



Co to jest npm

Build amazing things

Essential JavaScript development tools that help you go to market faster and build powerful applications using modern open source code.

NPM (Node Package Manager), to (1) menedżer pakietów dla JavaScript i (2) największa biblioteka oprogramowania - darmowe pakiety do wykorzystania w swoich projektach.

<https://www.npmjs.com/>

Jest instalowany razem z Node.js (jeśli mamy Node na swojej maszynie to mamy i npm). NPM, to domyślny menedżer pakietów dla Node.js.

npm --version lub **npm -v** // polecenie, które pokazuje zainstalowaną wersję npm

npm install npm -g // aktualizacja npm

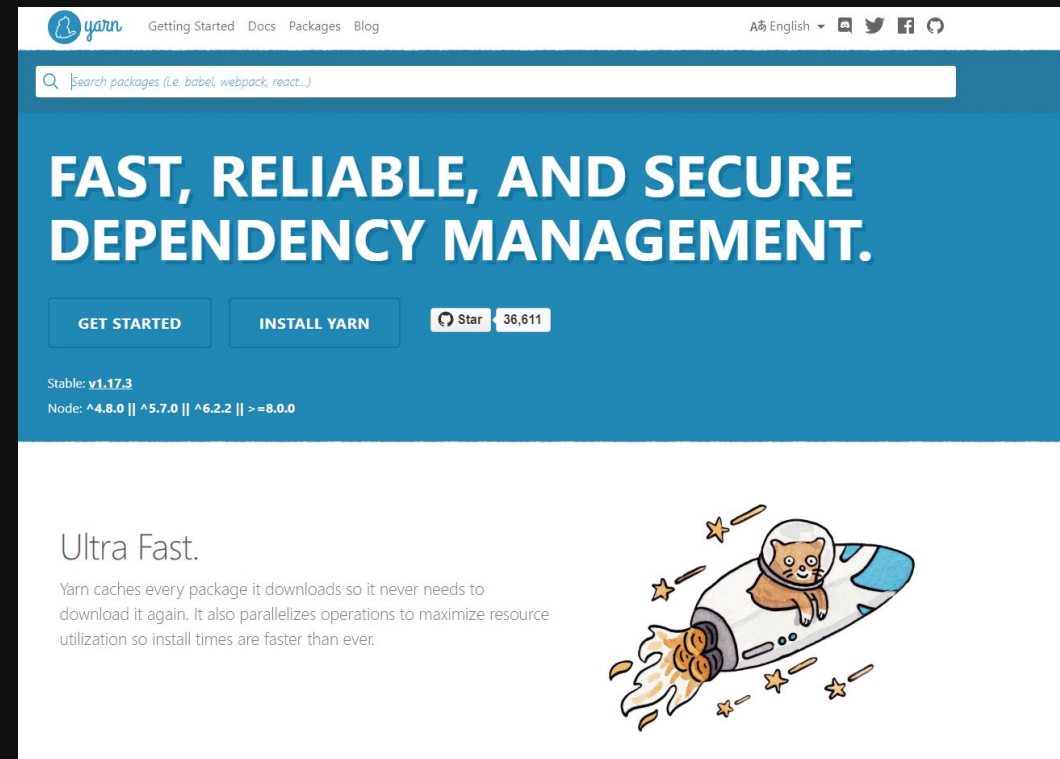
npm i alternatywy

Od wersji 6 (aktualna na 08.2019), npm jest zdecydowanie szybszy niż wcześniej (szybsza instalacja pakietów). NPM (jako organizacja) chwali się, że npm nie jest (już) wolniejszy niż yarn. Czym jest yarn? Yarn to alternatywny menedżer pakietów, którego głównym plusem jest szybkość. Na pewno npm jest popularniejszy, ale yarn ma także wielu zwolenników.

