JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 19 (2021) 75–80
Received: 1 February 2021
Accepted: 15 February 2021

Comparative analysis of JavaScript package managers - yarn and NPM

Analiza porównawcza menadżerów pakietów JavaScript – yarn oraz NPM

Michał Chodorowski*

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

Abstract
In this article, two leading solutions for managing packages in projects which are using JavaScript technology (yarn and npm) were subjected to a comparative analysis. As part of the implementation, two configuration files were created, one of which represents an empty application created on the basis of an application template based on the Angular framework in version 8. The second file reflects an extensive web application based on the same framework, but with the addiction of over 100 dependencies. The research was focused on the time efficiency of both solutions.

Keywords: NPM; yarn; package manager; performance testing

Streszczenie
W niniejszym artykule analizie porównawczej poddano dwa wiodące rozwiązania służące do zarządzania pakietami w projektach wykorzystujących technologię JavaScript (yarn oraz npm). W ramach realizacji powstały dwa pliki konfiguracyjne, z których jeden reprezentuje pustą aplikację stworzoną na podstawie szablonu aplikacji opartej na szkielecie programistycznym Angular w wersji 8. Drugi plik odzwierciedla rozbudowaną aplikację internetową opartą o ten sam szkielet programistyczny, lecz z dodatkiem ponad 100 zależności. Badania ukierunkowane zostały na wydajność czasową obu rozwiązań.

Słowa kluczowe: NPM; yarn; menadżer pakietów; testy wydajnościowe

*Corresponding author
Email address: mic.chodowrowski@gmail.com (M. Chodorowski)

1. Wstęp

Wraz z postępującym coraz szybciej zjawiskiem cyfryzacji, w naturalny sposób rośnie zapotrzebowanie na tworzenie szeroko rozumianego oprogramowania. To z kolei powoduje szybszy rozwój języków programowania, szkieletów, architektury oraz narzędzi do wytwarzania oprogramowania. Obecnie na rynku istnieje dostęp do bardzo wielu narzędzi, które mają za zadanie ułatwiać pracę programistom, często przez oszczędność czasu, który jest największym kosztem w całym procesie wytwarzania oprogramowania. Z tego powodu, przed przystąpieniem do pisania kodu, zespoły projektowe zastanawiają się nad doborem technologii oraz narzędzi, przy pomocy których będą tworzyć projekt. Jednym z fundamentalnych i niezbędnych narzędzi, bez których żadna współczesna aplikacja internetowa nie może się obyć, to menadżer pakietów. Wydawać by się mogło, że to mało istotna kwestia, ponieważ zarówno wybrany do analizy yarn oraz NPM, są rozwiązaniami używanymi przez miliony programistów na całym świecie. Jednak, przyglądając się bliżej tworzeniu oprogramowania można zaoszczędzić nawet kilka sekund na często wykonywanej operacji, to na przestrzeni dłuższego okresu czasu, oraz biorąc pod uwagę fakt, że nad daną aplikacją nie pracuje nigdy tylko jeden programista, zyskać można wiele godzin, które można by przeznaczyć na programowanie zamiast na czekanie. Wyniki analizy porównawczej zawarte w tym artykule mogą pomóc i oszczędzić wiele godzin koniecznych do wykonania analiz NPM i yarn.

2. Cel badań i hipoteza

Do badań wykorzystano dwa testowe pliki konfiguracyjne oparte o ten sam szkielet programistyczny, lecz z różną liczbą zależności, które zostały dobrane tak aby pierwszy plik konfiguracyjny odwzorowywał pustą aplikację zawierającą tylko niezbędne zależności, a drugi odwzorowywał rozbudowaną aplikację. Celem badań było przeanalizowanie wydajności czasowej tych dwóch rozwiązań.
Za hipotezę badawczą przyjęto:
Pomimo faktu, iż npm jest narzędziem chętniej wybieranym przez programistów, to yarn jest lepszym wyborem pod względem wydajności czasowej w aplikacjach zawierających dużą ilość pakietów.

