Zaawansowany Programista JS

Dzień 6

Plan

- Środowisko Node.JS i NPM
  - Praca z paczkami za pomocą komend `npm init` i `npm install`
  - Praca z package.json i własne `scripts`
- Poznanie biblioteki `parcel`
- Podstawy ES6 (spread operator, funkcje strzalkowe)
- Poznanie import/export ES6
- Praca na wielu plikach
- Poznanie biblioteki `json-server`
- Tworzenie aplikacji przy uzyciu `fetch`, `parcel` i `json-server`

Środowisko Node.JS i NPM

Node.js to otwarte i wieloplatformowe środowisko uruchomieniowe oparte na silniku JavaScript V8, stworzone w celu umożliwienia tworzenia aplikacji serwerowych za pomocą języka JavaScript. Node.js działa po stronie serwera i umożliwia programistom tworzenie skalowalnych, wysokowydajnych aplikacji sieciowych, które mogą obsługiwać wiele równoczesnych połączeń.

NPM (Node Package Manager) to menedżer pakietów dla języka JavaScript, który jest częścią środowiska Node.js. NPM umożliwia programistom łatwe zarządzanie zależnościami i bibliotekami, niezbędnymi do budowania aplikacji w języku JavaScript. Dzięki NPM można pobierać i instalować biblioteki i moduły wraz z ich zależnościami z centralnego repozytorium pakietów. NPM pozwala także na publikowanie i udostępnianie stworzonych przez programistów bibliotek i modułów dla innych użytkowników.

Node.js i NPM są bardzo popularnymi narzędziami w świecie JavaScriptu i stanowią podstawę do tworzenia wielu aplikacji internetowych, zarówno dla małych firm jak i dla dużych korporacji.

Praca z paczkami za pomocą komend `npm init` i `npm install`

Aby rozpocząć pracę z NPM, należy najpierw zainicjować nowy projekt za pomocą polecenia npm init. Polecenie to utworzy plik package.json, który będzie zawierał podstawowe informacje o projekcie, takie jak nazwa, wersja, opis, autorzy, zależności i skrypty.

Przykładowe użycie polecenia npm init

```
npm init
```

Po wprowadzeniu wszystkich wymaganych informacji, zostanie utworzony plik package.json w katalogu projektu.

Aby zainstalować wymagane pakiety i zależności, można użyć polecenia `npm install`. Polecenie to pozwala na instalowanie pakietów lokalnie dla projektu lub globalnie dla całego systemu.

Przykładowe użycie polecenia npm install:

```
npm install <pakiet>
```

Można również zainstalować wiele pakietów jednocześnie, np.:

```
npm install <pakiet1> <pakiet2> <pakiet3>
```

Jeśli chcemy zainstalować pakiet jako zależność deweloperską, np. do testów lub do automatyzacji budowania, należy użyć flagi --save-dev lub -D:

```
npm install <pakiet> --save-dev
```

lub

```
npm install <pakiet> -D
```

Flaga ta doda pakiet do sekcji "devDependencies" w pliku package.json.

W przypadku, gdy chcemy zainstalować pakiet globalnie dla całego systemu, należy użyć flagi -g:

```
npm install -g <pakiet>
```

Należy pamiętać, że instalacja pakietów globalnych wymaga uprawnień administratora.

Praca z package.json i własne `scripts`

Jeśli chcemy dodać nowy skrypt o nazwie "start", który będzie uruchamiał naszą aplikację, musimy dodać tę linijkę do sekcji "scripts":

```
"start": "node app.js"
```

W tym przykładzie app.js to plik, który chcemy uruchomić. Możemy również dodać argumenty do komendy, np.:

```
"start": "node app.js --port 3000"
```

Możemy również edytować istniejące skrypty, zmieniając ich wartości.

Po wprowadzeniu zmian w pliku package.json, należy zapisać plik i uruchomić polecenie `npm run <nazwa_skryptu>`, aby uruchomić skrypt.

Na przykład, jeśli chcemy uruchomić skrypt o nazwie "start", należy wpisać polecenie

```
npm run start
```

Możemy również użyć krótszej formy tego polecenia, tj. `npm start`, ponieważ "start" jest jednym z predefiniowanych skryptów w NPM.

Poznanie biblioteki `parcel`

Parcel jest to narzędzie typu "zero configuration" dla budowania aplikacji webowych w JavaScript, które automatycznie obsługuje procesy takie jak bundling (pakowanie) plików, kompilacja (np. z TypeScript) oraz transpilacja (np. z ES6 do ES5). Parcel jest rozwijany przez społeczność open source i jest dostępny na platformie NPM (Node Package Manager) jako biblioteka JavaScript.

Główną cechą Parcela jest jego prostota obsługi. W przeciwieństwie do innych narzędzi, takich jak Webpack czy Gulp, Parcel nie wymaga ustawiania skomplikowanych konfiguracji. W zasadzie, można zacząć używać Parcela bez żadnej konfiguracji.