Różnica występuje także na poziomie poleceń np.

yarn add sass (yarn)

npm install sass (node)



<https://yarnpkg.com/>

npm jako oprogramowanie

Oprogramowanie (narzędzie wiersza poleceń), które pozwala zarządzać pakietami (instalować, usuwać, aktualizować, weryfikować wersje) i zależnościami między nimi (jeden pakiet często korzysta, a więc i instaluje, wiele innych pakietów od których uzależnione jest jego działanie). W naszym projekcie, po pierwszej instalacji jakieś paczki, pojawia się katalog `node_modules`

PS D:\example> `npm install express` //proces instalacja, to kilka -kilkanaście sekund

+ `express@4.17.1`

added 50 packages from 37 contributors and audited 126 packages in 10.622s

```

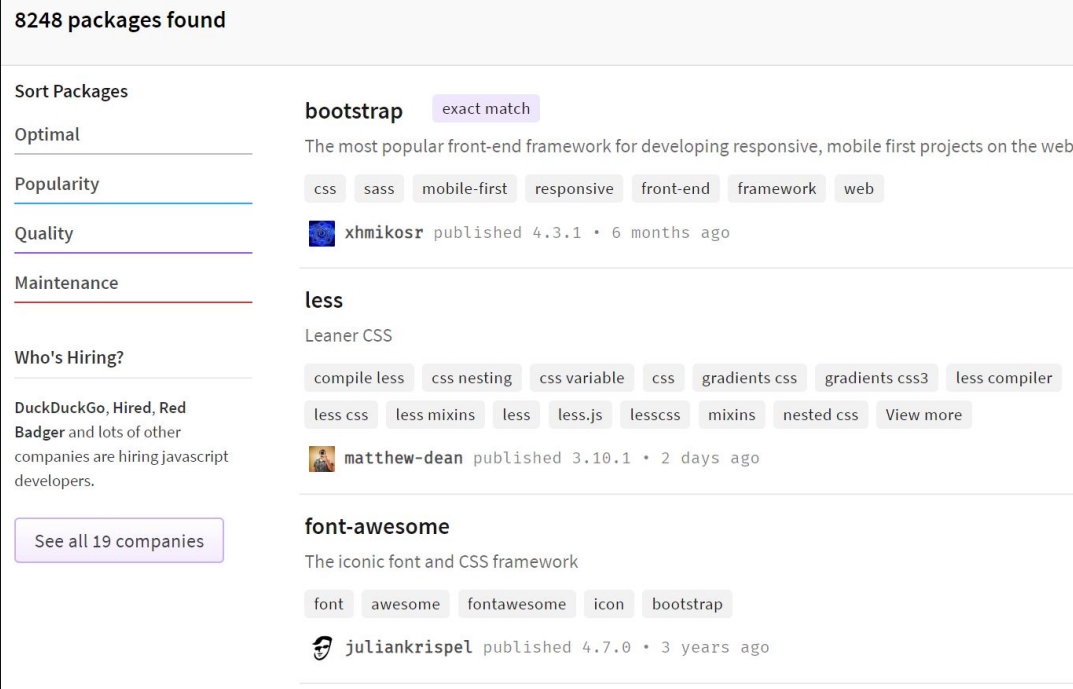
└─ node_modules
  ├── .bin
  ├── accepts
  ├── array-flatten
  ├── body-parser
  ├── bytes
  ├── content-disposition
  ├── content-type
  ├── cookie
  ├── cookie-signature
  ├── debug
  ├── depd
  ├── destroy
  ├── ee-first
  ├── encodeurl
  ├── escape-html
  ├── etag
  ├── express
  ├── finalhandler
  ├── forwarded
  ├── fresh
  ├── http-errors
  ├── iconv-lite
  ├── inherits
  └── ipaddr.js
```

npm jako zbiór (rejestr) pakietów

Kilkaset tysięcy darmowych pakietów dostępnych w zbiorze npm. Wyszukiwarka dostępna pod adresem npmjs.com

Bardzo duży wybór rozwiązań, najlepiej kierować się popularnością, poleceniami oraz liczbą aktualizacji.

Pakiety tworzone są przez firmy oraz społeczność i udostępniane poprzez rejestr npm.



8248 packages found

Sort Packages

Optimal

Popularity

Quality

Maintenance

Who's Hiring?


