JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 32 (2024) 194–198
Received: 23 May 2024
Accepted: 19 June 2024

Comparative analysis of tools for managing web application development

Analiza porównawcza narzędzi wspomagających zarządzanie budowaniem aplikacji internetowych

Paulina Wójcik*, Małgorzata Plechawska-Wójcik

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

Abstract

This article presents a comparative analysis of three bundlers: tools for managing web application development. The basic capabilities of the Webpack, Parcel and Vite tools were reviewed. Bundlers were examined in terms of performance, as well as ease of configuration. The article describes the configuration of each of the tools studied. Research scenarios were formulated, on the basis of which a series of tests were carried out consisting in building applications by each tool and measuring the parameters of this process. The obtained results of the research and the author's personal experience made it possible to choose the best tool for managing web application development.
Keywords: bundler; JavaScript; Webpack; Parcel; Vite

Streszczenie

W niniejszym artykule przedstawiono analizę porównawczą trzech narzędzi typu bundler wspomagających zarządzanie budowaniem aplikacji internetowych. Zapoznano się z podstawowymi możliwościami narzędzi Webpack, Parcel oraz Vite. Narzędzia typu bundlers zostały zbadane pod kątem wydajności, a także łatwości konfiguracji. W artykule opisano konfigurację każdego z badanych narzędzi. Sformułowano scenariusze badawcze, na podstawie których przeprowadzono serię badań polegającą na budowaniu aplikacji przez poszczególne narzędzia i pomiarze parametrów tego procesu. Otrzymane wyniki badań oraz personalne doświadczenia autorki pozwoliły na wybranie najlepszego narzędzia wspomagającego zarządzanie budowaniem aplikacji internetowych.

Słowa kluczowe: bundler; JavaScript; Webpack; Parcel; Vite

*Corresponding author
Email address: paulina.wojcik@pollub.edu.pl (P. Wójcik)

1. Wstęp

Wraz z rosnącym zapotrzebowaniem na bardziej zaawansowane i skomplikowane aplikacje internetowe, pojawiła się potrzeba efektywnego zarządzania złożonością kodu. Dzisiejsze aplikacje JavaScript są wysoce interaktywne. Wraz ze wzrostem poziomu zaawansowania aplikacji, liczba plików wzrasta, co prowadzi do długiego czasu jej ładowania i problemów z wydajnością. Na aplikację składa się wiele plików o różnych formatach, między którymi występują wszelkiego rodzaju zależności. Bez narzędzi do zarządzania tymi zależnościami uruchomienie takiego kodu w przeglądarce byłoby bardzo trudne. Kiedyś ręcznie ustawiano pliki we właściwej kolejności. Teraz z pomocą przychodzą narzędzia do zarządzania budowaniem aplikacji internetowych, czyli narzędzia typu bundlers. Ich działanie polega na spakowaniu plików różnych formatów do jednego pliku JavaScript poprzez przekształcenie i minifikację niepotrzebnego kodu. Korzystanie z nich staje się konieczne gdy aplikacja składa się z wielu plików oraz wykorzystuje biblioteki zewnętrzne.

2. Cel i zakres badań

Celem badań jest przeprowadzenie analizy porównawczej narzędzi wspomagających zarządzanie budowaniem aplikacji internetowych takich jak Webpack, Parcel oraz Vite.

Głównymi kryteriami porównawczymi były:

- szybkość zbudowania aplikacji,
- łatwość konfiguracji,
- rozmiar plików wynikowych,
- zajętość procesora podczas budowania aplikacji,
- popularność w środowisku deweloperskim.

W ramach artykułu postawiono tezę badawczą: „Webpack jest najlepszym narzędziem wspomagającym zarządzanie budowaniem aplikacji internetowych”.

3. Analiza literatury

Zaznajomienie się z literaturą traktującą o narzędziach wspomagających zarządzanie budowaniem aplikacji internetowych umożliwia podsumowanie dotychczasowej wiedzy na ten temat. Z uwagi na to, że narzędzia, które zostały poddane badaniom, są dosyć nowymi technologiami, nie udało się znaleźć wielu artykułów bezpośrednio podejmujących te tematy. Zaznajomiono się z natomiast z publikacjami, które podejmują tematy modułowości w języku JavaScript, a także opisują metody, które wykorzystywane są w narzędziach wspomagających budowanie aplikacji internetowych.

