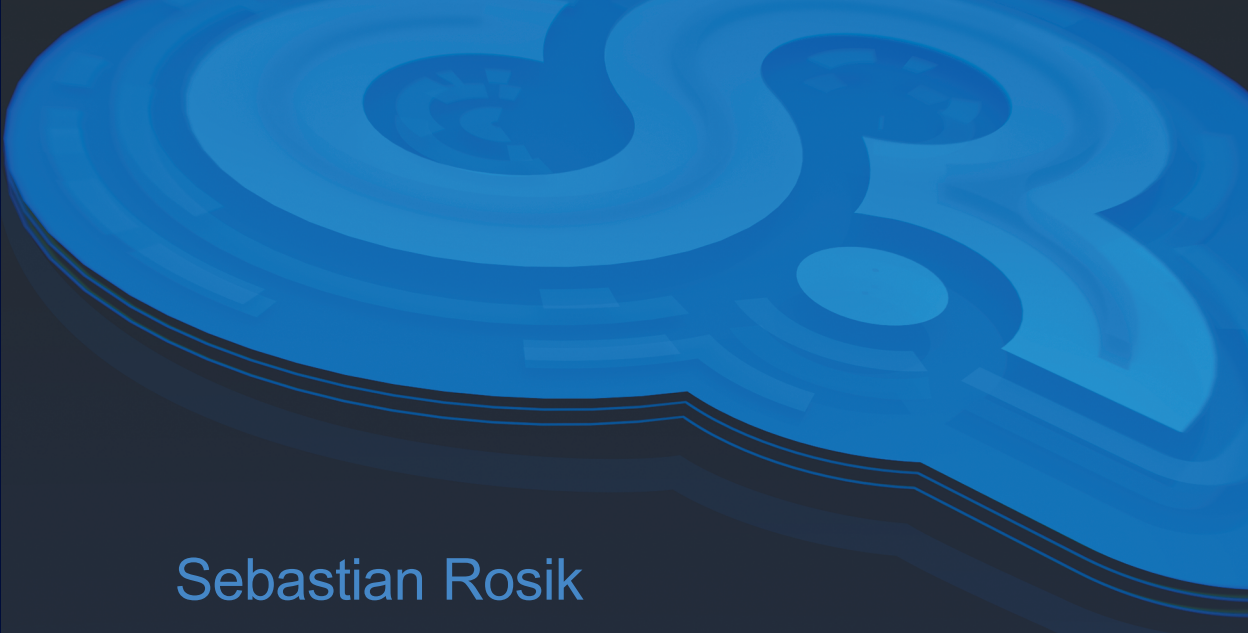
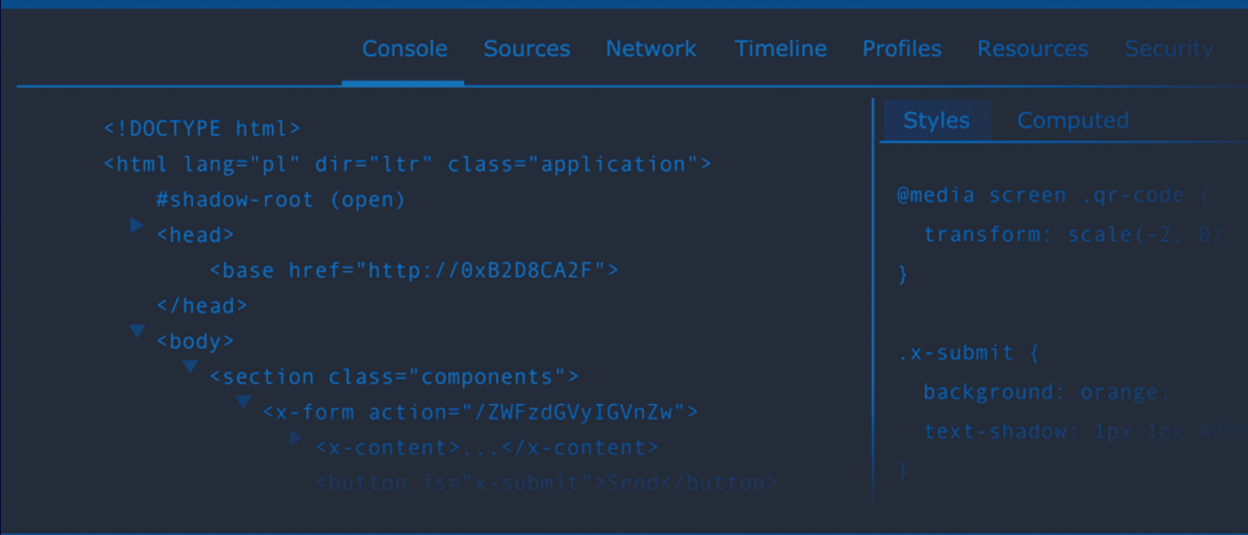


