JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 17 (2020) 396–400
Received: 2 November 2020
Accepted: 16 November 2020

Comparative analysis of the technology used to create multi-platform applications on the example of NW.js and Electron

Analiza porównawcza technologii tworzenia aplikacji wieloplatformowych na przykładzie NW.js i Electron

Maciej Hołowiński*, Beata Pańczyk

Department of Computer Science, Lublin University of Technology, ul. Nadbystrzycka 38, 20-618 Lublin, Poland

Abstract

Electron and NW.js are technologies that allow you to create multi-platform desktop applications. This article discusses the performance comparison of these frameworks. The research was undertaken in order to find an alternative to the Electron, which, despite its considerable popularity, is criticized in terms of its performance. For the purposes of the research, a test application with social media functionalities was designed and implemented. An identical application was created on both compared platforms and packaged for Windows 10 and Ubuntu 12.04 operating systems. Applications were tested in terms of the speed of data rendering, cache occupation and the size of the packed application. In each of the tests performed, NW.js obtained a better result than Electron, especially in tests consisting in downloading a relatively small amount of data. Based on the research carried out and the results obtained, it was found that in terms of performance, the NW.js framework is a much better solution than the Electron framework.

Keywords: NW.js; Electron; cross-platform development; comparative analysis

Streszczenie

Electron i NW.js to technologie pozwalające na tworzenie wieloplatformowych aplikacji desktopowych. Niniejszy artykuł omawia porównanie tych frameworków pod względem wydajności. Badania podjęto w celu znalezienia alternatywy dla Electron, który mimo swojej popularności, krytykowany jest pod względem oferowanej wydajności. Na potrzeby badań zaprojektowano i zaimplementowano aplikację testową z funkcjonalnościami mediów społecznościowych. Identyczna aplikacja została stworzona na obydwu porównywanych platformach oraz spakowana na systemy operacyjne Windows 10 i Ubuntu 12.04. Aplikacje badane były pod kątem szybkości renderowania danych, zajętości w pamięci podręcznej oraz rozmiaru spakowanej aplikacji. W każdym przeprowadzonym teście NW.js uzyskał lepszy wynik od Electron, szczególnie w testach polegających na pobraniu stosunkowo małej ilości danych. Na podstawie zrealizowanych badań i uzyskanych wyników stwierdzono, że pod względem wydajności framework NW.js jest znacznie lepszym rozwiązaniem od frameworka Electron.

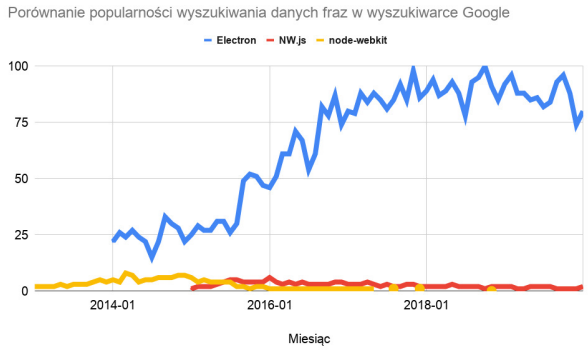
Słowa kluczowe: NW.js; Electron; tworzenie aplikacji wieloplatformowych; analiza porównawcza

1. Wstęp

Przed rozpoczęciem budowania aplikacji desktopowej programista musi zastanowić się nad wybraniem odpowiedniego systemu operacyjnego. Każdy z nich ma swoje własne API i środowiska programistyczne. Poznanie wszystkich, w przypadku chęci wydania aplikacji na wiele systemów, jest bardzo czasochłonne. Programista jednak ma również możliwość napisanie jednej aplikacji, która będzie mogła być rozprowadzana na wiele systemów operacyjnych (Windows, Linux, MacOS). Taki wybór pozwala oszczędzić czas oraz pomaga w obniżeniu kosztów produkcji [1]. Tworzenie aplikacji na różne platformy nie jest szczególnie nową koncepcją - przez ostatnie dziesięciolecie było to osiągalne za pomocą m.in.: Flash Air, JavaFX i Silverlight [2].