W artykule [1] zbadano zautomatyzowaną metodę migracji bazy kodu ECMAScript 5 (ES5) do modułów ECMAScript 6 (ES6). Metoda ma zastosowanie do kodu, który jest niemodułowy lub wykorzystuje formaty modułów AMD/CommonJS i działa na Module Dependence

Graph (MDG). Omawiane jest rozwinięcie modułowości w języku JavaScript od ES5 do ES6. Autorzy skupiają się na różnicach dotyczących hermetyzacji modułów, zasięgu zmiennych globalnych oraz kolejności oceniania zależności między modułami. W tym artykule autorzy opisują zalety projektowania modułów w taki sposób, który zakłada istnienie wielu niezależnych i reużywalnych funkcji w modułach, oraz precyzyjnych zależności między nimi poprzez używanie importów oraz eksportów w języku ES6. Takie podejście przynosi korzyści w procesie refaktoryzacji, optymalizacji kodu oraz śledzeniu błędów. Dodatkowo, taki sposób projektowania modułów wspiera narzędzia typu bundlers, takie jak np. Webpack, umożliwiając eliminację nieużywanego kodu poprzez technikę tree-shaking. Migracja ta stanowi ważny przełom w ewolucji starszych projektów w JavaScript, ponieważ modułowość w ECMAScript 6 jest podstawą do wykorzystywania wielu nowych funkcji wprowadzonych w ES6 lub później.

W kolejnym artykule [2] skupiono się na badaniu łańcucha dostaw oprogramowania w aplikacjach internetowych, ze szczególnym uwzględnieniem zjawiska bundlingu, czyli łączenia kodu zewnętrznego w pojedynczy plik. Autorzy przedstawiają nowatorską metodologię automatycznego wykrywania pakietów i ich częściowej inżynierii wstecznej. W artykule opisano sposób działania narzędzi typu bundlers jako łączenie kodu dewelopera z odpowiednią częścią zaimportowanych zależności, tworząc samodzielny plik JavaScript, który można bezpośrednio załadować po stronie klienta. Przedstawiają także korzyści z używania bundlerów, takie jak np. skrócenie czasu początkowego ładowania aplikacji, zmniejszenie rozmiaru kodu klienckiego oraz zminimalizowanie liczby żądań. Wyniki pokazują wysoki odsetek pakietów w ekosystemie internetowym oraz wskazują, że kod zawarty w pakietach jest podatny na ataki w łańcuchu dostaw.

Kolejny artykuł [3] podejmuje temat wykorzystania modułów z menadżera pakietów npm w aplikacjach JavaScript. Autorzy opisują problem, jakim jest nadmierny wzrost rozmiaru kodu poprzez instalowanie modułów ze wszystkimi funkcjonalnościami. Pobierane są nie tylko funkcje, które faktycznie są wykorzystywane w projekcie, ale cała zawartość paczki, wraz z nieużywanymi funkcjami. Autorzy wychodzą z rozwiązaniem jakim jest technika identyfikująca i eliminująca nieużywany kod poprzez konstruowanie wykresów wywołań z testów aplikacji i zastępowanie nieużywanego kodu tzw. stubami, Stuby są to przykładowe implementacje funkcji imitujące działanie tych właściwych. Narzędzie, które nazwano Stubbifier, współpracuje z bundlerami w celu łączenia aplikacji z jej zależnościami. Wykorzystano również fakt, że bundlery wykonują tzw. tree-shaking, usuwając nieużywany kod. Podczas badania, bundling został wykonany przed zastosowaniem Stubbifier, ponieważ bundlery wykonują własne transformacje kodu. Stubbifier działając równolegle z tymi narzędziami zredukował rozmiar aplikacji o 37%.