DuckDuckGo, Hired, Red Badger and lots of other companies are hiring javascript developers.

See all 19 companies

bootstrap exact match

The most popular front-end framework for developing responsive, mobile first projects on the web.

css sass mobile-first responsive front-end framework web


 **xhmikosr** published 4.3.1 • 6 months ago

less

Leaner CSS

compile less css nesting css variable css gradients css gradients css3 less compiler


less css less mixins less less.js lesscss mixins nested css View more

 **matthew-dean** published 3.10.1 • 2 days ago

font-awesome

The iconic font and CSS framework

font awesome fontawesome icon bootstrap

 **juliankrispel** published 4.7.0 • 3 years ago

npm init - pierwsze polecenie do zapamiętania i wykonania

Plik `package.json` jest przede wszystkim dokumentacją zależności naszego projektu. Zawiera m.in.:

- informacje o użytych paczkach w naszym projekcie (dependencies i devDependencies).
- obsługiwane polecenia (scripts) i informacje o projekcie.
- zawiera m.in. nazwę projektu, jego wersję czy opis (przy czym niezbędne są dwa pierwsze elementy).

Plik `package.json` umożliwia łatwe odtworzenie projektu (np. udostępnienie komuś innemu). Dzięki `package.json` odtworzymy folder `node_modules` (za pomocą polecenia `npm install`)

Od polecenia `npm init` zaczynamy tworzenie projektu. Polecenie wykonujemy w wierszu poleceń w katalogu, którym tworzymy projekt.

Plik `package.json` jest automatycznie aktualizowany gdy dodajemy jakieś paczki do naszego projektu.

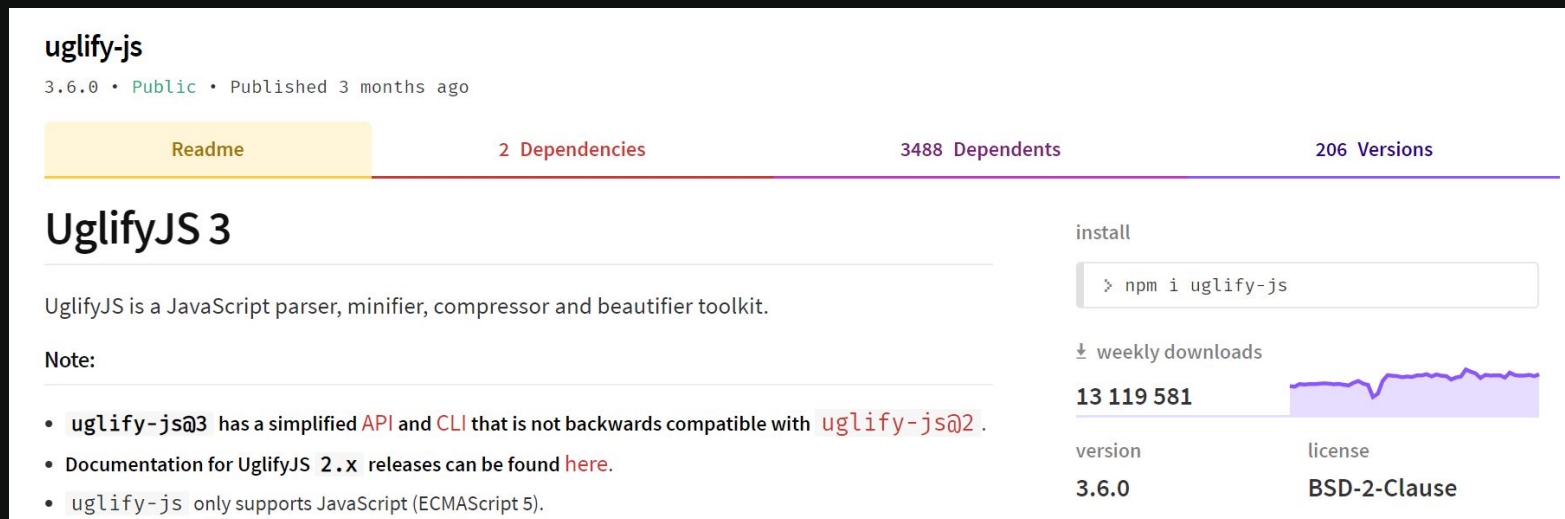
```
PS D:\projekty\kursy\css-js\npm\project> npm init
```

Pakiety/paczka (w npm)

Oprogramowanie, które może być użyte w naszym programie. Dodawane w formie modułów.

