JCSI 2 (2016) 98-103
Journal of Computer Sciences Institute

Wysłane: 2016-11-15
Przyjęte: 2016-11-18

Analiza porównawcza narzędzi do budowania aplikacji Single Page Application – AngularJS, ReactJS, Ember.js

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

Streszczenie. Tematem artykułu jest analiza porównawcza jednych z aktualnie najpopularniejszych frameworków języka JavaScript, służących do budowania aplikacji typu Single Page Application. Podczas analizy przyjęto cztery kryteria porównawcze: trudność nauki biblioteki, dodatkowe narzędzia charakterystyczne dla danego frameworka, wydajność na urządzeniach stacjonarnych oraz wydajność na urządzeniach mobilnych.

Słowa kluczowe: Angular; React; Ember; Single Page Application

*Autor do korespondencji.

Adres e-mail: radoslaw.nowacki@pollub.edu.pl

Comparative analysis of tools dedicated to building Single Page Applications – AngularJs, ReactJS, Ember.js

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

Abstract. The subject of this article is a comparative analysis of some of the most popular frameworks, of JavaScript programming language, dedicated to creating Single Page Applications. There were four criteria accepted for the purpose of the research. The criteria are: difficulty of learning the given library, additional tools specific for given framework, performance on desktop devices and performance on mobile devices.

Keywords: Angular; React; Ember; Single Page Application

*Corresponding author.

E-mail address: radoslaw.nowacki@pollub.edu.pl

1. Wstęp

Dynamiczny rozwój technologii webowych, takich jak HTML, CSS oraz w szczególności JavaScript wpłyną na ogromne zmiany, zarówno w procesie tworzenia, jak i w samej strukturze stron internetowych. Dodatkowe możliwości jakie oferują te technologie pozwoliły na znaczne poszerzenie zastosowania tych właśnie stron. Jeszcze niedawno służyły one głównie jako wizytówki firm zawierające bardzo dużo tekstu, i kilka obrazków. Obecnie wygląda to zupełnie inaczej. Tradycyjne strony internetowe zawierają obecnie w większości grafikę, a tylko najważniejsze informacje są umieszczone na stronie.

Zmiany wizualne nie są jednak największą zmianą jakie spowodował dynamiczny rozwój tych technologii. Przyczynił się on również do ogromnego wzrostu sposobów wykorzystania stron internetowych [1]. Ewoluowały one do ogromnych serwisów społecznościowych, serwisów informacyjnych i rozrywkowych, ale są także szeroko wykorzystywane do zarządzania całymi strukturami firm. Mogą to być przykładowo systemy klasy CRM (Customer Relationship Service), a nawet klasy HIS (Hospital Information System) służące do obsługi szpitali.

Taka zmiana w zastosowaniu spowodowała, że coraz częściej używana jest nazwa aplikacja internetowa lub aplikacja webowa, zamiast strona internetowa. Nazwa ta dużo lepiej odzwierciedla, że aktualnie takie aplikacje mają szerokie zastosowanie [2].

Jednym z najpopularniejszych rodzajów takich aplikacji są aplikacje typu Single Page Aplication (SPA). Są to serwisy, które po pierwszym wczytaniu nigdy nie odświeżają przeglądarki internetowej użytkownika, a wszystkie dane potrzebne do działania takiej strony są pobierane dynamicznie. Dzięki takiemu podejściu reakcja na akcję użytkownika następuje natychmiastowo, bez zbędnego oczekiwania na przeładowanie strony [1].

Oczywiście podejście to wiąże się z pewnymi trudnościami. Jest ono znacznie trudniejsze w implementacji, wymaga częstej ingerencji JavaScript w strukturę drzewa DOM. Dodatkowo, aby początkowe założenia aplikacji SPA, czyli płynne działanie i natychmiastowe akcje, były spełnione, bardzo ważna jest wydajność takich aplikacji.

Implementacja takich aplikacji jest wymagająca, ale zostaje ona znacznie ułatwiona dzięki zastosowaniu dostosowanych do tego celu frameworków języka JavaScript [3].