3. Systemy zarządzania pakietami JavaScript

3.1. NPM

NPM (ang. „Node Package Menager") jest domyślnym menadżerem pakietów dla środowiska uruchomieniowego Node.js dla języka JavaScript [1]. NPM dzieli się na dwie główne części [2]:

- CLI (ang. Command-Line Interface) –interfejs wiersza poleceń - narzędzie do pobierania oraz do udostępniania pakietów;
- internetowe repozytorium - zawiera pakiety JavaScript [3], które zostały stworzone i opublikowane przez społeczność programistów, którzy udostępnili swoje rozwiązania dla szerszego grona odbiorców.

Instalacja

Instalacja NPM sprowadza się do zainstalowania Node js, korzystając z oficjalnej strony producenta. NPM zostaje automatycznie zainstalowany wraz z Node js.

Skrypty NPM

Plik konfiguracyjny (szerszy opis znajduje się w rozdziale 3.3), wspiera własność ***scripts*** (Listing 1), która może być zdefiniowana w celu uruchomienia CLI, w celu zainstalowania pakietów, wykonania konkretnej operacji, lub zestawu operacji. Często programiści definiują takie skrypty, aby nie trzeba było za każdym razem wykonywać długich sekwencji wywołań poszczególnych komend przez CLI, usprawnić sobie prace i zaoszczędzić czas poprzez wywołanie jednego skryptu, który zadziała automatycznie wraz z wywołaniem np. komendy **`npm install`**.

Listing 1: Skrypt NPM wykorzystujący własność scripts.

```
{ "scripts": {
   "build": "tsc",
   "format": "prettier --write **/*.ts",
   "format-check": "prettier --check
           **/*.ts",
    "lint": "eslint src/**/*.ts",
    "pack": "ncc build",
    "test": "jest",
    "all": "npm run build && npm run for
     mat && npm run lint && npm test"
  }
}
```

Semantyka wersjonowania

W celu zapewnienia niezawodności, funkcjonowania oraz bezpieczeństwa projektu JavaScript, za każdym razem, gdy dokonywane są znaczące zmiany w jakimś pakiecie, zalecane jest publikowanie nowej wersji pakietu ze zmienionym numerem wersji w pliku konfiguracyjnym pozostającym w zgodności z dyrektywami semantyk NPM. Dokładne przestrzeganie semantyki wersjonowania pomaga innym programistom, którzy polegają na danym pakiecie, zrozumieć jakie zmiany zostały dokonane, i czy wskutek tych zmian Ci programiści muszą dostosowywać swoją aplikację tak aby nadal działała prawidłowo, z nową wersją pakietu. Zwiększenie numeru wersji jest zalecane, zwłaszcza jeżeli zmiany zrywają jakąkolwiek zależność lub zmieniają układ zależności. Zmiany tego typu są w stanie uszkodzić cały projekt.

Plik blokady

Plik blokady [5] używany jest w celu wyznaczenia dokładnej wersji zależności, która musi zostać użyta w danym projekcie, aby zapewnić jego poprawne działanie. Jeżeli w pliku blokady (package-lock.json) nie została zdefiniowana dokładna wersja, którejkolwiek z zależności – w takiej sytuacji zostanie pobrana najnowsza stabilna wersja danej zależności. Posiadanie pliku blokady jest szczególnie ważne na serwerach CI (ang. „Continous Integration"), gdzie spoczywają produkcyjne wersje aplikacji. Najważniejsze komendy NPM przedstawia Tabela 1.

Tabela 1: Najczęściej używane komendy w NPM CLI

Komenda	Opis działania
npm install	Instaluje zależności
npm ci	Generuje produkcyjną wersje projektu
npm audit	Skanuje projekt w poszukiwaniu zależności
npm audit fix	Automatyczna naprawa podatności

3.2. Yarn

Yarn jest menadżerem pakietów JavaScript, który został wprowadzony przez Facebooka w 2017 roku. Obecnie wspiera go również Google. Yarn stał się głównym substytutem dla istniejącego już na rynku konkurencyjnego rozwiązania NPM. Yarn jest w pełni kompatybilny wstecz ze strukturą NPM [4].