Sebastian Rosik

# HTML5
## KOMPONENTY




PWN

Sebastian Rosik

# HTML5
## KOMPONENTY

PWN

Projekt okładki **Sebastian Rosik**

Wydawca **Łukasz Łopuszański**

Redaktor prowadzący **Jolanta Kowalczuk**

Koordynator produkcji **Anna Bączkowska**

Skład i łamanie **Dariusz Ziach**



ISBN 978-83-01-18369-1

Wydanie I
Warszawa 2016

Wydawnictwo Naukowe PWN SA
02-460 Warszawa, ul. Gottlieba Daimlera 2
tel. 22 69 54 321, faks 22 69 54 288
infolinia 801 33 33 88
e-mail: pwn@pwn.com.pl
www.pwn.pl

Informacje w sprawie współpracy reklamowej: reklama@pwn.pl

Druk i oprawa: OSDW Azymut Sp. z o.o.

# Spis treści

# Wstęp

Przeglądarki internetowe są jednym z tych środowisk technologicznych, które zmieniają się najdynamiczniej. Sam jeszcze pamiętam, gdy wiele lat temu po raz pierwszy odkryłem, co to jest kod HTML i w jaki sposób tworzy się proste strony internetowe. Szczytem technologii przeglądarkowej były wtedy znaczniki MARQUEE oraz zegarki podążające za kursorem myszki (był to najprostszy sposób na zirytowanie użytkownika). Był to świat, w którym właśnie trwała wojna przeglądarek walczących o dominację na ryku. Skutkiem tej wojny był ich szybki rozwój. Z każdą kolejną wersją przeglądarki prezentowały coraz to większe możliwości. Od tamtej pory wiele się zmieniło – część technologii została już dawno zapomniana (np. DHTML, XHTML), wiele przeglądarek z pozycji lidera zeszło na drugi plan, a inne – przeglądarki niszowe zdominowały rynek. Jedno jednak się nie zmieniło – szybki rozwój technologii. My również, tak jak przeglądarki internetowe, aby przetrwać na tym rynku, musimy stale się rozwijać, uczyć się kolejnych bibliotek, szkieletów aplikacji, najnowszych API przeglądarek oraz technik.

Organizacje standaryzujące nieustannie tworzą nowe standardy lub zmieniają istniejące. Ich celem jest stawienie czoła problemom sieci jutra – nowym urządzeniom, nowym formom korzystania z sieci, konsumpcji oraz wytwarzania treści. Po powstaniu czy aktualizacji standardu twórcy przeglądarek implementują zmiany, po czym developerzy, tacy jak my, zapoznają się z daną technologią i puszczają ją w obieg, wykorzystując ją w swoich projektach.

By wyjść naprzeciw nadchodzącym zmianom, w książce tej poruszyłem takie tematy, jak np. ECMAScript 6 czy Web Components. Niektóre z tych technologii są tylko częściowo zaimplementowane w wybranych przeglądarkach. Niektóre z kolei będą zaimplementowane w najbliższej przyszłości. Jedno jest pewne – z czasem wszystkie przeglądarki będą je obsługiwać, a to znaczy, że już teraz powinniśmy nauczyć się je wykorzystywać, by być przygotowanym na jutro.

Fakt, iż przeglądarki potrzebują czasu, by dogonić standardy, nie oznacza wcale, że już dzisiaj nie możemy pracować z nowymi technologiami. W kolejnych rozdziałach opisałem na przykład, w jaki sposób możemy korzystać z nowych możliwości języka JavaScript w przeglądarkach, które jeszcze ich nie obsługują. Dowiemy się też, czym są Web Components oraz jakich narzędzi używać, by ułatwić sobie pracę.