Jednak według corocznej ankiety prowadzonej przez stronę StackOverflow, technologie te z roku na rok tracą popularność [3]. Bazując na tej samej ankiecie najpopularniejszą technologią w roku 2019 jest JavaScript, a w kategorii „Frameworki, biblioteki i narzędzia" na pierwszym miejscu stoi Node.js. Zatem w omawianiu technologii pozwalających na tworzenie aplikacji wieloplatformowych warto przytoczyć Electron oraz NW.js, u fundamentów których leży Node.js, co za tym idzie – JavaScript. Porównując Electron i NW.js (dawniej node-

*Corresponding author
Email address: holowinski.maciej@pollub.edu.pl (M. Hołowiński)



Rysunek 1: Porównanie popularności Electron – NW.js (node-webkit) [4]

webkit) pod względem popularności, Electron jest zdecydowanym faworytem (Rysunek 1). W artykule postawiono następującą tezę: "Framework NW.js, jest pod względem wydajnościowym lepszym rozwiązaniem od frameworka Electron".

2. Przegląd literatury

W wyniku przeglądu literatury stwierdzono niewielki udział artykułów poruszających tematykę wydajności frameworków Electron i NW.js. Publikacje na temat NW.js omawiają jedynie sposób działania technologii oraz etapy implementacji aplikacji [5]. Pod względem dostępnych artykułów i książek znacznie większą popularnością cieszy się Electron. Przegląd serwisów oferujących dostęp do artykułów naukowych takich jak Google Scholar, Research Gate oraz BazTech nie wykazał żadnej pracy porównującej omawiane technologie pod względem wydajnościowym. Dostępne są jedynie badania porównujące framework Electron z innymi technologiami pozwalającymi na budowanie aplikacji desktopowych.

Artykuł [6] omawia różnice pomiędzy Electron, a NW.js pod względem czysto teoretycznym. Nie przedstawione są w nim żadne badania, prezentowane są natomiast różnice w funkcjonalnościach omawianych technologii. Technologie są porównywane pod względem architektury, możliwości pozwalających na zmianę interfejsu użytkownika, integracji z systemem operacyjnym, dostępem do systemu plików i obsługą multimediów. Autorzy wywnioskowali, że Electron posiada o wiele większą funkcjonalność pod względem budowania aplikacji, przy czym NW.js jest o wiele łatwiejszą technologią do opanowania.

Artykuł [7] porównuje implementacje oraz wydajność aplikacji wieloplatformowej tworzonej w Electron oraz JavaFX. Praca skupia się na porównywaniu dostępnych funkcjonalności oraz różnic w implementacji. Badania polegały na zbudowaniu podobnych aplikacji w obydwu technologiach i porównanie ich pod względem zajętości pamięci podręcznej oraz na mierzeniu czasu odpowiedzi z bazy danych. Wyniki zaprezentowały, że aplikacja w Electron zapewniła krótszy czas wykonania operacji i mniejsze zużycie pamięci niż aplikacja JavaFX. Jednak autor zaznaczył, że implementacja koncepcji OOP w Electron przy użyciu JavaScript wiąże się z pewnymi obawami dotyczącymi hermetyzacji i dziedziczenia.

Po przeanalizowaniu dostępnych publikacji naukowych można stwierdzić, że brak jest badań porównujących obydwie omawiane technologie, co stało się podstawą do ich przeprowadzenia.

3. Obiekty badań

3.1. Electron

Wypuszczony pierwotnie pod nazwą Atom Shell w roku 2013, Electron to framework open-source rozwijany oraz wspierany przez firmę GitHub. Electron jest biblioteką używaną do tworzenia desktopowych aplikacji za pomocą technologii internetowych JavaScript, HTML oraz CSS [4]. Aplikacje tworzone w Electron mogą być pakowane i uruchamiane na systemach Mac, Windows i Linux. Wszystko za pomocą połączenia Chromium, Node.js oraz natywnych API dla każdego systemu, które pozwalają na obsługę plików, ikon oraz powiadomień.