Czwarta publikacja [4] opisuje system Unbundle-Rewrite-Rebundle (URR). Służy on do wykrywania inwazyjnych fragmentów kodu, które mogą naruszać prywatność. Narzędzie po wykryciu takiego fragmentu w spakowanym kodzie JavaScript przepisuje go w czasie rzeczywistym w taki sposób, aby pozbyć się wrażliwych części kodu bez naruszania logiki aplikacji. Wiele podejść do filtrowania treści opiera się na blokowaniu adresów URL. Dzisiejsze aplikacje internetowe w większości korzystają z narzędzi do budowania aplikacji, które umieszczają kod aplikacji tylko w jednym pliku (tzw. bundle). Przez to tradycyjne sposoby blokowania treści stały się nieskuteczne. Autorzy opisują powody, dla których aktualnie tworzy się aplikacje internetowe przy użyciu narzędzi typu bundlers oraz omawiają podstawowe koncepcje tych narzędzi. Przedstawiona jest istota działania systemu URR, który wykrywa pakiety i za pomocą inżynierii wstecznej przekształca je do modułów. Następnie identyfikuje moduły, które mogą naruszać prywatność i zastępuje je łagodnymi alternatywami.

Autorzy pracy [5] omawiają sposób, w jaki React.js wspomaga tworzenie interfejsów użytkownika. Przedstawiono kluczowe aspekty tego frameworka, w tym jego zalety w stosunku do konkurencyjnych frameworków, sposób działania i architekturę. W artykule opisano narzędzia, które stanowią ważne komponenty ekosystemu React, które pomagają w rozwijaniu i zarządzaniu aplikacjami front-endowymi, zapewniając zarządzanie stanem, strukturę plików oraz zależności w projekcie. Przedstawiono w nim m.in. Webpack, który jest odpowiedzialny za zbudowanie projektu i wszystkich jego zależności w końcowy pakiet, który następnie jest dostarczany klientowi. Autorzy podkreślają, że konfiguracja tego narzędzia jest niezwykle czasochłonna, jednak jest to narzędzie, które warto dodać do swojego projektu, a także, że warto posiadać wiedzę na jego temat.

4. Opis badanych narzędzi

W ramach badań porównano trzy popularne narzędzia: Webpack, Parcel oraz Vite. Wybór ten wynika z ich popularności oraz różnorodności funkcji jakie oferują, a także pozwala na analizę rozwiązań, które oferują unikalne podejścia do budowania aplikacji.

4.1. Webpack

Webpack [6] jest najczęściej używanym narzędziem do zarządzania plikami oraz zależnościami w projektach aplikacji internetowych. Analizuje on zależności pomiędzy plikami, a następnie łączy je w jeden plik wynikowy, który zostaje poddany różnym procesom optymalizacji i finalnie może zostać wdrożony na serwer. Udostępnia on również serwer deweloperski, który umożliwia wygodną implementację kodu aplikacji przy jednoczesnym podglądzie zmian. Webpack obsługuje wiele rodzajów plików, jednak wykorzystuje do tego moduły ładujące, czyli tzw. loadery [7]. Dodatkowo, narzędzie to oferuje bogaty ekosystem wtyczek, które rozszerzające możliwości tego bundlera poza standardowe przetwarzanie modułów i zależności. Wtyczki mogą między innymi minifikować i optymalizować pliki oraz korzystać ze zmiennych środowiskowych.

4.2. Parcel

Parcel [8] jest narzędziem służącym do budowania aplikacji internetowych, skupiającym się na prostocie użytkowania. Głównym jego zadaniem jest stworzenie drzewa zależności aplikacji, a następnie połączenie jego elementów w jeden plik wynikowy zwany bundlem. Jest on znany z braku konfiguracji. Automatycznie wykrywa i przetwarza różne typy plików na podstawie ich rozszerzeń i zawartości. Ma wbudowaną obsługę różnych typów zasobów, w tym obrazów, HTML czy CSS.

4.3. Vite

Najnowszym z badanych narzędzi jest Vite [9]. Służy ono do budowania i rozwijania aplikacji internetowych. Udostępnia ono serwer deweloperski umożliwiający tworzenie aplikacji z podglądem w czasie rzeczywistym. Vite umożliwia również budowanie aplikacji w trybie produkcyjnym. Na tle innych narzędzi wyróżnia się zdecydowanie szybkością, zarówno podczas startu serwera deweloperskiego, jak i procesu budowania. Vite nie wymaga pliku konfiguracyjnego, ponieważ jest już wstępnie skonfigurowany do generowania wysoce zoptymalizowanych zasobów statycznych. Można jednak rozszerzyć jego możliwości poprzez dodanie odpowiednich wtyczek.

