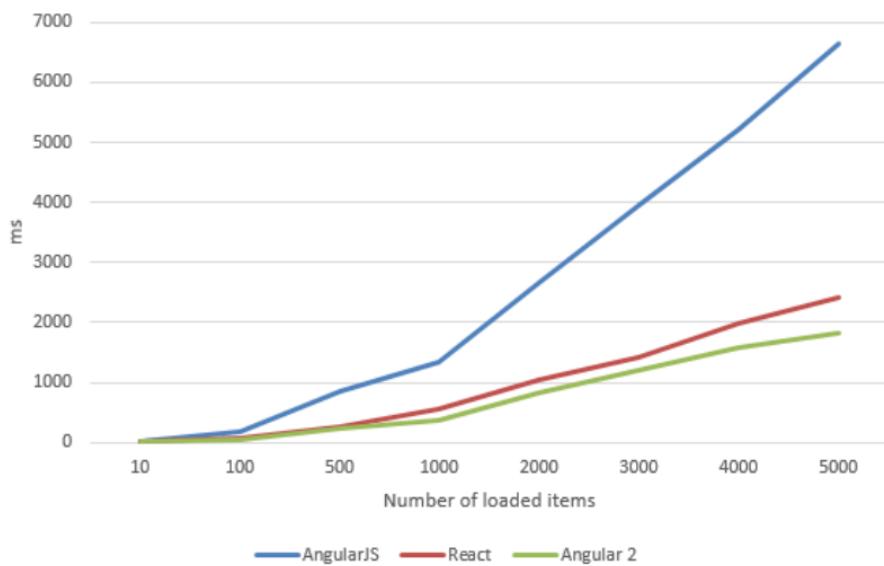
0.14.8 (użyta w pracy Pana Molina) do wersji 16.13.1 [8]. Najważniejszą zmianą którą należy tutaj wspomnieć jest zmiana silnika reacta nosząca nazwę React Fiber która nastąpiła w wersji 16.0.0. Był to kamień milowy, i z punktu widzenia badania wydajności, jest to całkowicie nowe rozwiązanie.

Następną sekcję która jest dla nas interesująca jest sekcja 5.2 w której autor prezentuje wyniki swoich testów. Wartym zauważenia jest wersja przeglądarki chrome, która wynosi 50.0.26661.102m. Najnowsza wersja chrome, w czasie pisania tego tekstu, to wersja 80.0.3987.16 czyli aż o 30 wersji nowszej.

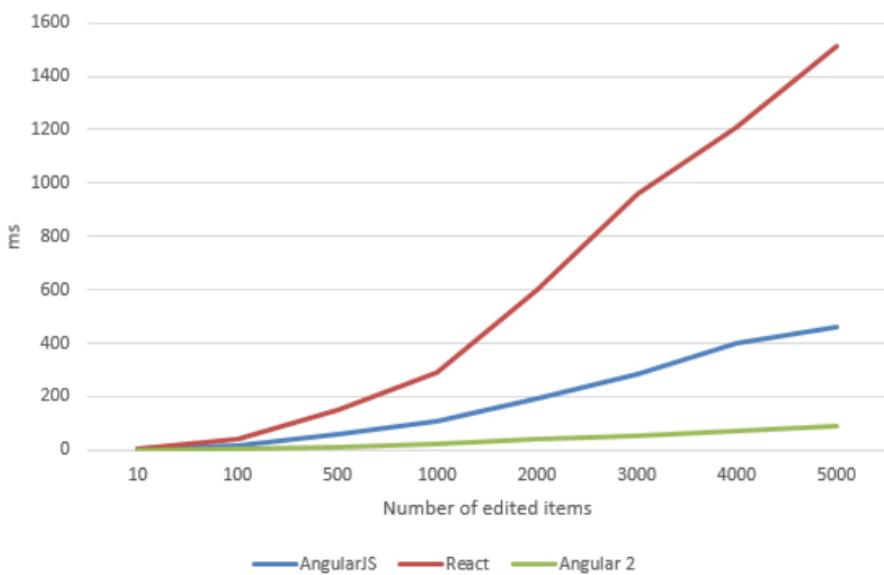
W pierwszym teście, autor porównuje wydajność pomiędzy rozwiązaniami na przykładzie ładowania elementów od 10 aż do 5000 przedstawionym na rysunku 2.2.

Z badania tego, możemy zauważyć jak bardzo AngularJS różni się od pozostałych, nowszych rozwiązań. Do pierwszych 100 elementów, wyniki są zbliżone, natomiast powyżej tej wartości, mamy nagłą zmianę stopnia nachylenia funkcji. Przyczyną takiego zachowania jest fakt, iż AngularJS nigdy nie był zoptymalizowany dla takich przypadków użycia podczas, gdy React oraz Angular2 już tak. W drugim teście, autor bada czas potrzebny do załadowania określonej ilości elementów. Wyniki przedstawiono na rysunku 2.3.

2. Część przeglądowa

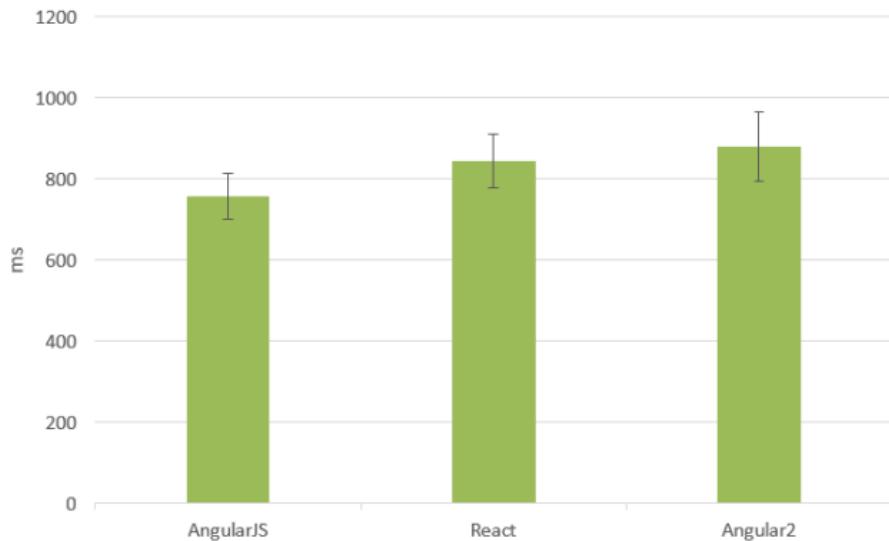


Rysunek 2.2. Grafika przedstawiająca wzrost czasu ładowania elementów od 10 do 5000 przy użyciu AngularJS, Angular2 oraz React [4]



Rysunek 2.3. Grafika przedstawiająca wykres zależności czasu do ilości edytowanych elementów na stronie [4]

Widzimy znaczną dysproporcję pomiędzy każdym z rozwiązań. Z jednej strony AngularJS przypomina funkcję wykładniczą gdzie po przeciwej stronie Angular2 zachowuje się jak funkcja liniowa. Pokazuje to jak bardzo istotne zmiany zaszły pomiędzy zmianą generacji oraz wpływ doświadczenia jakie zebraliśmy w okresie panowania frameworka AngularJS. Wszak AngularJS był pierwszym podejściem do generycznego narzędzia dla dynamicznych aplikacji internetowych. W ostatnim teście wydajnościowym, autor porównuje czas ładowania się aplikacji. Wyniki przedstawiono na rysunku 2.4.



Rysunek 2.4. Grafika przedstawiająca wykres czasu ładowania aplikacji [4]

O ile autor faktycznie porównuje wydajność rozwiązań, nie określa on w sposób klarowny jak zaimplementowano mechanizm dodawania oraz aktualizacji elementów w pierwszym oraz drugim teście. Są to informacje bardzo istotne, gdyż każdy z frameworków działa w inny sposób, i ma to bezpośredni wpływ na otrzymane wyniki. Dodatkowo, autor nie podaje błędu pomiaru [9] który wyznaczył podczas przeprowadzania badania. Jest to bardzo istotna informacja, gdyż frameworki takie opierają się na całej piramidzie warstw oprogramowania które współdzielą zasoby komputera z innymi procesami. Ponadto auto nie zastosował żadnej warstwy abstrakcji nad systemem operacyjnym co powoduje, że wyniki mogą znacznie różnić się pomiędzy różnymi wersjami systemu.

2.2.2 Lighthouse

Jest to narzędzie open-source stworzone przez firmę Google do badania wydajności stron internetowych oraz aplikacji webowych [10]. Jest ono integralną częścią przeglądarki Chrome, tak więc jest ono niezwykle łatwe w użytkowaniu. Narzędzie to pozwala nam na ocenę naszej aplikacji w kilku wybranych kategoriach w tym wydajności. Dodatkowo, narzędzie pozwala nam na określenie trybu badania gdzie do wyboru mamy tryb mobilny oraz tryb użytkownika komputera domowego. Ma to bardzo duży wpływ na wynik testu, gdyż wydajność urządzeń mobilnych jest z reguły bardzo ograniczona w porównaniu do pełnoprawnej platformy domowej. Metrykami jakie możemy otrzymać to:

- Czas rozpoczęcia pierwszego renderowania
- Czas gdy strona faktycznie została wyrenderowana w całości
- Indeks szybkości - czas, gdy główna treść strony została wyrenderowana
- Czas do interakcji - czyli czas, gdy użytkownik faktycznie może zacząć interakcję ze stroną

Jednak o ile informacje te są jak najbardziej przydatne z punktu widzenia ładowania strony, narzędziem to nie jest w stanie określić faktycznego czasu interakcji ze stroną. Wpływ na taki czas ma wiele czynników z których najważniejszym jest ilość elementów na stronie. Jest to istotne, gdyż JavaScript jest językiem jednowątkowym z mechanizmem event loop [11]. Tak więc jesteśmy w stanie otrzymać informacje o wydajności ładowania aplikacji przez przeglądarkę, jednak narzędziem w żaden sposób nie wspiera pomiarów działania aplikacji po etapie ładowania, a właśnie na tym skupia się ta praca.