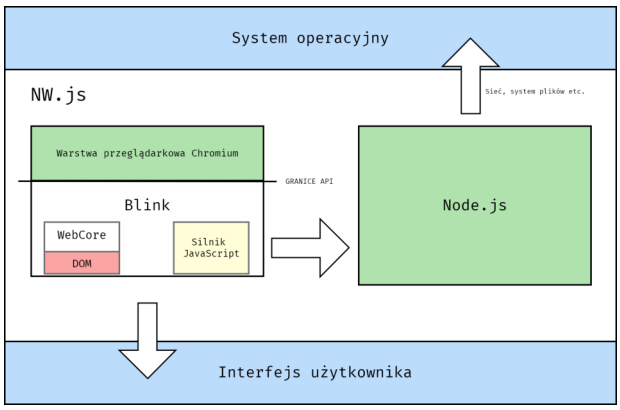
Aplikacje tworzone za pomocą Electron składają się z dwóch typów procesów: Main oraz Renderer. Pierwszy z nich jest procesem głównym, który określany jest w skrypcie package.json i odpowiedzialny jest za wyświetlanie graficznego interfejsu użytkownika poprzez tworzenie stron internetowych [8]. Każda ze stron uruchamia swój własny, drugi typ procesów - proces Renderer.

Proces główny tworzy strony za pomocą instancji okna BrowserWindow. Każde okno posiada swój własny, niezależny od innych, proces Renderer, który kończony jest po zamknięciu okna. Komunikacja między procesem Main, a procesami Renderer jest możliwa dzięki wbudowanym interfejsom API HTML5 np.: storage API, localStorage, sessionStorage oraz IndexedDB. Electron oferuje swoje własne rozwiązanie - system IPC. Przechowuje on obiekty w procesie głównym jako globalne zmienne oraz pozwala na dostęp do nich poszczególnym instancjom BrowserWindow.

3.2. NW.js

Projekt NW.js był wprowadzony w 2011 roku. początkowo pod nazwą node-webkit, przez Rogera Wanga ówcześnie pracującego w Intel's Open Source Technology Center (Shanghai). NW.js jest środowiskiem uruchomieniowym aplikacji internetowych stworzonym przy pomocy Node.js oraz Chromium [9]. Pozwala na wywoływanie modułów Node.js bezpośrednio z modelu obiektowego dokumentu (DOM). Oprócz Node.js u podstaw NW.js leży Blink - silnik przeglądarki internetowej rozwijany przez Google od 2013 roku, stworzony na podstawie silnika WebKit [10]. Blink rozwijany jest jako część projektu Chromium.

Diagram (Rysunek 2) przedstawia połączenie Node.js oraz Blink w celu uzyskania dostępu do graficznego interfejsu użytkownika oraz systemu operacyjnego.



Rysunek 2: Diagram połączenia Node.js oraz Blink [10]

4. Metoda badań

W celu przeprowadzenia badań zaprojektowano oraz stworzono aplikację mediów społecznościowych pozwalającą użytkownikom m.in. na udostępnianie zdjęć innym użytkownikom systemu, ich komentowanie oraz obserwowanie użytkowników. Implementacja została wykonana w NW.js oraz Electron. Aplikacja została podzielona na następujące moduły:

- **Rejestracja** - moduł pozwalający na stworzenie nowego użytkownika używając loginu, adresu e-mail oraz hasła,
- **Logowanie** - moduł sprawdzający poprawność danych oraz autoryzowanie użytkownika,
- **Profil** - moduł pozwalający użytkownikowi na wyświetlanie zdjęć, zmianę opisu profilu, zmianę zdjęcia profilowego, wyszukiwanie innych użytkowników, udostępnianie zdjęć oraz wyświetlanie listy użytkowników obserwowanych i obserwujących,
- **Interakcje** - moduł obsługujący obserwowanie użytkowników, usuwanie użytkowników z listy obserwowanych oraz komentowanie zdjęć,
- **Tablica** - moduł odpowiedzialny za wyświetlanie wszystkich użytkowników obserwowanych przez użytkownika zalogowanego.

