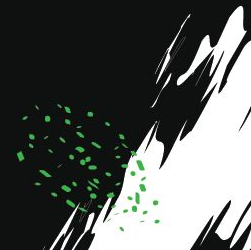



# **ES Modules w przeglądarce**



# Cechy modułów - przypomnienie

- wielokrotnego użytku (reusable)
- encapsulated (hermetyzacja - scope) + API modułów (udostępnienie poza moduł)
- pozwalają organizować kod



Ale czy można tego używać w przeglądarce??





Ale czy można tego używać w przeglądarce??

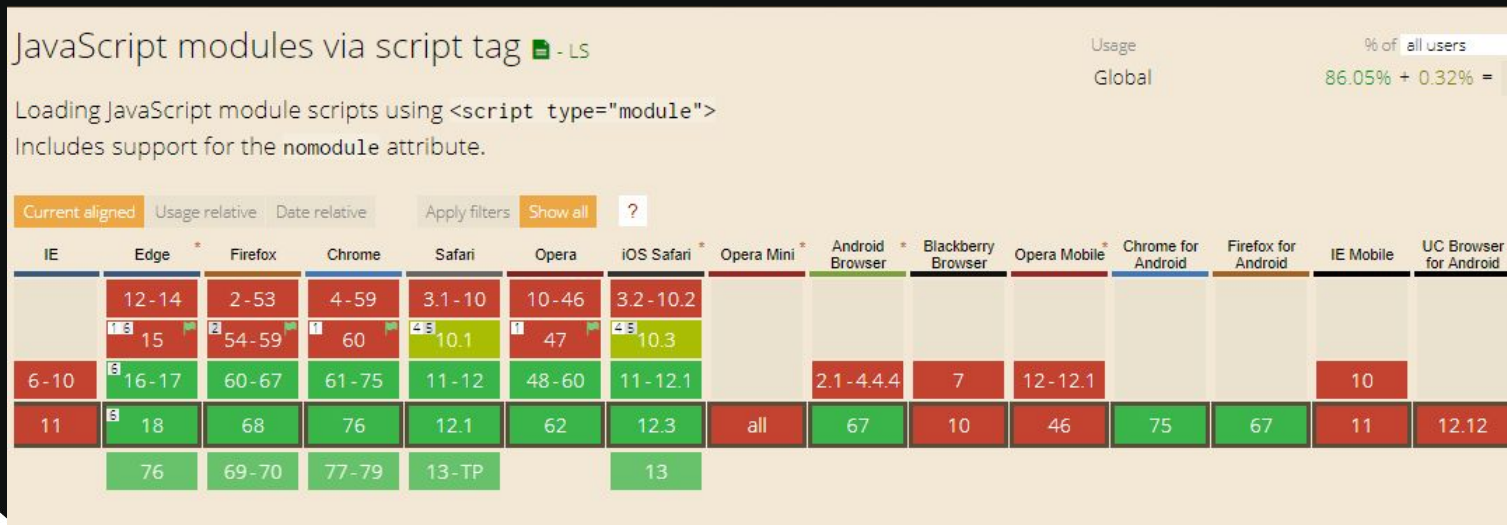
TAK!!!!



## Ale czy można tego używać w przeglądarce??

TAK!!!!

Choć nie w każdej ;) I nie każdej implementacji.



# ES Modules - webpack i przeglądarki

- ES Modules są rekomendowane do użycia wraz z webpackiem, który od wersji 2 dobrze sobie z nimi radzi, w przeciwieństwie np. do Node.js o czym już doskonale wiesz. Ale do [webpacka](#) jeszcze przejdziemy.
- Zamiast funkcji `require` i `module.exports` ([CommonJS](#)), w modułach działających w oparciu o ES Modules będziemy używać instrukcji `import` i `export` - poznamy ją więc trochę lepiej.
- **modułów opartych o CommonJS nie użyjemy natywnie w przeglądarce!** W przeciwieństwie do [ES Modules](#).

# ES Modules - historia natywnego wsparcia dla modułów

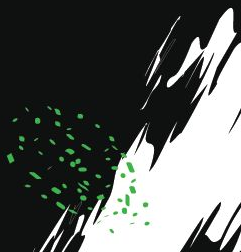
Oficjalnie pojawiły się w Javascript w 2015 roku (wraz ze specyfikacją ECMAScript 6). W webpacku natywne wsparcie już od wersji 2 (od 01.2017 r).

W przeglądarkach powolutku, ale do przodu. Obecnie zdecydowana większość przeglądarek ma już to wdrożone (jak i większość specyfikacji ES6).

