AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

Porównanie budowania i rozwoju aplikacji WWW w języku Elm i technologiach React+Redux

Comparision of building and development of web application in Elm language and React+Redux technologies

Autor: *Kamil Osuch*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr inż. Piotr Matyasik*

Kraków, 2017

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję . . . tu ciąg dalszych podziękowań np. dla promotora, żony, sąsiada itp.

Spis treści

1. Wprowadzenie 7
2. Podstawy teoretyczne 9
2.1. Aplikacja internetowa 9
2.2. SPA 9
2.2.1. Zalety 10
2.2.2. Wady 10
2.3. JavaScript 11
2.4. React.js 11
2.5. Redux 11
2.6. Elm 12
3. Porównanie 13
3.1. Virtual DOM i składnia 13
3.2. Jednokierunkowe flow aplikacji 17
3.3. Niemutowalność obiektów 17
3.4. Biblioteki i menedżery pakietów 17
4. Podsumowanie 19

1. Wprowadzenie

W ciągu ostatnich kilku lat sposób tworzenia stron internetowych przechodził intensywne i szybkie zmiany. W związku z rosnącą popularnością internetu na całym świecie okazało się, że zwykłe, proste strony internetowe nie są wystarczające. W związku z tym szybko one awansowały ze statycznych dokumentów hipertekstowych jedynie wyświetlających zawartość użytkownikowi, do poziomu aplikacji internetowych, z którymi może on wejść w interakcję, a nawet używać ich dokładnie tak samo, jak programów zainstalowanych w swoim systemie operacyjnym.

Rynek tworzenia oprogramowania webowego został przejęty głównie przez język JavaScript nazywany dziś przez niektórych asemblerem stron internetowych [1]. Coraz większe wymagania co do działania stron internetowych spowodowały, że tworzenie stron bezpośrednio w JavaScript'cie stało się zbyt skomplikowane i niewystarczające. Z pomocą przychodzą biblioteki, frameworki oraz języki kompilowane do JavaScriptu. Upraszczają one tworzony kod, często zmieniając też jego strukturę. Dzięki temu umożliwiają korzystanie z gotowych funkcjonalności w prosty i wygodny sposób. Przykładami tutaj mogą być biblioteki takie jak jQuery, Angular, React, Redux, czy Ember.js.

W przypadku języków kompilowanych do JavaScriptu zmiany są jeszcze większe. Programista nie zastanawia się nad tym, że ostateczna wersja stworzonego przez niego programu jest zapisana w zupełnie innym języku. Przykładami takich języków są chociażby CoffeeScript, TypeScript czy Elm.

Ilość takich rozwiązań tworzy nieograniczoną liczbę podejść do tworzenia aplikacji webowych, a ich liczba każdego dnia rośnie. W związku z tym powstaje wiele artykułów porównujących różne podejścia, zarówno pod względami architektury kodu, jak i szybkości działania samych aplikacji.

Celem niniejszej pracy jest porównanie budowania i rozwoju aplikacji webowych przy pomocy języka Elm, oraz kombinacji bibliotek React.js i Redux. Technologie te zostaną szczegółowo porównane pod względem dostępnych funkcjonalności, szybkości działania aplikacji, dostępności bibliotek oraz trudności zarówno w tworzeniu pierwszej wersji aplikacji, jak i rozwoju istniejącego już kodu.

Wybór tematu pracy był podyktowany przede wszystkim poszukiwaniem alternatywnych rozwiązań służących do tworzenia aplikacji webowych. Biblioteki takie jak React czy Redux zdominowały rynek, przez co ilość wykorzystywanych sposobów budowania stron internetowych w stosunku do ilości dostępnych możliwości jest bardzo mała.

Ważną częścią pracy jest implementacja dwóch wersji tej samej aplikacji webowej, stworzonych przy pomocy opisywanych w pracy rozwiązań. Pozwoli to na dokładniejszą analizę i porównanie obu podejść, w oparciu o praktyczny przykład.