Przyjmijmy na chwilę, że tworzymy aplikację, do której musimy wstawić kilka powiązanych ze sobą kontrolek. Każdy z nas na pewno wykonał kiedyś podobną listę kroków:

- Wybranie jednego z wielu dostępnych szkieletów (*frameworks*) aplikacji na rynku (lub bibliotek), które zawierają kontrolki nas interesujące.
- Pobieżne zapoznanie się z dokumentacją projektu, by dowiedzieć się, jakie zależności należy pobrać i w jaki sposób funkcjonuje API danego szkieletu aplikacji.
- Załączenie jednego lub więcej skryptów JavaScript oraz arkuszy stylów.
- Przekopiowanie ze strony projektu kodu inicjującego kontrolkę lub napisanie własnego kodu inicjującego kontrolki po zapoznaniu się z dokumentacją (jeżeli jesteśmy bardziej ambitni).
- W przypadku kłopotów kierujemy się na `stackoverflow.com` w celu znalezienia odpowiedzi na pytanie, czemu kontrolka informuje o błędzie JavaScript w konsoli narzędzi programistycznych przeglądarki.

Po kilku wieczorach może nam się udało opanować podstawy danego szkieletu aplikacji. Czy rozumiemy jednak, w jaki sposób te kontrolki działają? W jaki sposób są ze sobą połączone, czy się ze sobą komunikują? Jaka koncepcja za nimi stoi? Jakich wzorów projektowych się trzymać, a jakich unikać?

W niedalekiej przyszłości okazałoby się jednak, że nowy projekt, w którym bierzemy udział, korzysta z alternatywnego szkieletu. Co wtedy? Zacisnąć zęby i zaparzyć kolejną kawę?

Tworzenie warstwy prezentacji aplikacji nie polega na przekopiowaniu gotowych skryptów, lecz na tworzeniu interfejsu w sposób przemyślany i świadomy. Tylko wiedząc, jakie są podstawy technologii, zgodnie z którą działa nasz interfejs, możemy stworzyć interfejs użytkownika, który będzie ergonomiczny i wydajny. Znając tylko jeden szkielet aplikacji lub bibliotekę, ograniczamy swoje możliwości w sytuacji, w której nie będziemy mogli użyć właśnie jego (np. dołączymy do pracy nad projektem, w którym jest wykorzystywane inne rozwiązanie). W takiej sytuacji dogłębna wiedza z zakresu HTML5, CSS3, Web Components czy ECMAScript 6 okaże się niezbędna. Dzięki niej będziemy potrafili szybciej zapoznać się z działaniem alternatywnej biblioteki czy szkieletu aplikacji.

Web Components dają olbrzymie możliwości i w tym kierunku zmierzają zmiany środowiska przeglądarek. Web Components nie są gotową receptą na wszystkie bolączki, ale na pewno wprowadzają pewien porządek do procesu wytwarzania warstwy prezentacji naszych aplikacji.

Wiele koncepcji Web Components jest także widocznych w różnych szkieletach aplikacji, które pozwalają tworzyć komponenty interfejsu użytkownika, mimo że same nie są oparte na Web Components. Web Components można użyć jako bazy dla komponentów, które mogą współpracować z kilkoma różnymi szkieletami aplikacji.

Przed napisaniem książki myślałem, że to robota dla jednego. Miałem w głowie taki obraz pisarza co to odcina się od świata, zamyka w małym domku i pisze książkę przy kubku gorącej kawy, bez przerwy przez miesiąc czy dwa. W rzeczywistości wygląda to zupełnie inaczej. Otaczający nas świat i codzienne sprawy złośliwie nie pozwalają na pisanie książki w sposób, jakiego bym sobie życzył. Czas do oddania książki wydawcy mija nieubłaganie

szybko i kolejne ustalone terminy zostają przekroczone. Podziękowania zatem należą się Łukaszowi Łopuszańskiemu, który wykazał się niemałą cierpliwością w oczekiwaniu na moje dzieło. Dziękuję także mojej żonie, Agnieszce Zerce-Rosik, która poprawiała po mnie tysiące literówek i zamieniała niezrozumiały strumień myśli w spójne i klarowne zdania. Dziękuję jej także, że pozwalała mi tyle czasu spędzać nad kolejnymi stronami książki, wykazując się przy tym jeszcze bardzo dużym zrozumieniem dla udręki męża. Duże podziękowania dla Gynvaela Coldwinda, który był moją inspiracją. Dziękuję także Mikołajowi Koprasowi za przebrnięcie przez kilka rozdziałów i wyłapanie wielu błędów. Dziękuję również Karolinie Kowalskiej, która skutecznie przetestowała dużą część książki i wsparła mnie w wyłapywaniu błędów językowych.