W Node.js natywne wsparcie pojawi się pod koniec 2019 r. (od wersji 12 LTS)



# Instrukcije export i import





# instrukcja export

Sposób na udostępnienie elementów modułu. Takie udostępnione elementy mogą być potem importowane w innych modułach. Możemy udostępnić wiele elementów wielokrotnie używając instrukcji export. Zwróćmy uwagę, że w ten sposób można wyeksportować tylko element posiadający nazwę (identyfikator).

```
/* utils.js */  
export const add = (a, b) => a + b;  
export let users = ['Adam', 'Beti'];  
const number = 100;  
export const minus = function (a, b) { return a - b };
```

# instrukcja import

```
/* utils.js */  
export const add = (a, b) => a + b;  
export let users = ['Adam', 'Beti'];  
export const number = 100;
```

```
/* main.js */  
import { add, users } from './utils.js'  
const result = add(2,3); //przypisze 5
```

Za pomocą instrukcji import w module możemy zaimportować wybrane elementy z innego modułu. Jeden moduł może być importowany w wielu modułach.

# instrukcja import - budowa

```
import { add, users } from './utils.js';  
import { params } from '../settings.js';  
import { submit, validation } from '/components/form.js';  
import { data } from 'https://jakas-strona.com/users/data.js';
```

`{}` - lista elementów (wiązań) do zaimportowania - nie muszą być wszystkie

`from '/path/name.js'` - ścieżka do modułu

`./` - katalog bieżący

`../` - katalog nadrzędny

`/` - katalog główny

# instrukcja import - koniecznie pamiętaj - rozszerzenie

```
/* main.js */  
import { add } from './utils.js';
```

Pamiętaj że:

- jeśli stosujesz import w przeglądarce, to nazwa modułu musi obecnie posiadać zakończenie .js. W Node.js (webpack) rozszerzenie nie jest wymagane.

```
import { add } from './utils';
```

```
// GET http://127.0.0.1:5500/esm/js/utils net::ERR_ABORTED 404 (Not Found)
```

# instrukcja import - koniecznie pamiętaj - te same nazwy

```
/* main.js */  
import { add } from './utils.js';
```

Pamiętaj że:

- nazwa w tym wypadku musi być taka sama jak nazwa eksportowanego elementu

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

Jeśli nazwa nie będzie pokrywała się z tą eksportowaną to zobaczymy błąd np.:

```
import { addd } from './utils.js' //Uncaught SyntaxError: The requested module './utils.js' does not  
provide an export named 'addd'
```

# instrukcja import - koniecznie pamiętaj

```
/* main.js */  
import { add, users } from './utils.js';
```

Pamiętaj że:

- w module w którym importujemy (w naszym przypadku main.js) stworzą się wiązanie z add i users z modułu utils. Moduł main zarezerwuje też te dwie nazwy w przestrzeni nazw swojego modułu. "add" i "users" będą się zachowywały tak jak stałe (const), czyli nie będzie można do nich przypisać nic nowego.

```
import { add, users } from './utils.js';  
add = 10; // Uncaught TypeError: Assignment to constant variable.
```



default export



# default export - eksport domyślny

Z modułu możemy też dokonać eksportu domyślnego za pomocą instrukcji export default.

```
/* double.js */  
export default (a) => {a*2};
```

W imporcie w takim wypadku nie podajemy już nazwy importowanego elementu, a wskazujemy dowolną nazwę, bez nawiasów {}. Pod taką nazwą zostanie umieszczony element eksportowany defaultowo.

```
/* main.js */  
import dowolna_nazwa from './double.js';  
console.log(dowolna_nazwa); //(a) => { a * 2 }
```



## default export - co trzeba pamiętać - nazwa nie jest konieczna

```
/* double.js */  
export default (a) => {a*2};  
/* main.js */  
import mathAdd from './double.js';
```

- eksportowany element nie musi mieć nazwy, a nawet jeśli ma nazwę to nazwa w imporcie może być użyta dowolna ponieważ i tak zostanie do niego przypisany domyślny eksport.

default export - co trzeba pamiętać - tylko jeden default na moduł

```
/* double.js */  
const name = "coś";  
export default 100;  
export default class User {  
  constructor() { }  
  remove() { }  
};
```

- Tylko jeden element może być eksportowany domyślnie z modułu. W tym wypadku pojawi się błąd: Uncaught SyntaxError: Duplicate export of 'default'

# default export - co trzeba pamiętać

```
/* double.js */  
const name = "coś";  
class User {  
  constructor() { }  
  remove() { }  
};  
export default User;
```

- bardzo często export default umieszcza się na końcu skryptu. Nie musi on też poprzedzać deklaracji elementu, a jedynie wystarczy umieścić instrukcję export default i wskazać na eksportowany domyślnie element.