Powody, dla których zespół Facebook'a zdecydował się na stworzenie własnego menadżera pakietów Yarn:

- możliwość działania w trybie bez dostępu do internetu, ponieważ Yarn posiada mechanizm zapamiętywania pakietów w pamięci podręcznej, dzięki czemu raz załadowane pakiety, zostają w tzw. cache, zamiast być pobierane ponownie, co przekłada się bezpośrednio na wydajność czasową,
- uruchamianie instalacji domyślnie w trybie deterministycznym, dzięki czemu struktura folderu zależności na każdej maszynie pozostaje identyczna. Zabieg ten zapobiega powstawaniu tzw. „piekła zależności", zwłaszcza gdy nad projektem pracuje w danym momencie wielu programistów jednocześnie, na różnych maszynach. Piekło zależności (ang. dependency hell) – potoczny termin używany przez programistów, który określa błędnie zdefiniowane lub trudne do spełnienia zależności, uniemożliwiające lub utrudniające poprawną instalacje lub działanie aplikacji.

Instalacja

W celu zainstalowania Yarn wystarczy posiadać Node.js na maszynie, na której odbywać się będzie instalacja. Yarn jest pakietem dostępnym w repozytorium pakietów NPM. Aby zainstalować go globalnie (posiadając Node.js) wystarczy wykonać komendę:

```
npm install -g yarn
```

Skrypty yarn

Yarn podobnie do NPM ma możliwość uruchamiania zdefiniowanych skryptów w pliku konfiguracyjnym, jednak yarn umożliwia programiście przekazanie jako wartość cały plik JavaScript zawierający skrypt, który ma się uruchomić po podaniu klucza, pod którym zapi-

sany został w pliku konfiguracyjnym. Służy do tego komenda **`yarn run <nazwa_skryptu>.`**

Listing 2: Przypisanie skryptu do pliku konfiguracyjnego.

```
"scripts": {
"build-project": "node build-project.js"
}
```

W przypadku wywołania komendy

`yarn run build-project`

zostanie uruchomiona komenda wywołująca instrukcję znajdujące się w pliku o tej samej nazwie, z rozszerzeniem js. Nazewnictwo skryptów yarn przekładać się może na moment ich wywołania. Przykładowo nazwa skryptu zawierająca frazę *preinstall* spowoduje, że skrypt zostanie wywołany przed zainstalowaniem pakietu, natomiast w przypadku fraz *postinstall, install, postinstall, prepublish, prepare* skrypt zostanie wywołany po zakończeniu instalacji pakietu.

Plik blokady

W momencie, w którym do zarządzania pakietami używany jest yarn, plik blokady zostaje wygenerowany automatycznie, co jest olbrzymią zaletą tego rozwiązania. Plik blokady tworzony przez yarn nosi nazwę ***yarn.lock***. Kolejną zaletą jest fakt, iż za każdym razem, gdy uruchamiana jest komenda ***yarn install,*** plik blokady jest aktualizowany w sposób automatyczny. Plik blokady w yarn, podobnie jak w przypadku NPM, pozwala programistom uniknąć zjawiska zwanego piekłem zależności. Warto zwrócić uwagę na złożoność oraz mnogość informacji, które dostarcza programistom plik blokady yarn. Plik ten zawiera nazwę pakietu, wersję, link do rejestru repozytoriów, hash integralności oraz zależności, które bazują na tym pakiecie.

Semantyka wersjonowania

Yarn respektuje identyczną politykę wersjonowania co NPM. Dokładniejszy opis tej semantyki podano w rozdziale 3.1.