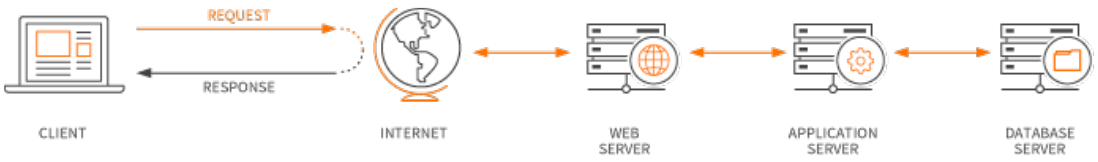
W rozdziale 2 zostały krótko opisane charakterystyka języka JavaScript oraz podstawowe pojęcia związane z tworzeniem aplikacji webowych. Następnie krótko zostały opisane biblioteki React i Redux oraz język Elm. Rozdział 4 zawiera krótkie podsumowanie przeprowadzonej analizy porównawczej. Opisuje on wnioski oraz możliwości rozwoju pracy.

2. Podstawy teoretyczne

W tym rozdziale zostaną omówione podstawowe pojęcia związane z tematem pracy. Pierwsze dwa podrozdziały skupiają się na opisie czym w ogóle jest aplikacja internetowa, oraz opisują jedno z najpopularniejszych podejść do budowania aplikacji internetowej, SPA. W kolejnych podrozdziałach zostały opisane technologie, które są porównywane w dalszej części pracy.

2.1. Aplikacja internetowa

Aplikacja internetowa jest czymś więcej niż zwykłą stroną internetową. Z definicji jest to aplikacja typu klient-serwer, w której klient jest uruchamiany przy pomocy przeglądarki internetowej. Oprogramowanie klienta jest pobierane na komputer klienta podczas wizyty na odpowiedniej stronie internetowej, przy użyciu standardowych procedur, takich jak HTTP. Aktualizacje oprogramowania klienta mogą odbywać się za każdym razem, gdy odwiedzana jest strona internetowa. W czasie trwania sesji przeglądarka internetowa interpretuje i wyświetla strony, oraz działa jako uniwersalny klient dla dowolnej aplikacji internetowej.

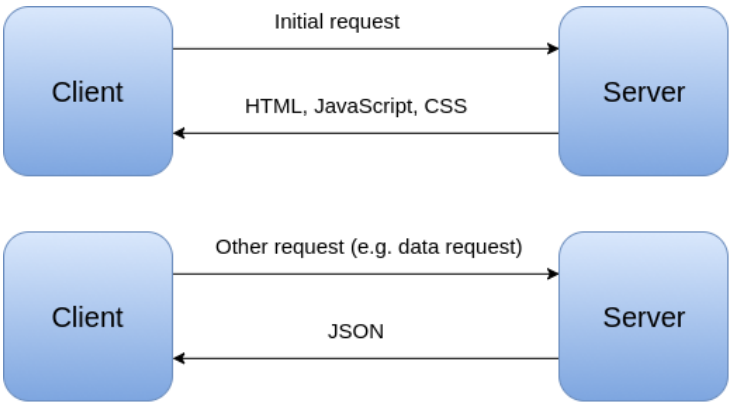


Rysunek 2.1. Podstawowy schemat działania aplikacji internetowej [2]

2.2. SPA

Skrót SPA pochodzi od *Single-page Application*. Jest to aplikacja internetowa działająca wewnątrz przeglądarki, która podczas użytkowania strony nie wymaga odświeżania strony. W tym podejściu, cały niezbędny kod źródłowy — HTML, JavaScript i CSS — jest pobierany przy pojedyńczym załadowaniu strony, lub odpowiednie zasoby są ładowane dynamicznie i dodawane do strony jedynie wtedy, kiedy jest to potrzebne. Idea jaka stoi za takim rozwiązaniem, to przede wszystkim lepszy, bardziej naturalny user

expierience. Użytkownik po wejściu na stronę, nie musi przy każdej interakcji oczekiwać na ponowne załadowanie się strony, czy też jej odświeżanie.



Rysunek 2.2. Podstawowy cykl życia SPA

Choć koncept SPA zaczął być częściej używany po spopularyzowaniu AJAX'a[1], to tak naprawdę dopiero od kilku lat jest on powszechnie wykorzystywany podczas budowania aplikacji internetowych. Serwisy takie jak Gmail, Google Maps, Twitter czy GitHub są właśnie aplikacjami typu single-page application. Także większość najpopularniejszych bibliotek JavaScript'owych umożliwiają implementację aplikacji internetowej zgodnie z zasadami SPA.