Pakiet to przede wszystkim kod, który realizuje jakieś zadanie.

Pod słowem pakiet może kryć się biblioteka, narzędzie, framework itd. Często zamiast pakiet, w kontekście node, używamy moduł czy paczka.



The screenshot shows the npm page for the 'uglify-js' package. At the top, it displays the package name 'uglify-js', version '3.6.0', and status 'Public'. Below this, there are four tabs: 'Readme' (highlighted in yellow), '2 Dependencies', '3488 Dependents', and '206 Versions'. The main section is titled 'UglifyJS 3' and contains a description: 'UglifyJS is a JavaScript parser, minifier, compressor and beautifier toolkit.' A 'Note:' section follows, listing three bullet points: 'uglify-js@3 has a simplified API and CLI that is not backwards compatible with uglify-js@2.', 'Documentation for UglifyJS 2.x releases can be found here.', and 'uglify-js only supports JavaScript (ECMAScript 5)'. On the right side, there is an 'install' section with a code block showing the command '> npm i uglify-js'. Below this is a 'weekly downloads' section with a line graph and the number '13 119 581'. At the bottom right, there is a table with two columns: 'version' and 'license'. The 'version' column shows '3.6.0' and the 'license' column shows 'BSD-2-Clause'.

uglify-js
3.6.0 • Public • Published 3 months ago

Readme 2 Dependencies 3488 Dependents 206 Versions

UglifyJS 3

UglifyJS is a JavaScript parser, minifier, compressor and beautifier toolkit.

Note:

- `uglify-js@3` has a simplified **API** and **CLI** that is not backwards compatible with `uglify-js@2`.
- Documentation for UglifyJS 2.x releases can be found [here](#).
- `uglify-js` only supports JavaScript (ECMAScript 5).

install

```
> npm i uglify-js
```

weekly downloads

13 119 581

version	license
3.6.0	BSD-2-Clause

Pakiety/paczka - wersja

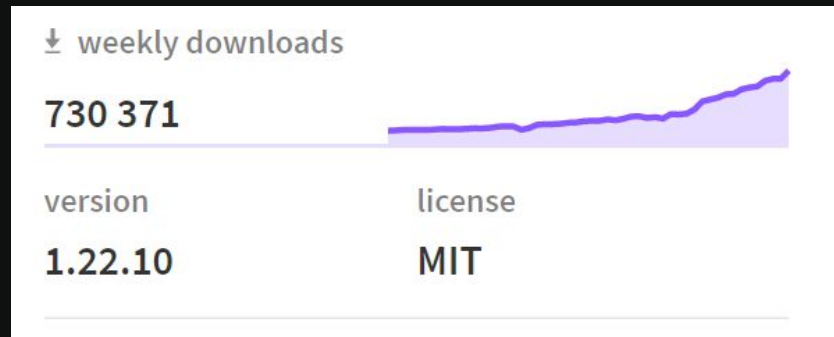
Każda paczka dostępny w npm posiada swoją wersję, która składa się z trzech liczb np. 0.8.12 (trzy liczby oddzielone kropkami). Paczka posiada też swoją dokumentację dostępną często bezpośrednio na stronie npm czy na GitHubie.