3.3. Plik konfiguracyjny

Plik konfiguracyjny (*package.json*) – każdy projekt napisany w JavaScript, niezależnie od tego czy jest to pełnoprawna aplikacja serwerowa napisana z Node.js, czy jest to zwykła aplikacja internetowa, może zostać skondensowana do pakietu NPM, zawierającego swoje własne informacje o tym pakiecie oraz swój własny plik konfiguracyjny, który opisuje ten projekt. Plik konfiguracyjny zostaje wygenerowany w momencie wywołania komendy ***npm init***, inicjalizującej projekt JavaScript z domyślnymi metadanymi projektowymi:

- ***name:*** nazwa projektu lub biblioteki,
- ***version:*** wersja projektu, która zazwyczaj jest pomijana, ponieważ w większości przypadków nie ma konieczności jej ustawiania na tym etapie, lub ustawiana jest później bazując np. na dokumentacji projektowej,
- ***description:*** opis projektu,
- ***licenses:*** licencja jaką objęty jest tworzony projekt.

4. Analiza porównawcza

Kryteria, które uwzględniono w prowadzonej analizie porównawczej to [7]:

- instalacja,
- wydajność,
- prędkość,
- niezawodność,
- komendy CLI,
- ilość logów,
- ilość zajmowanego miejsca na dysku przez folder zależności,
- popularność,
- bezpieczeństwo.

Instalacja NPM oraz instalacja yarn została przedstawiona odpowiednio w rozdziałach 3.1. oraz 3.2. Porównując oba te procesy stwierdzono, że NPM jest prostszy w instalacji. Porównania wydajności znajduje się w rozdziale 5, a wyniki przedstawiono w rozdziale 6.

Obydwa rozwiązania pobierają pakiety z tego samego repozytorium, używając w przypadku yarn komendy yarn add a w przypadku NPM jest to npm install. Jednakże yarn jest o wiele szybszy niż NPM, co przedstawione zostało na Rysunku 2, ponieważ yarn instaluje wszystkie pakiety jednocześnie, ponadto yarn zapisuje pobrane pakiety w pamięci podręcznej, co zapobiega pobieraniu tego samego pakietu po raz drugi.
Menadżer pakietów, który stworzyła olbrzymia firma z sektora wytwarzania oprogramowania, ma pewne zalety (yarn), rozwiązanie to jest niezawodne i stabilne, dużo bardziej od NPM, ponieważ yarn używa pliku blokady, oraz deterministycznego algorytmu podczas instalacji każdego z pakietów, yarn również gwarantuje, że jeżeli coś działa w jednym środowisku programistycznym to będzie działało również na każdym innym. Jeżeli w yarn pojawia się jakiś błąd jest bardzo szybko naprawiany przez zespół programistyczny Marka Zuckerberga, czego niestety nie można powiedzieć na temat NPM.

Programiści muszą spędzić wiele godzin studiując komendy wykorzystywane w CLI, każdego narzędzia, w aspekcie menadżerów pakietów nie jest inaczej. W tabeli 2 przedstawiono komendy CLI dla obu pakietów.

Porównanie liczby logów zostało wykonane na podstawie instalacji tego samego pakietu (lodash). Yarn wygenerował 13 linii logów, natomiast NPM około 40 linii. Yarn generuje skondensowane logi, przekazując programiście tylko najważniejsze informacje, więc w liczbie logów wygrywa yarn.

Porównanie ilości zajmowanej przestrzeni dyskowej sprawdzono po wykonaniu komendy instalacyjnej i sprawdzeniu ilość zajmowanej przestrzeni dyskowej.

Dla yarn folder ***node_modules*** zajmował **307 MB**, zaś NPM **317 MB**.

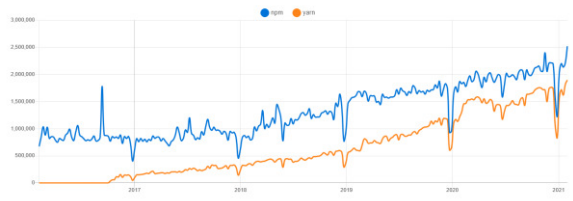
Wybierając menadżer pakietów należy również zwrócić uwagę na popularność rozważanego rozwiązania. Jeżeli wybrana technologia jest bardziej popularna, wówczas można liczyć na szerokie spektrum pomocy od społeczności. Jednak nie warto skreślać nowych rozwiązań, bowiem wszystko co jest nowe i sygnowane przez giganta technologicznego (Facebook w przypadku yarn), będzie zyskiwać szybko na popularności. Rysunek 1 przedstawia wykres liczby pobrań obu rozwiązań na przestrzeni ostatnich pięciu lat. Obserwując ten wykres bez trudu można określić, które rozwiązanie było popularniejsze na przestrzeni tych lat, ale również można dostrzec, że w roku 2020 popularność yarn znacząco wzrosła.