2.2.3 Porównanie narzędzi do tworzenia aplikacji typu SPA na przykładzie Angular2 i React

Praca Pań Jadwiga Kalinowska oraz Beata Pańczyk [12] także ma na celu porównanie aplikacji typu SPA, tym razem Angular2 oraz React. Autorzy przedstawiają szereg metryk według których porównywały będą obydwa narzędzia. Pierwszą metryką jest sama struktura aplikacji czyli jak wygląda architektura wewnętrzna rozpatrywanego frameworka. I jak autorki zauważają, obydwa rozwiązania czerpią garściami z nowoczesnej szkoły tworzenia wysoce skalowalnych i wydajnych rozwiązań implementując naturę tworzenia złożonych aplikacji z małych komponentów. Z komponentów takich składamy coraz to większą i bardziej złożoną aplikację. Drugą metryką rozpatrywaną przez autorki jest metryka kodu przedstawiona na rysunku 2.5. Innymi słowy, jak wiele kodu boilerplate należy napisać, aby osiągnąć ten sam efekt. Ma to wpływ przede wszystkim na końcowy rozmiar pliku który należy przesyłać pomiędzy serwerem a klientem. Im niższa wartość, tym lepiej.

		React	Angular
Komponent dodawania plików graficznych	Rozmiar plików	3,43 KB	3,16 KB
	Liczba plików	1	4
	Liczba linii kodu	103	107
Rozmiar biblioteki		240MB	339MB
Rozmiar całego projektu		241MB	340MB

Rysunek 2.5. Porównanie metryk kodu dla pojedynczego komponentu [12]

Wnioskiem płynącym z otrzymanych danych jest fakt, iż React jest finalnie znacznie mniejszym frameworkiem co bezpośrednio redukuje koszt tworzenia oraz utrzymania kodu. Ostatnim interesującym punktem jest metryka badająca bezpośrednią wydajność aplikacji przedstawiona na rysunku 2.6. Badanie miało na celu zmierzenie czasu, jaki potrzebny jest aby pobrać rekordy z przygotowanej bazy danych.

Interesującym wynikiem jest fakt, iż React w przeglądarce Google Chrome dla 1,5 oraz 10 wyników otrzymał bardzo zbliżone czasy bliskie błędowi pomiaru. Jednak im większa liczba

Liczba rekordów	Angular			React		
	Czas transferu [s]			Czas transferu [s]		
	Google Chrome	Mozilla Firefox	Microsoft Edge	Google Chrome	Mozilla Firefox	Microsoft Edge
1	7,74	12,79	13,48	6,96	7	8,17
5	8,96	13,22	24,73	7,03	7,53	11,35
10	11,78	15,44	35,66	7,08	7,99	14,26
25	15,4	18,75	66,47	12,94	12,88	52,71
50	21,63	23,49	107,15	14,87	18,59	76,1
75	25,34	28,97	152,6	22,88	23,49	94,36

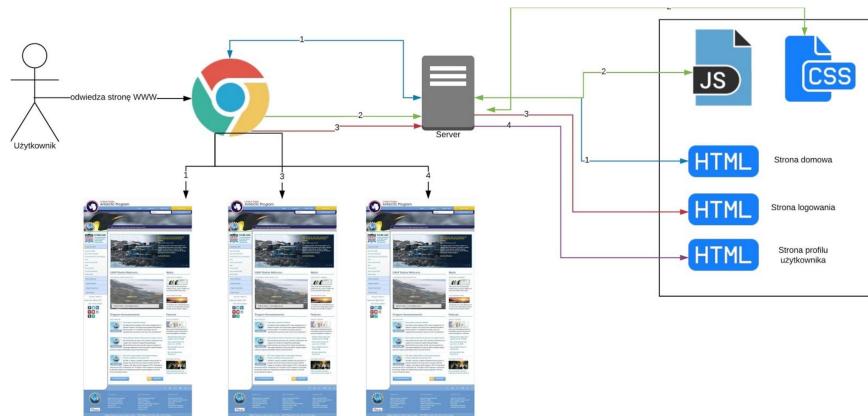
Rysunek 2.6. Porównanie czasów pobrania danych z bazy [12]

rekordów pobrana z bazy, tym większą przewagę osiągał React. Wszystkie metryki które zostały zastosowane podczas porównania dobrze obrazują różnicę pomiędzy tymi rozwiązaniami, ale tylko w ujęciu wysoko poziomowym. Badanie zawiera w sobie serwer WWW oraz bazę danych, których czasy stanowią znaczną część czasu uzyskanego w wynikach. Nie pozwala to nam na jednoznaczne stwierdzenie, które narzędzie jest wydajniejsze, gdyż wykonano tylko jedno badanie, które nie pozwala na ocenę tych narzędzi pod kątem różnych przypadków użycia.

Część praktyczna

3. Część praktyczna

Cykl życia standardowej strony internetowej przedstawiono na rysunku 3.1



Rysunek 3.1. Ilustracja przedstawiająca cykl życia statycznej strony internetowej

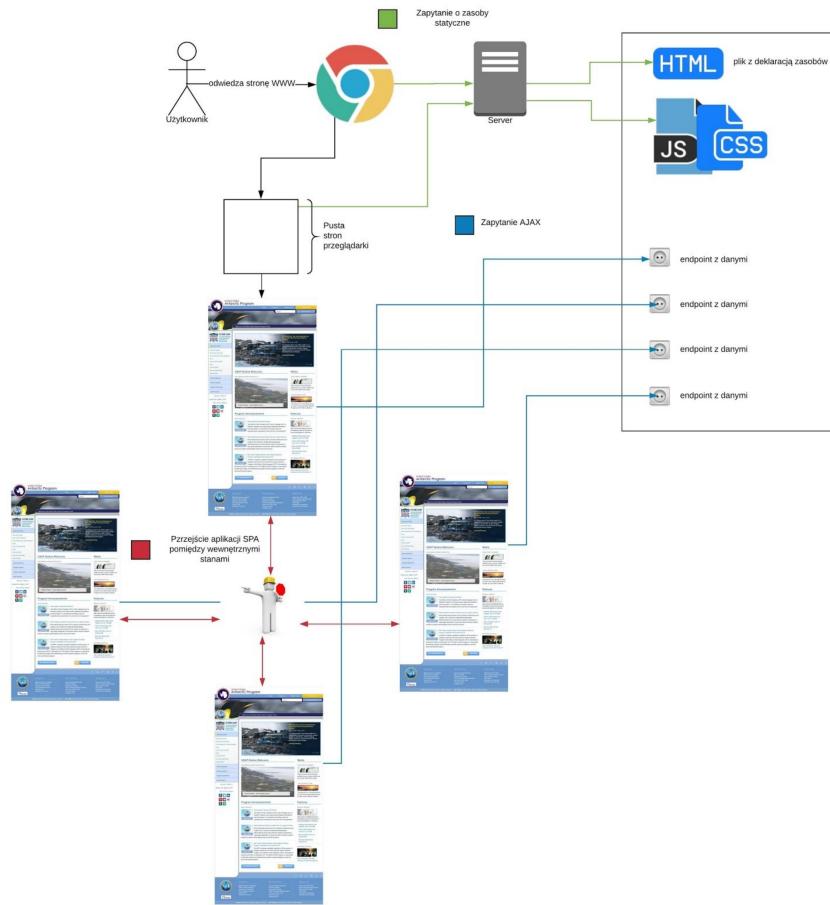
- Użytkownik otwiera przeglądarkę i odwiedza stronę WWW.
- Przeglądarka wysyła zapytanie do serwera z żądaniem zasobów.
- Serwer odsyła plik HTML w którym znajdują się odnośniki do plików JavaScript oraz CSS.
- Przeglądarka pobiera pliki JavaScript oraz CSS
- Przeglądarka renderuje stronę dla użytkownika.
- Użytkownik kliknie w odnośnik do kolejnej strony (w przykładzie strona logowania)
- Przeglądarka pobiera cały nowy plik HTML
- Przeglądarka renderuje nową stronę od nowa.
- Przeglądarka ponownie pobiera dodatkowe pliki JavaScript oraz CSS (zakładamy, że przeglądarka nie przechowuje plików w pamięci podręcznej).
- Przeglądarka renderuje stronę od nowa.

Jak widzimy, cały proces odwiedzenia dwóch stron w tej samej domenie wymaga każdorazowego przeładowania strony oraz pobrania tych samych danych. Proces ten wymaga ciągłego pobierania plików HTML, pobierania plików dodatkowych co obciąża procesor, pamięć łącze oraz sam serwer.

Dla porównania, przedstawiona poniżej ilustracja pokazuje cykl działania aplikacji SPA (rysunek 3.2)

- Użytkownik odwiedza tę samą stronę .
- Przeglądarka pobiera plik HTML z definicją potrzebnych zasobów. Zazwyczaj sekcja body pliku jest pusta.
- Przeglądarka pobiera dodatkowe pliki JavaScript oraz CSS.

3. Część praktyczna



Rysunek 3.2. Ilustracja przedstawiająca cykl działania aplikacji SPA

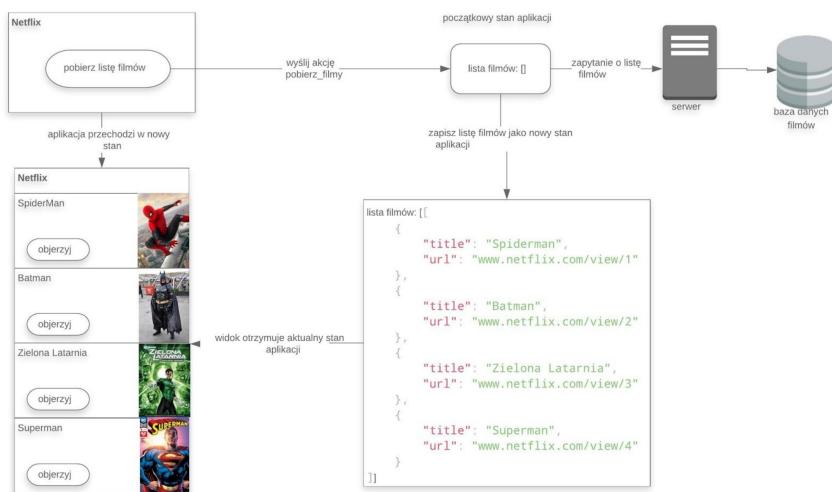
- Często pliki te są zoptymalizowane poprzez kompresję i sklejone w jeden plik tak, aby wykonać tylko jedno zapytanie zamiast dwóch.
- Pliki JavaScript, generują treść strony HTML.
- Przeglądarka może (ale nie musi) wykonać dodatkowe zapytanie o dane do jednego z endpointów serwera.
- Użytkownik klik na odnośnik do kolejnej strony. Przeglądarka nie pobiera już dodatkowego pliku HTML, jako, że aplikacja jest dynamiczna. Skrypt JavaScript może, ale nie musi pobrać dodatkowych informacji z serwera.

