Spis treści

[**1.** **RWD** 4](#_Toc508549839)

[1.1. max-device-width ogarnąć 4](#_Toc508549840)

[**2.** **CSS** 5](#_Toc508549841)

[2.1. BEM 5](#_Toc508549842)

[2.2. Kaskadowość 5](#_Toc508549843)

[2.3. Dziedziczenie 5](#_Toc508549844)

[2.4. Specyficzność 5](#_Toc508549845)

[**3.** **Bootstrap** 5](#_Toc508549846)

[3.1. kiedy i dlaczego używać 5](#_Toc508549847)

[**4.** **SASS** 5](#_Toc508549848)

[4.1. po co używać 6](#_Toc508549849)

[4.2. loader do webpacka 6](#_Toc508549850)

[4.3. co ma ponad CSS 6](#_Toc508549851)

[4.4. zobaczyć LESS – co to i podstawowe różnice do SASS 6](#_Toc508549852)

[4.5. SCSS 6](#_Toc508549853)

[**5.** **GIT i jego funkcje** 6](#_Toc508549854)

[5.1. Mergowanie 6](#_Toc508549855)

[5.2. Branchowanie 6](#_Toc508549856)

[5.3. pull requesty 6](#_Toc508549857)

[5.4. cofanie się do commitów 6](#_Toc508549858)

[5.5. Stash 7](#_Toc508549859)

[5.6. Konflikty - skąd się biorą i jak je rozwiązywać 7](#_Toc508549860)

[5.7. poprosić Ozza 7](#_Toc508549861)

[5.8. git flow 7](#_Toc508549862)

[**6.** **Npm, a yarn** 7](#_Toc508549863)

[6.1. popytać Ozza 7](#_Toc508549864)

[**7.** **JS** 7](#_Toc508549865)

[7.1. Ify 7](#_Toc508549866)

[7.2. Pętle 8](#_Toc508549867)

[7.3. Funkcje 8](#_Toc508549868)

[7.4. Prototypy 8](#_Toc508549869)

[7.5. operatory logiczne 9](#_Toc508549870)

[7.6. undefined vs null 9](#_Toc508549871)

[7.7. Asynchroniczność 9](#_Toc508549872)

[**8.** **JQuery** 9](#_Toc508549873)

[8.1. Funkcjonalności 10](#_Toc508549874)

[8.2. dlaczego wypadł z łask 10](#_Toc508549875)

[8.3. zalety na swój czas 10](#_Toc508549876)

[**9.** **OOP** 10](#_Toc508549877)

[9.1. dlaczego obiektowość 10](#_Toc508549878)

[9.2. implementacja w react 10](#_Toc508549879)

[**10.** **Ajax** 10](#_Toc508549880)

[10.1. ajax a redux 10](#_Toc508549881)

[10.2. po co używać 10](#_Toc508549882)

[10.3. zady i walety 10](#_Toc508549883)

[**11.** **API** 10](#_Toc508549884)

[11.1. api na stronach 10](#_Toc508549885)

[11.2. api do mongo w mern 10](#_Toc508549886)

[11.3. hash użytkownika API 10](#_Toc508549887)

[**12.** **Node** 10](#_Toc508549888)

[12.1. przypomnieć sobie jak postawić serwer na czystym node 10](#_Toc508549889)

[12.2. różnice z php 11](#_Toc508549890)

[12.3. back-end – serwer i zapytania do niego 11](#_Toc508549891)

[**13.** **ReactJS** 11](#_Toc508549892)

[13.1. szablony (PuG EJS) 11](#_Toc508549893)

[13.2. struktura danych 11](#_Toc508549894)

[13.3. cykl życia komponentu 11](#_Toc508549895)

[13.4. ReactDOM 11](#_Toc508549896)

[13.5. state vs props 11](#_Toc508549897)

[13.6. dlaczego React 11](#_Toc508549898)

[13.7. podstawowe różnice z AgularJS 11](#_Toc508549899)

[**14.** **ES6** 11](#_Toc508549900)

[14.1. co wprowadza 11](#_Toc508549901)

[14.2. jaki sposób kompiluje się do es2015 (babel) 11](#_Toc508549902)

[**15.** **Webpack** 11](#_Toc508549903)