Tabela 2: Komendy CLI różne w obu menadżerach

Komenda	NPM	yarn
Inicjalizacja projektu	npm init	yarn init
Uruchomienie testów dla konkretnego pakietu	npm test	yarn test
Sprawdzenie czy, którykolwiek pakiet uległ przedawnieniu	npm outdated	yarn outdated
Publikacja pakietu do repozytorium NPM	npm publish	yarn publish
Uruchomienie skryptu	npm run	yarn run
Czyszczenie pamięci podręcznej	npm cache clean	yarn cache clean
Proces logowania i wylogowania	npm login / npm logout	yarn login / yarn logout

Tabela 3: Przedstawia komendy CLI identyczne w obu menadżerach.

Komenda	NPM	yarn
Instalacja zależności	npm install	yarn
Instalacja pakietu	npm install [nazwa_pakietu]	yarn add [nazwa_pakietu]
Odinstalowanie pakietu	npm uninstall [nazwa_pakietu]	yarn remove [nazwa_pakietu]
Aktualizacja środowiska	npm update	yarn upgrade
Aktualizacja pakietu	npm update [nazwa_pakietu]	yarn upgrade [nazwa_pakietu]
Instalacja pakietu globalnie	npm install --global [nazwa_pakietu]	yarn global add [nazwa_pakietu]
Odinstalowanie pakietu globalnie	npm uninstall --global [nazwa_pakietu]	yarn global remove [nazwa_pakietu]
Interaktywna aktualizacja	npm run upgrade-interactive	yarn upgrade-interactive



Rysunek 1: Popularność yarn i npm w latach 2016-2021 [6]

Bezpieczeństwo jest jedną z ważniejszych kwestii w porównaniu menadżerów pakietów, zaraz po wydajności. Początkowo yarn był uznawany za najbezpieczniejszy menadżer pakietów, ale zespół programistyczny NPM wykonał dużo pracy, aby zmienić ten fakt. Od wersji 6.0 NPM wprowadził mechanizm wbudowanych zabezpieczeń, dzięki któremu, jeżeli programista będzie chciał zainstalować pakiet ze znaną luką w zabezpieczeniach, NPM wyświetli stosowny komunikat ostrzegawczy.

Yarn natomiast ma inny mechanizm, zapewniający bezpieczeństwo, który polega na weryfikacji sumy kontrolnej każdego pakietu oraz licencji jaką objęty jest dany pakiet, i tego w przypadku NPM wciąż brakuje. Mimo dużego nakładu pracy wykonanego przez zespół NPM, yarn nadal pozostaje lepiej zabezpieczonym menadżerem pakietów niż NPM.

5. Środowisko badawcze

Badania zostały przeprowadzone na komputerze o specyfikacji:

- procesor – Intel Core i5 4460 3.2Ghz, 6 MB,
- dysk – SSD ADATA 256GB 2,5” SATA Ultimate S800,
- karta graficzna – MSI Radeon R9 270X HAWK, 2GB GDDR5,
- płyta główna – Gigabyte GA-H81-D3,
- prędkość łącza internetowego – 250 Mb/s,
- system operacyjny – Windows 10 Pro 64 bit,
- środowisko programistyczne IntelliJ IDEA Ultimate wer. 2020.2.2.

5.1. Aplikacja testowa

Stworzenie aplikacji testowej polegało na wygenerowaniu dwóch nowych projektów za pomocą IntelliJ IDEA wykorzystujących szkielet programistyczny Angular, który stanowił tylko bazę imitującą kompletny i działający projekt. Do pliku konfiguracyjnego projektu pełniącego role dużej aplikacji dodane zostało około 100 dodatkowych losowych zależności. Drugi plik pozostał bez zmian.