Pierwsza liczba oznacza główną wersję (major version), druga liczba oznacza podwersję (minor version), a trzecia łatkę/poprawkę (patch)

2.4.2 -> 2.4.3 (drobna poprawka, błąd)

2.4.3 -> 2.5.0 (większa zmiana, nowa funkcja)

2.5.0 -> 3.0.0 (nowa wersja, może być problem z kompatybilnością, zmiana zasad)



Budowa paczki

Paczka, którą instalujemy jest najczęściej oparta na wielu katalogach plikach. Każda paczka ma plik `package.json` który zawiera informacje na temat paczki, w tym wskazuje główny plik (entry point), którym najczęściej jest `index.js` (i w nim `require` do innych modułów).

Instalując jedną paczkę bardzo często instalujemy także kilka (kilkadziesiąt) innych, ponieważ dana paczka potrzebuje ich do działania.

```
└─ colors
   └─ examples
   └─ lib
      └─ custom
      └─ maps
      └─ system
      JS colors.js
      JS extendStringPrototype.js
      JS index.js
      JS styles.js
      └─ themes
      TS index.d.ts
      LICENSE
      {} package.json
      ⓘ README.md
      TS safe.d.ts
      JS safe.js
```

Instalowana paczka staje się zależnością projektu

dependency

Domyślnie paczka po instalacji staje się zależnością projektu, co możemy zobaczyć w pliku package.json. Dependencies ma oznaczać, że paczka jest potrzebna do uruchomienia aplikacji czyli musi znaleźć się w wersji produkcyjnej np. Bootstrap, jQuery, Font Awesome czy React.

devDependency (zależność deweloperska)

Taka paczka nie jest potrzebna na produkcji do działania aplikacji, ale jest używana podczas tworzenia (developmentu) aplikacji. Np. nie potrzebujemy kompilatora Sass -> CSS w wersji produkcyjnej, podobnie jak transpilatora JavaScript.

```
{} package.json ●
{} package.json > ...
1  {
2    "name": "npm-edu",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {},
7    "author": "",
8    "license": "ISC",
9    "devDependencies": {
10     "nodemon": "^1.19.1"
11   },
12   "dependencies": {
13     "font-awesome": "^4.7.0"
14   }
15 }
```

Instalowanie paczek lokalnie

`npm install nazwa-paczki --save (lub -S)`

Paczka, po instalacji, zostanie dopisana do obiektu dependencies w package.json

Ten parametr (`--save` lub `-S`) nie jest wymagany od wersji 6 NPM. Zainstalowana paczka znajdzie się domyślnie w dependencies. W ten sposób instalujemy paczki, które będą potrzebne w wersji produkcyjnej naszego projektu.

`npm install nazwa-paczki --save-dev (lub -D)`

Paczka zostanie dopisana do obiektu devDependencies pliku package.json. W ten sposób instalujemy paczki, których potrzebujemy przy developmencie naszej aplikacji. Przykładowo Babel czy kompilator Sass.

Instalowanie paczek globalnie

`npm install nazwa-paczki -g (lub --global)`

Paczka zostanie zainstalowana globalnie. Dostęp do niej uzyskujemy w każdym projekcie i z każdego miejsca naszej maszyny. Uwaga: Tak zainstalowana paczka oczywiście nie zostaje dopisana do package.json projektu!

Jest to dobre rozwiązanie dla niektórych narzędzi, ale jego minus jest taki, że przy przenoszeniu projektu na inną maszynę czy gdy pracuje na nim inny developer, to może on nie mieć zainstalowanych tych paczek. Jeśli paczka jest zainstalowana lokalnie (jako devDependency czy dependency), to przy każdym użyciu npm install zostanie zainstalowana (bo zostanie zainstalowane to co jest w package.json)

Często mamy więc alternatywa czy zainstalować to samo narzędzie globalnie czy lokalnie.

Instalowanie globalne ma sens jeśli z tego samego narzędzie będziemy korzystać w różnych projektach. Zobaczmy to na przykładzie narzędzia nodemon. Część deweloperów instaluje też paczkę jednocześnie lokalnie i globalnie. Taką instalację można zrobić w jednym poleceniu np. `npm install nodemon -g nodemon -D`

package-lock.json

Gdy instalujemy paczki lokalnie (w projekcie, a nie globalnie), to pojawia się także plik package-lock.json (od wersji 5 npm). Zawiera on dodatkowe informacje o paczkach m.in. dokładne wersje paczek użytych w projekcie (w tym zależności instalowanych przez nas paczek). Informacje te zwiększają bezpieczeństwo naszego projektu i szybkość instalacji.