[15.1. Ozz 11](#_Toc508549904)

[15.2. Pluginy 11](#_Toc508549905)

[15.3. Loadery 11](#_Toc508549906)

[15.4. HMR 12](#_Toc508549907)

[**16.** **ExpressJS** 12](#_Toc508549908)

[16.1. Middleware 12](#_Toc508549909)

[16.2. routing – endpointy 12](#_Toc508549910)

[16.3. CRUD 12](#_Toc508549911)

[16.4. JSON w JS 12](#_Toc508549912)

[16.5. autoryzacja 12](#_Toc508549913)

[16.6. serwowanie plików 12](#_Toc508549914)

[**17.** **Inne** 12](#_Toc508549915)

[17.1. Makiety 12](#_Toc508549916)

[17.2. SCRUM 12](#_Toc508549917)

[17.3. Polling 12](#_Toc508549918)

[17.4. Streaming 12](#_Toc508549919)

[17.5. Long-polling 12](#_Toc508549920)

[17.6. Socket.io 12](#_Toc508549921)

[**18.** **Redux** 12](#_Toc508549922)

[18.1. Flux 12](#_Toc508549923)

[18.2. Cqrs 12](#_Toc508549924)

[18.3. event sourcing 13](#_Toc508549925)

[18.4. zasady korzystania 13](#_Toc508549926)

[18.5. programowanie funkcyjne 13](#_Toc508549927)

[18.6. store w mern 13](#_Toc508549928)

[18.7. React-redux 13](#_Toc508549929)

[18.8. middleware redux 13](#_Toc508549930)

[18.9. data flow 13](#_Toc508549931)

[**19.** **React-router** 13](#_Toc508549932)

[19.1. co to jest i jak działa 13](#_Toc508549933)

[**20.** **MongoDB** 13](#_Toc508549934)

[20.1. nierelacyjne bazy danych 13](#_Toc508549935)

[20.2. SQL 13](#_Toc508549936)

[20.3. Mongoose 13](#_Toc508549937)

[20.4. operacje na bazie Mongo 13](#_Toc508549938)

[**21.** **MERN** 13](#_Toc508549939)

[21.1. inne wersje np. MEAN 13](#_Toc508549940)

[21.2. kompletnie inne technologie PERL 13](#_Toc508549941)

[21.3. aplikacje izomorficzne 13](#_Toc508549942)

[21.4. server side rendering 14](#_Toc508549943)

[21.5. wirtualizacja (VBox, vagrant, docker) – Ozz 14](#_Toc508549944)

[21.6. generator kodu 14](#_Toc508549945)

[21.7. Redux-thunk 14](#_Toc508549946)

[21.8. Fetch 14](#_Toc508549947)

[21.9. przejrzeć dokumentację 14](#_Toc508549948)

[**22.** **Pozostałe** 14](#_Toc508549949)

[22.1. REST API 14](#_Toc508549950)

[22.2. normalizacja danych 14](#_Toc508549951)

[22.3. ReactDnD 14](#_Toc508549952)

[22.4. compose i connect 14](#_Toc508549953)

[22.5. obadać punkty połączenia frontu z back-endem 14](#_Toc508549954)

[**23.** **Zasady** 14](#_Toc508549955)

[23.1. DRY 14](#_Toc508549956)

[23.2. KISS 14](#_Toc508549957)

[23.3. poziomy abstrakcji 14](#_Toc508549958)

[23.4. SOLID Principles 14](#_Toc508549959)

[**24.** **Wzorce projektowe** 14](#_Toc508549960)

[**25.** **Migracje** 15](#_Toc508549961)

[**26.** **Zagadnienia serwerowe** 15](#_Toc508549962)

[26.1. deployowanie, CI/CD 15](#_Toc508549963)

[26.2. Cache’owanie (memcache, redis, cloudflare) 15](#_Toc508549964)

[26.3. Domeny (DNS, Vhosty) 15](#_Toc508549965)

1. **RWD**

<https://www.w3schools.com/css/css_rwd_intro.asp>

* 1. max-device-width ogarnąć

DEPRECATED!!! Nie używać!

Rzeczywista ilość pikseli wyświetlacza (są np. wyświetlacze, w których na jeden piksel w CSS składają się 2 piksele).