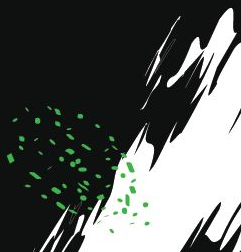
# default export - nazwy, konwencja

```
/* user.js */  
class User {  
  constructor() { }  
};  
export default User;
```

```
/* main.js */  
import User from './user.js';  
console.log(User);  
const player = new User()
```



Co jeszcze o import/export trzeba wiedzieć



## default export i export w jednym module? OK

```
/* user.js */  
class User {  
  constructor() { }  
};  
export default User;  
export const limit = 200;
```

```
/* main.js */  
import User from './user.js';  
console.log(User);  
const player = new User()
```

## Import wszystkich eksportowanych elementów jako obiekt

```
// parameters.js
```

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

```
export let users = ['Adam', 'Beti'];
```

```
export const number = 100;
```

```
export const minus = function (a, b) { return a - b };
```

```
// index.js
```

```
import * as params from './parameters.js'
```

```
const result = params.add(2, 3)
```

Import wszystkich eksportowanych elementów jako obiekt, także default

```
// parameters.js
```

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

```
export let users = ['Adam', 'Beti'];
```

```
export default add;
```

```
// index.js
```

```
import * as params from './parameters.js';
```

```
console.log(params);
```

*Nazwa dowolna  
(w przykładzie params)*

*Module*

*default: (...)*

*users: (...)*



# Zmiana nazwy podczas importu

```
/* utils.js */
```

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

```
export let users = ['Adam', 'Beti'];
```

```
export const number = 100;
```

```
/* main.js */
```

```
import { add as addFn, users as userTable, number } from './utils.js'
```

```
const result = addFn(2,3); //przypisze 5
```

# Import wartości domyślnej jednocześnie z innymi wartościami

```
/* utils.js */  
export const add = (a, b) => a + b;  
export let users = ['Adam', 'Beti'];  
export default 100;
```

```
/* main.js */  
import result, { add, users as userTable } from './utils.js'  
//result - odnosi się do wartości default - w naszym wypadku 100  
//add - funkcja  
//userTable -tablica
```



skrypt to moduł?



# skrypt vs moduł

Skrypt nie jest modułem - nie posiada jego cech, m.in. zmienne w zasięgu globalnym nie są prywatne (brak własnego zakresu), nie działa w trybie ścisłym, nie importuje/eksportuje.

```
<script src="index.js" type="module"></script>
```

```
<script src="script.js"></script>
```

# Ścieżki do zasobów - przypomnienie

# Różne przykłady ścieżek

```
// aktualny katalog (ten sam co plik html w którym znajduje się skrypt)
```

```
<script src="./js/index.js"></script>
```

```
// aktualny katalog (ten sam co plik html w którym znajduje się skrypt) ps. nie zadziała node/webpack
```

```
<script src="js/index.js"></script>
```

```
// główny katalog
```

```
<script src="/js/index.js"></script>
```

```
// katalog wyżej
```

```
<script src="../../js/index.js"></script>
```

```
//ścieżka absolutna
```

```
<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
```



# Umieszczenie modułów w projekcie

# Umieszczanie modułów i radzenie sobie z przeglądarkami

```
//SKRYPT
```

```
<script src="./index.js"></script>
```

```
//ODCZYTAJ JAKO MODUŁ - ważne, przeglądarka ignoruje znacznik script jeśli nie rozpoznaje type
```

```
<script type="module" src="./main.js"></script>
```

```
<!-- lub -->
```

```
<script type="module">
```

```
  export /***/
```

```
</script>
```

```
// NOMODULE - nie obsługuje modułów
```

```
<script nomodule src="./main-noes6.js"></script>
```



# Umieszczanie innych typów plików i bibliotek JS

## inne rodzaje danych i biblioteki

```
import style from './style.css'
```

```
import './style.css';
```

Nie da rady :) Failed to load module script: The server responded with a non-JavaScript MIME type of "text/css". Strict MIME type checking is enforced for module scripts per HTML spec.

```
import 'https://code.jquery.com/jquery-3.4.1.min.js';
```

```
$('body').css('background-color', 'red')
```

// zadziała choć nic nie jest tu importowane! Import oznacza tu tylko wczytanie kodu.

Biblioteka jQuery będzie dostępny (wykonana) w tym module

jQuery nie ma wersji ES Modules.

# Przykładowe wady

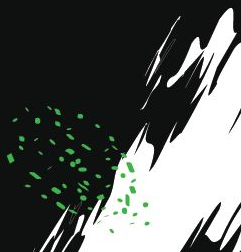
# ES Modules bezpośrednio w przeglądarce

- nie tak szybko (wiele requestów przeglądarki)
- niepełne wsparcie w przeglądarkach - trzeba wersję dla przeglądarek, które nie wspierają modułów (czyli wersję skryptową).
- w dużej części popularne biblioteki (obecnie) nie obsługują ES Modules, więc bez bundlingu ich nie użyjemy

Z tych i innych powodów Webpack i bundling są rozwiązaniem ciągle lepszym.



Zobaczmy przykład w przeglądarce





Przejdźmy do webpacka