W tym artykule opisany został proces analizy trzech bibliotek: AngularJS, React oraz Ember.js. Zostały one

wybrane, ponieważ wszystkie są narzędziami open-source z licencją MIT, oznacza to, że ich kod źródłowy jest dostępny na platformie GitHub dla każdego użytkownika, a biblioteki mogą być wykorzystywane komercyjnie bez żadnych opłat. Kolejnym kryterium wyboru tych frameworków jest ich ogromna popularność oraz sukces prestiżowych aplikacji napisanych za ich pomocą. Ember został wykorzystany w takich projektach jak Playstation Now oraz Apple Music. Za pomocą React napisane zostały portale Airbnb oraz Instagram. Angular natomiast został wykorzystany przy tworzeniu stron: Ryanair oraz Google Cast.

2. Kryteria analizy

2.1. Trudność nauki biblioteki

Pierwszym kryterium jest trudność nauki frameworka. Jest to bardzo ważne kryterium podczas wybierania technologii do projektu. W tym artykule miarą trudności biblioteki będzie tzw. próg wejścia.

Próg wejścia jest określeniem opisującym zakres wiedzy jaki należy przyswoić w celu stworzenia prostej, w pełni działającej aplikacji za pomocą badanej biblioteki. Trudność nauki jest wartością, której nie da się zmierzyć, w związku z tym na potrzeby tej pracy przyjęto, że jej wyznacznikiem będzie liczba zagadnień, które muszą zostać przyswojone w celu stworzenia prostych, ale w pełni funkcjonalnych, aplikacji demonstracyjnych.

2.2. Wydajność na urządzeniach stacjonarnych

Wydajność jest jednym z najważniejszych aspektów aplikacji SPA. Każde opóźnienie negatywnie wpływa na odczucia użytkownika i może potencjalnie wpłynąć na porażkę strony.

Na wydajność składają się dwa elementy, którymi są czas ładowania aplikacji, oraz czas reakcji na akcję użytkownika. Aby zbadać ten aspekt, stworzone aplikacje demonstracyjne zostały zapełnione dużymi zestawami danych, a następnie zostały zmierzone wspomniane wcześniej czasy. Zestawy danych zawierały dane osobowe fikcyjnych użytkowników, zawierające sześć pól: imię, nazwisko, miasto, telefon, pesel oraz email. Przeprowadzono po pięć prób dla zestawów danych składających się kolejno z: 50, 100, 500 oraz 1500 rekordów.

W celu przeprowadzenia analizy dla frameworków Angular oraz React, wykorzystano narzędzie webpack, posiadające wbudowany serwer: webpack-dev-server umożliwiający uruchomienie aplikacji [4]. W przypadku Embera skorzystano z wbudowanego narzędzia, ember-server, dającego podobne możliwości.

2.3. Wydajność na urządzeniach mobilnych

W dzisiejszych czasach aplikacje internetowe są użytkowane równie często na urządzeniach mobilnych co na urządzeniach stacjonarnych. W związku z tym utrzymanie wydajności stało się jeszcze większym wyzwaniem, gdyż bardzo często przenośne urządzenia nie posiadają tak dobrej specyfikacji technicznej jak tradycyjne komputery [5].

Analiza wydajności na tych urządzeniach została przeprowadzona w sposób analogiczny do analizy na urządzeniach stacjonarnych. Aplikacje testowe zostały poddane stresorowi w postaci dużych zestawów danych, następnie zbadano czas ładowania strony oraz czas reakcji na zmiany modelu aplikacji.

Zarówno webpack-dev-server jak i ember-server dają programiście możliwość przekazania parametru –host, którego ustawienie na wartość 0.0.0.0 umożliwia uruchomienie lokalnie uruchomionej aplikacji internetowej na urządzeniu mobilnym połączonym z tą samą siecią.

Aplikacja była uruchamiana na przeglądarce mobilnej Google Chrome, ponieważ daje ona możliwość badania aplikacji internetowej za pomocą Chrome DevTools po podłączeniu narzędzia mobilnego do komputera stacjonarnego lub laptopa. Chrome DevTools zawierają natomiast zakładkę linii czasu, która umożliwia wykonanie potrzebnych pomiarów.

Wszystkie testy zostały przeprowadzone na smartphone HTC One m8, z systemem Android w wersji 6.0.