1. **CSS**

<https://www.w3schools.com/css/default.asp>

* 1. BEM

<http://getbem.com/>

Block - Standalone entity that is meaningful on its own. – naming: .block

Element - A part of a block that has no standalone meaning and is semantically tied to its block. – naming: .block\_\_elem

Modifier - A flag on a block or element. Use them to change appearance or behavior. – naming: .block--mod, .block\_\_elem--mod

* 1. Kaskadowość

<https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade>

Najpierw autor, potem użytkownik, potem przeglądarka. CSS Animations przed nimi. Przed tym wszystkim te z flagą !important user-agent, user, author.

* 1. Dziedziczenie

Properties such as color, font-family are inherited, and border, margin are not. Check if property is inherited by default when used. Can use inherit, initial and unset to set inheritance.

* 1. Specyficzność

<https://the-awwwesomes.gitbooks.io/html-css-step-by-step/content/pl/css-basics/css-specificity/index.html>

Selektory: najpierw #identyfikator, potem .klasa/pseudoklasa, potem element/pseudoelement. Im dłuższy (bardziej dokładny selektor) tym wyżej w hierarchii.

1. **Bootstrap**

<https://getbootstrap.com/>

* 1. kiedy i dlaczego używać

Używać kiedy stylowanie nie ma znaczenia. Np. dla potrzeb własnych, na potrzeby testów, dla aplikacji na użytek własny, dla prostych single page, które prawie nie mają elementów. Przyspiesza proces stylowania, ale tworzy generycznie wyglądający layout.

1. **SASS**

<https://sass-lang.com/>

* 1. po co używać

Dostępne są frameworki takie jak Bourbon, Compass, itp. To co oferuje ponad CSS (patrz punkt 4.3).

* 1. loader do webpacka

Należy dodać task runner do webpacka aby wykonywało się preprocesorowanie SASS do CSS

<https://www.jonathan-petitcolas.com/2015/05/15/howto-setup-webpack-on-es6-react-application-with-sass.html>

<https://jonathanmh.com/webpack-sass-scss-compiling-separate-file/>

* 1. co ma ponad CSS

Oferuje: zmienne, wielostopniowe zagnieżdżanie, mixiny, dziedziczenie i importowanie, użycie operatorów +, - ,\* , %, itp.

* 1. zobaczyć LESS – co to i podstawowe różnice do SASS

<https://x-coding.pl/blog/developers/sass-vs-less/>

* 1. SCSS

Posiada składnię kompatybilną z CSS (parametry w klamrach i oddzielone średnikami) co ułatwia przerobienie CSS.

1. **GIT i jego funkcje**

<https://git-scm.com/>

* 1. Mergowanie

Checkout na branch, do którego chcemy mergować

Merge brancha który chcemy mergować

Np. git checkout master

git merge hotfix

* 1. Branchowanie

git branch branch\_name

git checkout branch\_name

lub git checkout –b branch\_name

* 1. pull requesty

Podajemy z jakiego brancha na jakim repozytorium chcemy zmergować do jakiego branch na jakim repozytorium. Po code review i wprowadzeniu poprawek używamy merge pull request.

* 1. cofanie się do commitów

git log –oneline // pokazuje historię commitów z ich ID można wywołać git log branch\_name dla jednego brancha

git checkout commitID // przenosi nas do commita o danym ID

git revert HEAD~3 // reverts changes in 4th last commit and creates new commit after reverting, -n does not create new commit after reverting, branch\_name~5..branch\_name~2 zamiast HEAD cofa commity od 3 do 5 (included) na branchu branch\_name

git reset --hard commitID // cofa nas do commita o commitID i usuwa wszystkie commity, które były po nim z historii. Wystąpi problem przy próbie pusha. W takim przypadku lepiej użyć revert

* 1. Stash

git stash // usuwa niezcommitowane zmiany z working tree i umieszcza w schowku

git stash pop // usuwa to co jest w schowku i wprowadza do working tree

git stash apply // pozostawia zmiany w schowku i wprowadza do working tree

git stash list // pokazuje wszystkie dostępne istniejące stashe

git stash pop stash@{nr\_stosu} // przywraca to co jest w danym stosie i usuwa ten stos

* 1. Konflikty - skąd się biorą i jak je rozwiązywać

Przy mergowaniu jeżeli linia kodu w pliku została zmieniona już przez kogoś innego znim zmergowaliśmy naszą zmianę, lub plik został usunięty.

