

Programowanie Front-end

Ekosystem programisty front-end

mgr inż. Jakub Gogola

Ekosystem JavaScript

- Język JavaScript, jak każdy współczesny język programowania, ma swój ekosystem, czyli zbiór narzędzi, bibliotek i framework'ów, które mają za zadanie ułatwić zarządzanie projektem i organizację kodu
- Do takich narzędzi możemy zaliczyć:
 - interpreter kodu,
 - biblioteki zapewniające formatowanie kodu oraz sprawdzającego jego stylistykę (*code style*),
 - menadżer pakietów,
 - design systems

Node.js

- Node.js to interpreter języka JavaScript (środowisko uruchomieniowe)
- Pozwala uruchamiać kod JavaScript poza przeglądarką
- Node.js umożliwia wykorzystanie języka JavaScript do zastosowań serwerowych (backend)

Node.js

- Node.js powstał w 2009 roku
- Stworzył go Ryan Dahl
- Wykorzystuje V8 - silnik JavaScript - ten sam, który został użyty w przeglądarce Google Chrome

Node.js

- Środowisko Node.js można zainstalować na swoim komputerze podążając za instrukcjami umieszczonymi na oficjalnej stronie: <https://nodejs.org/en/download/package-manager>

Install Node.js v22.11.0 (LTS) on macOS using nvm

```
1 # installs nvm (Node Version Manager)
2 curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.0/install.sh | bash
3
4 # download and install Node.js (you may need to restart the terminal)
5 nvm install 22
6
7 # verifies the right Node.js version is in the environment
8 node -v # should print `v22.11.0`
9
10 # verifies the right npm version is in the environment
11 npm -v # should print `10.9.0`
```

Bash [Copy to clipboard](#)

Node.js

Powiązane narzędzia

- Razem ze środowiskiem Node.js instalowane są dodatkowe narzędzia:
 - NPM - menadżer pakietów dla języka JavaScript
 - NVM - menadżer środowiska Node.js pozwalający na dynamiczne zarządzanie wersją środowiska
 - NPX - *in flight package runner* dla pakietów NPM

Node.js

- Dokumentacja: <https://nodejs.org/docs/latest/api/>
- Biblioteka standardowa Node.js pozwala realizować wiele funkcji typowych dla systemów backendowych, niedostępnych w API przeglądarki:
 - Serwer HTTP/WebSocket/...
 - Operacje na systemie plików
 - Połączenie z bazą danych
 - ...

Node.js

Przykład - prosty serwer HTTP

```
import http from 'http';

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello, Node.js Server!');
});

server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```


NPM

- Narzędzie NPM służy do zarządzania projektem oraz jego zależnościami
 - Strona narzędzia: <https://www.npmjs.com/>
 - Dokumentacja: <https://docs.npmjs.com/>

NPM

Funkcjonalności

- Narzędzie NPM (*Node Package Manager*) posiada kilka podstawowych funkcjonalności:
 - zarządzanie projektem stworzonym w oparciu o Node.js,
 - zarządzanie zależnościami (pakietami) w projekcie,
 - uruchamianie skryptów,
 - zarządzanie wersjami oraz publikowanie paczek w repozytorium NPM,

NPM

Podstawowe komendy

- `npm init` - tworzy nowy projekt
- `npm install` - instaluje nowe zależności w projekcie
- `npm start` - uruchamia aplikację
- `npm test` - uruchamia testy aplikacji
- `npm version patch|minor|major|...` - podbija wersję paczki
- `npm publish` - publikuje nową wersję paczki do repozytorium NPM

NPM

Części składowe projektu

- `package.json` - serce projektu, w którym zawarte są wszystkie podstawowe informacje o projekcie, lista zależności oraz zdefiniowane skrypty
- `package-lock.json` - plik jest automatycznie tworzony przez NPM i aktualizowany z każdym wywołaniem komendy `npm install`, zapewnia spójność wersji zależności pomiędzy różnymi środowiskami
- `node_modules` - katalog, w którym instalowane są wszystkie zależności zapisane w pliku `package.json`

NPM

Przykładowy plik package.json

```
{
  "name": "vite-template-react",
  "version": "1.4.0",
  "type": "module",
  "scripts": {
    "start": "vite --port 3000 --open",
    "dev": "vite --port 3000 --open",
    "build": "vite build",
    "serve": "vite preview --open",
    "test": "vitest",
    "eslint": "eslint .",
    "eslint:fix": "eslint --fix ."
  },
  "dependencies": {
    "@reduxjs/toolkit": "^2.3.0",
    "axios": "^1.7.7",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-redux": "^9.1.2",
    "react-router-dom": "^6.27.0"
  },
```

```
  "devDependencies": {
    "@eslint/js": "^9.10.0",
    "@stylistic/eslint-plugin": "^2.9.0",
    "@testing-library/jest-dom": "^6.1.6",
    "@testing-library/react": "^16.0.1",
    "@vitejs/plugin-react": "^4.2.1",
    "@vitejs/plugin-react-swc": "^3.7.1",
    "eslint": "^9.13.0",
    "eslint-config-prettier": "^9.1.0",
    "eslint-plugin-import": "^2.31.0",
    "eslint-plugin-prettier": "^5.2.1",
    "eslint-plugin-react": "^7.37.1",
    "eslint-plugin-react-hooks": "^5.0.0",
    "globals": "^15.9.0",
    "jsdom": "^25.0.1",
    "prettier": "3.3.3",
    "redux-logger": "^3.0.6",
    "vite": "^5.0.8",
    "vite-plugin-mkcert": "^1.17.6",
    "vitest": "^2.1.3"
  }
}
```

Node.js + NPM

Przykład - serwer HTTP z wykorzystaniem biblioteki Express.js