2.4. Dodatkowe narzędzia

Każdy z frameworków oferuje pewien zestaw narzędzi ułatwiających lub przyspieszających proces tworzenia aplikacji z ich wykorzystaniem. Mogą być to zarówno narzędzia deweloperskie, ułatwiające pisanie i analizowanie kodu oraz wykrywanie błędów, ale również zestawy gotowych rozwiązań, a nawet platformy umożliwiające tworzenie natywnych aplikacji mobilnych używając danego frameworka języka JavaScript.

W celu analizy frameworków pod tym kątem, zostały wybrane po dwa najważniejsze narzędzia wpierające pracę programistów. Następnie wartość jaką one wnoszą została oceniona w celu wyłonienia najlepszego pod tym względem frameworka.

3. Przedstawienie aplikacji demonstracyjnych

Wydajność jest jednym z najważniejszych aspektów aplikacji SPA. Każde opóźnienie negatywnie wpływa na odczucia użytkownika i może potencjalnie spowodować porażkę komercyjną strony. Wykonanie analizy wymagało stworzenia wersji demonstracyjnych dla każdego z badanych frameworków. Aby wyniki analizy były wiarygodne, wszystkie z aplikacji posiadają identyczne funkcjonalności. Ponadto, przedmiotem badań są biblioteki JavaScript, ważne było więc również aby kod CSS i HTML był jak najbardziej zbliżony w każdej z wersji demonstracyjnych.

Jako aplikacje demonstracyjne, stworzone zostały strony typu CRUD (Create Read Update Delete), posiadające możliwość tworzenia, odczytywania, aktualizowania

i usuwania rekordów. Rekordami były dane osobowe fikcyjnych użytkowników.

Dane te zostały wyświetlone w tabeli. W jej drugim wierszu osadzony został formularz dodawania nowego użytkownika. Od trzeciego wiersza zaczyna się lista fikcyjnych użytkowników. Dodatkowo aplikacje musiały umożliwiać modyfikowanie użytkowników, jednakże wymaganiem było, aby zmiany były aktualizowane na bieżąco, bez konieczności akceptacji, ponieważ ciągłe aktualizowanie treści strony jest charakterystyczne dla aplikacji SPA. Ponadto, aby sprawdzić jakie możliwości daje framework, formularz edycji każdego z wierszy jest początkowo ukryty, dopiero przejście w tryb edycji danego wiersza sprawia, że ten formularz jest widoczny dla użytkownika. Ukrywanie treści również jest bardzo ważnym elementem aplikacji tego typu.

Ostatnią funkcjonalnością była wyszukiwarka użytkowników. Wymaganiem było filtrowanie listy po każdej zmianie w polu wyszukiwania, dodatkowo wyszukiwarka powinna działać bez względu na wielkość wpisanych liter.

4. Analiza porównawcza

4.1. Trudność nauki bilbioteki

Ocena progu wejścia dla każdego z frameworków została wykonana na podstawie procesu tworzenia aplikacji demonstracyjnej. Głównym wyznacznikiem była obszerność materiałów, które należało przyswoić w celu stworzenia aplikacji. Ponadto negatywnie wpływały na nią nieoczekiwane problemy napotkane podczas tworzenia danej aplikacji.

Początki z AngularJS mogą wydawać się trudne, framework ten zawiera bowiem bardzo wiele terminów dla niego specyficznych takich jak digest cycle, scope lub watchers, jednakże jest to wrażenie mylne, ponieważ ich zrozumienie jest konieczne dopiero w późniejszych etapach nauki frameworka. Początki okazały się bardzo łatwe, doskonałą pomocą była dokumentacja techniczna. Odtwarzając kroki z dokumentacji, należało stworzyć moduł, a następnie zdefiniować w nim komponenty [6].

W przypadku aplikacji demonstracyjnej były to dwa komponenty – lista użytkowników oraz użytkownik. Następnie do każdego z nich należało zadeklarować szablon HTML, co również okazało się bardzo proste. Listę udało się wyświetlić za pomocą dyrektywy ng-repeat doskonale opisanej w dokumentacji technicznej. Podczas wyświetlania listy za pomocą komponentu użytkownika pojawił się jednak problem. Wykorzystanie komponentu użytkownika jako wiersz tabeli zaburzyło strukturę tabeli HTML, ponieważ wewnątrz znacznika *tbody* dopuszczalne są tylko kolejne wiersze *tr*. Aby rozwiązać ten problem, do wyświetlenia użytkownika wykorzystano dyrektywę atrybutową. Następnie należało stworzyć kontrolery dla danych komponentów, odpowiedzialne za sterowanie zachowaniem aplikacji. Kontrolery są jednak zwykłymi klasami, w podstawowym przypadku ich stworzenie nie wymaga więc żadnej wiedzy na temat frameworka. Ostatnim krokiem było dodanie filtrowania listy. Efekt ten osiągnięto za pomocą dyrektywy ng-model oraz zastosowania filtra dyrektywy ng-repeat.