Parcel automatycznie wykrywa pliki wejściowe oraz zależności i tworzy dla nich odpowiednie pliki wynikowe. Parcel obsługuje różne rodzaje plików, takie jak JavaScript, CSS, HTML, obrazy, filmy i wiele innych.

Dzięki swojej prostocie i funkcjonalności, Parcel stał się popularnym narzędziem w świecie front-end developmentu.

Podstawy ES6 (spread operator, funkcje strzalkowe)

Spread Operator w ES6 to operator rozpraszający, który pozwala na rozbijanie elementów tablicy lub obiektu na oddzielne argumenty lub właściwości. Operator ten składa się z trzech kropek "..." i może być stosowany do tablic i obiektów.

Przykładowo, w przypadku tablic, Spread Operator umożliwia rozbicie jednej tablicy na kilka oddzielnych elementów, co ułatwia operacje na tablicach. Na przykład:

```
const arr1 = [1, 2, 3];
const arr2 = [4, 5, 6];

const arr3 = [...arr1, ...arr2];

console.log(arr3); // [1, 2, 3, 4, 5, 6]
```

W tym przykładzie, Spread Operator "rozszerza" (rozbija) tablice arr1 i arr2 i tworzy nową tablicę arr3 zawierającą wszystkie elementy z obu tablic.

W przypadku obiektów, Spread Operator umożliwia kopiowanie obiektów oraz łączenie ich właściwości. Na przykład:

```
const obj1 = { a: 1, b: 2 };
const obj2 = { c: 3, d: 4 };

const obj3 = { ...obj1, ...obj2 };

console.log(obj3); // { a: 1, b: 2, c: 3, d: 4 }
```

W tym przykładzie, Spread Operator umożliwia połączenie dwóch obiektów obj1 i obj2 w jeden obiekt obj3, który zawiera wszystkie właściwości obu obiektów.

Funkcje strzałkowe w ES6 to nowy sposób definiowania funkcji w JavaScript, który pozwala na zapisanie funkcji w bardziej zwięzły i czytelny sposób.

Składnia funkcji strzałkowych składa się z parametrów funkcji, strzałki oraz ciała funkcji

```
(parametry) => { ciało funkcji }
```

```
parametr => { ciało funkcji }
```

lub

```
() => { ciało funkcji }
```

Główną cechą funkcji strzałkowych jest to, że nie posiadają one własnego kontekstu `this`. Zamiast tego, kontekst `this` jest dziedziczony z kontekstu, w którym funkcja została zdefiniowana. Dzięki temu, funkcje strzałkowe pozwalają na uniknięcie niektórych problemów związanych z kontekstem this w JavaScript.

Funkcje strzałkowe są szczególnie przydatne w przypadku krótkich funkcji, które wykonują jedną prostą operację. Przykładowo, funkcja strzałkowa może wyglądać tak:

```
const add = (a, b) => a + b;
```

W tym przykładzie, funkcja strzałkowa add przyjmuje dwa argumenty a i b i zwraca ich sumę. Dzięki skróconej składni, funkcja ta jest bardziej czytelna i zwięzła niż tradycyjna funkcja zapisana w sposób:

```
function add(a, b) {
  return a + b;
}
```

Podsumowując, funkcje strzałkowe są nowym sposobem definiowania funkcji w JavaScript, który pozwala na zwięzły i czytelny zapis krótkich funkcji oraz rozwiązuje pewne problemy związane z kontekstem this.

Poznanie import/export ES6

Importy i eksporty w JavaScript to mechanizmy pozwalające na dzielenie kodu między plikami oraz na używanie kodu z zewnętrznych modułów w projekcie.

Eksport polega na zdefiniowaniu, które części kodu z danego pliku mają być dostępne z zewnątrz, a import polega na zaimportowaniu tych elementów do innego pliku, w którym można ich użyć.

W JavaScript od wersji ES6 (ES2015) importy i eksporty są obsługiwane nadrzędnie przez specjalne słowa kluczowe import i export. Można nimi importować i eksportować poszczególne funkcje, klasy, stałe czy zmienne.

Przykład eksportu funkcji o nazwie `sayHello` z pliku `hello.js`:

```
// hello.js
export const sayHello = name => {
  console.log(`Hello, ${name}!`);
}
```

Przykład importu funkcji `sayHello` z pliku `hello.js` do pliku `index.js`:

```
// index.js
import { sayHello } from './hello.js';

sayHello('John'); // wyświetli "Hello, John!"
```

W przypadku eksportu można także użyć słowa kluczowego `default`, aby zdefiniować element, który ma być domyślnie eksportowany. W takim przypadku przy imporcie można użyć dowolnej nazwy dla importowanego elementu.

