Java.script – dzień 2

Wzorce i Praktyki Programistyczne

Michał Jabłoński

[Road].map()

- Dzień 2

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

Plan szkolenia
#3 Wzorce w JavaScript

- Reużywanie kodu
- Uzyskiwanie modularnych i bezpiecznych rozwiązań
- Wybrane wzorce GOF w oparciu o praktyczne przykłady Wzorce / rozwiązania stosowane w popularnych bibliotekach

Reużywanie kodu

3. Wzorce w JavaScript

- Najprostszy sposób daje nam możliwość kreowania pomocniczych funkcji, które będą realizowały znane z programowania funkcyjnego zachowanie PURE
  - brak mutowania danych wejściowych
  - brak wewnętrznych zależności
  - dla tych samych danych wejściowych te same dane wyjściowe

Uzyskiwanie modularnych i bezpiecznych rozwiązań

3. Wzorce w JavaScript

- Z pomocą przychodzi nam głównie Node.js, które traktuje pliki .js jako oddzielne moduły, które muszą importować lub eksportować zależności.

- Z dodatkową pomocą przychodzą wzorce kreacyjne:
  - Fabryka – np. możliwość tworzenia obiektu na podstawie istniejącego API, które wymaga od nas za każdym razem podjęcia określonych kroków.
  - Budowniczy – w momencie gdy chcemy skorzystać z kilku elementów i połączyć je w całość -> np. podczas budowania struktury DOM w oparciu o JavaScript.

Wybrane wzorce GOF

3. Wzorce w JavaScript

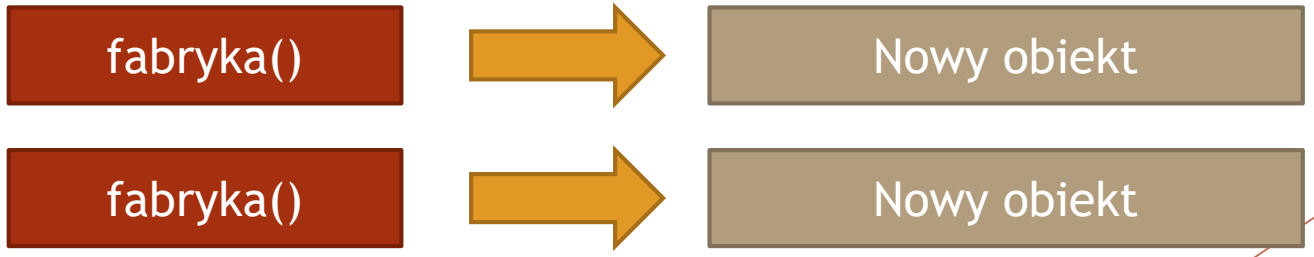
- W teorii przeniesienie dowolnego wzorca projektowego z OOP do JavaScript, nie byłoby problemem. Kwestia zastosowania pewnych uproszczeń z powodu braku interface'ów.
- W praktyce jednak. Tylko niektóre będą miały zastosowanie praktyczne i będą pomocne w dobrym przygotowaniu naszej aplikacji do działania.
- W praktyczny i specyficzny dla JavaScript sposób zrealizujemy je w ćwiczeniach.

Fabryka / Metoda wytwórcza

3. Wzorce w JavaScript

- Często w JavaScript korzystamy z jakiegoś API, albo chcemy przygotować własne metody pomagające nam w przygotowaniu komponentów Web, bądź pozwalające na utworzenie nowych obiektów w oparciu o konkretne warunki wejściowe

