Java.script – dzień 1

Wzorce i Praktyki Programistyczne

Michał Jabłoński

console.log('Kilka słów o mnie');

Michał Jabłoński

/michaljabi

Front – End deweloper od 2007 r.

- 2007 – (X)HTML + CSS + JavaScript
- 2009 – Flash + HTML + AS 2.0 + PHP
- …
- 2011 – Flex + AS 3.0 + Java EE + GWT + JavaScript
- …
- 2017 – Angular + React + JS ES6 + TypeScript

[Road].map()

- Dzień 1
- A – JavaScript wprowadzenie
- B – Nowy syntaks ES6+
- C – Dobre i złe praktyki

- Dzień 2
- D - Wzorce Projektowe
- E – Testowanie
- F - Wydajność

Przykłady i narzędzia

- Będziemy korzystali z IDE
  - ItelliJ IDEA / WebStrom
  - Visual Studio Code
- Środowisko uruchomieniowe:
  - Node.js
- Dodatkowa wtyczka do sprawdzenia kodu w czasie rzeczywistym:
  - Quokka.js
- Pozostałe narzędzia do pomocy z DOM i ES6+
  - Webpack
  - Babel

Plan szkolenia
#1 Wprowadzenie

- Podsumowanie istotnych koncepcji języka
- Standard ES6 i jego wsparcie
- Zasady budowania zarządzalnego kodu
- Stosowanie konwencji kodowania
- Praktyki prowadzące do niskiego sprzężenia kodu

Podsumowanie istotnych koncepcji języka

1. Wprowadzenie

- Prototypowalność
- Funkcje jako First Class Citizens
- Zasięg zmiennych
- Kontekst wywołania
- Obiektowa natura JavaScriptu

Standard ES6 i jego wsparcie

1. Wprowadzenie

- W 2009 pojawia się „Node.js” – który zmienia sposób pisania kodu
  - Od tej pory można traktować pliki .js jako oddzielne moduły
  - Możliwe staje się też wykorzystywanie JavaScript’u – server-side.
  - Okazuje się że Node + dodatkowe narzędzia (biblioteki) może również przysłużyć się do generowania finalnego kodu dla front-endu (bundling) oraz w procesie developerskim (hot reloading dev-servers)
- Po 2015 pojawiają się „cukiereczki” tzw. Sintactic sugar
  - Nowe słowo kluczowe: class, zasięgi leksykalne zmiennych itp. itd.
  - Nowe metody w natywnych obiektach!

Standard ES6 i jego wsparcie

1. Wprowadzenie

- Standard ES6 (jak również ES7, -8 , ES-NEXT) jest już wspierany przez nowoczesne przeglądarki, jak również przez Node.js

- Aktualne postępy we wdrażaniu funkcjonalności nowego JavaScriptu można np. zobaczyć pod:

  - https://node.green/
  - https://caniuse.com/

Kompatybilność wstecz ?

A co jeśli chce pisać na Starsze Przeglądarki (albo ktoś mi kazał) ?

- To jest całe piękno rozwoju JavaScript.
- Zmienił się syntax, wprowadzają go przeglądarki – jedne szybciej inne wolniej.
- Jednak ponieważ nie zmieniono koncepcji języka, a nowe słowa kluczowe języka to tylko lukier składniowy

- Wystarczy tylko TRANSPILOWAĆ kod do poprzedniej wersji języka np., ES5

  - Inne słowo na „kompilacja” z tą różnicą, że po procesie transpilacji mamy język na podobnym poziomie abstrakcji (np. dalej edytowalny kod)

Zasady budowania zarządzalnego kodu

1. Wprowadzenie

- Zalety i wady pisania kodu w JavaScript:

Zalety	Wady
wolność !	(do)wolność ☹

- Trzeba wprowadzić zasady:
  - Ograniczenie wpływu na przestrzeń globalną „Modułowość”
    - np. idea AMD albo CommonJS
  - Podział kodu na podstawie „namespacing’u”
  - Faworyzowanie komponowania obiektów nad ich dziedziczenie

Stosowanie konwencji kodowania

1. Wprowadzenie

- Najlepiej w projekcie zastosować narzędzia typu:
  - Linter
- Dostępnych jest kilka wariantów, w zależności od wersji języka.
- Np. ESLint https://eslint.org/
- Warto polegać na sprawdzonych zasadach „Clean Code” autorstwa Roberta C. Martina
- Najważniejsze żeby zespół programistów ustalił praktyki a developerzy nawzajem pilnowali się co do ich stosowania