Przykład eksportu klasy Person jako domyślnego elementu z pliku `person.js`:

```
// person.js
export default class Person {
  constructor(name) {
    this.name = name;
  }

  sayHello() {
    console.log(`Hello, my name is ${this.name}.`);
  }
}
```

Przykład importu klasy `Person` jako domyślnego elementu z pliku `person.js` do pliku `index.js`

```
import Person from './person.js';

const john = new Person('John');
john.sayHello(); // wyświetli "Hello, my name is John."
```

Praca na wielu plikach

Kilka dobrych praktyk dotyczących struktury folderów w projekcie JavaScript

- Oddzielaj kod zależnie od jego funkcjonalności: Podziel kod na moduły według ich funkcjonalności. Na przykład, jeśli masz stronę internetową, możesz mieć oddzielne moduły dla funkcji logowania, obsługi zamówień, wyświetlania produktów, itp.
- Nazwij pliki zgodnie z ich zawartością: Pliki powinny być nazwane zgodnie z ich zawartością, aby ułatwić ich identyfikację. Na przykład plik zawierający funkcje logowania powinien być nazwany "login.js".
- Zgrupuj pliki według typu: Pliki powinny być zgrupowane według ich typu, takie jak skrypty, style, obrazy, szablony HTML, itp.
- Unikaj umieszczania dużej liczby plików w jednym folderze: Jeśli masz wiele plików, lepiej podzielić je na mniejsze grupy i umieścić je w oddzielnych folderach.
- Używaj konwencji nazewnictwa: Używaj konwencji nazewnictwa, aby ułatwić czytanie kodu i uniknąć nieporozumień. Na przykład, stosuj snake_case lub camelCase dla nazw plików i funkcji.

- Użyj pliku .gitignore: Użyj pliku .gitignore, aby wykluczyć niepotrzebne pliki i foldery z repozytorium git.

Poznanie biblioteki `json-server`

`json-server` to biblioteka w języku JavaScript, która umożliwia łatwe tworzenie API opartych na plikach JSON. Jest to narzędzie często używane do testowania i prototypowania aplikacji, ponieważ pozwala na szybkie utworzenie "fałszywego" API, bez potrzeby tworzenia i konfigurowania serwera.

`json-server` pozwala na uruchomienie serwera API na lokalnym komputerze i przetwarzanie żądań HTTP, takich jak GET, POST, PUT, DELETE i inne. Serwer może obsługiwać wiele zasobów, z których każdy jest przechowywany w osobnym pliku JSON.

`json-server` jest łatwy w użyciu i konfiguracji, ponieważ większość konfiguracji jest automatyczna. Po zainstalowaniu biblioteki wystarczy utworzyć plik JSON z danymi i uruchomić serwer poleceniem json-server, podając ścieżkę do pliku JSON.

Na przykład, aby utworzyć API zawierające listę użytkowników, można utworzyć plik db.json z następującą zawartością

```
{
  "users": [
    { "id": 1, "name": "John Doe", "email": "john.doe@example.com" },
    { "id": 2, "name": "Jane Smith", "email": "jane.smith@example.com" },
    { "id": 3, "name": "Bob Johnson", "email": "bob.johnson@example.com" }
  ]
}
```

Następnie wystarczy uruchomić serwer poleceniem

```
json-server --watch db.json
```

Serwer będzie dostępny pod adresem http://localhost:3000/users i będzie obsługiwał żądania HTTP związane z listą użytkowników.

Tworzenie aplikacji przy uzyciu `fetch`, `parcel` i `json-server`

1. Stwórz Projekt przy pomocy biblioteki parcel
2. Wgraj potrzebne biblioteki do uruchomienia projektu
3. Stwórz bazę danych `trains.json` i odpal ją za pomocą `json-server`

```
{
  "trains": [
    {
      "id": 1,
      "from": "Warsaw",
      "to": "Wroclaw",
      "date": "07.11.2022",
      "isPremium": true
    },
    {
```

```
      "id": 2,
      "from": "Kraków",
      "to": "Wroclaw",
      "date": "05.11.2022",
      "isPremium": false
    },
    {
      "id": 3,
      "from": "Gdańsk",
      "to": "Warszawa",
      "date": "01.11.2022",
      "isPremium": false
    },
    {
      "id": 4,
      "from": "Kołobrzeg",
      "to": "Wrocław",
      "date": "10.11.2022",
      "isPremium": true
    },
    {
      "id": 5,
      "from": "Szczecin",
      "to": "Wrocław",
      "date": "4.11.2022",
      "isPremium": false
    },
    {
      "id": "613d0ade-aee2-4427-b271-b7fb72ccc594",
      "from": "Wrocław",
      "to": "Warszawa",
      "date": "07.11.2022",
      "isPremium": false
    }
  ]
}
```


4. Stwórz aplikację której zadaniem będzie

- Wyswietlenie listy pociagów z API
- Mozliwosc przefiltrowania pociagow po miejscowosci (from lub to)
- Mozliwosc dodania pociagow do bazy danych
- UWAGA: Jesli dodasz nowe pociagi, musisz miec mozliwosc filtrowania po nich.