Jak widzimy, wykonano znacznie mniej zapytań do serwera. Dodatkowe zapytania które skrypt może wykonać w celu pobrania danych zazwyczaj są znacznie mniejsze od pełnego pliku HTML z wybraną treścią. Dodatkowo, przeglądarka nie przeładowuje strony przy każdej zmianie podstrony, co także odciąża procesor jak i łącze sieciowe.

3.1 Problem pomiaru wydajności aplikacji SPA

Z racji, że w aplikacjach SPA nigdy nie następuje przeładowanie strony, a ich działanie jest ciągłe pojawia się problem następującej natury.

W celu ułatwienia zrozumienia problemu posłużę się przykładem przedstawionym na rysunku 3.3. Założmy że jesteśmy na platformie Netflix. Otworzyliśmy stronę z listą dostępnych filmów do obejrzenia.



Rysunek 3.3. Grafika przedstawiająca mechanizm działania aplikacji dynamicznych

Aplikacja w stanie początkowym nie posiada żadnych filmów. Użytkownik kliknie przycisk "pobierz listę filmów". Efektem ubocznym (side effect) jest wysłanie informacji o akcji do menedżera stanu. Menedżer stanu wysyła zapytanie do serwera o listę filmów do zaproponowania. Serwer odpowiada z danymi, i menedżera stanu przechodzi w nowy stan. Podstrona otrzymuje informacje o nowym stanie. Pobiera nowy stan z menedżera stanu i prze renderuje widok.

Problem pojawia się gdy chcemy określić kiedy cała procedura została zakończona. Samo sprawdzenie czy przycisk został naciśnięty nie jest wystarczające, gdyż z perspektywy użytkownika, procedura obejmuje także rezultat efektu ubocznego. Zapytanie do serwera zajmie od kilkudziesięciu milisekund do nawet i kilku czy kilkunastu sekund. Samo renderowanie także nie jest deterministyczny w kontekście czasu operacji gdyż wpływ na nie mają inne usługi korzystające z procesora czy pamięci komputera. Z racji, iż frameworki internetowe muszą być generyczne, nikomu jeszcze nie udało się stworzyć idealnego i jednolitego mechanizmu wyznaczania, czy dana akcja została zakończona czy nie.

Tak więc na potrzeby pracy istotnym jest, wprowadzenie jednoznacznej definicji akcji w aplikacji.

Akcja aplikacji - jest to zbiór procedur oraz zdarzeń i ich obsługi następujący w chwili interakcji użytkownika z aplikacją.

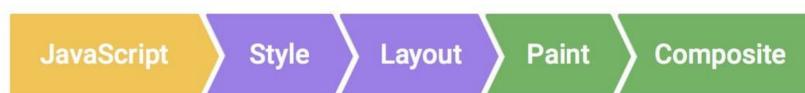
3.2 Specyfikacja wymagań

3.3 Architektura rozwiązania

- 3.3.1 Projekt testu wydajności
- 3.3.2 Budowa aplikacji
- 3.3.3 Przygotowanie aplikacji do badania
- 3.3.4 Konteneryzacja aplikacji
- 3.3.5 Moduł badania
- 3.3.6 Moduł analizy

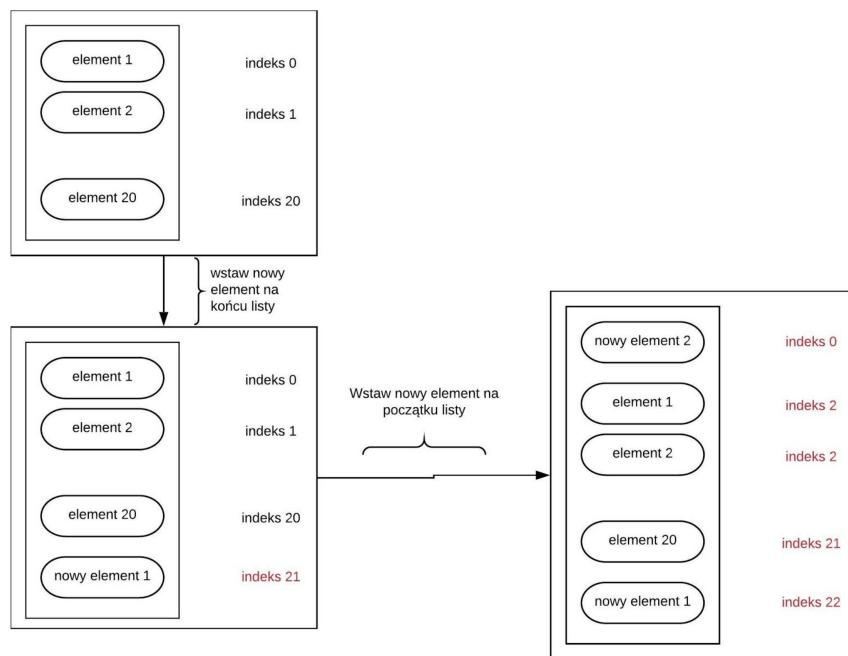
3.4 Analiza wyników

- 3.4.1 Czas załadowania aplikacji
- 3.4.2 Dodanie 1000 wierszy na początku listy
- 3.4.3 Dodanie 1000 wierszy na końcu listy
- 3.4.4 Zamiana wszystkich wierszy na liście
- 3.4.5 Aktualizacja 500 wierszy
- 3.4.6 Zamiana miejscami dwóch wierszy
- 3.4.7 Usunięcie pojedynczego wiersza
- 3.4.8 Usunięcie wszystkich wierszy

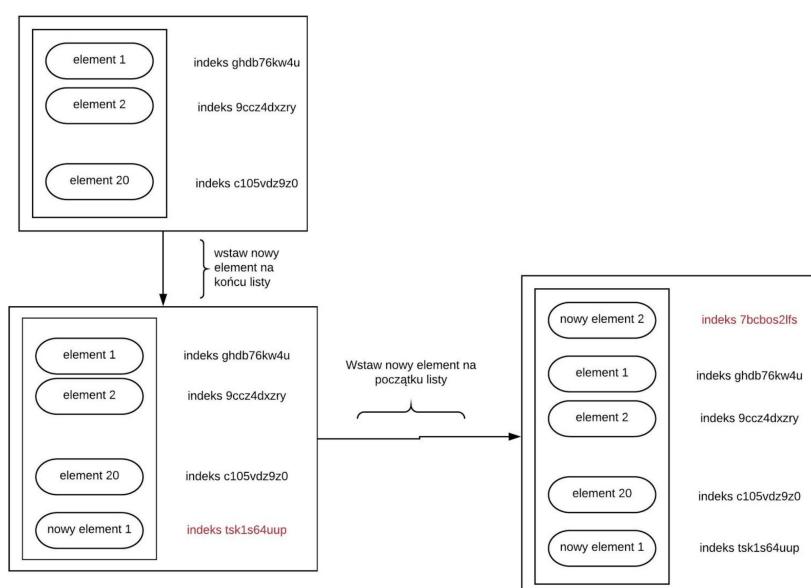


Rysunek 3.4. Grafika przedstawiająca kolejność faz renderowania w przeglądarce

3. Część praktyczna

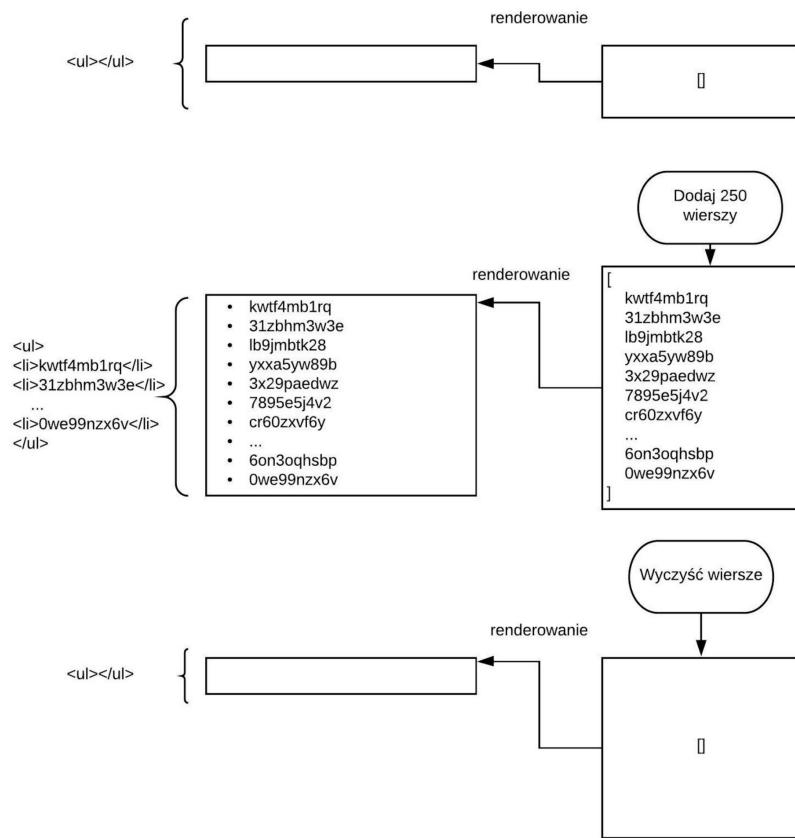


Rysunek 3.5. Ilustracja przedstawiająca problem dopisywania elementu na koniec listy



Rysunek 3.6. Ilustracja przedstawiająca problem dopisywania elementów na początek listy