5. Plan badań

W ramach przygotowania środowiska badawczego stworzono prostą aplikację internetową, która umożliwi porównanie narzędzi pod kątem szybkości, rozmiaru pliku wynikowego oraz zajętości procesora podczas budowania aplikacji.

Aplikacja została napisana w języku JavaScript. Pobiera ona dane z publicznego API oraz wyświetla je w różnej formie na ekranie przy użyciu bibliotek zewnętrznych. Aplikacja została sklonowana, aby każde z narzędzi mogło zostać przetestowane na osobnej aplikacji.

5.1. Scenariusze badawcze

Dla każdego z badanych narzędzi przeprowadzono następujące badania:

- pomiar czasu budowania aplikacji,
- pomiar rozmiaru pliku wynikowego,
- pomiar zajętości procesora podczas budowania.

Pomiar czasu budowania oraz rozmiaru pliku wynikowego nie wymaga użycia dodatkowych narzędzi, ponieważ informacje te są podane na w konsoli po zakończeniu procesu budowania aplikacji.

Badanie zajętości procesora będzie wymagało dodatkowych czynności. Przed rozpoczęciem budowania aplikacji należy uruchomić „Menadżer zadań” oraz rozpocząć nagrywanie ekranu, a następnie zbudować aplikację. Pozwoli to na zapis aktywności procesora w czasie budowania aplikacji.

Dla wymienionych badań zostały wykonane poniższe scenariusze:

Scenariusz 1: Pierwsze zbudowanie projektu po instalacji wszystkich zależności
1. Zainstalowanie bundlera.
2. Konfiguracja bundlera.
3. Zainstalowanie zależności projektu.
4. Zbudowanie aplikacji.
5. Odnotowanie wyników.

Scenariusz 2: Drugie zbudowanie projektu po usunięciu katalogu z plikami wynikowymi
1. Usunięcie katalogu *dist* z plikami wynikowymi pozostałymi po poprzednim zbudowaniu aplikacji.
2. Zbudowanie aplikacji.
3. Odnotowanie wyników.

Scenariusz 3: Trzecie zbudowanie projektu bez wprowadzania żadnych zmian
1. Zbudowanie aplikacji.
2. Odnotowanie wyników.

5.2. Konfiguracja narzędzi

Dla wszystkich narzędzi konfiguracja została ograniczona do podstawowych ustawień bundlerów z wykorzystaniem ewentualnych różnic pomiędzy nimi, poprzez dodanie odpowiednich wtyczek oraz loaderów. Każde z narzędzi zostało pobrane z repozytorium pakietów npm.

Parcel jest znany z tego, że nie wymaga konfiguracji. W celu zbudowania aplikacji określono plik wejściowy w pliku package.json, a następnie podano komendę budującą aplikację.

W przypadku bundlera Vite struktura projektu została nieco zmieniona, aby mógł on poprawnie odczytać plik wejściowy, którym jest „index.html”. Następnie, tak samo jak w poprzedniej konfiguracji, zdefiniowano polecenie służące do zbudowania aplikacji. Vite domyślnie zapewnia optymalizację kodu JS i CSS w trybie produkcyjnym, jednak do minifikacji kodu HTML należało skorzystać z odpowiedniej wtyczki. W tym celu utworzono plik „vite.config.js”, który zawierał w sobie obiekt konfiguracyjny z uprzednio zainstalowaną wtyczką ViteMinifyPlugin.