*Sebastian Rosik*

Wrocław, marzec 2016 r.

# Informacje techniczne

## Serwer lokalny

W książce tej znajduje się wiele przykładów, których uruchomienie bezpośrednio z dysku (gdzie protokół to `file://`) może spowodować niepoprawne działanie części przykładu ze względu na ograniczenia w dostępie do plików z dysku w przeglądarkach internetowych. Podczas pracy z książką zalecam zatem korzystanie z lokalnego serwera HTTP.

Przy bardziej skomplikowanych zadaniach jest podana informacja o sposobie uruchomienia lokalnego serwera do prac deweloperskich.

## Komentarze

W przykładach większe komentarze do kodu zostały przeniesione poza kod (by zwiększyć czytelność). W samym kodzie natomiast znajdują się referencje do odpowiedniego komentarza, np. w postaci `/* <1> */`. Sposób komentowania kodu zależy od języka, dlatego dla HTML jest to `<!-- 1 -->`, natomiast dla CSS `/* <1> */`.

Przykładowy kod z komentarzem:

```
:host {  /* <3> */
  background: black;
  display: block;
  user-select: none;
}
```

<3> Selektor `:host` umożliwi nam ostylowanie elementu, który zawiera treść znajdującą się w cieniu. Oznacza to, że w tym miejscu jesteśmy w stanie „wyjrzeć” poza cień i nadać domyślny styl naszemu elementowi `x-player`.

## Praca z kodem

Podczas tworzenia przykładów może się zdarzyć, że będziemy wracać do napisanych wcześniej funkcji, by dopisywać do nich kolejne kawałki kodu.

Przykładowo, na początku stworzymy funkcję, która jedynie będzie deklarować odpowiednie zmienne:

```
function foo() {
  var x = 1;
  var y = 2;
}
```

Natomiast w dalszej części rozdziału dopiszemy do niej pozostały kod. W listingu będzie to wyglądać jak funkcja o tej samej nazwie, lecz jej zawartość będzie się różnić. Komentarz z wielokropkiem `// ...` będzie oznaczać miejsce, w którym znajduje się kod wcześniej przez nas napisany.

```
function foo() {
  // ...
  return x + y;
}
```

## Odniesienia do elementów

W książce tej w wielu miejscach odnoszę się do elementów w drzewie DOM. Czasami będę pisał ogólnie o elementach, np. o elementach `button`, lub też będę pisał o konkretnym elemencie, np. `input[type="submit"]`. Zawsze jednak będę się do tych elementów odnosił w postaci selektora CSS3[1].

## Wersje oprogramowania

Oprogramowanie, z którego najczęściej korzystamy, jest wciąż rozwijane przez producentów. Co kilka miesięcy pojawiają się większe lub mniejsze zmiany w obsłudze standardów w każdej z przeglądarek, dlatego nie mogę zagwarantować, że przykłady podane w książce będą na pewno działać bez żadnych zmian w najnowszych wersjach przeglądarek, po kilku latach od wydania książki. Mogę natomiast podać informację, w jakich wersjach podane przykłady na pewno działają.

[1] Selektory CSS: http://www.w3.org/TR/css3-selectors/

Wersje przeglądarek internetowych:

- Google Chrome 45,
- Mozilla Firefox 41,
- Opera 32,
- Microsoft Internet Explorer 11,
- Microsoft Edge 25,

inne:

- Node.js 4.0.