2.2.1. Zalety

– Szybkość działania – większość zasobów jest ładowana tylko raz podczas cyklu życia aplikacji, jedynymi informacjami, które są wymieniane z serwerem cały czas, są dane

– Brak ciągłego przeładowywania strony

– Znacznie prostszy proces wdrożenia aplikacji – jedyne co jest potrzebne to statyczny serwer serwujący minimalnie 3 pliki – pojedynczą stronę HTML, oraz 2 pakiety: jeden zawierający wszystkie style, drugi skupiający w sobie cały kod JavaScriptu

– Odciążenie strony serwerowej – serwer, zamiast generować za każdym razem pełny kod strony, transmituje jedynie potrzebne w danej chwili dane

2.2.2. Wady

– Powolne początkowe uruchomienie strony – wymaga ono załadowania frameworku, oraz przynajmniej części aplikacji, która później już nie jest ponownie ściągana.

– Ze względu na zależność SPA od JavaScriptu, bardzo łatwo o pojawienie się wycieków pamięci pomiędzy długimi okresami czasu między przeładowaniami strony

[1] Asynchronous JavaScript And XML

2.3. JavaScript

JavaScript jest to wysokopoziomowy, słabo typowany, wieloparadgymatowy, skryptowy i interpretowany język programowania stworzony w 1995 roku przez Brendana Eicha dla firmy Netscape. W roku 1997 organizacja Ecma International wydała na podstawie JavaScriptu standard języka skryptowego nazwany ECMAScript, na którym bazowanych jest większość silników JavaScript'owych.

JavaScript jest jedną z trzech głównych technologii wykorzystywanych przy tworzeniu treści związanych z siecią internetową. Obecnie 94,9% spośród 10 milionów najbardziej popularnych stron internetowych wykorzystuje JavaScript (stan na grudzień 2017 [3]).

2.4. React.js

React jest biblioteką języka JavaScript, wykorzystywaną do tworzenia graficznych interfejsów użytkownika. Pozwala ona na tworzenie rozbudowanych aplikacji internetowych, które używają danych i mogą zmieniać swoją zawartość w czasie bez ponownego ładowania strony. Biblioteka została stworzona przez jednego z programistów Facebooka, Jordana Walke, który zainspirował się rozszerzeniem języka PHP, XHP, także stworzonym przez Facebooka. Pierwsze wersje biblioteki zostały użyte w 2011 roku na stronie aktualności Facebooka, a od 2012 roku jest ona wykorzystywana w serwisie Instagram. Od 2013 roku React stał się wolnym oprogramowaniem, co spowodowało jego nagły wzrost popularności, a także umożliwiło społeczności pomoc w rozwoju oprogramowania.

Ze względu na ogromny wpływ rynku urządzeń mobilnych, Facebook ogłosił w 2015 roku bibliotekę React Native, pozwalającą na korzystanie z architektury Reacta podczas budowania aplikacji mobilnych. To co wyróżniało tę bibliotekę od dotychczasowych sposobów tworzenia mobilnego oprogramowania, to brak konieczności powielania funkcjonalności dla każdej platformy z osobna. React Native pozwalał na wykorzystanie tego samego kodu źródłowego zarówno w aplikacji dla systemu Android jak i iOS.

Obecnie React jest jedną z najpopularniejszych bibliotek służących do tworzenia aplikacji internetowych. Jest wykorzystywany w ogromnej ilości znanych serwisów takich jak Facebook, Instagram, Spotify, Netflix czy eBay.

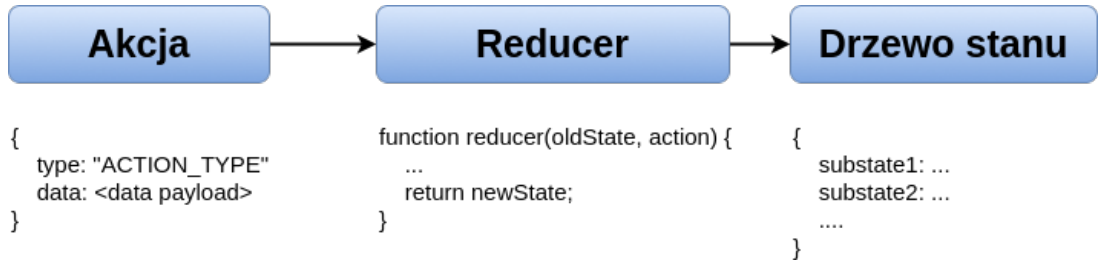
2.5. Redux

Redux jest biblioteką języka JavaScript o otwartym kodzie źródłowym zaprojektowaną do zarządzania stanem aplikacji. Została stworzona w 2015 roku przez Dana Abramova. Działanie Reduxa może być opisane przez trzy zasady [4]:

1. Istnieje jedno źródło prawdy – stan całej aplikacji jest przechowywany w pojedyńczym obiekcie, który przechowuje całe drzewo stanu aplikacji.