W przypadku AngularJS stworzenie aplikacji wymagało więc podstawowej wiedzy na temat komponentów i wbudowanych dyrektyw. Dodatkowo z racji wspomnianego problemu również wiedza na temat tworzenia własnych dyrektyw była wymagana.

React okazał się na początku dużo trudniejszy. Sam framework nie zapewnia dobrej obsługi modelu aplikacji, dlatego do stworzenia aplikacji demonstracyjnej wykorzystano bibliotekę Redux. Jest to implementacja narzędzia do zarządzania modelem zgodnie z architekturą Flux. Użycie tego narzędzia wymaga więc znajomości tej architektury, ale wymaga również umiejętności budowania czystych funkcji, czyli takich, które nie posiadają efektów ubocznych i nie mutują stanu aplikacji. Opanowanie tych podstaw okazało się dużo trudniejsze niż w przypadku AngularJS. Stworzenie aplikacji wymagało znajomości komponentów frameworka React, ale również umiejętności tworzenia funkcji redukujących, akcji, kontenerów oraz zarządzaniem obiektem store. Dodatkowym utrudnieniem może być początkowo przyzwyczajenie do JSX, które jest rozszerzeniem do języka JavaScript, dodającym do niego składnię przypominającą XML. JSX umożliwia pisanie kodu w pliku JavaScript, podobnego do kodu HTML, który jest następnie kompilowany do odpowiednich funkcji – w przypadku React jest to funkcja *createElement*.

W przypadku Ember, wydawać by się mogło, że dzięki zastosowaniu Ember-CLI stworzenie aplikacji demonstracyjnej będzie bardzo proste, jednakże próg wejścia jest dość wysoki. O ile tworzenie elementów takich jak komponenty czy szablony sprowadza się do wpisania jednej linijki w terminalu, to zrozumienie jak one ze sobą współgrają jest dużo trudniejsze [7]. Do stworzenia badanej aplikacji wymagana była znajomość narzędzia Mirage, Ember-CLI, komponentów oraz wiedzy na temat działania obiektu store, odpowiedzialnego za operacje na rekordach. Dodatkową trudnością jest konieczność znajomości handlebars, czyli narzędzia do tworzenia szablonów. Jest ono podobne do klasycznego HTML, jednakże mylące może być przykładowo definiowanie elementów takich jak input wewnątrz podwójnych nawiasów klamrowych zamiast tradycyjnego znacznika.

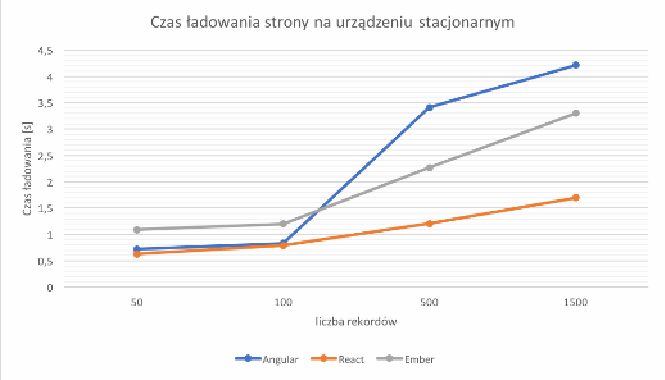
Angular jest więc zdecydowanie frameworkiem o najniższym progu wejścia. Nieco trudniejszy jest Ember, gdyż wymaga znajomości większej liczby elementów charakterystycznych dla niego, ale jednocześnie umożliwia ich łatwiejszą implementację za pomocą Ember-CLI. Najtrudniejszy okazał się React, ponieważ poza znajomością samej biblioteki, należało zapoznać się z Reduxem. Wymagał on również umiejętności pisania funkcji niemutujących stanu aplikacji.