Należy przejść do pliku z konfliktem, rozwiązać konflikt ręcznie, usunąć markupy gita i scommitować wykonane zmiany, które rozwiązały konflikt. Można już kontynuować mergowanie. Przy usuniętym pliku należy dodać lub usunąć plik z naszego repozytorium (w zależności od decyzji jak rozwiązać konflik) i scommitoać zmianę.

* 1. poprosić Ozza

**…**

* 1. git flow

[**https://datasift.github.io/gitflow/IntroducingGitFlow.html**](https://datasift.github.io/gitflow/IntroducingGitFlow.html)

1. **Npm, a yarn**

<https://www.npmjs.com/>

<https://yarnpkg.com/lang/en/>

<https://www.keycdn.com/blog/npm-vs-yarn/>

* 1. popytać Ozza

**…**

1. **JS**

<http://devdocs.io/javascript/>

hoisting – przenosi deklaracje zmiennych do początku zakresu (scope) – jest to domyślne działanie JS

* 1. Ify

if(condition) {}

else(condition) {}

else if(condition) {}

switch(expression) {

case n: {

break;

}

default: {

statement;

}

}

Użycie takie samo jak w większości języków.

* 1. Pętle

for, while, do while

nie używać jeżeli możliwe, ale: break – wychodzi z pętli, continue – przeskakuje jedną iterację pętli, label i break label – umożliwia skok do miejsca gdzie mamy label – nie używać pod żadnym pozorem, bo wprowadza ogromny bałagan w kodzie

for(variable in object) {} – iteracja po elementach obiektu

for(variable of object {} – iteracja tylko po iterowalnych elementach obiektu

Array.forEach(function) – iteruje po elementach tablicy

* 1. Funkcje

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function>

Standardowa implementacja function name(arg1, arg2) {}

Arrow (ES6) (arg1, arg2) => {}

Mogą w JS być przypisywane do zmiennych!

Function expression – może być przypisana do zmiennej lub być anonimowa i może być wywołana od razu przy deklaracji, może być zagnieżdżona wewnątrz expression (np. obiektu) nie działa na nie hoisting

Function generator – function\* - może być wznawiana i przerywana – next przechodzi do kolejnego yield, który zwraca wartość (lub yield\* który zwraca kolejny generator), return kończy funkcję, może być użyty jako function expression, nie mogą być definiowane przy użyciu konstruktora

Funkcje mogą być deklarowane przy użyciu konstruktora – new Function

Można używać rest parameters – reprezetuje to dowolną liczbę parametrów przekazanych w tablicy

Funkcje mogą być użyte jako metody wewnątrz obiektu

* 1. Prototypy

<https://developer.mozilla.org/pl/docs/Web/JavaScript/Reference/Classes>

Deklaracje klas nie są hoistowane

Class Klasa {

constructor(args) {

this.args = args;

}

}

Static – metoda, którą możemy wywołać tylko bez tworzenia obiektu danej klasy

Aby zachować kontekst używamy deklaracji metod poza klasą poprzez Klasa.prototype.metoda = function() {}

Dziedziczenie – class KlasaDziecko extends KlasaRodzic {} – aby uruchomić konstruktor klasy rodzica musimy użyć super()

Aby dziedziczyć po elementach niekostruowalnych potrzebujemy użyć Object.setPrototypeOf(KlasaDziecko.prototype, Rodzic);

Species – pozwala na zwrócenie obiektu klasy rodzica - static get [Symbol.species]() { return KlasaRodzic; }

Super – pozwala na wywołanie metody rodzica – super.metodaRodzica();

Class templates – mix-iny – pozwala na dodawanie obiektów do mixina (rozszerzanie mixina)

Pamiętać aby tworzyć możliwie proste struktury danych! Najpierw spróbować funkcji, jeżeli potrzeba stanu, to dopiero wtedy klasę, jeżeli potrzebujemy specyficznej obsługi, która ma się wykonywać dla danego rodzaju obiektów klasy, lub w dla danego celu ma się na tych obiektach wykonywać jakieś zadanie, to możemy skorzystać z dziedziczenia, a na końcu jeżeli musimy tworzyć obiekty, które trzeba potem wedle potrzeb rozszerzać o kolejne obiekty to używamy mix-inów (nadal unikać jeżeli tylko się da!).