Do celów badawczych wykorzystano moduł Tablica oraz moduł Profil. Mierzono czas w jakim aplikacja wyrenderuje odpowiednie elementy. Dla modułu Tablica jest to odpowiednio 10, 50 oraz 150 zdjęć, a dla modułu Profil jest to czas odświeżenia strony profilu użytkownika oraz czas wyświetlenia odpowiednio 10, 50 oraz 250 wierszy zawierających nazwy obserwatorów. Każde ze zdjęć wyświetlanych na stronie Tablica jest o rozdzielczości 1920x1080 i rozmiarze 350kb.

Testy zostały wykonane na komputerze o następujących specyfikacjach:

- procesor - Intel Core i5 - 4690k;
- pamięć RAM - 8 GB;
- dysk - 256GB SSD;
- karta graficzna – NVIDIA GeForce GTX 970;
- system operacyjny – Windows 10 / Ubuntu 12.04;

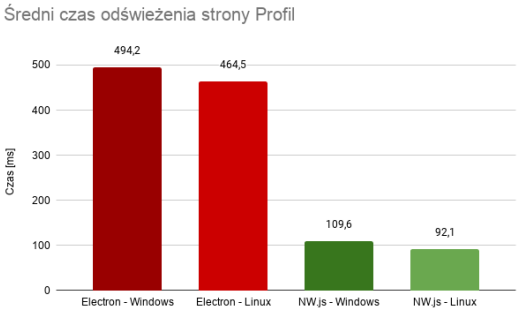
5. Wyniki badań

Testy wydajnościowe zostały przeprowadzone według następujących scenariuszy:

- S1 - odświeżanie strony, czas mierzony od kliknięcia w przycisk "Profile" do wyrenderowania wszystkich elementów profilu zalogowanego użytkownika;
- S2 - wyświetlanie obserwatorów, czas mierzony od kliknięcia w przycisk "Followers" do wyświetlenia 10, 50, 250 wierszy z nazwami obserwatorów;
- S3 - wyświetlanie zdjęć, czas mierzony od kliknięcia przycisk "Home" do wyrenderowania 10, 50, 150 zdjeć;

5.1. Scenariusz testowy S1

Rysunek 3 przedstawia wykres obrazujący średnie czasy odświeżania strony Profil omawianych technologii na poszczególnych systemach operacyjnych. Electron uzyskał w tym przypadku zaskakująco słaby wynik. W NW.js strona zawierająca podstawowe informacje o użytkowniku ładuje się prawie 5 razy szybciej niezależnie od systemu operacyjnego. Nawet bez badań, do takiego samego wniosku może dojść użytkownik używający aplikacji na porządku dziennym – w odświeżaniu widoku, czy też w przechodzeniu między widokami, zauważalne jest opóźnienie. W porównaniu z NW.js ładowanie strony jest niemal natychmiastowe.

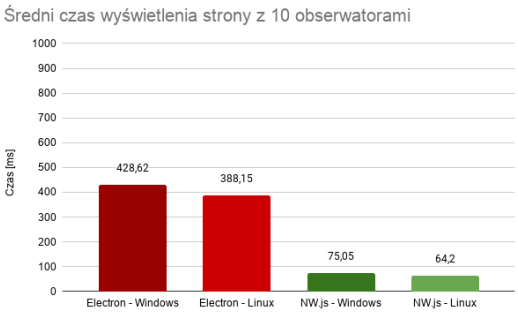


Rysunek 3: Diagram przedstawiający średni czas odświeżania strony Profil

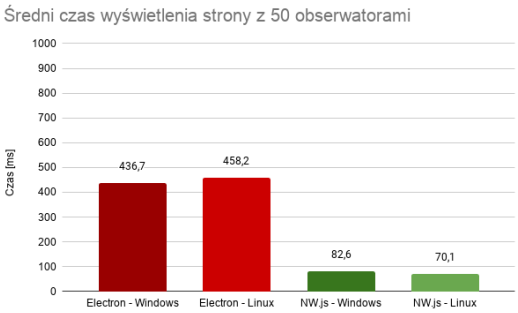
5.2. Scenariusz testowy S2

Rysunki 4, 5 i 6 przedstawiają wyniki badań przeprowadzone według scenariusza testowego S2.