4.2. Wydajność na urządzeniach stacjonarnych

W celu zbadania wydajności na urządzeniach mobilnych, aplikacje zostały zasilone fikcyjnymi zestawami danych.

Następnie aplikacje te zostały uruchomione za pomocą webpack-dev-server lub ember-server. Do mierzenia czasu reakcji aplikacji użyte zostały narzędzia developerskie przeglądarki Google Chrome.

Pierwszym badanym elementem był czas ładowania strony w zależności od liczby rekordów wyświetlanych jednocześnie na stronie. Wyniki przedstawione zostały na rysunku 1. Na osi X znajduje się liczba rekordów na stronie, natomiast na osi Y – badana wartość, czyli czas ładowania strony.

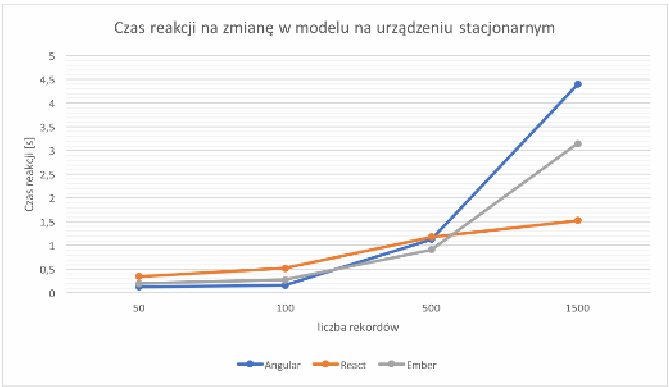


Rys. 1. Czas ładowania strony na urządzeniu stacjonarnym.

Najszybszym frameworkiem pod tym względem okazał się React. Różnica przy 50 i 100 wierszach tabeli jest znikoma, jednakże przy próbach przeprowadzonych dla 500 i 1500 elementów łatwo można zauważyć dużą różnicę w wydajności tej biblioteki pod względem prędkości ładowania strony.

Przy niewielkim zestawie danych Ember poradził sobie najgorzej, jednakże jest on dużo bardziej wydajny niż AngularJS przy większym zestawie danych. Dla 1500 rekordów był on o sekundę szybszy.

Podobnie wyglądała sytuacja w przypadku czasu reakcji na zmianę w modelu aplikacji. Charakterystyczne dla aplikacji typu SPA jest ciągłe aktualizowanie treści na stronie po wykryciu, dlatego wydajność pod tym względem jest tak istotna. Wyniki analizy przedstawiono na rysunku 2.

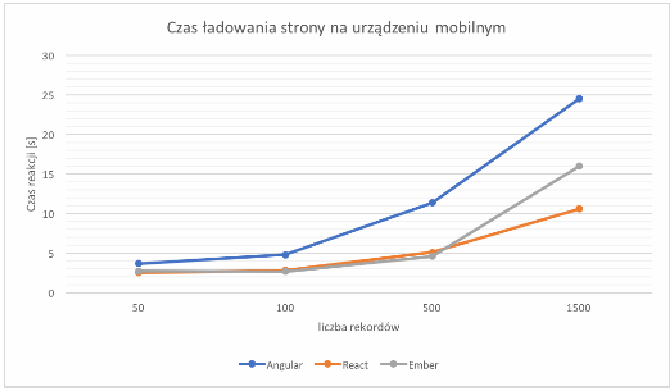


Rys. 2. Czas reakcji na zmianę w modelu na urządzeniu stacjonarnym.

Dla małego zestawu danych najszybszy okazał się Angular, jednakże jego skalowalność pozostawia dużo do życzenia. Już przy 500 elementach na stronie Ember znacznie szybszy od pozostałych. Jednakże podobnie jak w przypadku czasu ładowania strony, różnice w wydajności najłatwiej jest ocenić przy bardzo dużym zestawie danych. Można więc zauważyć, że React był znacznie szybszy od rywali, gdyż przy 1500 rekordach na stronie, zareagował na zmianę w czasie o ponad połowę krótszym od Ember, a niemalże trzy razy krótszym niż w przypadku Angular.