2. Stan służy tylko do odczytu – jedynym sposobem na zmianę stanu jest wysłanie akcji – obiektu opisującego co się stało, zwykle zawierającego typ akcji, oraz ewentualne dane, które mają wpływ na zmianę stanu.

3. Zmiany stanu są dokonywane przy użyciu czystych funkcji – aby określić, w jaki sposób drzewo stanu jest modyfikowane przez akcje, tworzy się funkcje zwane *reducerami*, przyjmujące poprzedni stan i akcję jako argumenty, a zwracające nowy stan.



Rysunek 2.3. Schemat działania Reduxa

2.6. Elm

Elm jest funcyjnym językiem programowania kompilowanym do JavaScriptu. Język został stworzony w 2012 roku przez Evana Czaplickiego, w ramach jego pracy dyplomowej. Na dalszy rozwój języka pozwoliła firma Prezi, która w 2013 roku zatrudniła twórcę języka. W 2016 roku Czaplicki zmienił firmę na NoRedInk i postanowił stworzyć organizację non-profit — Elm Software Foundation — której celem ma być promowanie, ochrona i rozwój języka Elm, oraz wszelka pomoc społeczności programistów tworzących oprogramowanie w Elmie [5].

Evan Czaplicki twierdzi, że jego język może konkurować z projektami takimi jak React, jako narzędzie do tworzenia aplikacji internetowych. Język kładzie bardzo duży nacisk na prostotę, łatwość użycia i jakość narzędzi [6].

3. Porównanie

Choć porównywanie kombinacji bibliotek z językiem programowania wydaje się dziwne, to jeżeli spojrzy się na funkcjonalności oferowane przez Reacta i Reduxa w porównaniu do możliwości Elma, można zauważyć pewne podobieństwa w sposobie budowania aplikacji. W tym rozdziale zostały opisane funkcjonalności, które są oferowane przez oba rozwiązania, oraz różnice występujące między Reactem i Reduxem a Elmem dla każdej z funkcjonalności.

3.1. Virtual DOM i składnia

DOM [1] jest niezależnym od platformy i języka programowania interfejsem, który pozwala programom i skryptom na dynamiczny dostęp i aktualizację treści, struktury i stylu dokumentu. Kiedy strona internetowa jest ładowana, przeglądarka tworzy DOM strony, będący obiektową reprezentacją dokumentu HTML. Służy ona jako interfejs umożliwiający pobieranie oraz modyfikację elementów HTML, które w DOM-ie są zdefiniowane jako obiekty.

Każda akcja na stronie powoduje zmianę DOM-u. Ze względu na jego drzewiastą strukturę, sama modyfikacja DOM-u jest szybka. Jednak każdy z modyfikowanych elementów oraz wszystkie jego dzieci muszą dodatkowo przejść przez dwa kosztowne etapy:

1. Reflow będący procesem, podczas którego przeliczane zostają wymiary oraz pozycja elementu. Dokładnie ten sam proces jest uruchamiany na węzłach dzieci, a także elementach, które pojawiają się w DOM-ie później niż główny element. Reflow jest kosztowny, ponieważ zmiana pojedynczego elementu w strukturze DOM-u może spowodować wywołanie Reflow na wielu innych elementach.