6. Badania wydajności czasowej

Badania zostały przeprowadzone w seriach po 100 prób na każdą komendę. Dwie komendy na dwóch plikach konfiguracyjnych, dla obu menadżerów pakietów dało łącznie 800 prób. Z czasów uzyskanych, w przypadku każdej z komend, na poszczególnych plikach konfiguracyjnych, została wyciągnięta średnia arytmetyczna. Wyniki przedstawiono w Tabeli 3.

6.1. Charakterystyka zbioru danych pierwotnych

Dane podchodzą z testów wydajnościowych przeprowadzonych na dwóch plikach konfiguracyjnych za pomocą biblioteki gnomon oraz z wykorzystaniem terminala, dla obu menadżerów pakietów. Wyniki otrzymano z dokładnością do 1 ms. Biblioteka gnomon została wykorzystana do pomiaru czasów wykonania poszczególnych komend, wywoływanych za pomocą terminala.

6.2. Algorytmy służące do obliczania poszukiwanych wartości

Do obliczania poszukiwanych wielkości użyto wzoru na średnią arytmetyczną. Uzyskane wyniki zostały zaokrąglone do dwóch miejsc po przecinku i są zestawione w rozdziale 6.

6.3. Przeprowadzenie badań

Przed przystąpieniem do badania wydajności czasowej menadżerów pakietów, upewniono się, iż żaden zbędny proces, obciążający środowisko badawcze nie pozostał włączony, sprawdzono również obciążenie środowiska badawczego i upewniono się, że żadna usługa sieciowa nie działa w tle i nie pobiera zasobów. Badania były przeprowadzone dla następujących scenariuszy:

- **S1** - plik konfiguracyjny zawiera podstawowe zależności dla projektu Angular.
- **S2** - plik konfiguracyjny zawiera dodatkowe zależności dla projektu Angular.

Scenariusz S1 dla podstawowego projektu Angular

1. Stworzono pusty projekt poprzez użycie komendy *npm init* dla NPM oraz *yarn init* dla yarn.
2. Wykonano komendy *yarn install | gnomon* dla folderu, w którym znajduje się projekt stworzony za pomocą yarn i analogicznie *npm install | gnomon* dla projektu utworzonego przez NPM.
3. Wykonano komendy *npm update | gnomon*, oraz *yarn upgrade | gnomon.*

Scenariusz S2 dla rozbudowanego projektu Angular

Procedura testowa jest zbliżona do S1, z tym, że zanim przystąpiono do wykonania testów, dodane zostało ponad sto dodatkowych zależności, aby odzwierciedlić wykonanie każdej z komend na rozbudowanym projekcie.

7. Wyniki badań

W Tabeli 4 przedstawiono średnie czasy wykonania scenariuszy S1 i S2 dla yarn, natomiast w Tabeli 5 – dla NPM. Rysunki 2 i 3 obrazują rezultaty uzyskane dla obu pakietów.

Tabela 4: Wydajność czasowa yarn

Komenda CLI	Średni czas wykonania komendy dla S1	Średni czas wykonania komendy dla S2
yarn install	45,89s	107,4s
yarn upgrade	19,9s	49,09s

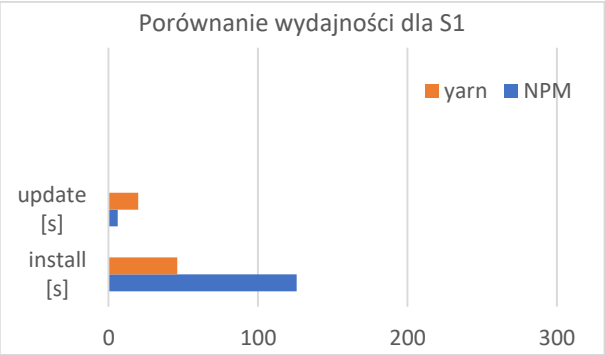
Tabela 5: Wydajność czasowa NPM

Komenda CLI	Średni czas wykonania komendy dla S1	Średni czas wykonania komendy dla S2
npm install	125,8s	258,43s
npm update	6,08s	19,4s