Ostatnim narzędziem do skonfigurowania był Webpack. Utworzono plik „webpack.config.js”, który zawiera obiekt konfiguracyjny, odczytywany przy procesie budowania. Określono pliki wejściowy oraz wyjściowy, a następnie ustawiono tryb pracy na produkcyjny. Webpack domyślnie nie obsługuje innych plików niż tych z rozszerzeniem .js. Aby wszystkie pliki zostały uwzględnione, należało zainstalować odpowiednie wtyczki oraz loadery. Do obsługi plików CSS zainstalowano css-loader. Dodatkowo, zainstalowano MiniCssExtractPlugin, czyli wtyczkę, która wyodrębnia CSS do oddzielnych plików. Oprócz tego, do obiektu dodano również wtyczkę HtmlWebpackPlugin, która minifikuje i wyodrębnia plik „index.html” do folderu z plikami wynikowymi. Do obsługi plików graficznych wykorzystano file-loader który dołącza pliki statyczne do folderu z plikami wynikowymi procesu budowania. Ostatnim elementem konfiguracji było zdefiniowanie wtyczek

służących do minifikacji plików. Wykorzystano TerserPlugin służący do optymalizacji oraz minifikacji plików JavaScript, natomiast CssMinimizerPlugin do plików CSS.

6. Wyniki

Badania polegały na zbudowaniu aplikacji skonfigurowanej dla danego bundlera, a następnie odczytaniu rozmiaru plików wynikowych, czasu budowania, a także na monitorowaniu oraz zapisaniu średniej wartości zajętości procesora podczas tego procesu.

W wyniku zbudowania aplikacji powstały trzy pliki wynikowe, które zawierały w sobie zminifikowany kod aplikacji, gotowy do użycia przez przeglądarkę oraz plik z grafiką. Niezależnie od scenariusza badawczego, rozmiary plików wynikowych były niezmienne. W Tabeli 1 zestawiono wszystkie rozmiary plików. W przypadku plików JavaScript, CSS oraz HTML najmniejsze pliki wygenerował Webpack, zaś największe – Parcel. Nieco inaczej przedstawiają się wyniki pomiarów rozmiaru pliku graficznego. W przypadku Vite i Webpack rozmiary plików są identyczne i wynoszą 49,57 kB, natomiast Parcel wygenerował najmniejszy plik, którego rozmiar wynosi 36,67 kB. Parcel ma wbudowaną obsługę optymalizacji obrazów, a więc stąd mogą wynikać te różnice. Biorąc pod uwagę wszystkie typy plików, różnice pomiędzy rozmiarami plików są niewielkie, aczkolwiek należy wziąć pod uwagę, że nie jest to duży projekt. W przypadku bardziej zaawansowanych aplikacji te różnice mogą być dużo większe, a więc rozmiary tych plików mogą mieć duże znaczenie przy wyborze odpowiedniego narzędzia do projektu.

Tabela 1: Zestawienie rozmiarów plików wynikowych

	JS	HTML	CSS	PNG
Webpack	299,00 kB	862 B	1,44 kB	49,57 kB
Parcel	315,55 kB	970 B	1,56 kB	36,67 kB
Vite	310,72 kB	958 B	1,48 kB	49,57 kB

Pierwszy scenariusz badawczy zakładał pierwsze zbudowanie aplikacji, bezpośrednio po zainstalowaniu zależności projektu. Zgodnie z wynikami przedstawionymi w Tabeli 2, pierwsze zbudowanie aplikacji przez Webpack zajmuje najwięcej czasu ze wszystkich narzędzi. Zgodnie z oczekiwaniami, najszybszy okazał się Vite, którego czas budowania jest ponad 3-krotnie mniejszy niż w przypadku Webpacka. Parcel zbudował aplikację w nieco ponad 5 sekund, a więc jest to większy czas od Vite, jednak w porównaniu z Webpackiem, jest to bardzo dobry wynik.

Tabela 2: Zestawienie czasów procesu budowania aplikacji oraz wykorzystania CPU

	Czas [ms]	Wykorzystanie CPU [%]
Webpack	13 124	52
Parcel	5 460	67
Vite	4 330	71

Sytuacja jest odwrotna w przypadku wyników dotyczących drugiego parametru. Średnie obciążenie procesora podczas procesu budowania wynosiło najmniej w przypadku Webpacka, więc chociaż czas budowania jest dłuższy, to proces ten nie obciąża tak bardzo procesora. Największe średnie wykorzystanie procesora zanotowano w przypadku Vite.