3. Część praktyczna

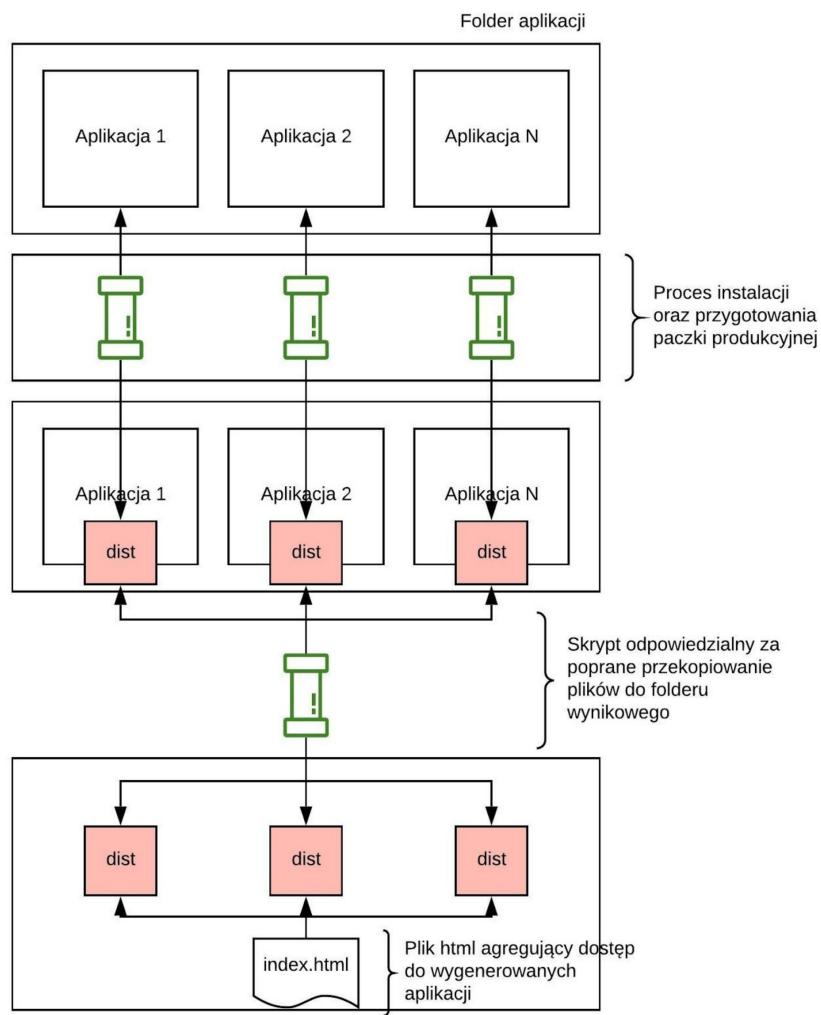


Rysunek 3.7. Ilustracja mechanizmu przebiegu badania

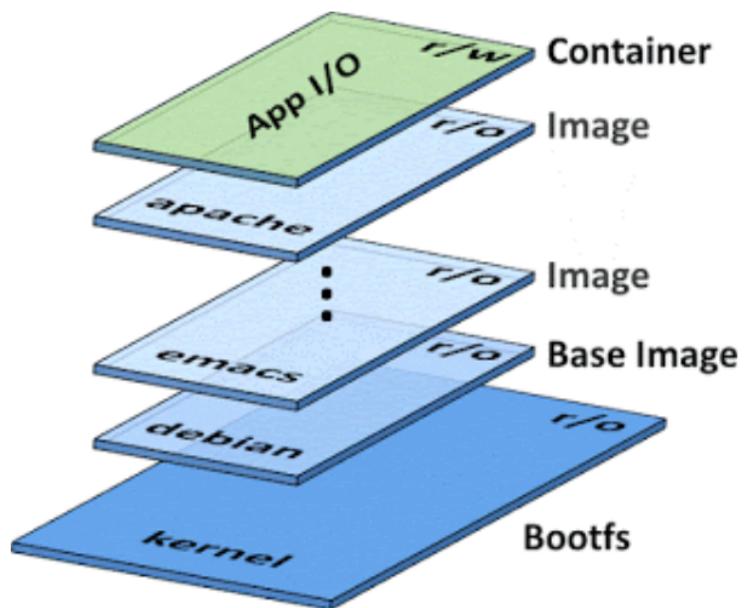
Append 250 rows start | Append 250 rows end | replaceAllRows | updateNRows | Swap Rows | Remove Row | Clear rows

Rysunek 3.8. Grafika przedstawiająca zaimplementowane przyciski w aplikacji

3. Część praktyczna



Rysunek 3.9. Ilustracja procesu przygotowania aplikacji do konteneryzacji



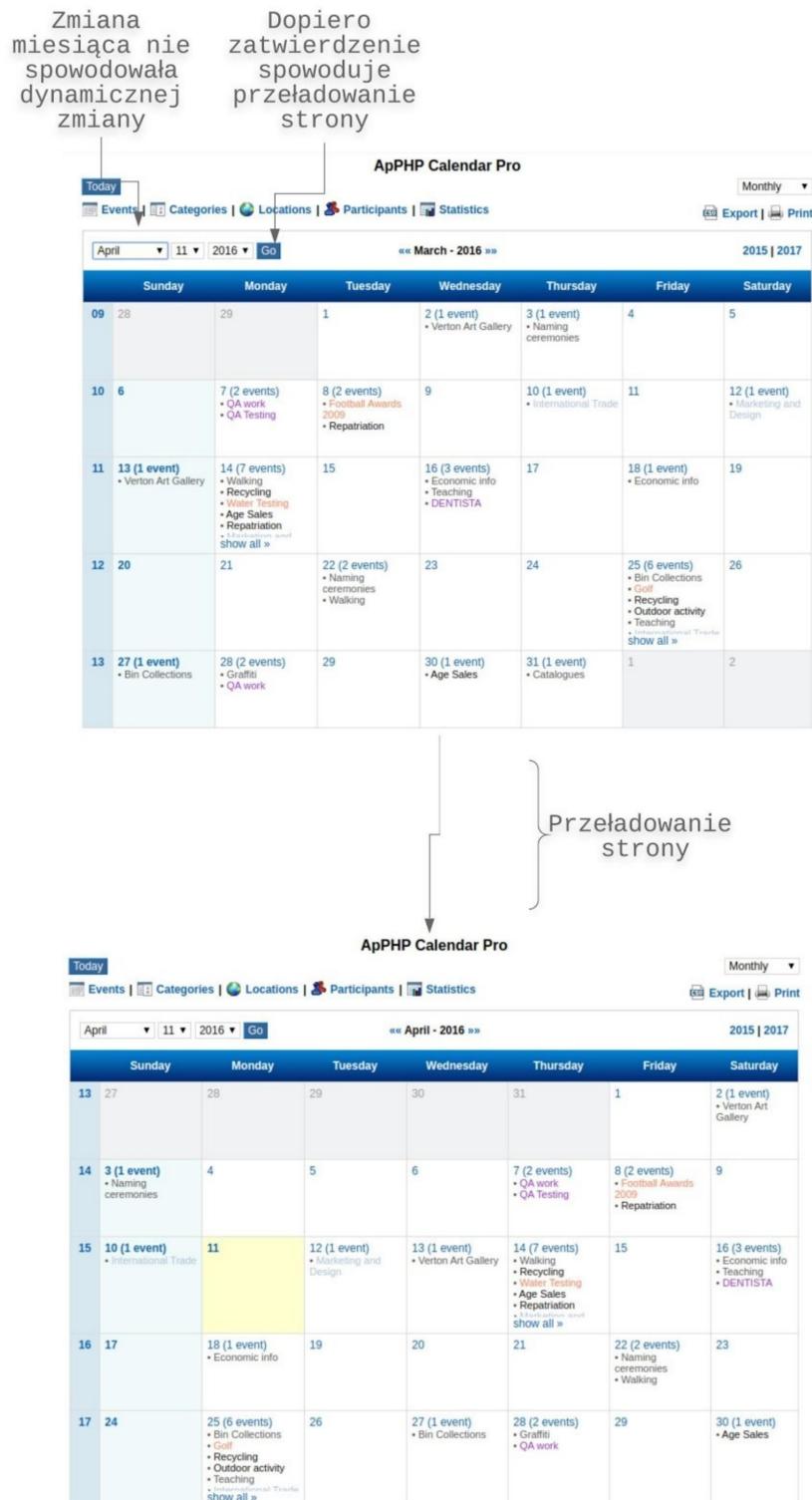
Rysunek 3.10. Ilustracja przedstawiająca warstwy składające się na przykładowy obraz dockera

List of available apps:

- [angular2](#)
- [react](#)
- [vue](#)

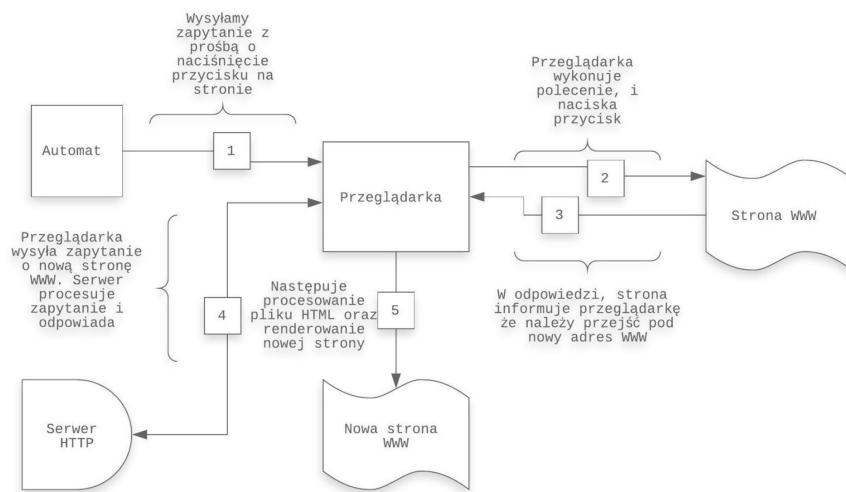
Rysunek 3.11. Grafika przedstawiająca plik index.html wraz z dostępnymi aplikacjami do badania

3. Część praktyczna

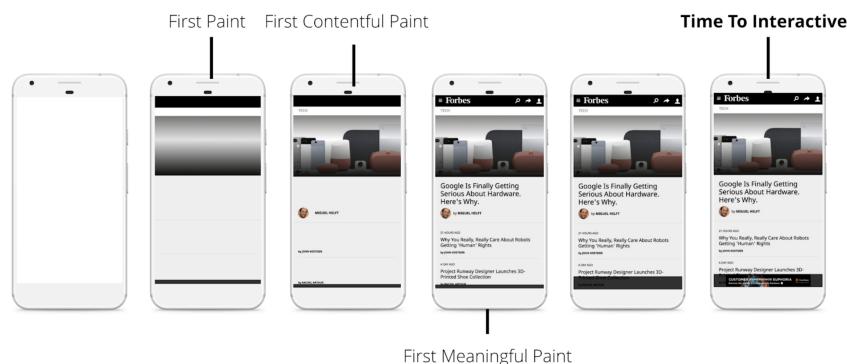


Rysunek 3.12. Ilustracja mechanizmu działania strony statycznej na przykładzie aplikacji kalendarza przy użyciu języka PHP

