RAI - Laboratorium z JavaScript

Narzędzia i biblioteki

- Przydatne może byc użycie Firefox developer tools:
  `https://firefox-source-docs.mozilla.org/devtools-user/index.html`
  lub Chrome developer tools :
  `https://developer.chrome.com/docs/devtools/`
- Dokumentacja do npm-a :
  `https://docs.npmjs.com/`
- Testy z uzyciem mocha i chai:
  `https://codeburst.io/javascript-unit-testing-using-mocha-and-chai-1d97d9f18e71`
- dokumentacja mocha:
  `https://mochajs.org/#installation`
- W3C schools dokumentacja javascript:
  `https://www.w3schools.com/js/js_es6.asp`
- Dokumentacja Babel:
  `https://babeljs.io/docs/en/usage,`
- Babel presety:
  `https://www.tutorialspoint.com/babeljs/babeljs_babel_presets.htm`
- i babel CLI :
  `https://www.tutorialspoint.com/babeljs/babeljs_cli.htm`
- Dokumentacja WebPacka:
  `https://webpack.js.org/`
  lub Parcela:
  `https://parceljs.org/`

Jak skonfigurować nodejs w laboratoriach przy braku uprawnień administratora

1. należy sciągnąć archiwum zip z binarna wersja `nodejs` dla windows i rozpakować do katalogu np. `d:\bin\node`
2. otworzyć `cmd`
3. ustawić zmienną `NODEJS_HOME` na katalog z binarkami np.:
   `set NODEJS_HOME=d:\bin\node\node-v14.16.0-win-x64`
4. dodać katalog na początek ścieżki np.: `set PATH=%NODEJS_HOME%;%PATH%`
5. w ramach tego cmd nodejs powinien działać poprawnie.

Zadania:

I. Uruchamianie testów jednostkowych 10%

- Zapoznać się z dokumentacją nt. `chai` i `mocha`.
- Zapoznać się z przykładem `1-UTNode.zip` i zademonstrować uruchamianie i debugowanie testów uruchamianych w NodeJs

1. Zapoznać się z przykładem projektu, z podzialem na testy i kod oraz wykorzystaniem modułów w formacie `CommonJS`. Odtworzyć zależności i sprawdzić czy moduł się uruchamia. Ew. można jeśli to konieczne zaktualizować zalezności np. przez `npm audit`.
2. uruchomienie testu przez `npm test` lub `npm run test`
3. doprowadzić testy do zieleni
4. Żeby debugować przykład w VSCode –

    - konieczne może być dodanie konfiguracji debugowej w VSCode: `Add configuration` -> `"Nodejs mocha tests"`
    - Prawdopodobnie należy poprawić sciezkę do runnera `"program": "${workspaceFolder}/node_modules/mocha/bin/_mocha",`
    - W przypadku błędu uruchomienia należy ew. zmodyfikować konfigurację przez zmianę w pliku `lauch.json` "tdd" na "bdd"

- Zapoznać się z przykładem `2-UTNodeInBrowser.zip` i zademonstrować uruchamianie oraz debugowanie testów.
    1. Zapoznać się z przykładem projektu pozwalającego na uruchamianie testów w przeglądarce. Dla Chrome konieczna może być aktualizacja zależności np. przez `npm audit` i/lub wskazanie lokalizacji chrome przez ustawienie zmiennej `CHROME_BIN`.
    2. Należy doprowadzić testy do zieleni
    3. W katalogu `coverage` można znaleźć raport nt. pokrycia kodu testami
    4. Projekt działa w trybie wykrywania zmian - każda zmiana zapisana na dysku powinna powodować ponowne uruchomienie kodu w przeglądarce. Proszę zweryfikować to zachowanie.
    5. Oprócz obserwacji proszę pokazać jak debugować kod (przy użyciu narzędzia F12).
    6. Proszę przygotować i zademonstrować konfigurację testów tak by uruchamiane były w dodatkowej przeglądarce np. Chrome/Edge (należy skonfigurować: `karma.config` sekcja `browsers`: + instalacja odpowiedniego modułu `karma-xxx-launcher` przy pomocy `npm`. Należy zadbać by zależności trafiały do sekcji `devDependencies`.