W drugim scenariuszu zmierzono czas budowania aplikacji po usunięciu dotychczasowych plików wynikowych oraz ewentualnych plików z pamięci podręcznej. W Tabeli 3 przedstawiono średnie wyniki wraz z odchyleniem standardowym dla dwudziestu prób budowania aplikacji dla każdego bundlera. Można zauważyć, że tak samo jak w poprzednim scenariuszu, najwyższe czasy osiąga Webpack, a najkrótsze Vite. Najbardziej obciążające były procesy budowania aplikacji z konfiguracją Parcel, a następnie Vite oraz Webpack. projektu.

Tabela 3: Zestawienie czasów procesu budowania aplikacji oraz wykorzystania CPU

	Czas [ms]	Wykorzystanie CPU [%]
Webpack	8 200 ± 596	48 ± 7
Parcel	4 568 ± 282	58 ± 5
Vite	3 381 ± 370	52 ± 7

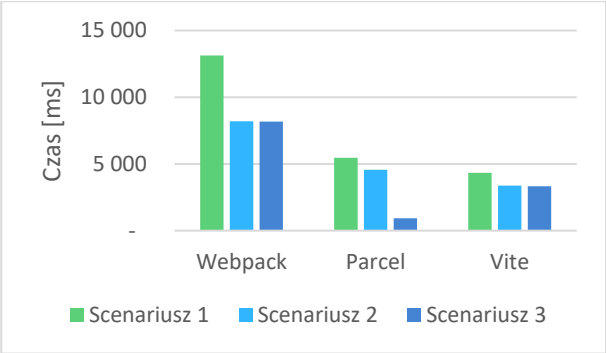
W ostatnim scenariuszu również przeprowadzono 20 prób, podczas których budowano aplikację jednocześnie badając zajętość procesora. W tym scenariuszu występuje bardzo duża różnica pomiędzy najszybszym a najwolniejszym bundlerem. Webpack, tak jak w poprzednich scenariuszach, utrzymuje najdłuższe czasy. Tym razem jednak to Parcel najszybciej budował aplikację. Średni czas dla tego bundlera w tym scenariuszu był aż 9-krotnie mniejszy od najwolniejszego narzędzia. W Tabeli 4 przedstawiono uśrednione wyniki wraz z odchyleniem standardowym dla tego scenariusza. Dla drugiego i trzeciego scenariusza obliczono odchylenie standardowe dla uzyskanych wyników. W obu scenariuszach Parcel wykazuje najmniejszą zmienność wyników pomiarów. Największa zmienność występuje dla Webpacka. projektu.

Tabela 4: Zestawienie czasów procesu budowania aplikacji oraz wykorzystania CPU dla Scenariusza 3

	Czas [ms]	Wykorzystanie CPU [%]
Webpack	8 176 ± 416	50 ± 4
Parcel	931 ± 83	51 ± 5
Vite	3 330 ± 181	51 ± 4

Na Rysunku 1 przedstawiono zestawienie średnich czasów ze wszystkich trzech scenariuszy. Można zauważyć, że Webpack oraz Vite największy spadek zanotowały pomiędzy Scenariuszem 1 a Scenariuszem 2, kiedy to pomiędzy procesami usuwane były foldery z plikami wynikowymi. W przypadku Scenariusza 2 oraz Scenariusza 3 różnice były bardzo małe i dla obu narzędzi wynosiły setne części procenta. Sytuacja wygląda nieco inaczej w przypadku Parcela. Pomiędzy pierwszym i drugim scenariuszem widać spadek średniego czasu budowania aplikacji. Wysoce istotny w tym przypadku jest folder „.parcel-cache”, dzięki któremu pomiędzy Scenariuszem 2 a Scenariuszem 3 zanotowano spadek czasu budowania

aplikacji o niemalże 80%, co stanowi ogromną redukcję czasu budowania aplikacji.



Rysunek 1: Porównanie czasów ze Scenariusza 1, Scenariusza 2 i Scenariusza 3.

W Tabeli 5 umieszczono wszystkie kryteria porównawcze. Dodano również kryterium dotyczące dostępności wtyczek rozszerzających możliwości bundlerów. Dla każdego kryterium narzędzia otrzymały 1, 2 lub 3 punkty, gdzie 3 stanowi najlepszy wynik, a 1 – najgorszy.