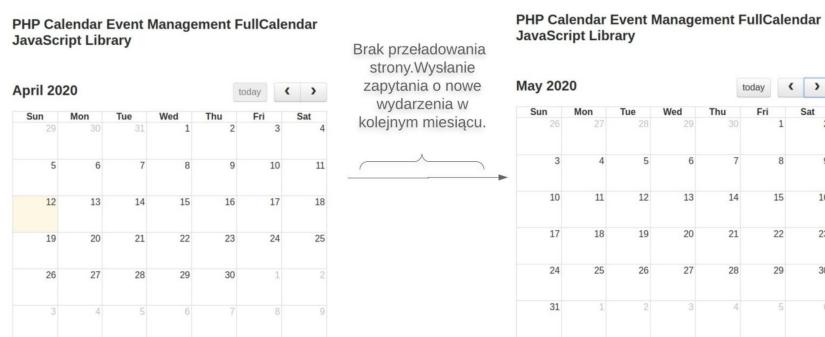
3. Część praktyczna



Rysunek 3.13. Grafika przedstawiająca proces przeładowania strony statycznej

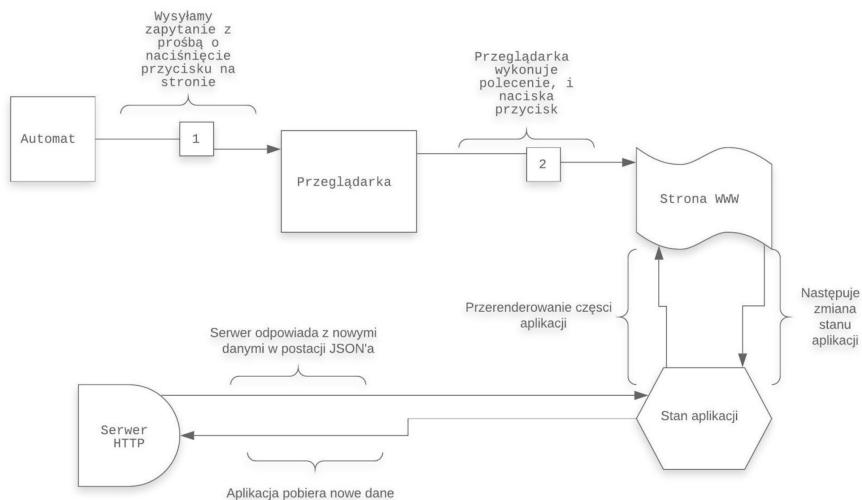


Rysunek 3.14. Ilustracja procesu ładowania aplikacji oraz zdarzenia rejestrowane przez przeglądarkę

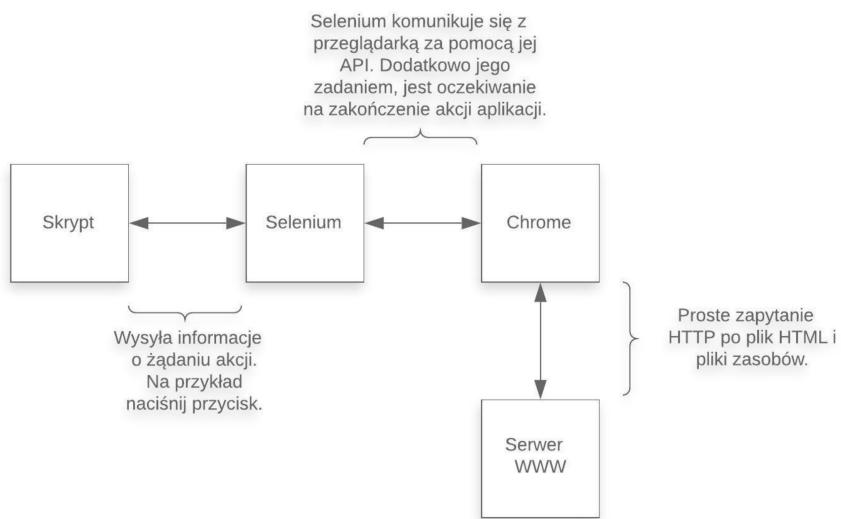


Rysunek 3.15. Ilustracja mechanizmu zmiany daty w przypadku aplikacji dynamicznej

3. Część praktyczna



Rysunek 3.16. Ilustracja przedstawiająca proces zmiany treści strony w przypadku aplikacji dynamicznej



Rysunek 3.17. Ilustracja procesu współpracy pomiędzy Selenium a przeglądarką

```

chrome_1 2020-04-13 10:48:35,598 INFO Included extra file "/etc/supervisor/conf.d/selenium.conf" during supervisor start
chrome_1 2020-04-13 10:48:35,598 INFO supervisor started with pid 7
selenium-hub 2020-04-13 10:48:35,197 INFO Included extra file "/etc/supervisor/conf.d/selenium-hub.conf"
selenium-hub 2020-04-13 10:48:35,198 INFO supervisor started with pid 7
Waiting for Selenium Server
Waiting for Web Server
172.18.0.3 - [13/04/2020:10:48:35 +0000] "GET / HTTP/1.1" 200 70 "-" "axios/0.19.2" "-"
[172.18.0.3:4444] at connectToPage (/app/src/utils.js:23:11)
at processTickAndRejections (internal/process/task_queues.js:97:5)
Connected successfully to Web Server!
2020-04-13 10:48:36,200 spawned: 'selenium-hub' with pid 10
Starting Selenium Hub with configuration:
2020-04-13 10:48:36,209 INFO success: selenium-hub entered RUNNING state, process has stayed up for > 1 second

```

Widzimy próbę połączenia

Nie udaje się połączyć jednak do serwera selenium

Serwer HTTP odpowiada z kodem 200

Każdy wpis ma jasno określone źródło z którego pochodzi

Sukces połączenia do serwera HTTP

Rysunek 3.18. Wycinek wpisów skryptu przeprowadzającego badanie w środowisku docker-compose. Ilustruje on inicjalizację skryptu oraz mechanizm uzyskiwania połączenia pomiędzy Skryptem - Przeglądarką - Selenium

3. Część praktyczna

```

chrome_1 2020-04-13 10:48:35.500 INFO Included extra file "/etc/supervisor/conf.d/selenium.conf" during startup
chrome_1 2020-04-13 10:48:35.500 INFO supervisor started with pid 7
selenium-hub 2020-04-13 10:48:35.197 INFO Included extra file "/etc/supervisor/conf.d/selenium-hub.conf"
selenium-hub 2020-04-13 10:48:35.198 INFO supervisord started with pid 7
selenium_benchmark_1 Waiting for Web Server
master_web_server_1
selenium_benchmark_1 Waiting for Selenium Server
selenium_benchmark_1 172.18.0.5 - - [13/Apr/2020:10:48:35 +0000] "GET / HTTP/1.1" 200 470 "-" "axios/0.19.2"
selenium_benchmark_1 at connect to page (/app/src/utils.js:23:11)
selenium_benchmark_1 at processTicksAndRejections (internal/process/task_queues.js:97:5)
selenium_benchmark_1 Connected successfully to Web Server!
selenium-hub 2020-04-13 10:48:36.200 INFO spawned: 'selenium-hub' with pid 10
selenium-hub Starting Selenium Hub with configuration:
selenium-hub 2020-04-13 10:48:36.209 INFO success: selenium-hub entered RUNNING state, process has stayed up for > 0 seconds (startsecs=1).

```

Annotations:

- Każdy wpis ma jasno określone źródło z którego pochodzi
- Sukces połączenia do serwera HTTP
- Widzimy próbę połączenia
- Serwer HTTP odpowiada z kodem 200
- Nie udaje się połączyć jednak do serwera selenium

Rysunek 3.19. Wycinek skryptu ukazujący uzyskanie połączenia do Selenium pomimo początkowych problemów

```

chrome_1 10:48:38.438 INFO [RemoteSessionsFactory.lambda$per
selenium_benchmark_1 ★ Driver created
master_web_server_1 | 172.18.0.4 - - [13/Apr/2020:10:48:38 +0000] "GET /
selenium_benchmark_1 | Found: [ 'angular2', 'react', 'vue' ]
master_web_server_1 | 172.18.0.4 - - [13/Apr/2020:10:48:38 +0000] "GET //
master_web_server_1 | 172.18.0.4 - - [13/Apr/2020:10:48:38 +0000] "GET //
(masterHTML, like Gecko) HeadlessChrome/79.0.3945.117 Safari/537.36" -
master_web_server_1 | 172.18.0.4 - - [13/Apr/2020:10:48:38 +0000] "GET //
Kit/537.36 (KHTML, like Gecko) HeadlessChrome/79.0.3945.117 Safari/537.36"
master_web_server_1 | 172.18.0.4 - - [13/Apr/2020:10:48:38 +0000] "GET //
WebKit/537.36 (KHTML, like Gecko) HeadlessChrome/79.0.3945.117 Safari/537.3
master_web_server_1 | 172.18.0.4 - - [13/Apr/2020:10:48:38 +0000] "GET //
it/537.36 (KHTML, like Gecko) HeadlessChrome/79.0.3945.117 Safari/537.36" -
selenium_benchmark_1 | result_foler ./results/angular2/single_action/0
master_web_server_1 | 172.18.0.4 - - [13/Apr/2020:10:50:18 +0000] "GET //
rome/79.0.3945.117 Safari/537.36" -
selenium_benchmark_1 | result_foler ./results/angular2/single_action/1
master_web_server_1 | 172.18.0.4 - - [13/Apr/2020:10:51:55 +0000] "GET //
rome/79.0.3945.117 Safari/537.36" -
selenium_benchmark_1 | result_foler ./results/angular2/single_action/2
master_web_server_1 | 172.18.0.4 - - [13/Apr/2020:10:53:34 +0000] "GET //
rome/79.0.3945.117 Safari/537.36" -
selenium_benchmark_1 | result_foler ./results/angular2/single_action/3
master_web_server_1 | 172.18.0.4 - - [13/Apr/2020:10:55:10 +0000] "GET //
rome/79.0.3945.117 Safari/537.36" -
selenium_benchmark_1 | result_foler ./results/angular2/single_action/4
master_web_server_1 | 172.18.0.4 - - [13/Apr/2020:10:56:45 +0000] "GET //
rome/79.0.3945.117 Safari/537.36" -