II. Struktura projektu w JS, zalezności i narzędzia 10%

- Należy utworzyć nowy projekt w JS - zainicjować go w `npm (np. npm init).`
- W katalogu `test` umieścić plik `SimpleTest.js` a w nim:

```
var expect = require('chai').expect;

describe('canary-tests', function()
{
  it('should always pass the canary test', function()
  {
        expect(true).to.eql(true);
```

```
  });
});
```

- Proszę dodać jako zależności (lokalnie) mocha (frameworki testowy) i chai (bibliotekę do asercji). Proszę zadbać żeby zależnosci znalazły się na liście zależności developerskich w pliku konfiguracyjnym (co powinno zostać użyte: `save` czy `save-dev`?)
- Proszę skonfigurować uruchamianie testów przy pomocy nodeJs:
  W sekcji `directories` nalezy dodac `"test": "test",` a w sekcji `scripts`: wpis `"test": "mocha"`

III. Testy prototypów/domknięć. 20%

- To zadanie proszę zrealizować w formie kodu podzielonego na kod produkcyjny (domenowy) i testy jednostkowe. Kod powinien używać dyrektywy `strict`.
- Kod testowany powinien być oddzielony od testów i zawarty w oddzielnych katalogach

1. **Emulacja zmiennych prywatnych z wykorzystaniem mechanizmu domknięcia por. przykład podany na wykładzie.**
   - Proszę zdefiniować konstruktor "`Pojazd`" z atrybutami np.: `id`, `max_predkosc`, `predkosc` i interfejsem – funkcjami np.: `status`, `start(predkosc), stop()`
   - Następnie proszę napisać testy weryfikujące działanie funkcji oraz test weryfikujący czy próba zapisu lub/odczytania pól obiektu (np. `id,predkosc` itd) daje oczekiwany efekt.
2. **Przetestować mechanizm prototypów por. przykład podany na wykładzie.**
   - Proszę zdefiniować konstruktor "`Pojazd`" z atrybutami jak wcześniej, ale umieszczając funkcje w prototypie konstruktora
   - Następnie proszę napisać testy weryfikujące działanie funkcji
   - Dodatkowo proszę napisać testy weryfikujący czy "pola" obiektu są dostępne (odczyt/zapis) publicznie.
   - Sprawdzić dostępność pól `prototype/constructor`, `_prototype`.
   - Sprawdzić jak wpływa dodanie funkcji do prototypu obiektu po stworzeniu obiektu.

IV. Implementacja logiki biznesowej i testów jednostkowych. 20%

Przygotować testy i implementacje (przy użyciu klas javascript, bez baz danych itd.) prostej logiki biznesowej w bibliotece:

1. `Ksiazka` z atrybutami np. `autor`, `tytul`, `cena`, `wydawnictwo`, `slowa kluczowe` oraz `biblioteka` z `lista_wypozyczen` (i zwrotow) i interfejsem tj. funkcjami: `wypozycz`, `zwroc`, `kto_wypozyczyl`, `szukaj` wg. słów kluczowych. W miarę możliwości dla kolekcji proszę używać nowoczesnych metod (bez pętli)

V. Konstrukcje języka akceptowane przez `node.js`/przeglądarki. 20%

Proszę przygotować kod (można zrobić to w kodzie zadania IV) używający kilku (co najmniej 10 :-)) mechanizmów języka zdefiniowanych przez starsze i nowsze standardy.

Można to zrobić w ćwiczeniu IV. Pewną wskazówką mogą być `https://compat-table.github.io/compat-table/es2016plus/` lub propozycje poniżej

1. klasy (definiowane przez słowo kluczowe class)
2. uproszczona składnia "strzałkowa" dla funkcji anonimowych
3. zmienne o zasięgu bloku
4. parametry domyślne funkcji
5. przypisania dekomponujące struktury i listy
6. interpolacja napisów
7. parametry "resztkowe"
8. zachowanie this w funkcji zagnieżdzonej i =>
9. operator ?. (optional chaining)
10. replaceAll dla stringów
11. separatory w liczbach
12. prywatne akcesory i metody

Dla kilku wybranych konstrukcji znaleźć (w sensie być w stanie podać) niekompatybilną przeglądarkę lub wersję node.js.

Zainstalować Babel oraz wybrane presety

```
npm install --save-dev @babel/core @babel/cli @babel/preset-env
```

i zaprezentować wykorzystanie Babel dla wybranych wersji wykonawczych i pokazać różnice w generowanym kodzie

```
./node_modułes/.bin/babel src --out-dir src_env --presets=@babel/env
```

lub

```
npm install -g npx + npx src --out-dir src_env --presets=@babel/env
```

VI. Użycie bundlerów i narzędzi 20%

Zaprezentować użycie WebPacka lub Parcela

Zagadnienia:

1. Konfiguracja i uruchomienie bundlera
2. Dev server
3. Hot module reloading
4. Jak skonfigurować np.: Babel z wybranym bundlerem?
5. Na czym polega source mapping?