* 1. operatory logiczne

|| and – oba muszą być spełnione

&& or – jedno musi być spełnione

! not – zaprzeczenie (true na false i odwrotnie)

Zwracają boolean

* 1. undefined vs null

<http://www.ajaymatharu.com/javascript-difference-between-undefined-and-null/>

Undefined – oznacza że zadeklarowanej zmiennej nie przypisano jeszcze wartości

NULL – oznacza, że dla danego obiektu, przypisany jest brak posiadanej wartości

Undefined jest typem samym w sobie, null odnosi się do typu obiekt i oznacza, jest przypisany do tego obiektu tak samo jak wartość i oznacza brak wartości (pusty obiekt)

* 1. Asynchroniczność

Fajny artykuł – próbuje opowiedzieć w zrozumiały sposób o co chodzi jak sobie radzić z podstawowymi problemami

<https://www.pluralsight.com/guides/front-end-javascript/introduction-to-asynchronous-javascript>

1. **JQuery**

<https://jquery.com/>

* 1. Funkcjonalności

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.

<http://api.jquery.com/>

* 1. dlaczego wypadł z łask

Nie rozwiązuje problemów (takich jak dostęp do stworzonych elementów HTML i opakowanie ich w strukturę danych), tylko ułatwia korzystanie z JS.

* 1. zalety na swój czas

Przyspieszał i ułatwiał pracę programistów. Oferował szereg gotowych rozwiązań dla częstych implementacji JS.

1. **OOP**
   1. dlaczego obiektowość
   2. implementacja w react
2. **Ajax**
   1. ajax a redux
   2. po co używać
   3. zady i walety
3. **API**
   1. api na stronach
   2. api do mongo w mern
   3. hash użytkownika API
4. **Node**
   1. przypomnieć sobie jak postawić serwer na czystym node
   2. różnice z php
   3. back-end – serwer i zapytania do niego
5. **ReactJS**
   1. szablony (PuG EJS)
   2. struktura danych
   3. cykl życia komponentu
   4. ReactDOM
   5. state vs props
   6. dlaczego React
   7. podstawowe różnice z AgularJS
6. **ES6**
   1. co wprowadza
   2. jaki sposób kompiluje się do es2015 (babel)
7. **Webpack**
   1. Ozz
   2. Pluginy
   3. Loadery
   4. HMR
8. **ExpressJS**
   1. Middleware
   2. routing – endpointy
   3. CRUD
   4. JSON w JS
   5. autoryzacja
   6. serwowanie plików
9. **Inne**
   1. Makiety
   2. SCRUM
   3. Polling
   4. Streaming
   5. Long-polling
   6. Socket.io
10. **Redux**
    1. Flux
    2. Cqrs
    3. event sourcing
    4. zasady korzystania
    5. programowanie funkcyjne
    6. store w mern
    7. React-redux
    8. middleware redux
    9. data flow
11. **React-router**
    1. co to jest i jak działa
12. **MongoDB**
    1. nierelacyjne bazy danych
    2. SQL
    3. Mongoose
    4. operacje na bazie Mongo
13. **MERN**
    1. inne wersje np. MEAN
    2. kompletnie inne technologie PERL
    3. aplikacje izomorficzne
    4. server side rendering
    5. wirtualizacja (VBox, vagrant, docker) – Ozz
    6. generator kodu
    7. Redux-thunk
    8. Fetch
    9. przejrzeć dokumentację
14. **Pozostałe**
    1. REST API
    2. normalizacja danych
    3. ReactDnD
    4. compose i connect
    5. obadać punkty połączenia frontu z back-endem
15. **Zasady**
    1. DRY
    2. KISS
    3. poziomy abstrakcji
    4. SOLID Principles
16. **Wzorce projektowe**
17. **Migracje**
18. **Zagadnienia serwerowe**
    1. deployowanie, CI/CD
    2. Cache’owanie (memcache, redis, cloudflare)
    3. Domeny (DNS, Vhosty)