```

Annotations:

- Skrypt pobiera listę dostępnych aplikacji
- Web serwer odpowiada z plikiem index.html
- Następuje rozpoczęcie badania
- Wyniki poszczególnych badań zostają zapisane na dysku

Rysunek 3.20. Grafika przedstawia rozpoczęcie badania

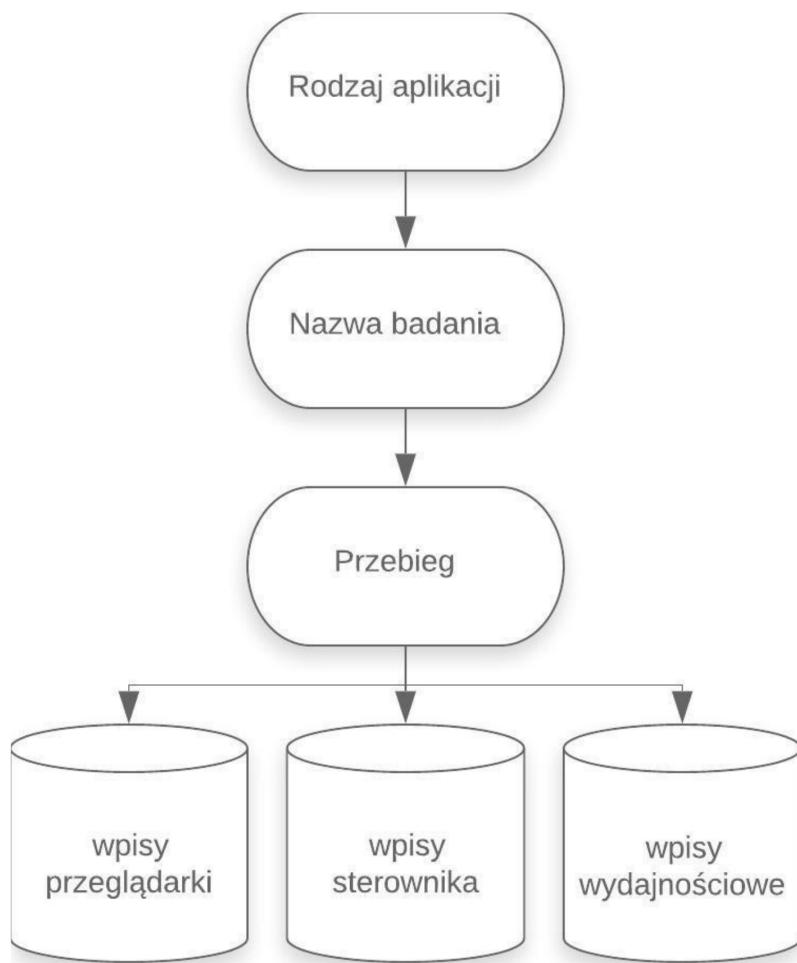
```

selenium_benchmark_1 | result_foler ./results/vue/single_action/3
master_web_server_1 | 172.18.0.4 - - [13/Apr/2020:11:10:55 +0000] "GET
3945.117 Safari/537.36" "-"
selenium_benchmark_1 | result_foler ./results/vue/single_action/4
master_web_server_1 | 172.18.0.4 - - [13/Apr/2020:11:12:28 +0000] "GET
3945.117 Safari/537.36" "-"
selenium_benchmark_1 | result_foler ./results/vue/simple_benchmark/0
master_web_server_1 | 172.18.0.4 - - [13/Apr/2020:11:12:30 +0000] "GET
3945.117 Safari/537.36" "-"
selenium_benchmark_1 | 🚀 Finished benchmark successfully for vue
selenium_benchmark_1 | ❤️ Finished all benchmarks!
selenium_benchmark_1 | 💀 Killing driver
chrome_1 [DZIENNIK AUTORSKIE] 11:12:30.877 INFO [ActiveSessions$1.onStop] - Ren

```

Rysunek 3.21. Grafika przedstawia skrypt zakończający badanie po zapisaniu zebranych danych na dysk - widzimy, że ważnym elementem jest zamknięcie połączenia do Selenium

3. Część praktyczna

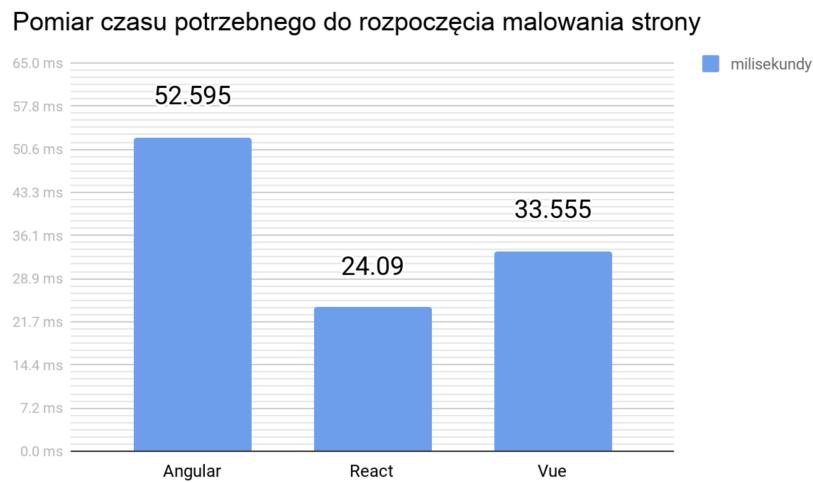


Rysunek 3.22. Ilustracja struktury wyniku badania na które zostanie poddane dalszej obróbce

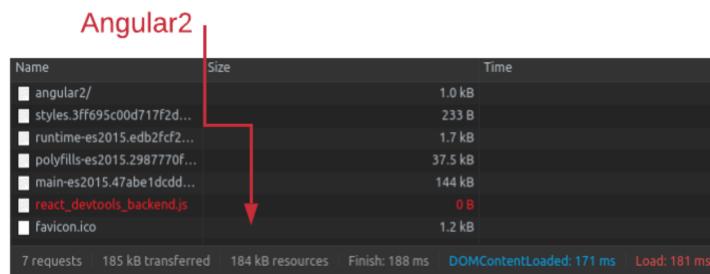
```
{
  "name": "measure_replace_all_click-1586714207563}",
  "entryType": "measure",
  "startTime": 61.564999996335246,
  "duration": 49.28500000096392
},
[
  {
    "name": "measure_updateNRows_click-1586714207631}",
    "entryType": "measure",
    "startTime": 130.34499999776017,
    "duration": 48.45000000204891
]
},
```

Rysunek 3.23. Grafika przedstawiająca wpis zebranego pomiaru

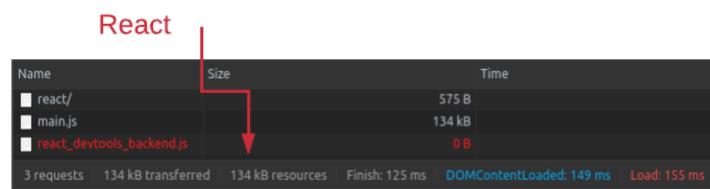
3. Część praktyczna



Rysunek 3.24. Diagram kolumnowy ukazujący wynik badania czasu potrzebnego do rozpoczęcia malowania strony