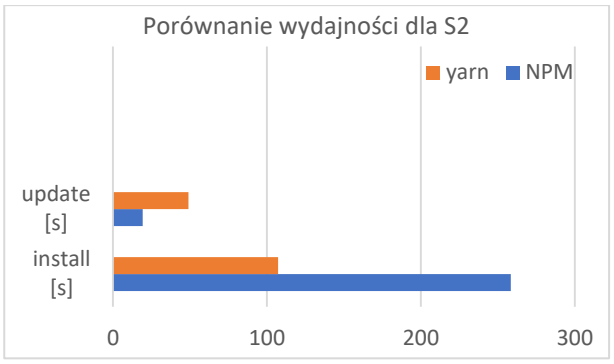
8. Wnioski

Na podstawie przeprowadzonych badań wyciągnięto następujące wnioski:

- Yarn jest wydajniejszy pod względem pobierania wszystkich zależności z użyciem komendy instalacyjnej od NPM, niemal trzykrotnie w przypadku małych aplikacji internetowych oraz nieco ponad dwa i pół razy wydajniejszy w przypadku rozbudowanych aplikacji internetowych.
- NPM jest wydajniejszy pod względem aktualizacji pakietów od yarn, w przypadku zarówno małych aplikacji internetowych, ponad trzykrotnie jak i w przypadku rozbudowanych aplikacji, w przypadku których NPM, jest wydajniejszy ponad dwukrotnie.
- Pomimo faktu, iż NPM jest narzędziem chętniej wybieranym przez programistów, to yarn jest lepszym wyborem pod względem wydajności czasowej w aplikacjach zawierających dużą ilość pakietów.
- Wyniki uzyskane w artykule autorstwa Jacobs Alexander z 2019 roku, które nie dały jednoznacznej odpowiedzi, które rozwiązanie autor poleca do użytku, najbliżej tego wyniku w porównaniu tego autora był NPM. Co tylko potwierdza ogólnie znaną tezę, że w świecie programowania technologie zmieniają się bardzo dynamicznie. Nie inaczej jest w przypadku JavaScript, w 2019 roku NPM wiódł prym, a w 2020 prym wiódł yarn.
- Hipoteza postawiona na początku artykułu została potwierdzona.



Rysunek 2: Porównanie wydajności dla aplikacji z podstawową liczbą zależności



Rysunek 3: Porównanie wydajności aplikacji z rozbudowaną siatką zależności

Yarn osiągnął lepszy wynik od NPM w następujących kategoriach:

- bezpieczeństwo,
- ilość zajmowanej przestrzeni dyskowej,
- ilość produkowanych logów,
- niezawodność,
- prędkość,
- wydajność czasowa.

NPM okazał się lepszym rozwiązaniem pod kątem instalacji oraz popularności wśród społeczności programistów.

Literatura

[1] MSR '16: Proceedings of the 13th International Conference on Mining Software Repositories https://dl.acm.org/doi/abs/10.1145/2901739.2901743, [24.01.2021].

[2] Charakterystyka NPM https://www.freecodecamp.org/news/what-is-npm-a-node-package-manager-tutorial-for-beginners/, [03.01.2021].

[3] Działanie menadżerów pakietów JavaScript https://www.freecodecamp.org/news/javascript-package-managers-101-9afd926add0a/, [03.01.2021].

[4] Charakterystyka yarn https://engineering.fb.com/2016/10/11/web/yarn-a-new-package-manager-for-javascript/, [11.02.2021].

[5] E. Wittern, P. Suter, S, Rajagopalan, A look at the dynamics of the JavaScript package ecosystem, MSR'16: Proceedings of the 13 Conference of Mining Software Repositories, (2016) 351-361, https://dl.acm.org/doi/10.1145/2901739.2901743.

[6] A. Jacobs, Comparsion of Javascript Package Managersm 2019, https://www.theseus.fi/handle/10024/227945, [24.01.2021].

[7] Wykres popularności obu rozwiązań https://www.npmtrends.com/npm-vs-yarn, [24.01.2021].