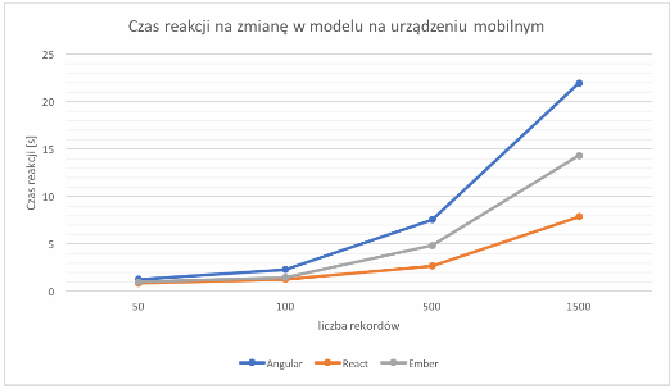
4.3. Wydajność na urządzeniach mobilnych

W celu zbadania wydajności na urządzeniach mobilnych, wykonano próby analogiczne do tych, wykonanych na urządzeniu stacjonarnym. Wyniki pomiarów czasu ładowania strony zostały przestawione na rysunku 3, natomiast reakcji na zmianę w modelu na rysunku 4.



Rys. 3. Czas ładowania strony na urządzeniu mobilnym.

Wyniki analizy na urządzeniu mobilnym wyglądają podobnie jak w przypadku urządzenia stacjonarnego. Pomimo tego, że Ember jest minimalnie szybszy przy małych zestawach danych, to po ostatnim zestawie danych łatwo możemy zaobserwować, że React jest dużo lepiej skalowalny. Angular ponownie został zdeklasowany przez rywali. Jego czas już przy 500 użytkownikach na liście był zdecydowanie zbyt duży, aby strona była oddana do użytku.



Rys. 4. Czas reakcji na zmianę w modelu na urządzeniu mobilnym.

Warto również zauważyć, że czasy ładowania wszystkich stron są znacznie większe niż w przypadku urządzenia stacjonarnego. Właśnie dlatego, pisząc aplikację dostosowaną do urządzeń mobilnych tak ważne jest badanie jej wydajności.

W przeciwieństwie do wyników analizy czasu ładowania strony, wyniki pomiarów czasu reakcji na zmianę w modelu są jednoznaczne. Już w przypadku najmniejszego zestawu danych React uzyskał najlepszy czas. Angular natomiast ponownie okazał się najwolniejszy, co widać w szczególności przy ostatnim zestawie danych.

4.4. Wnioski z analizy wydajności

Z powyższych badań można wywnioskować, że wszystkie frameworki doskonale radzą sobie z niewielkimi oraz średnimi zestawami danych. Różnice między nimi są dla takich zestawów znikome, a wydajność jest wystarczająca do zapewnienia płynnego działania aplikacji. Wraz ze wzrostem liczby wyświetlanych elementów, różnice między badanymi bibliotekami stają się bardziej wyraziste. Najbardziej wydajny dla dużych zestawów danych, czyli tym samym najlepiej skalowalny jest React. Dzieje się tak ponieważ został w nim zaimplementowany mechanizm Virtual DOM, działający tak, że po wykryciu zmian wymagających modyfikacji drzewa DOM, React tworzy coś w rodzaju wirtualnej kopii tego drzewa, a następnie bardzo wydajne mechanizmy znajdują najlepszą akcję pozwalającą osiągnąć pożądaną strukturę strony [8]. Przykładowo zamiast usuwać elementu z środka listy, zostanie usunięty ostatni wiersz, a zaktualizowana zostanie tylko treść pozostałych tak aby zaoszczędzić czas potrzebny na wykonanie kosztownych manipulacji.

Nieco wolniejszy okazał się Ember, jednakże jego wyniki również były stosunkowo zadowalające. Pomimo tego, że przy 1500 elementach aplikacja stawała się dość trudna w użytku a czas oczekiwania na zmiany przekroczył 3s, co jest niedopuszczalne, to przy nieco mniejszym zestawie danych radził on sobie podobnie, a nawet nieco lepiej od React.