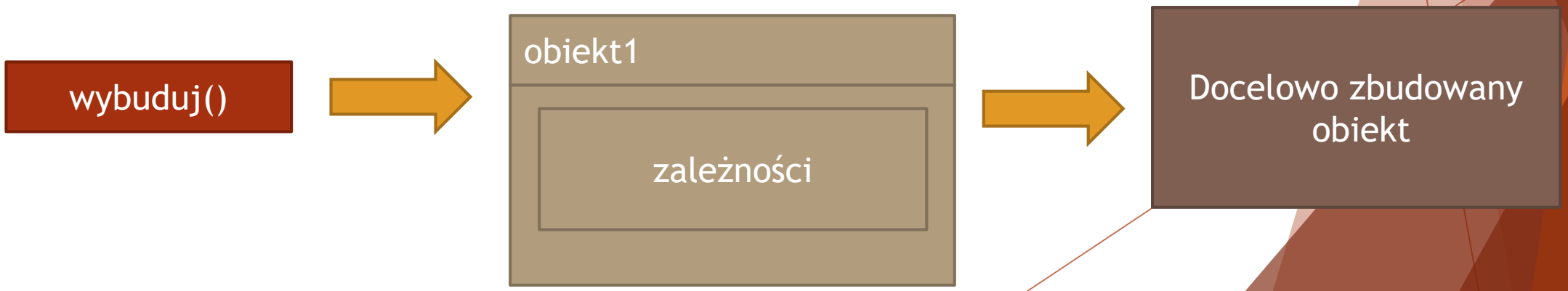
- Z pomocą przychodzi wykorzystanie Fabryki.
- Przygotowujemy ją sobie jako metodę, która zakrywa logikę odpowiedniego utworzenia i ustawienia naszego nowego obiektu.



Budowniczy

3. Wzorce w JavaScript

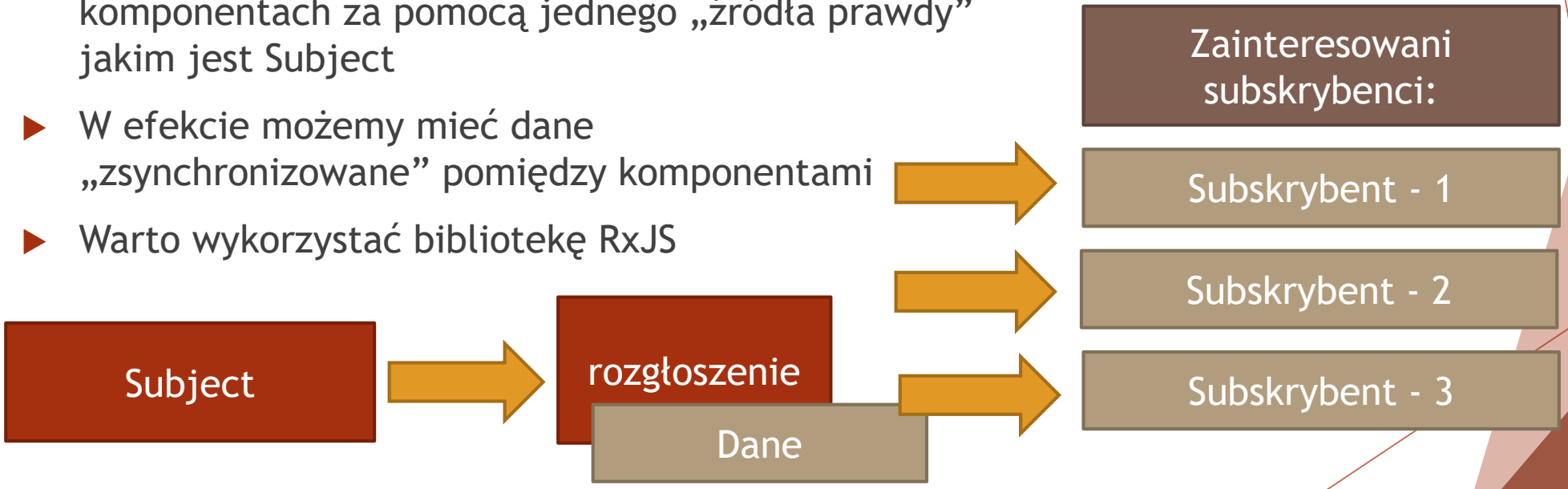
- Metoda, która „składa” dla nas już utworzone obiekty w jedną orkeśloną całość.
- Dzięki temu zakrywa część logiki odpowiedzialnej za to żeby powstała dana konstrukcja.
- Przydaje się zwłaszcza tam, gdzie mamy do czynienia z powtarzalnymi operacjami ustawiania Obiektów w określony sposób. Jednocześnie zauważamy, że za każdym razem, kiedy potrzebujemy tego określonego obiektu – musi on być złożony z innych.