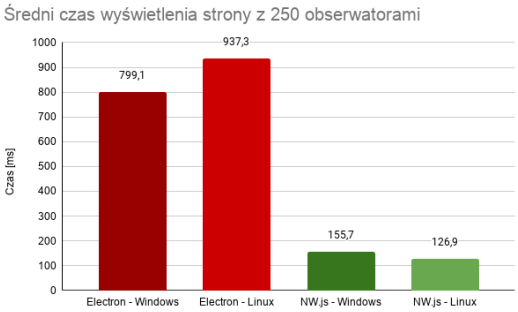
Scenariusz S2 miał na celu sprawdzenie w jakim czasie badane frameworki wyświetlą odpowiednio 10, 50 oraz 250 wierszy zawierających nazwę użytkownika oraz jego zdjęcie profilowe. W tym przypadku testy zdecydowanie umocniły pozycję NW.js. W przypadku testów dla 10 i 50 wierzsy, NW.js jest około 4 razy szybszy od konkurenta. Warto również zauważyć, że przy średniej prędkości ładowania 70ms, ładowanie jest dalej płynne i natychmiastowe. Dopiero przy renderowaniu 250 wierszy NW.js przy średniej prędkości ładowania 130ms, zauważalne jest opóźnienie w ładowaniu. Ciekawą rzeczą, którą pokazało ładowanie większej liczby wierszy jest fakt, że system Windows wyświetla treści 15% szybciej od



Rysunek 4: Średni czas wyświetlenia strony z 10 obserwatorami



Rysunek 5: Średni czas wyświetlenia strony z 50 obserwatorami



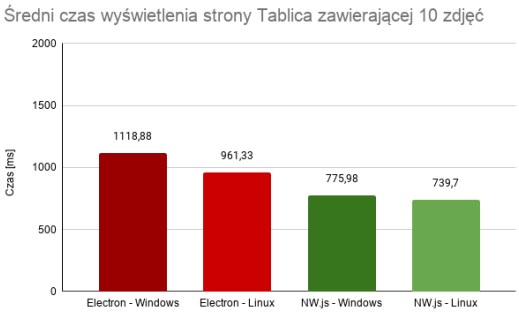
Rysunek 6: Średni czas wyświetlenia strony z 250 obserwatorami

systemu Linux. Jest to jedynie ciekawostka, która wymagałaby przeprowadzenia kolejnych badań, szczególnie zwracając uwagę na stosunkowo mały wpływ systemu operacyjnego na szybkość renderowania w poprzednich wynikach testów.

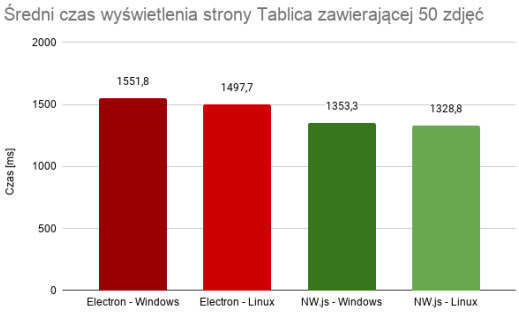
5.3. Scenariusz testowy S3

Rysunki 7, 8 i 9 przedstawiają wyniki nadań przeprowadzonych według scenariusza testowego S3.