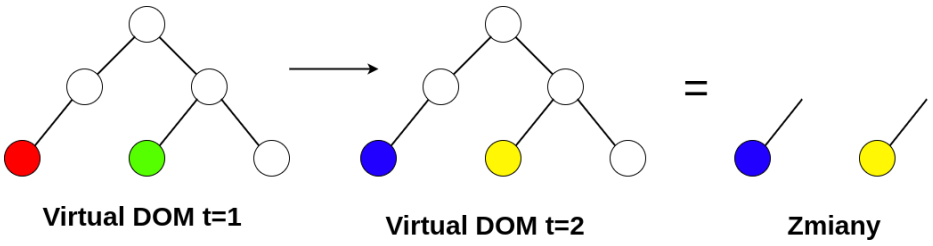
2. Repaint, w którym niektóre partie ekranu muszą zostać zaktualizowane, czy to ze względu na modyfikacje wymiarów i pozycji elementu, czy przez zmiany stylistyczne, takie jak zmiana koloru tła. Etap ten jest kosztowny ponieważ przeglądarka musi sprawdzić widoczność innych węzłów w DOM-ie.

Virtual DOM jest to lekka, niezależna od przeglądarki abstrakcyjna reprezentacja DOM-u. Służy ona między innymi do zminimalizowania kosztu stworzonego przez etapy Reflow i Repaint. Zamiast tworzyć za każdym razem drzewo składające się z węzłów DOM-u, Virtual DOM pozwala na stworzenie tego

[1] z. ang. Document Object Model

drzewa przy pomocy abstrakcyjnych węzłów, które są odpowiednikami faktycznych elementów DOM-u. Dzięki temu wszystkie operacje modyfikujące widok mogą być wykonane na abstrakcyjnej strukturze, a dopiero końcowy rezultat powoduje modyfikację faktycznego DOM-u.

Koncepcja Virtual DOM-u została wykorzystana do tego, aby w każdej klatce budować zupełnie nową scenę. Choć taka operacja wydaje się kosztowna, to tak naprawdę zbudowanie pełnego drzewa Virtual DOM-u jest szybkie, i jest wykorzystywane przy każdej aktualizacji widoku. W momencie gdy następuje zmiana powodująca modyfikację widoku, algorytm porównuje ze sobą stare i nowe drzewo Virtual DOM-u. Wszystkie komponenty, w których nastąpiła jakakolwiek zmiana są oznaczane specjalną flagą, która określa, że dany węzeł został zmodyfikowany. Na tej podstawie budowana jest dokładna lista zmian jakie nastąpiły w widoku. Następnie lista ta jest wykorzystywana do modyfikacji faktycznego DOM-u, lecz nie jako pojedynczo wprowadzane zmiany, a jedna aktualizacja drzewa dokumentu.



Rysunek 3.1. Schemat działania algorytmu porównywania różnic

Zarówno React, jak i Elm posiadają własne implementacje Virtual DOM-u. W Elmie abstrakcyjną reprezentację węzła otrzymujemy przy pomocy funkcji `node`, która jako atrybuty przyjmuje tag, listą atrybutów HTML, oraz listę węzłów dzieci:

```
node : String -> List Attrbiute -> List Html -> Html
```

W przypadku użycia tagów HTML, takich jak `div`, Elm udostępnia funkcje pomocnicze, które posiadają już uzupełniony atrybut tag, pozostawiając nam do określenia atrybuty węzła oraz jego dzieci. Elm nie ma specjalnej składni, która służyłaby do budowania widoków. Wszystkie elementy, z których budowany jest widok aplikacji, są funkcjami. W przypadku budowania widoku jedynym wymaganiem jest, aby funkcja go budująca zwracała rekord specjalnego typu `Html msg`, który jest głównym blokiem służącym do tworzenia wyjściowego kodu HTML.

W przypadku Reacta mamy do czynienia ze znacznie bardziej rozszerzonym podejściem. Bazowym elementem reprezentującym abstrakcyjny węzeł Virtual DOM-u jest ReactElement. Analogicznie jak w przypadku funkcji `node` w Elmie jest to obiekt posiadający informację o tagu, który reprezentuje, atrybutach zdefiniowanego węzła, oraz listę dzieci. Przykład tworzenia takiego elementu można zobaczyć we fragmencie kodu 3.1.

```
1  var divHello = React.createElement(
2  "div",
3  { className: "myclass" },
4  "Hello world!"
5  );
```

Listing 3.1. Javascript

```
1  var divHello = (
2  <div className="myclass">
3  Hello world!
4  </div>
5  );
```