Stosowanie konwencji kodowania

1. Wprowadzenie

- Używanie tylko jednego rodzaju cudzysłowów dla string.
  - Preferowane ’string’ zamiast ”string” – to 2gie zostawiamy dla HTML’a
- Nazwy zmiennych pisane camelCase
- Funkcje konstruujące (klasy) pisane PascalCase

Praktyki prowadzące do niskiego sprzężenia kodu

1. Wprowadzenie

- **Problem: Asynchroniczność**
  - ograniczenie callbacków (callback hell'u)
  - używanie Promises i strumieni
- **Problem: Zmiany stanu**
  - brak mutacji danych
  - programowanie funkcyjne
- **Problem: Rozbudowana logika biznesowa**
  - podział na mniejsze części
  - „ogłupianie" komponentów
  - wyodrębnianie zależności

Plan szkolenia
#2 Praktyki programistyczne

- Poprawne wykorzystanie podstawowych elementów języka
- Funkcje i związane z nimi konstrukcje
- Tworzenie i reużywanie obiektów
- Obsługa zdarzeń
- Praca z przeglądarką
- Wykorzystanie DOM API

Poprawne wykorzystanie podstawowych elementów języka

2. Praktyki programistyczne

- Pisanie pętli warunkowych zawsze razem z nawiasami klamrowymi
- Unikanie niejawnego rzutowania (tj. stosowanie === zamiast ==)
- Używanie domknięć (closures) do osiągania prywatności zmiennych
- Używanie let i const zamiast var – unikanie hoistingu zmiennych

szczegóły i przykłady w ćwiczeniach.

Funkcje i związane z nimi konstrukcje

2. Praktyki programistyczne

- Wykorzystywanie Higher Order Functions i programowania funkcyjnego, zwłaszcza tam gdzie mamy do czynienia z tranformacją danych oraz procesami zachodzącymi „krok po kroku”
- Wykorzystanie najnowyszch (od ES6 >) możliwości:
  - Arrow functions do callbacków (ubezpieczenie kontekstu wywołania)
  - Domyślne argumenty funkcji (określanie intencji programisty)
  - Używanie rest operatora tam gdzie wcześniej „arguments”
- Wyodrębianie zależności (Smart i Dumb function, component etc.) Po przez przekazanie ich w argumentach metod

Tworzenie i reużywanie obiektów

2. Praktyki programistyczne

- Skorzystanie z lukru składniowego „class”
  - Zabezpiecza nas przed nieprawidłowym wywołaniem konstruktora bez „new”
  - Niestety nie gwarantuje prywatności zmiennych
- Stosowanie fabryk dla obiektów i domknięć w fabrykach – tam gdzie potrzebna prywatność zmiennej
- Ubezpieczenie: brak mutowania obiektów w funkcjach, które mają być PURE (brak mutacji danych wejściowych, deterministyczność: te same dane wejściowe dają odpowiednie dane wyjściowe)

Obsługa zdarzeń

2. Praktyki programistyczne

- Zdarzenia w DOM
  - Obsługiwane przez EventListenery + Ustalenie Callback'u
- Pobieranie danych z Back-end (XMLHttpRequest)
  - Najlepiej obsługiwać przez Promise API
  - Do zapytań HTTP, biblioteka: (jQuery, axios)
  - Nowe Promise API: fetch
  - https://developer.mozilla.org/pl/docs/XMLHttpRequest
- Można również potaktować zdarzenia jako „Observable" i użyć biblioteki np. RxJS

Praca z przeglądarką

2. Praktyki programistyczne

- Nowoczesny projekt oprzyjmy o środowisko Node.js
- Jeśli jest to projekt front-endowy. Wyposażmy się w biblioteki służące do bundlowania. Natomiast nasz projekt trzymajmy w oparciu o praktyki CommonJS (oddzielny plik to oddzielny moduł)
- Najczęściej jeden, zawierający całość aplikacji JavaScript – zbundlowany, plik – osadzamy na samym końcu dokumentu html, przed końcowym znacznikiem „</body>”
- Żeby uniknąć „cachowania” nowych wersji produkcyjnych wyposażmy plik bundle w sumę kontrolną.

Wykorzystanie DOM API

2. Praktyki programistyczne

- https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

- Najprostrzy sposób na osadzenie w DOM i dostęp do API Elementu:

```
const p = document.createElement('p');

// selektor do id=„root”:
$root.appendChild(p);
```

- Różnica pomiędzy DOM Node a DOM Element
  - DOM Node jest wpięty i istnieje w strukturze DOM tree
  - DOM Element może istnieć poza DOM tree – po prostu w pamięci