Nie edytujemy tego pliku sami.

```
"ajv": {
  "version": "6.10.2",
  "resolved": "https://registry.npmjs.org/ajv",
  "integrity": "sha512-70zldj/9RWKYzI6+3oG5egc2Isp0p4uKID8sU6QW25LzR1Tq4iZn8oR1Bq305v1q12I4YUgJLrVgKw==",
  "dev": true,
  "requires": {
    "fast-deep-equal": "^2.0.1",
    "fast-json-stable-stringify": "^2.0.0",
    "json-schema-traverse": "^0.4.1",
    "uri-js": "^4.2.2"
  }
},
"amdefine": {
  "version": "1.0.1",
  "resolved": "https://registry.npmjs.org/amdefine",
  "integrity": "sha1-SlKCrBZHKek2GbZ90tFR+E",
  "dev": true
},
"ansi-regex": {
  "version": "2.1.1",
```

nodemon

Narzędzie, które automatycznie restartuje aplikację przy każdej wykrytej zmianie w pliku.



nodemon reload, automatically.

nodemon

W przykładzie zainstalujemy go globalnie jak (można też bezpośrednio w projekcie jako zależność developerską). Następnie w wierszu poleceń zamiast node wpisujemy nodemon.

- (1) `npm install -g nodemon`
- (2) `nodemon app.js`

install

```
> npm i nodemon
```

↓ weekly downloads

1 568 682



version

1.19.1

license

MIT

kompilator Sass

node-sass

Paczka dzięki, której możemy skompilować nasz kod sass/scss do css.

- (1) `npm install -g node-sass`
- (2) `node-sass input output`

`np. node-sass sass/style.scss css/style.css`

install

```
> npm i node-sass
```

↓ weekly downloads

3 386 696

version

4.12.0

license

MIT

dodanie konkretnej wersji

npm install nazwa-paczki

// najnowsza stabilna wersja np. 5.2.7

npm install nazwa-paczki@4

// ostatnia stabilna wersja 4 np. 4.8.6)

npm install nazwa-paczki@4.2

// ostatnia wersja 4.2 np. 4.2.5

npm install nazwa-paczki@4.3.3 (precyzyjnie wskazana wersja)

// zainstalowana wersja 4.3.3



npm install i package.json

- instalowanie zależności na podstawie package.json
- wersje paczek i aktualizacja w procesie instalacji projektu
- usuwanie paczek



zainstalowanie wszystkich dependencies i devDependencies - npm install

Korzystamy z zawartości `package.json` - devDependencies i dependencies

`npm install` (wszystkie paczki)

`npm install --production` (tylko paczki z dependencies)

`^3.4.2` - ostatnia stabilna wersja 3 (czyli np. 3.7.2) Jeśli pojawi się wersja 4 to nie będzie brana pod uwagę.

`~3.4.2` - wersja i podwersja zostaną zachowane. W instalacji zostanie wzięta najnowsza poprawka np. 3.4.12 ale już nie 3.5.0

`3.4.2` - paczka dokładnie we wskazanej wersji

Odinstalowanie - npm uninstall

Przykładowe polecenia:

```
npm uninstall nazwa-paczki
```

```
npm uninstall --global nazwa-paczki
```

```
npm un nazwa-paczki (un - alias, podobnie jak to jest przy i dla install)
```

Inne przykładowe polecenia

npm list (lub dla paczek globalnych npm list -g)

npm list --depth=0 (lub npm list --depth 0)

npm outdated (czy mamy jakieś stare paczki w projekcie/lokalnie)

npm outdated -g (czy mamy jakieś stare paczki zainstalowane globalnie)

npm update (aktualizacja wszystkich paczek lokalnych)

npm update nazwa-paczki (aktualizacja konkretnej paczki)

npm update -g (aktualizacja paczek globalnych)