Listing 3.2. JSX

Biorąc pod uwagę to, że każdy element Virtual DOM-u w React'cie jest zbudowany w podobny sposób, można zauważyć, że w przypadku rozbudowanej aplikacji, kod bardzo szybko staje się skomplikowany i niezrozumiały. Aby tego uniknąć, Facebook stworzył specjalne rozszerzenie składni dla JavaScriptu – JSX. Z wyglądu przypomina składnię języka HTML, lecz zasadniczo dostarcza cukier syntaktyczny dla funkcji `createElement`. Fragmenty kodu 3.2 oraz 3.1 są dokładnie tymi samymi wyrażeniami, z tą różnicą, że w drugim przypadku został wykorzystany JSX. Takie podejście pozwala na użycie JSX-a wewnątrz instrukcji JavaScriptu, przypisywanie go do zmiennych, czy też zwracanie z funkcji. Operacja zachodzi także w drugą stronę, co znaczy, że można korzystać z kodu JavaScriptu wewnątrz składni JSX-a. Taki kod musi być objęty nawiasami klamrowymi, aby odróżnić fragmenty napisane w JavaScript'cie od kodu JSX-a.

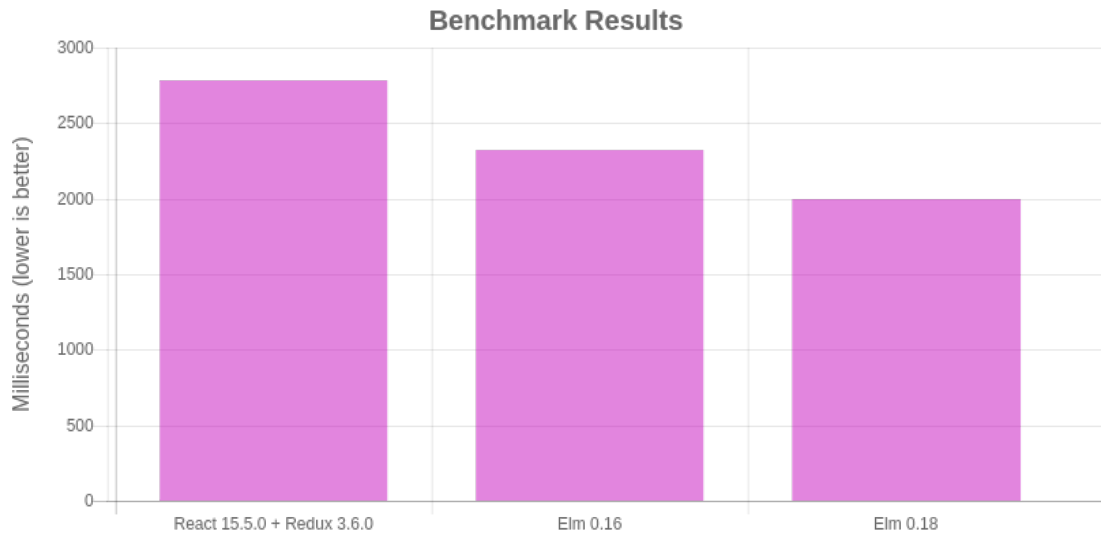
Choć główne założenia Virtual DOM-u w Elmie i React'cie są podobne, to jego implementacje nie są identyczne, w związku z czym można je porównać pod względem wydajnościowym. W tym przypadku wykorzystany został test wydajnościowy stworzony przez twórcę Elma [7]. Pozwala on na porównanie czasu renderowania różnych implementacji aplikacji TodoMVC. Jest to prosty projekt listy zadań, umożliwiający dodawanie i usuwanie wpisów, oznaczanie ich jako zakończone, a także odfiltrowywanie na podstawie statusu wpisów. Test zakłada realistyczny scenariusz, w którym każda zmiana jest wyświetlona jako pojedyncza klatka, tak jakby to faktyczny użytkownik przeprowadzał test. Algorytm scenariusza wykonywany w trakcie pomiaru średniego czasu renderowania wygląda następująco:

1. Stworzenie strony niezawierającej wpisów
2. Dodanie 100 wpisów do listy
3. Oznaczenie każdego z elementów jako zakończony
4. Usunięcie wszystkich wpisów

Dodatkowo zostały przyjęte następujące założenia, które sprawiają, że przeprowadzony test jest sprawiedliwy:

1. Brak zgrupowanych zdarzeń – oznacza to, że zamiast generować zdarzenia w pojedynczej pętli, algorytm tworzy jedno zdarzenie na raz, przechodząc do następnego dopiero po wyrenderowaniu całego widoku. Gdyby takie założenie nie zostało przyjęte, to przykładowo w przypadku dodawania wpisów, zmiany następowałyby na tyle szybko, że zamiast wyświetlić 100 klatek, przeglądarka wyświetliłaby tylko jedną.