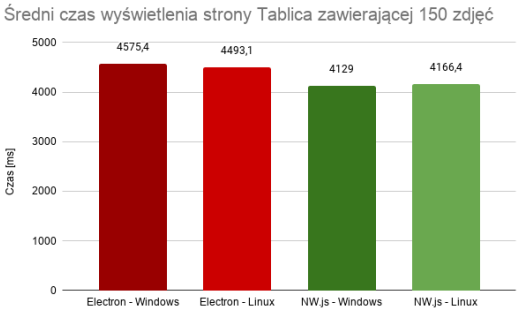
Scenariusz S3 sprawdzał szybkość wyświetlania odpowiednio 10, 50 i 150 zdjęć w wysokiej rozdzielczości. W tym przypadku wyniki pomiędzy obydwoma frameworkami były bardziej zbliżone. W ładowaniu 10 zdjęć NW.js był o ok. 31% szybszy, w 25 zdjeć o ok. 22%, a w przypadku ładowania 150 zdjęć – ok. 11%. Ten test uwidocznia fakt, że wraz ze zwiększoną ilością danych potrzebnych do pobrania, wydajność NW.js w porównaniu do Electrona spada. Test pozolił również na zaobser-



Rysunek 7: Średni czas wyświetlenia strony Tablica zawierającej 10 zdjęć



Rysunek 8: Średni czas wyświetlenia strony Tablica zawierającej 50 zdjęć



Rysunek 9: Średni czas wyświetlenia strony Tablica zawierającej 150 zdjęć

wowanie wpływu systemu operacyjnego na wydajność. W przypadku ładowania 10 zdjęć Electron działający na systemie Linux jest szybszy od Electron działającego na Windows o ok. 20%. W przypadku ładowania 50 i 150 zdjęć wyniki pomiędzy systemami oraz omawianymi technologiami nie są duże. Przy większej ilości danych na wynik mogą wpływać czynniki zewnętrzne takie jak ograniczenia sprzętowe czy też sama wydajność Node.js.

6. Wnioski

W tym artykule przedstawiono porównanie wydajności wieloplatformowej aplikacji Electron i NW.js. Wyniki przeprowadzonych badań pokazują różnicę wydajności pomiędzy porównywanymi technologiami. NW.js był technologią szybszą w każdym analizowanym przypad-

ku. Electron, nawet w operacjach pobierających małą ilość danych wykazywał znaczne opóźnienia wpływające na płynność obsługi aplikacji. Wpływ systemu operacyjnego na wydajność aplikacji jest trudny do oceny. O ile w większości przypadków nie wpływa on znacznie na szybkość działania aplikacji, tylko niektóre testy wykazały wyraźną przewagę jednego z systemów, co nie pozwala na wydanie jednoznacznej opinii w tej kwestii. Na podstawie uzyskanych wyników, można stwierdić, że teza "Framework NW.js, jest pod względem wydajnościowym lepszym rozwiązaniem od frameworka Electron" jest prawdziwa.

Literatura

[1] P. Lindhol, Web technologies for cross-platform desktop applications–a feasible option?, 2017.

[2] D. Alymkulov, Desktop Application Development Using Electron Framework: Native vs. Cross-Platform, 2019.

[3] Developer Surver Results 2019, https://insights.stackoverflow.com/survey/2019, [10.10.2020].

[4] Porównanie popularności Electron – NW.js, https://trends.google.com/trends/explore?date=all&q=%2Fg%2F11bw˙559wr,nw.js,node-webkit,,[10.10.2020].

[5] D. Sheiko, Cross-platform Desktop Application Development: Electron, Node, NW.js, and React: Build desktop applications with web technologies, Packt Publishing, 2017.

[6] Z. Hussein, An In-Depth Comparison of Software Framework for Developing Desktop Applications Using Web Technologies, 2019.

[7] A. Alkhars, Cross-Platform Desktop Development (JavaFX vs. Electron), 2017.

[8] S. Kinney, Electron in Action, Manning Publications, 2018.

[9] NW.js, https://nwjs.readthedocs.io/en/latest/, [10.10.2020],

[10] A. Benoit, NW.js Essentials, Packt Publishing, 2015.