Obserwator

3. Wzorce w JavaScript

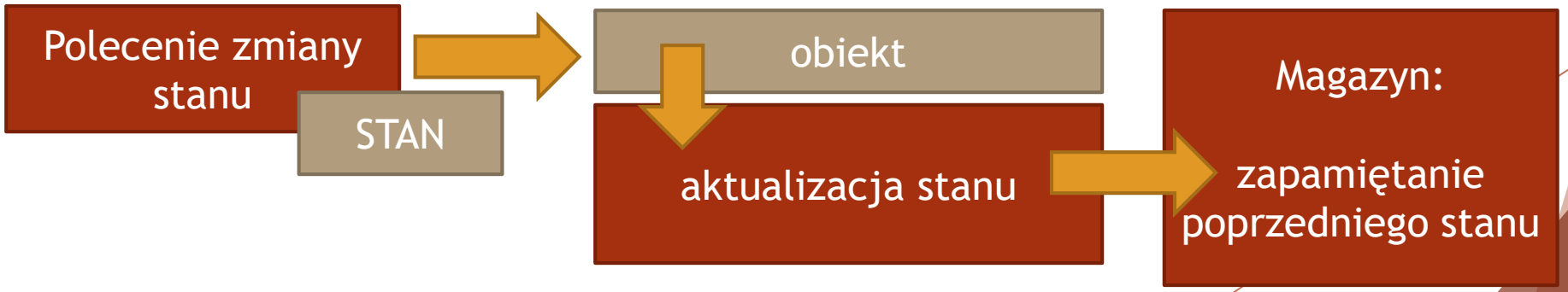
- Jedno z lepszych rozwiązań dla front-endowych aplikacji Web gdzie potrzebujemy tych samych danych w kliku miejscach
- Pozwala np. dynamicznie zaktualizować dane w kilku komponentach za pomocą jednego „źródła prawdy” jakim jest Subject
- W efekcie możemy mieć dane „zsynchronizowane” pomiędzy komponentami
- Warto wykorzystać bibliotekę RxJS



Stan + Polecenie

3. Wzorce w JavaScript

- Celowo połączone nazwy tych dwóch wzorców – aby bliżej odnieść się do biblioteki REDUX.
- Wykorzystywana np. przy aplikacjach napisanych w React. Istnieją też implementacje na inne frameworki „na czasie” np. Anglar ngrx/store
- Często zdarza się tak iż chcemy pokazać użytkownikowi końcowemu określone komponenty na naszej stronie – w zależności, od wybranych opcji, od tego czy po drodze nie wystąpił gdzieś błąd, albo dane komponentu się ładują.
- Określając i zmieniając konkretny stan obiektu, możemy mieć możliwość jego przywrócenia. Zabezpieczamy się przed mutowaniem danych w obiektach wykorzystując nowe „polecenia” zmiany stanu w obiekcie.



Dekorator

3. Wzorce w JavaScript

- Rozszerzenie funkcjonalności utworzonego wcześniej obiektu, po przez dynamiczne dodanie nowych możliwości
- W JavaScript dynamicznie rozszerzyć obiekt możemy bezproblemowo. Na przykład wykorzystując prototyp.

- Można też korzystać np. z już istniejącego API dla obiektu:
- Przykładowo dodając nowe atrybuty do wcześniej utworzonych elementów DOM.

Fasada

3. Wzorce w JavaScript

- Często zdarza się tak, że np. nie oddzielamy logiki ładowania danych w naszej aplikacji od samego widoku.
- W efekcie mogą zdarzyć się np. odwołania bezpośrednio do serwera REST po kliknięciu w przycisk

- Zamiast tego chcemy „zakryć” sposób przygotowania zapytań po dane lub inną powtarzalną logikę.
- Z pomocą przychodzi Fasadowanie, czyli przeniesienie tych funkcjonalności do innego poziomu. Komponent nie będzie wiedział skąd pochodzą dane i jak ma o nie zapytać. Wyrazi jedynie chęć ich pozyskania.
- Jako przykład, warto zauważyć to zachowanie w Serwisach frameworku Angular. Służących oddzielaniu logiki pozyskiwania danych od ich renderowania (przez komponent)

Plan szkolenia
#4 Testowanie

- Wykorzystanie obiektów typu mock i stub
- Walidowanie poprawności składni skryptu
- Popularne narzędzia wspierające testowanie

Wykorzystanie obiektów typu mock i stub

4. Testowanie

- Generalnie obiekty tego typu możemy zastosować wszędzie tam gdzie wyodrębniliśmy jakieś przekazywane np. jako argumenty metody lub konstruktora zależności.