2. Brak użycia `requestAnimationFrame` – funkcja ta informuje przeglądarkę o zamiarze wykonania animacji i żąda od przeglądarki wywołania określonej funkcji w celu odświeżenia animacji przed następną zmianą w widoku. Oznacza to, że odświeżenie animacji jest wyrównane do 60 razy na sekundę, niezależnie od tego, jak wiele klatek wygeneruje JavaScript. Elm wykorzystuje tą funkcję do pomijania części klatek, które i tak nie będą widoczne dla użytkownika. W z związku z realistycznym scenariuszem, oraz brakiem podobnej optymalizacji w innych implementacjach, funkcja ta musiała zostać usunięta z Elma w ramach przeprowadzanego testu.



Rysunek 3.2. Porównanie czasu renderowania aplikacji TodoMVC (w oparciu o [7])

Na rysunku 3.2 można zauważyć, że zostały wzięte pod uwagę dwie wersje Elma. Powodem jest tutaj zmiana używanej implementacji Virtual DOM-u. Od początku istnienia Elma aż do wersji 0.16, wykorzystywana była implementacja Matta Escha, która była silnie inspirowana wersją Virtual DOM-u wykorzystywaną w React'cie. Jednak z powodu dużych zmian wprowadzonych w nowszych wersjach Elma, twórca języka był zmuszony stworzyć własną implementację dopasowaną do nowego API.

Wyniki testu pokazują, że implementacja aplikacji w Elmie jest szybsza o ponad sekundę. Twórca Elma w jednym ze swoich artykułów [8] pisze o wykorzystanych technikach, które są powodem takich wyników:

1. Używanie tablic zamiast słowników – iteracja po tablicy jest zawsze o wiele szybsza operacją niż przechodzenie po kluczach słownika.

2. Minimalizacja ilości alokacji – garbage collection jest jednym z kosztownych elementów w analizowanych implementacjach. Im mniej obiektów jest alokowanych, tym lepsza jest wydajność aplikacji. Sposób wykorzystany w Elmie polega na alokowaniu obiektów z pustymi polami. Dzięki temu silniki JavaScriptowe radzą sobie o wiele lepiej z optymalizacją takich obiektów, a obiekt nie zmienia się, nawet gdy wypełnimy go większą ilością informacji.

3. Unikanie powolnych operacji, takich jak pobieranie konkretnego elementu z tablicy.

3.2. Jednokierunkowe flow aplikacji

3.3. Niemutowalność obiektów

3.4. Biblioteki i menedżery pakietów

4. Podsumowanie

Bibliografia

[1] Scott Hanselman. *JavaScript is Assembly Language for the Web: Sematic Markup is Dead! Clean vs. Machine-coded HTML*. Dostęp: 07-12-2017. URL: *https://www.hanselman.com/blog/JavaScriptIsAssemblyLanguageForTheWebSematicMarkupIsDeadCleanVsMachinecodedHTML.aspx*.

[2] Amos Ndegwa. *What is a Web Application?* Dostęp: 13-12-2017. URL: *https://www.maxcdn.com/one/visual-glossary/web-application/*.

[3] *Usage of JavaScript for websites*. Spraw. tech. *https://w3techs.com/technologies/details/cp-javascript/all/all*. 2017.

[4] *Redux documentation*. *https://redux.js.org*. 2017.

[5] Evan Czaplicki. *New Adventures for Elm. Joining NoRedInk and creating Elm Software Foundation*. *http://elm-lang.org/blog/new-adventures-for-elm*. 2016.

[6] *An Introduction to Elm*. *https://guide.elm-lang.org*. 2017.

[7] Evan Czaplicki i Rogério Chaves. *rogeriochaves/react-angular-ember-elm-performance-comparison: Comparing performance of Elm, React, Ember, and Angular*. URL: *https://github.com/rogeriochaves/react-angular-ember-elm-performance-comparison*.

[8] *Blazing Fast HTML. Round Two*. *http://elm-lang.org/blog/blazing-fast-html-round-two*. 2016.