Tabela 5: Ocena narzędzi pod kątem poszczególnych kryteriów

Kryterium	Webpack	Parcel	Vite
Szybkość	1	2	3
Rozmiar plików	3	1	2
Wykorzystanie CPU	3	2	1
Łatwość konfiguracji	1	3	2
Dostępność wtyczek	3	1	2
Większa społeczność	3	1	2
Suma	14	10	12

7. Wnioski

Wyniki zebrane po serii testów przeprowadzonych na aplikacjach skonfigurowanych pod dane narzędzia pozwoliły na ich porównanie pod kątem szybkości, rozmiaru pliku wynikowego oraz zajętości procesora podczas procesu budowania aplikacji. Po dokonaniu analizy otrzymanych rezultatów testów można potwierdzić postawioną tezę, że Webpack jest najlepszym narzędziem służącym do budowania aplikacji internetowych.

Potwierdzeniem tej tezy jest fakt, że Webpack generuje najlżejsze pliki, a przy tym jest on najmniej obciążający dla procesora. Poza tym, jest on najbardziej uniwersalnym narzędziem, które możemy wykorzystać dla każdego rodzaju projektu. Dzięki szerokiej gamie wtyczek oraz loaderów, które rozszerzają możliwości tego narzędzia, umożliwia dostosowanie procesu budowania aplikacji do indywidualnych potrzeb deweloperów.

Webpack okazał się najwolniejszy ze wszystkich badanych bundlerów, jednak biorąc pod uwagę wszystkie pozostałe kryteria, można stwierdzić, że jest to najlepszy wybór. Konfiguracja tego narzędzia, która jest wysoce skomplikowana, jest również jego zaletą. Dzięki temu, oferuje dużo więcej możliwości niż pozostałe narzędzia.

Głównym wnioskiem wynikającym z tej analizy jest to, że wybór odpowiedniego bundlera zależy od konkretnych potrzeb projektu, jego rozmiaru, złożoności oraz priorytetów w zakresie wydajności, elastyczności i łatwości użycia. Vite, który z dnia na dzień staje się co raz bardziej popularny, jest bardzo cennym narzędziem ze względu na swoją szybkość. Parcel z kolei, z uwagi na niski próg wejścia, jest idealny dla osób rozpoczynających swoją przygodę z programowaniem aplikacji w JavaScript. Każde z omówionych narzędzi ma wiele do zaoferowania i może być doskonałym wyborem w odpowiednim kontekście.

Literatura

[1] K. Paltoglou, V.E. Zafeiris, N.A. Diamantidis, E.A. Giakoumakis, Automated refactoring of legacy JavaScript code to ES6 modules, Journal of Systems and Software 181 (2021) 111049-11059, https://doi.org/10.1016/j.jss.2021.111049.

[2] J. Rack, C.A. Staicu, Jack-in-the-box: An Empirical Study of JavaScript Bundling on the Web and its Security Implications, In CCS '23: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (2023) 3198–3212, https://doi.org/10.1145/3576915.3623140.

[3] A. Turcotte, E. Arteca, A. Mishra, S. Alimadadi, F. Tip, Stubbifier: debloating dynamic server-side JavaScript applications, Empirical Software Engineering 27 (2022) 161-172, https://doi.org/10.48550/arXiv.2110.14162.

[4] M.M. Ali, P. Snyder, C. Kanich, H. Haddadi, Unbundle-Rewrite-Rebundle: Runtime Detection and Rewriting of Privacy-Harming Code in JavaScript Bundles, arXiv:2405.00596 (2024), https://doi.org/10.48550/arXiv.2405.00596.

[5] S. Chen, U. R. Thaduri, V. K. R. Ballamudi, Front-End Development in React: An Overview. Engineering International 7(2) (2019) 117–126, https://doi.org/10.18034/ei.v7i2.662.

[6] Porównanie liczby pobrań Webpack, Parcel i Vite, https://npmtrends.com/parcel-vs-vite-vs-webpack/, [17.11.2023].

[7] Dokumentacja Webpack, https://webpack.js.org/, [18.05.2024].

[8] Dokumentacja Parcel, https://en.parceljs.org/, [18.05.2024].

[9] Dokumentacja Vite, https://vitejs.dev/, [18.05.2024].