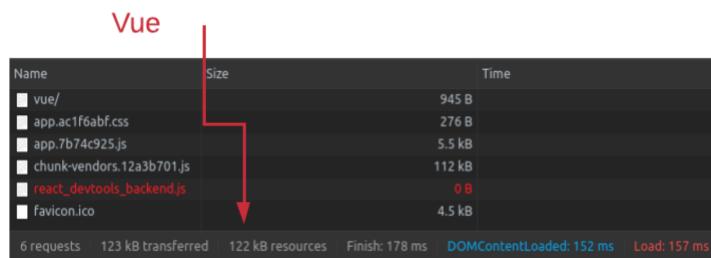


Rysunek 3.25. Grafika ukazująca rozmiar plików zasobów dla aplikacji Angular2

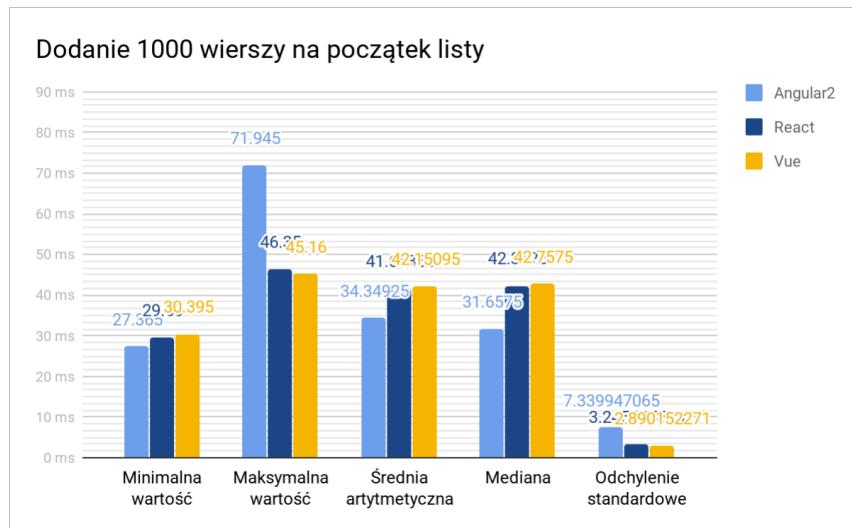


Rysunek 3.26. Grafika ukazująca rozmiar plików zasobów dla aplikacji React

3. Część praktyczna

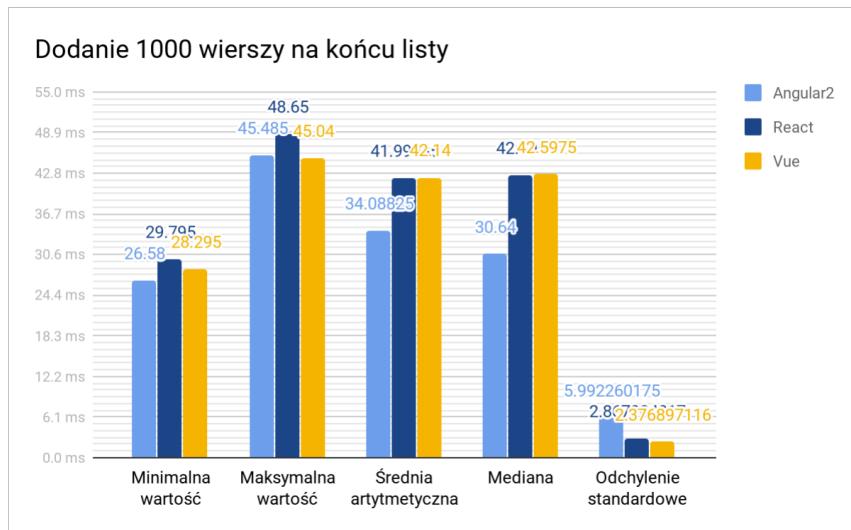


Rysunek 3.27. Grafika ukazująca rozmiar plików zasobów dla aplikacji Vue

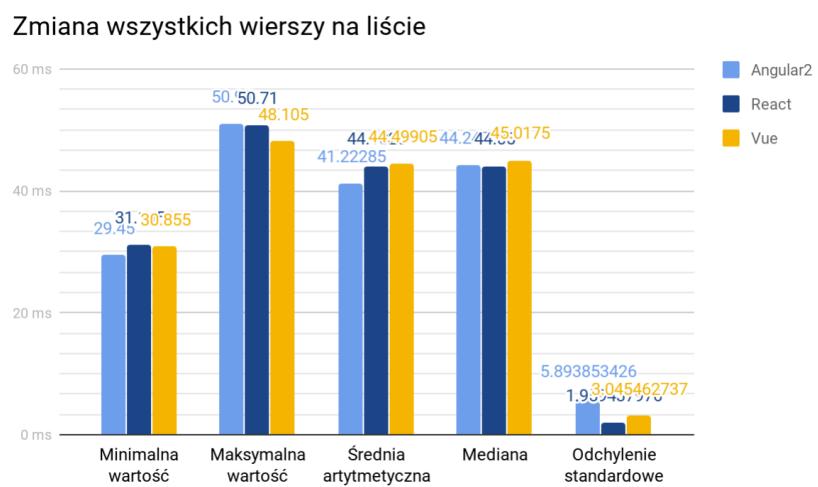


Rysunek 3.28. Diagram kolumnowy obrazujący wynik badania pomiaru czasu dodania 1000 elementów na początek listy

3. Część praktyczna

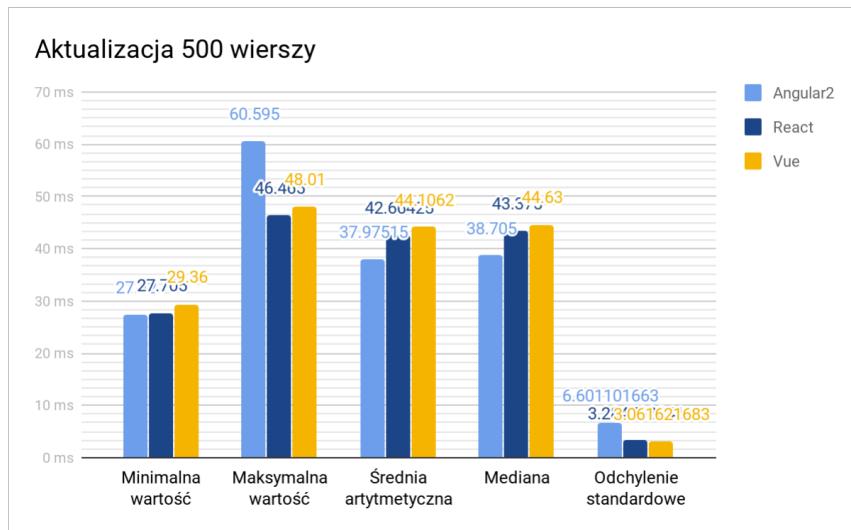


Rysunek 3.29. Diagram kolumnowy obrazujący wynik badania pomiaru czasu dodania 1000 elementów na koniec listy

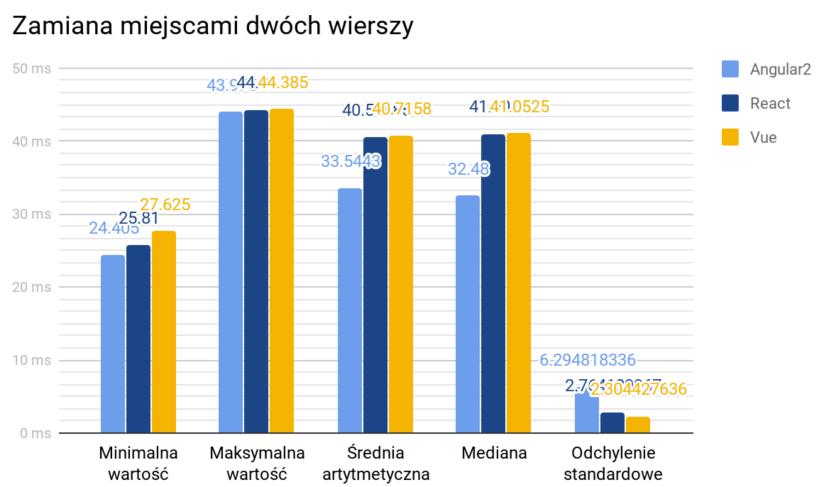


Rysunek 3.30. Diagram kolumnowy obrazujący wynik badania pomiaru czasu zmiany wszystkich wartości listy dla 1000 elementów

3. Część praktyczna



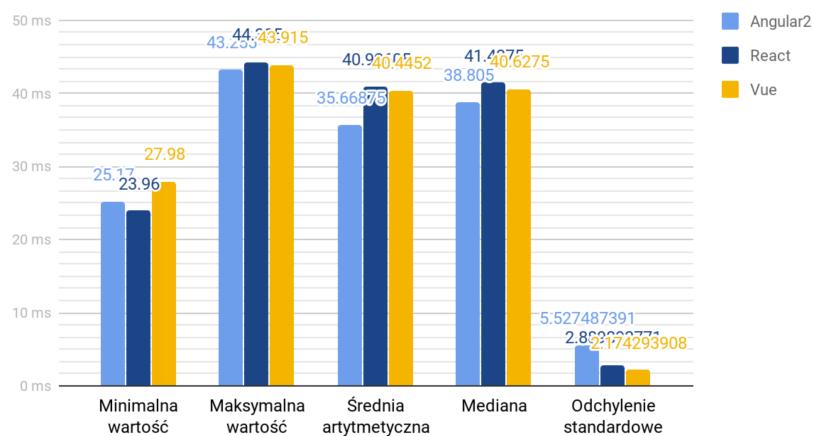
Rysunek 3.31. Diagram kolumnowy obrazujący wynik badania pomiaru czasu zmiany wartości dla 500 elementów listy



Rysunek 3.32. Diagram kolumnowy obrazujący wynik badania pomiaru czasu zamiany miejscami dwóch wartości na liście elementów

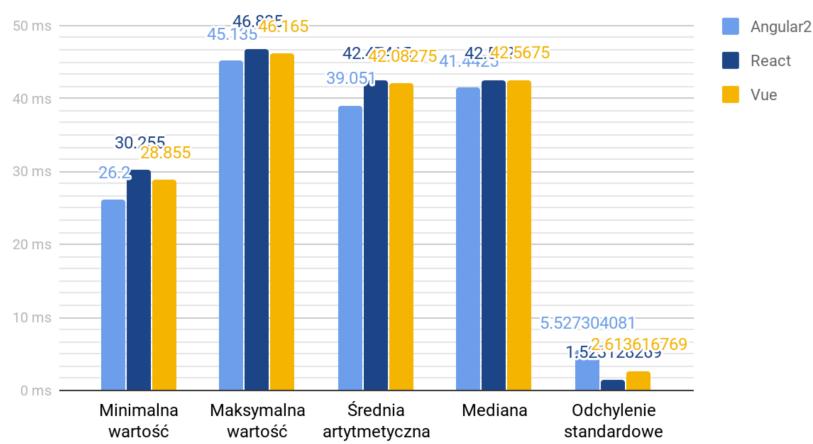
3. Część praktyczna

Usunięcie pojedynczego wiersza



Rysunek 3.33. Diagram kolumnowy obrazujący wynik badania pomiaru czasu usunięcia pojedynczej wartości z listy elementów

Usunięcie wszystkich wierszy



Rysunek 3.34. Diagram kolumnowy obrazujący wynik badania pomiaru czasu usunięcia wszystkich wartości z listy elementów

4. Podsumowanie

Bibliografia

- [1] JSON.org, <https://www.json.org/>
- [2] Peleke Sengstacke *JavaScript Transpilers: What They Are and Why We Need Them*, Kwiecień 25, 2016
<https://scotch.io/tutorials/javascript-transpilers-what-they-are-why-we-need-them>
- [3] Sławomir Kołodziej *What Are Single Page Applications? What Is Their Impact on Users' Experience and Development Process?*, 3 Lipiec 2019
<https://www.netguru.com/blog/what-are-single-page-applications>
- [4] Eric Molin, *Comparison of Single-Page Application Frameworks*, Instytut KTH w Sztokholmie
<https://pdfs.semanticscholar.org/fa9f/f75f32de61cddafa8805ea433d4d8a0e20da.pdf>
- [5] <https://octoverse.github.com/>
- [6] Maja Nowak, Reasons, *Why Vue.js Is Getting More Traction Every Month*, 19 Grudzień 2018,
<https://www.monterail.com/blog/reasons-why-vuejs-is-popular>
- [7] <https://github.com/angular/angular/blob/master/CHANGELOG.md>
- [8] <https://github.com/facebook/react/blob/master/CHANGELOG.md>
- [9] *Probabilistyczny opis błędu jako podstawa definiowania niepewności pojedynczego wyniku pomiaru*
<http://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-article-BSW4-0034-0011>
- [10] Google Developers, *Lighthouse*
<https://developers.google.com/web/tools/lighthouse>
- [11] Kyle Simpson *You Don't Know JS: Async & Performance*
http://cdn.lxqnsys.com/05_You_Don't_Know_JS_Async_&_Performance.pdf