- W JavaScript możemy łatwo „imitować” potrzebne obiekty czy funkcjonalności
- Najlepsze do testowania będą np. PURE functions – tam mamy dostęp i możemy przekazać dane wejściowe i spodziewać się odpowiednich danych wyjściowych

Walidowanie poprawności składni skryptu

4. Testowanie

- Głównie osiągalne dzięki użyciu Linerów:
  - https://eslint.org/
  - Instalacja dzięki npm:

```
npm install -g eslint

eslint --init
```

Popularne narzędzia wspierające testowanie

4. Testowanie

- Mocha, Jasmine
  - środowiska uruchomieniowe (frameworki) do testowania.
- Sinon
  - Biblioteka umożliwiająca „szpiegowanie” oraz tworzenie obiektów Mock i Stub
- Wallaby.js – działa podobnie do Quokka.js – z tym że pokazuje dodatkowo pokrycie testowe projektu

Plan szkolenia
#5 Wydajność

- Przegląd narzędzi profilujących / testujących wydajność
- Efektywne ładowanie i wykonywanie skryptów
- Praca z obiektami, zasięg zmiennych oraz kontekst wykonania
- Algorytmy i sterowanie przepływem
- DOM Scripting
- Komunikacja sieciowa
- Najlepsze praktyki

Przegląd narzędzi profilujących / testujących wydajność

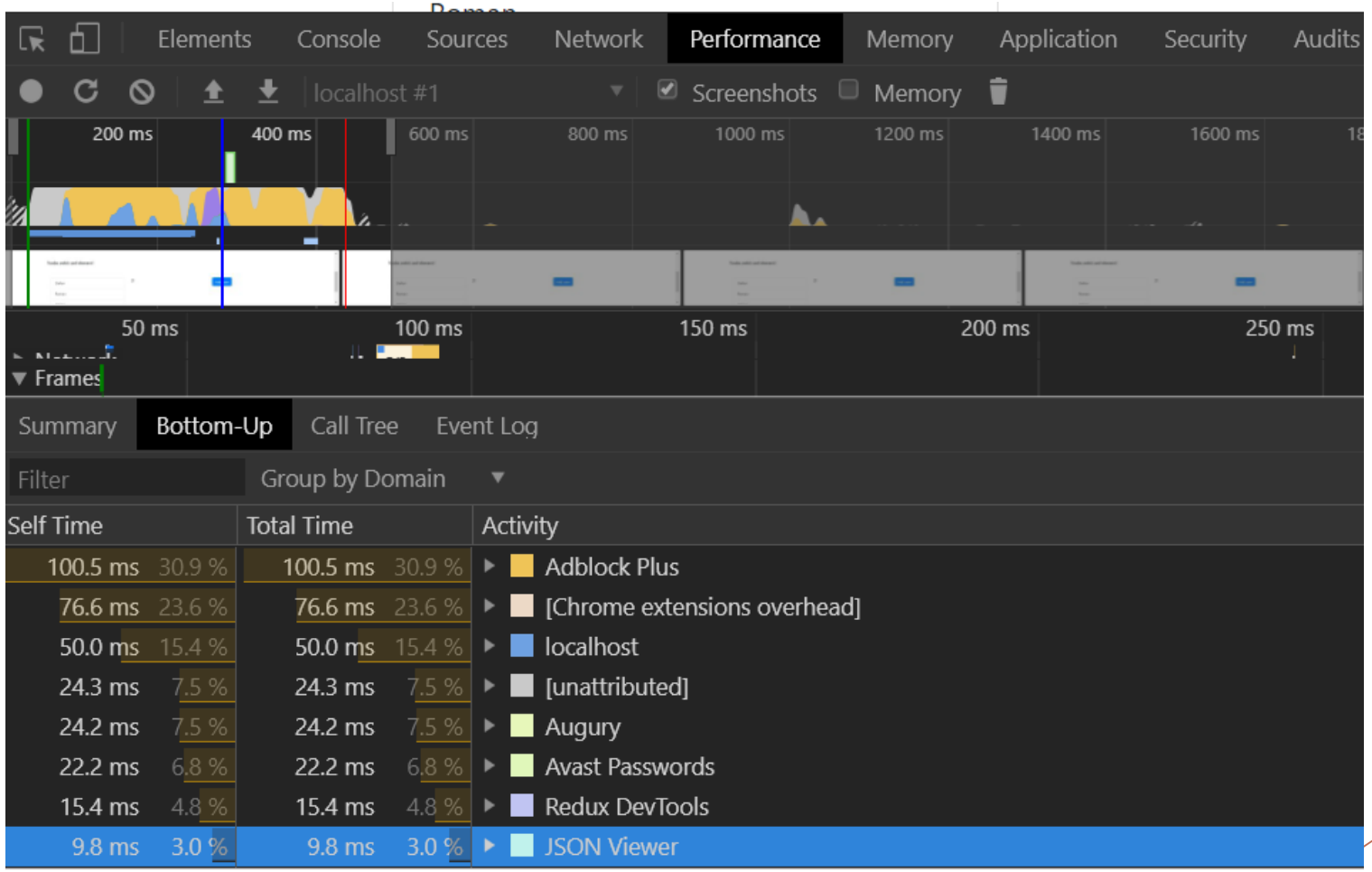
5. Wydajność

- Najprostrze testowanie wydajności opierać się może o wykorzystanie komend:
- console.time() , console.timeEnd();

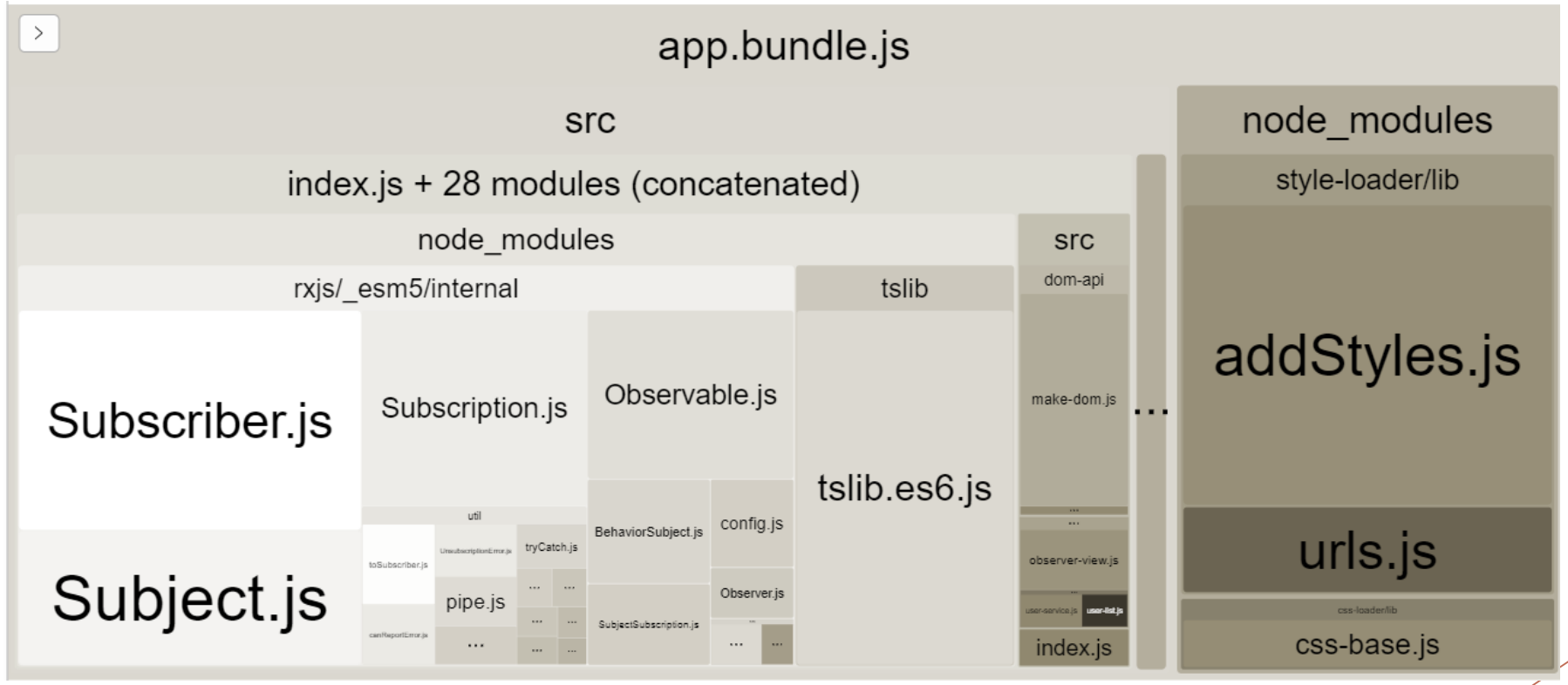
- Dobrym narzędziem obrazującym nam które elementy naszej strony są ładowane w jakim czasie, jest wbudowany w przeglądarkę Chrome – profiler, dostępny w narzędziach developerskich Chrome