```
import express from 'express';

const app = express();

app.get('/', (req, res) => {
  res.send('Hello, Express Server!');
});

const port = 3000;
app.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}`);
});
```

Node.js + NPM

Przykład - serwer HTTP z wykorzystaniem biblioteki Express.js

```
import express from 'express';

const app = express();

const greetings = {
  en: `Hello`,
  es: `¡Hola`,
  fr: `Bonjour`,
  id: `Halo`,
  pl: `Cześć`
};

app.get('/api/greet/', (req, res) => {
  const name = req.query.name || 'Guest';
  const language = greetings[req.query.language] ? req.query.language : 'en';

  res.json({
    message: `${greetings[language]}, ${name}!`,
    name: name,
    language: language
  });
});

const port = 3000;
app.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}`);
});
```

NVM

- Narzędzie NVM (*Node Version Manager*) pozwala na zarządzanie wersjami Node.js oraz NPM pomiędzy różnymi projektami
- Przez zastosowanie NVM unika się problemu z kompatybilnością Node.js pomiędzy różnymi projektami

NVM

Podstawowe komendy

- `nvm install <node_version>` - instaluje zadaną wersję Node.js wraz z narzędziem NPM
- `nvm use <node_version>` - przełącza wersję Node.js oraz NPM na wskazaną wersję

NVM

Plik .nvmrc

- Plik `.nvmrc` zawiera wersję Node.js, która powinna być użyta w projekcie i pozwala na automatyczne wykrycie przez narzędzie NVM wymaganej wersji poprzez użycie komendy `nvm use` (bez dodatkowych argumentów oraz flag).

NVM

Integracja z powłoką systemową

- Narzędzie NVM można zintegrować z powłoką systemową (Bash, ZSH, Fish,...) w taki sposób, aby pliki `.nvmrc` były automatycznie wykrywane i w konsekwencji wybierana była kompatybilna wersja Node.js
- Skrypt należy umieścić w pliku odpowiedzialnym za konfigurację powłoki przy jej uruchomieniu:
 - Bash - `.bashrc`
 - ZSH - `.zshrc`

NVM

Integracja z powłoką Bash

```
#!/bin/bash
export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s "${HOME}/.nvm" || printf %s "${XDG_CONFIG_HOME}/nvm")"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm

# place this after nvm initialization!
autoload -U add-zsh-hook

load-nvmrc() {
  local nvmrc_path
  nvmrc_path="$(nvm_find_nvmrc)"

  if [ -n "$nvmrc_path" ]; then
    local nvmrc_node_version
    nvmrc_node_version=$(nvm version "$(cat "${nvmrc_path}")")

    if [ "$nvmrc_node_version" = "N/A" ]; then
      nvm install
    elif [ "$nvmrc_node_version" != "$(nvm version)" ]; then
      nvm use
    fi
  elif [ -n "$(PWD=$OLDPWD nvm_find_nvmrc)" ] && [ "$(nvm version)" != "$(nvm version default)" ]; then
    echo "Reverting to nvm default version"
    nvm use default
  fi
}

add-zsh-hook chpwd load-nvmrc
load-nvmrc
```

NPX

- NPX pozwala na uruchamianie pakietów z repozytorium NPM bez konieczności instalowania ich w projekcie lub globalnie
- Użyteczny dla *jednorazowych operacji*, na przykład inicjalizacja projektu

```
#!/bin/bash
```

```
npx create-react-app my-app --template typescript
```

ESLint

- ESLint to narzędzie służące do statycznej analizy kodu:
 - wykrywa błędy składniowe,
 - analizuj kod pod kątem stylistycznym i umożliwia (w dużej mierze) automatyczne poprawianie takich błędów,
 - pozwala dbać o jakość kodu,
- Dokumentacja: <https://eslint.org/>

Prettier

- Prettier jest narzędziem pozwalającym na formatowanie kodu według ustalonych zasad i dbać o jego czytelność.
- Możliwe jest jego zintegrowanie z narzędziem ESLint
- Dokumentacja <https://prettier.io/>

ESLint + Prettier

- Oba narzędzia można ze sobą zintegrować . Każde z nich posiada jednak osobną konfigurację umieszczanych w osobnych plikach:
 - Prettier – `.prettierrc` lub `prettier.js`
 - ESLint – `.eslintrc` lub `eslint.js`

ESLint + Prettier

Przykład

- Konfiguracja obu narzędzi zostanie omówiona na przykładzie projektu z poprzedniego wykładu dostępnym pod następującym linkiem: <https://github.com/JakubGogola-IDENTT/dsw-frontend-lecture-2024/tree/main/lecture-4/react>

Design systems

- **Design system** to zbiór zasad, dobrych praktyk oraz komponentów, które mają na celu tworzenie spójnego, skalowalnego i przejrzystego interfejsu użytkownika oraz zapewnić dobry *user experience* podczas korzystania z aplikacji.

Design system

Kluczowe funkcje

- predefiniowane, używalne elementy interfejsu
- spójność wyglądu pomiędzy wszystkimi elementami
- zasady i dobre praktyki tłumaczące w jaki sposób używać zapewnianych przez design system elementów oraz jak tworzyć nowe

Design System

Material Design/Material UI

- Material Design - <https://m3.material.io/>
- Material UI (biblioteka komponentów dla React'a) - <https://mui.com/material-ui/>

Dziękuję za uwagę!