- [12] Jadwiga Kalinowska, Beata Pańczyk, *Porównanie narzędzi do tworzenia aplikacji typu SPA na przykładzie Angular2 i React*, Politechnika Lubelska, Instytut Informatyki
<http://yadda.icm.edu.pl/yadda/element/bwmeta1.element.baztech-5c6271b4-27e3-42d0-9762-a240dc3a9973>
- [13] Paul Lewis, *Google Developers, Rendering Performance*,
<https://developers.google.com/web/fundamentals/performance/rendering>
- [14] *ReactJS, Lists and Keys*,
<https://reactjs.org/docs/lists-and-keys.html>
- [15] *ReactJS, Virtual DOM and Internals*,
<https://reactjs.org/docs/faq-internals.html>
- [16] ReactJS, Reconciliation,
<https://reactjs.org/docs/reconciliation.html>
- [17] GNU.org, *Makefile*,
https://www.gnu.org/software/make/manual/html_node/Introduction.html
- [18] NpmJs.com, *npm | build amazing things*,
<https://www.npmjs.com/>
- [19] ReactJS, *Optimizing Performance*,
<https://reactjs.org/docs/optimizing-performance.html>
- [20] magnifier.pl, *Konteneryzacja - czym jest i dlaczego staje się tak popularna?*, 24 Październik. 2019,
<https://magnifier.pl/konteneryzacja-docker-kubernetes/>
- [21] MDN web docs, *Promise - JavaScript*,
https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Promise
- [22] MDN web docs, *What is a web server?*, 18 Stycznia 2019,
https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server
- [23] Docker Docs, *Docker*,
<https://docs.docker.com/get-started/#docker-concepts>

- [24] Nginx, *Nginx for Windows*,
<http://nginx.org/en/docs/windows.html>
- [25] Nginx, *Linux packages*,
http://nginx.org/en/linux_packages.html
- [26] Wiki Alpine Linux, *Docker*,
<https://wiki.alpinelinux.org/wiki/Docker>
- [27] Docker Hub, *Ubuntu*,
https://hub.docker.com/_/ubuntu
- [28] Meggin Kearney, Addy Osmani, Kayce Basques, Jason Miller, *Measure Performance with the RAIL Model | Web Fundamentals*, 12 Luty. 2019,
<https://developers.google.com/web/fundamentals/performance/rail>
- [29] Jesus Castello, *Ruby Templating Engines: ERB, HAML & Slim - RubyGuides*,
<https://www.rubyguides.com/2018/11/ruby-erb-haml-slim/>
- [30] Whatsabyte, *What Are Threads in a Processor?*, 24 Sierpień 2018,
<https://whatsabyte.com/blog/processor-threads/>
- [31] Dimiter Petrov, *A tale of flaky Cypress tests*, 24 Październik 2019,
<https://dimiterpetrov.com/blog/a-tale-of-flaky-cypress-tests/>
- [32] Selenium Dev, *The Selenium project and tools*,
https://www.selenium.dev/documentation/en/introduction/the_selenium_project_and_tools/
- [33] Docker Docs, *Overview of Docker Compose*,
<https://docs.docker.com/compose/.>
- [34] Docker Documentation, *Networking in Compose*.
<https://docs.docker.com/compose/networking/>
- [35] MDN - Mozilla, *Memory Management*, 4 Marca 2020,
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management
- [36] MDN - Mozilla, *Performance - Web APIs*, 30 Styczeń 2020,
<https://developer.mozilla.org/en-US/docs/Web/API/Performance>

- [37] MDN - Mozilla, *DOMHighResTimeStamp - Web APIs*, 19 Lusty 2020,
<https://developer.mozilla.org/en-US/docs/Web/API/DOMHighResTimeStamp>
- [38] MDN - Mozilla, *First paint - MDN Web Docs Glossary*, 10 Marca. 2020,
https://developer.mozilla.org/en-US/docs/Glossary/First_paint

Spis rysunków

2.1	Grafika przedstawiająca ocenę wpływu wybranego elementu na atrakcyjność danego narzędzia [4]	7
2.2	Grafika przedstawiająca wzrost czasu ładowania elementów od 10 do 5000 przy użyciu AngularJS Angular2 oraz React [4]	8
2.3	Grafika przedstawiająca wykres zależności czasu do ilości edytowanych elementów na stronie [4]	8
2.4	Grafika przedstawiająca wykres czasu ładowania aplikacji [4]	9
2.5	Porównanie metryk kodu dla pojedynczego komponentu [12]	10
2.6	Porównanie czasów pobrania danych z bazy [12]	11
3.1	Ilustracja przedstawiająca cykl życia statycznej strony internetowej	13
3.2	Ilustracja przedstawiająca cykl działania aplikacji SPA	14
3.3	Grafika przedstawiająca mechanizm działania aplikacji dynamicznych	15
3.4	Grafika przedstawiająca kolejność faz renderowania w przeglądarce	16
3.5	Ilustracja przedstawiająca problem dopisywania elementu na koniec listy	17
3.6	Ilustracja przedstawiająca problem dopisywania elementów na początek listy	17
3.7	Ilustracja mechanizmu przebiegu badania	18
3.8	Grafika przedstawiająca zaimplementowane przyciski w aplikacji	18
3.9	Ilustracja procesu przygotowania aplikacji do konteneryzacji	19
3.10	Ilustracja przedstawiająca warstwy składające się na przykładowy obraz dockera	20
3.11	Grafika przedstawiająca plik index.html wraz z dostępnymi aplikacjami do badania	20
3.12	Ilustracja mechanizmu działania strony statycznej na przykładzie aplikacji kalendarza przy użyciu języka PHP	21
3.13	Grafika przedstawiająca proces przeładowania strony statycznej	22
3.14	Ilustracja procesu ładowania aplikacji oraz zdarzenia rejestrowane przez przeglądarkę	22
3.15	Ilustracja mechanizmu zmiany daty w przypadku aplikacji dynamicznej	22
3.16	Ilustracja przedstawiająca proces zmiany treści strony w przypadku aplikacji dynamicznej	23
3.17	Ilustracja procesu współpracy pomiędzy Selenium a przeglądarką	23
3.18	Wycinek wpisów skryptu przeprowadzającego badanie w środowisku docker-compose. Ilustruje on inicjalizację skryptu oraz mechanizm uzyskiwania połączenia pomiędzy Skryptem - Przeglądarką - Selenium	23
3.19	Wycinek skryptu ukazujący uzyskanie połączenia do Selenium pomimo początkowych problemów	24

3.20 Grafika przedstawia rozpoczęcie badania	24
3.21 Grafika przedstawia skrypt zakańczający badanie po zapisaniu zebranych danych na dysk - widzimy, że ważnym elementem jest zamknięcie połączenia do Selenium	24
3.22 Ilustracja struktury wyniku badania na które zostanie poddane dalszej obróbce .	25
3.23 Grafika przedstawiająca wpis zebranego pomiaru	25
3.24 Diagram kolumnowy ukazujący wynik badania czasu potrzebnego do rozpoczęcia malowania strony	26
3.25 Grafika ukazująca rozmiar plików zasobów dla aplikacji Angular2	26
3.26 Grafika ukazująca rozmiar plików zasobów dla aplikacji React	26
3.27 Grafika ukazująca rozmiar plików zasobów dla aplikacji Vue	27
3.28 Diagram kolumnowy obrazujący wynik badania pomiaru czasu dodania 1000 elementów na początek listy	27
3.29 Diagram kolumnowy obrazujący wynik badania pomiaru czasu dodania 1000 elementów na koniec listy	28
3.30 Diagram kolumnowy obrazujący wynik badania pomiaru czasu zmiany wszystkich wartości listy dla 1000 elementów	28
3.31 Diagram kolumnowy obrazujący wynik badania pomiaru czasu zmiany wartości dla 500 elementów listy	29
3.32 Diagram kolumnowy obrazujący wynik badania pomiaru czasu zamiany miejscami dwóch wartości na liście elementów	29
3.33 Diagram kolumnowy obrazujący wynik badania pomiaru czasu usunięcia pojedynczej wartości z listy elementów	30
3.34 Diagram kolumnowy obrazujący wynik badania pomiaru czasu usunięcia wszystkich wartości z listy elementów	30

Wrocław, dnia 2020-05-08

Wydział Informatyki, Administracji i Fizjoterapii

Kierunek studiów: **informatyka (INF)**

Daniel Słaby

.....
(imię i nazwisko studenta)

6781

.....
(nr albumu)

**OŚWIADCZENIE O UDOSTĘPNIANIU PRACY
DYPLOMOWEJ**

Tytuł pracy dyplomowej: Analiza porównawcza nowoczesnych rozwiązań technologicznych aplikacji SPA za pomocą jednakowej aplikacji zaimplementowanej w każdym z wybranych rozwiązań

Wyrażam zgodę (nie wyrażam zgody)¹ na udostępnianie mojej pracy dyplomowej.

.....
(podpis studenta)

¹Niepotrzebne skreślić.

Wrocław, dnia 2020-05-08

Wydział Informatyki, Administracji i Fizjoterapii

Kierunek studiów: **informatyka (INF)**

Daniel Słaby

.....
(imię i nazwisko studenta)

6781

.....
(nr albumu)

OŚWIADCZENIE AUTORSKIE

Oświadczam, że niniejszą pracę dyplomową pod tytułem:

Analiza porównawcza nowoczesnych rozwiązań technologicznych aplikacji SPA za pomocą jednakowej aplikacji zaimplementowanej w każdym z wybranych rozwiązań

napisałem/am samodzielnie. Nie korzystałem/am z pomocy osób trzecich, jak również nie dokonałem/am zapożyczeń z innych prac.

Wszystkie fragmenty pracy takie jak cytaty, rycinę, tabele, programy itp., które nie są mojego autorstwa, zostały odpowiednio zaznaczone i zamieszczono w pracy źródła ich pochodzenia. Treść wydrukowanej pracy dyplomowej jest identyczna z wersją pracy zapisaną na przekazywanym nośniku elektronicznym.

Jednocześnie przyjmuję do wiadomości, że jeżeli w przypadku postępowania wyjaśniającego zebrany materiał potwierdzi popełnienie przeze mnie plagiatu, skutkować to będzie niedopuszczeniem do dalszych czynności w sprawie nadania mi tytułu zawodowego do czasu wydania orzeczenia przez komisję dyscyplinarną oraz złożenie zawiadomienia o popełnieniu przestępstwa.

.....

(podpis studenta)