Pomimo bardzo dobrych wyników dla bardzo małych zestawów danych, najgorzej wypadł Angular. Aplikacje posiadające 50 elementów nie są zbyt częstym zjawiskiem w realnym świecie, dlatego największe znaczenie miały późniejsze próby. Angular nie jest zbyt dobrze skalowalny i wymaga uważnego projektowania komponentów. Dzieje się tak ponieważ wykrywanie zmian w tej bibliotece jest zaimplementowane jako tzw. dirty checking. Oznacza to, że jeśli jakakolwiek z wartości obserwowanych, czyli tzw. watcher, zmieni się, Angular rozpocznie cykl digest, który sprawdzi wszystkie pozostałe watchery i jeśli to konieczne, zaktualizuje je. Jest to mechanizm bardzo niewydajny, i powoduje, że już przy 2000 takich obserwatorów strona może działać mniej płynnie [9. 10].

4.5. Dodatkowe narzędzia

W celu analizy frameworków pod tym kątem, z każdego z nich wybrano po dwa najważniejsze narzędzia ułatwiające pracę z daną biblioteką.

W przypadku Angular, wybrane narzędzia to ui-bootstrap oraz AngularJS Batarang. Pierwsze z nich to zestaw dyrektyw, napisany z zastosowaniem bardzo popularnego frameworka CSS – Bootstrap. Jest to zestaw uniwersalnych rozwiązań, które mogą być w bardzo łatwy sposób zaimplementowane w danym projekcie. Dodatkowo są one łatwo modyfikowalne, dlatego można je dostosować na potrzeby większości projektów. Jest to świetne narzędzie, gdyż pozwala zaoszczędzić bardzo dużo czasu na implementację często powtarzających się w projektach funkcjonalności, jak np. Accordion.

AngularJS Batarang to wtyczka do przeglądarki Google Chrome, pozwalająca na głębszą inspekcję aplikacji. Po zainstalowaniu wtyczki, pojawia się dodatkowa zakładka w narzędziach deweloperskich, zawierająca informacje charakterystyczne dla Angular, czyli przykładowo strukturę drzewa $scope, ich zawartość oraz liczbę aktywnych funkcji obserwujących.

W przypadku React wybranymi narzędziami są React Native oraz Redux devtools. React Native zasługuje na miano oddzielnego frameworka, jednakże jego ideą jest pozwolenie programiście na tworzenie natywnych aplikacji mobilnych za pomocą React. Różni się on od obecnych do tej pory na rynku narzędzi, ponieważ powstała aplikacja ma strukturę identyczną jak w przypadku standardowych aplikacji iOS i Android, przez co jest bardzo wydajna.

Drugim narzędziem są Redux DevTools. Podobnie jak AngularJS Batarang jest to wtyczka do Google Chrome, udostępniająca dodatkową zakładkę w narzędziach deweloperskich. Narzędzie daje to w połączeniu z aplikacją napisaną w Redux daje ogromne możliwości. Wyświetlane są tam wszystkie wykonane w aplikacji akcje użytkownika oraz ich wpływ na stan aplikacji. Dzięki temu, że w Redux stan ten jest przechowywany w jednym obiekcie store, Redux DevTools umożliwia dowolne cofanie, przyspieszanie, odtwarzanie wykonywanych akcji. Znacznie ułatwia to proces tworzenia aplikacji i przyspiesza odnajdywanie błędów.

Wybranymi narzędziami dla Ember są Ember-CLI oraz Mirage. Pierwsze z nich służy do wspomagania tworzenia aplikacji poprzez udostępnienie programiście dodatkowych komend w konsoli lub terminalu. Pozwala ono na stworzenie bazowej struktury projektu oraz dodanie do aplikacji każdego dostępnego w Ember elementu, czyli m.in. komponentów, ścieżek i szablonów.

Drugim narzędziem jest Mirage. Umożliwia ono definiowanie w łatwy sposób sztucznych odpowiedzi serwera. Jest to bardzo duże ułatwienie podczas tworzenia aplikacji typu front-end, kiedy strona serwerowa nie została jeszcze zaimplementowana. Wbrew pozorom jest to sytuacja bardzo częsta i bywa bardzo uciążliwa. Mirage umożliwia zwracanie różnych kodów odpowiedzi, danych i bardzo wspomaga pracę programisty.