## Oficjalna witryna World Wide Web

Dla książki powstał dedykowany serwis dostępny pod adresem:

`http://sebastianrosik.pl/html5-komponenty`

W serwisie tym znajdują się między innymi:

- Materiały do pobrania – będą przydatne podczas pracy nad przykładami zaprezentowanymi w książce.
- Aktualizacje informacji zawartych w książce. Sieć jest medium niezwykle dynamicznym i podlegającym zmianom, tak jak i technologie, które za nią stoją. Książki już wydrukowanej nie można niestety zaktualizować, dopiero w jej nowym wydaniu – mam nadzieję, że nastąpi to w odpowiednim momencie.
- Errata – żadna książka nie jest niestety pozbawiona błędów. Największym błędem byłoby więc nieudostępnienie poprawek w formie erraty.

Rozdział 1.

# Przydatne narzędzia

Do tworzenia oprogramowania w HTML5 nie potrzeba specjalistycznych narzędzi, wystarczy podstawowy edytor tekstu oraz przeglądarka, w której możemy podejrzeć nasz projekt. Jest to, co prawda, zestaw absolutnie minimalny do rozpoczęcia pracy nad prostym projektem, jak np. statyczna strona HTML, może się więc okazać niewystarczający do bardziej zaawansowanych rzeczy. Tworząc coraz to bardziej skomplikowane projekty szybko znajdziemy się w sytuacji, w której będziemy potrzebować nowych narzędzi wspomagających pracę.

Podczas tworzenia skryptów JavaScript pomocne będzie narzędzie, które pozwoli nam śledzić działanie skryptu i analizować jego poszczególne kroki, co ułatwi wykrycie oraz naprawę kryjących się tam błędów.

W przypadku tworzenia arkuszy stylów zdarza się, że deklarację atrybutów musimy powtarzać w różnych miejscach arkusza, np. kopiując wartość koloru, by użyć go dla tekstu i obramowania elementu. Użycie narzędzia wprowadzającego zmienne do arkuszy stylów na pewno ułatwiłoby pracę.

Podczas codziennej pracy nad projektem wprowadzenie narzędzia do automatyzacji powtarzalnych czynności oszczędzi nam czas. Za każdym razem, gdy dokonamy zmiany w plikach `*.less`, musimy samodzielnie wykonać polecenie, które skompiluje pliki `*.less` do `*.css`, odpowiednie narzędzie natomiast mogłoby obserwować dokonywane zmiany na plikach i samo wykonywać odpowiednie polecenia.

## 1.1. Google Chrome DevTools

Obecnie dostępne na rynku przeglądarki mają wbudowane narzędzia, dzięki którym możemy w znaczący sposób przyspieszyć pracę nad projektem, ponieważ umożliwiają podgląd działania kodu w najdrobniejszych szczegółach. Narzędzia te pozwalają nam na debugowanie kodu oraz jego analizę pod względem wydajności. Prawie każda przeglądarka ma wbudowane narzędzia programistyczne, które trochę się różnią. Nie jestem niestety w stanie dogłębnie przedstawić w książce wszystkich narzędzi w dostępnych na rynku przeglądarkach, dlatego też skupimy się na pracy jedynie z Google Chrome DevTools. Nie oznacza to jednak, że powinniśmy tylko i wyłącznie pracować z tym narzędziem. Dla każdego programisty wytwarzającego oprogramowanie przeznaczone dla różnych przeglądarek

powinno być naturalnym przełączanie się między nimi i testowanie swojego oprogramowania na bieżąco, co oczywiście wiąże się z posiadaniem podstawowych umiejętności pracy z narzędziami oferowanymi przez każdą z nich.

Google Chrome DevTools są wbudowane w przeglądarkę; możemy je uruchomić dla aktualnej strony, używając klawisza F12 lub skrótu klawiszowego CTRL+SHIFT+I. Narzędzia deweloperskie przeglądarki Google Chrome oferują wiele funkcjonalności przydatnych zarówno przy tworzeniu oprogramowania, jak i rozwiązywaniu problemów. Nie sposób w jednej książce omówić je wszystkie w najdrobniejszych szczegółach, dlatego też skupimy się na przeglądzie najczęściej używanych paneli.