- W przypadku Webpacka i wielkości skryptów w bundle – możemy „zobaczyć co się dzieje” używając np: BundleAnalyzerPlugin.

Profiler w Chrome



Webpack bundle-analizer



https://www.npmjs.com/package/webpack-bundle-analyzer

Efektywne ładowanie i wykonywanie skryptów

5. Wydajność

- Skrypty nie powinny blokować ładowania strony www. Dlatego umieszczone powinny być na końcu strony, tuż przed znacznikiem </body>
- Wykonywanie kodu JavaScript może blokować ładowanie pozostałych elementów strony.
- Najlepiej żeby strona posiadała tylko jedno odwołanie do pojedynczego pliku ze skryptem .js
- Zmniejszamy ilość przetwarzania zapytań HTTP

Algorytmy i sterowanie przepływem

5. Wydajność

- Optymalizacja i zwiększenie wydajności dzięki znalezieniu zagnieżdżeń pętli w innych pętlach etc. I uproszczeniu logiki kodu

- Używanie konstrukcji typu rekurencje, gdzie funkcja posiada wywołanie samej siebie – tylko dla niewielkiej liczby elementów
- Optymalizacją tej konstrukcji mogą okazać się generatory

DOM Scripting

5. Wydajność

- DOM jest „powolny z natury”
- Kolejne do niego odwołania i przerysowanie elementów oraz ich dołączanie za pomocą metod dostępnych w „document” jest kosztowne.
- Warto również trzymać wskaźniki do elementów których wyszukujemy za pomocą „document.querySelector” lub za pomocą bibliotek (np. jQuery)

  Tak aby (cache’ować) - nie ponawiać swoich wyszukiwani.

Komunikacja sieciowa, najlepsze praktyki

5. Wydajność

- Najbardziej optymalnym rozwiązaniem w naszym projekcie może okazać się rozwiązanie typu: Single Page Aplication
- Wtedy nie musimy „doładowywać” skryptów (.js) wykorzystując zapytania XMLHttpRequest. Przygotowujemy całość widoków i naszego „front-endu” do jednorazowego ściągniecia, po pierwszym odwołaniu do strony
- Przeglądarka cache’uje skrypty – warto jest posiadać mechanizm „sumy kontrolnej” np. z wykorzystaniem budowania w Webpacku. Wtedy tylko każde nowe wydanie wersji naszego projektu będzie „odświeżane” dla powracającego użytkownika.
- Warto w projekcie wykorzystać aktualnie cieszące się uznaniem frameworki, które bardzo optymalizują naszego JavaScript’a zwłaszcza w kontekście odniesienia do DOM. Przykładowo React korzysta z VirtualDOM, najpierw ustala co ma się „odświeżyć” i zaktualizować na DOM – optymalizuje ilość „przerysowań”