Wyniki analizy narzędzi nie są jednoznaczne. Wszystkie badane narzędzia okazały się bardzo pomocne. Mimo to, za najlepszy można uznać React, ponieważ React Native jest technologią pionierską, oferującą tworzenie aplikacji natywnych w JavaScipt, bez utraty wydajności, jak było to w przypadku rozwiązań hybrydowych takich jak np. Cordova. Również Redux DevTools okazało się imponujące, możliwość dowolnego cofania i przywracania akcji użytkownika to coś,

z czym ciężko spotkać się w przypadku innych technologii. Walkę o drugie miejsce między Angular i Ember można uznać za remis, gdyż wszystkie narzędzia oferowane przez te biblioteki są na bardzo wysokim poziomie.

5. Wnioski

W celu łatwiejszej analizy otrzymanych wyników, stworzona została skala punktowa. Najlepsza biblioteka w danej kategorii otrzymała 2 punkty, druga w kolejności biblioteka 1 punkt, a najgorsza 0 punktów. Wyniki zostały przedstawione w tabeli 1.

Tabela 1. Zestawienie wyników analizy

	Angular	**React**	**Ember**
Stopień trudności przyswajania wiedzy	2	0	1
Wydajność aplikacji	0	2	1
Wydajność aplikacji na urządzeniach mobilnych	0	2	1
Dodatkowe narzędzia	1	2	1
Suma	3	6	4

Jak widać najlepszym z frameworków według niniejszej analizy okazał się React. Jest to zasługa głównie świetnej wydajności, która była głównym elementem tej analizy. Dodatkowym atutem są udostępniane narzędzia.

Nieco gorszy okazał się Ember. We wszystkich kategoriach zajął on drugie miejsce. Jego wydajność jest nieco słabsza od React, jednak znacznie lepsza od Angular. Ponadto początki w tym frameworku są znacznie łatwiejsze od React. Oferuje on również przydatne narzędzia deweloperskie.

Najgorszym według niniejszej analizy okazał się Angular. Przyczyną tak rozczarowującego wyniku okazała się słaba wydajność. Angular ma jednak swoje bardzo silne strony. Przede wszystkim jest on bardzo przyjazny początkującym programistom, Znacznie łatwiej zacząć pracę z nim pracę niż z pozostałymi bibliotekami. Dodatkowo oferuje on równie dobre narzędzia deweloperskie co Ember.

Wyniki analizy są jednoznaczne, jednak warto podkreślić, że cztery kryteria nie są wystarczające, aby jednoznacznie określić który z frameworków jest najlepszy. Mają jedynie podkreślić silne i słabe cechy badanych frameworków w porównaniu do ich konkurencji. React jest więc bardzo wydajny i oferuje najlepsze narzędzia, kosztem jest jednak duży próg wejścia. Ember oferuje umiarkowaną wydajność i niezłe narzędzia, ale również wymaga sporej wiedzy nawet przy tworzeniu małych aplikacji. Pierwsze kroki są najłatwiejsze z Angular, oferuje on także dobre dodatkowe narzędzia, kosztem jest jednak wydajność, dlatego dobrze sprawdza się on podczas do budowania serwisów, które wyświetlają jednocześnie niewielką ilość danych wymagających ciągłej aktualizacji.

Literatura

[1] Beda B.: Single Page Web Applications Security, Bucharest University of Economic Studies, ISSN: 2067-4074 (Print), 2015.

[2] Minović M., Vesic S.: Single Page Applications: Trend or Future, Info, 2015.

[3] Enache M. C.: Web Application Frameworks, Dunarea de Jos University of Galati ISSN: 1584-0409, 2015.

[4] Vepsäläinen J.: Webpack and React from apperentice to master, Survivejs, 2015.

[5] Weyl E.: HTML5 : strony mobilne, Wydawnictwo Helion, 2014.

[6] Darwin B. P., Kozłowski P.: Mastering Web Applications Development with AngularJs, Packt Publishing, 2013.

[7] Brady T., Cravens J.: Ember.js dla webdeveloperów, Wydawnictwo Helion, 2015.

[8] Fedosejev A.: React.js Essentials, Packt Publishing, 2015.

[9] Freeman A.: AngularJS : profesjonalne techniki, Wydawnictwo Helion, 2015.

[10] Green B., Seshadri S.: AngularJS, Wydawnictwo Helion, 2014.