## Elementy (*elements*)

Panel elementów pozwala na podgląd aktualnego stanu drzewa DOM[2]; możemy w nim też zaznaczyć dowolny element tego drzewa, a następnie dokonać modyfikacji jego atrybutów. Zaznaczony element możemy także edytować jako zwykły kod HTML wraz z jego potomkami. Między elementami można nawigować na dwa sposoby:

- Przez wyszukiwanie elementów w drzewie DOM. Wyszukiwanie odbywa się przez podanie odpowiedniego selektora, który będzie pasował do szukanego przez nas elementu. Po użyciu kombinacji klawiszy CTRL + F w panelu elementów ukaże się pole wyszukiwania, w którym będziemy mogli podać selektor. Pasujący element zostanie natychmiast podświetlony.
- Dzięki użyciu klawiszy ze strzałkami góra i dół do przemieszczania się między elementami sąsiadującymi oraz lewo i prawo do przemieszczania się między rodzicami i ich potomkami.

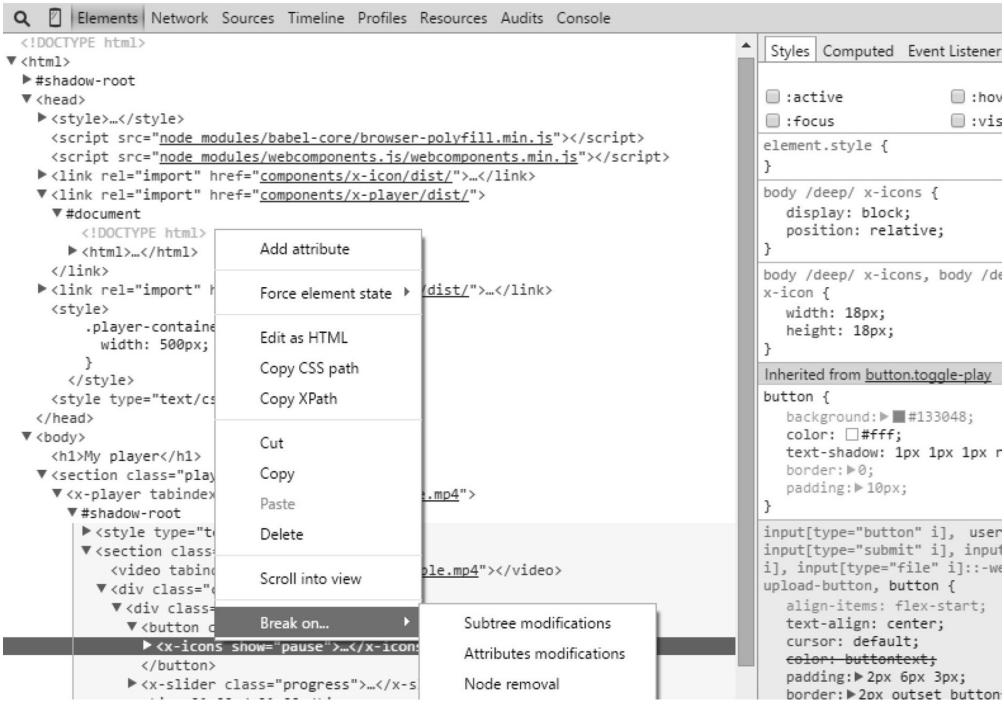
W kwestii modyfikacji elementów w drzewie DOM z poziomu panelu mamy do wyboru kilka opcji z poziomu menu kontekstowego. W zależności od tego, w którą część zaznaczonego elementu klikniemy prawym przyciskiem myszy, dostaniemy różne opcje.

```
<div class="myClass">treść</div>
```

Kliknięcie w nazwę elementu (np. `div`) pozwoli nam między innymi dodać nowy atrybut, natomiast kliknięcie prawym przyciskiem myszy w istniejący atrybut da nam opcje edycji atrybutu.

Podczas edycji atrybutu jest on zaprezentowany w postaci pola tekstowego, gdzie jego nazwa i wartość mogą zostać zedytowane (np. `class="myClass"`). Wszelkie zmiany, jakich dokonamy podczas edycji atrybutu, potwierdzamy klawiszem Enter. Usunięcie samej wartości atrybutu (np. `"myClass"`) sprawi, że atrybut nadal będzie istniał na elemencie, ale bez przypisanej wartości, natomiast jeżeli usuniemy tylko nazwę atrybutu (np. `class`) lub wykasujemy całą zawartość pola tekstowego podczas edycji, to atrybut zostanie całkowicie usunięty z zaznaczonego elementu.

[2] *Document Object Model* (Obiektowy Model Dokumentu): http://www.w3.org/DOM/



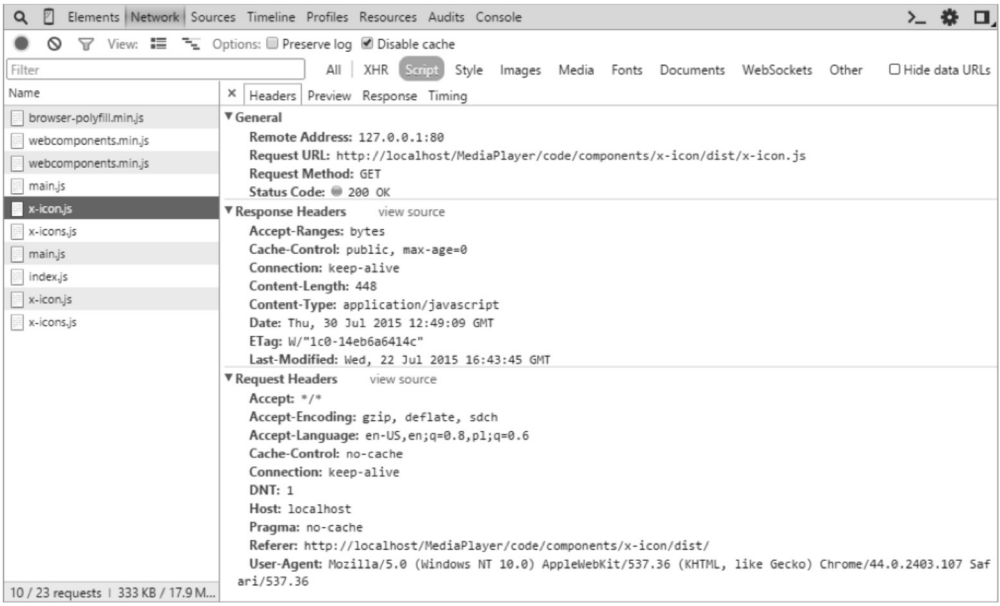

**Rysunek 1.1.** Panel elementów drzewa DOM. Dostęp do niego i innych paneli znajduje się w górnej części narzędzi programistycznych

## Ruch sieciowy (*network*)

Panel ten pozwala śledzić ruch sieciowy w danym dokumencie. Dzięki niemu jesteśmy w stanie ustalić:

- Jakie zapytania zostały wykonane?
- Jaki jest ich stan?
- Czy zakończyły się sukcesem?
- Ile czasu zajął każdy z etapów życia danego zapytania?

Dodatkowo dzięki wsparciu dla WebSockets jesteśmy w stanie na bieżąco podglądać informacje, jakie są wymieniane z serwerem w danym połączeniu WebSocket.




**Rysunek 1.2.** Panel podglądu ruchu sieciowego

## Źródła *(sources)*

Panel źródeł oprócz wyświetlania zawartości plików JavaScript znajdujących się w pamięci pozwala na wstrzymanie wątku maszyny wirtualnej JavaScript oraz na podgląd jej stanu. Jest to niezwykle przydatne podczas debugowania. Debugerem możemy sterować interfejsem graficznym, a także skrótami klawiszowymi:

- F8 – wstrzymanie lub kontynuowanie wątku;
- F10 – przejście od aktualnie wykonywanej instrukcji do następnej; pozwala na podgląd działania skryptu linia po linii;
- F11 – wejście w głąb aktualnie wykonywanej funkcji;
- SHIFT + F11 – wyjście z aktualnie wykonywanej funkcji.

Oczywiście debugowanie polegające na czekaniu na odpowiedni moment, aby nacisnąć klawisz F8, byłoby dość frustrujące, dlatego też warto wiedzieć, że DevTools w panelu źródeł oferują też funkcjonalność punktów wstrzymania (*break points*).

Przez kliknięcie w dany numer linii w pliku źródłowym jesteśmy w stanie ustawić na niej punkt wstrzymania. Wykonywany skrypt zostanie wstrzymany, gdy tylko maszyna JavaScript przeglądarki spróbuje wykonać instrukcje na danej linii.

```
15
16 function askForAge() {
17   return prompt("Your age?");
18 }
19
20 var myAge = askForAge();
21
22 document.write("Age: " + myAge);
23
```

**Rysunek 1.3.** Punkt wstrzymania w kodzie źródłowym na linii 22

Możemy także dodać warunkowy punkt wstrzymania na danej linii, klikając na jej numer prawym klawiszem myszy i z menu kontekstowego wybierając pozycję: „Dodaj warunkowy punkt wstrzymania” (*Add conditional breakpoint*). Wpisane tam wyrażenie, jeżeli tylko zwróci wartość `Boolean` równą `true`, wstrzyma wykonywany skrypt w danym miejscu.

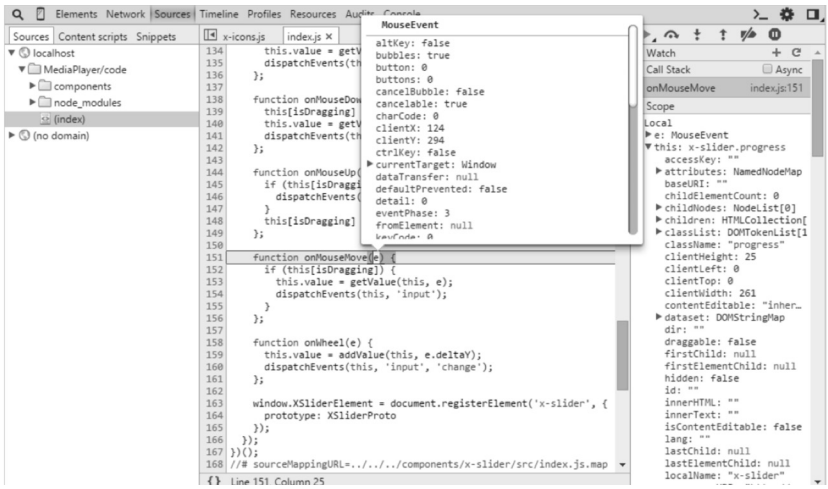
```
16 function askForAge() {
17   return prompt("Your age?");
18 }
19
20 var myAge = askForAge();
21
22 document.write("Age: " + myAge);
```

The breakpoint on line 22 will stop only if this expression is true:

```
myAge === "0"
```

**Rysunek 1.4.** Dodawanie warunkowego punktu wstrzymania w kodzie źródłowym

Gdy wątek JavaScript jest wstrzymany, możemy podejrzeć wartości każdej zmiennej albo zobaczyć źródło każdej funkcji przez najechanie myszą na interesującą nas zmienną i odczekanie ułamka sekundy. Ukaże nam się panel zawierający dane, które się kryją pod danym obiektem w chwili, w której wątek został wstrzymany.




**Rysunek 1.5.** Panel źródeł

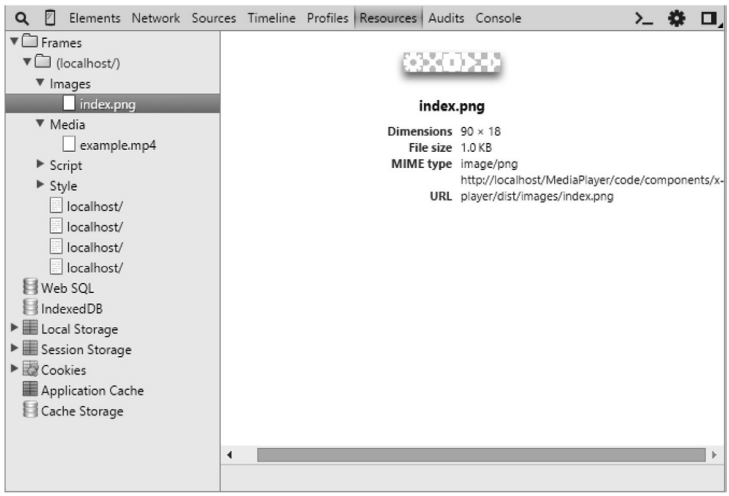
Inną ciekawą funkcjonalnością DevTools jest obsługiwanie wyrażenia `debugger`, które możemy umieścić bezpośrednio w kodzie. Jeżeli moduł debugera przeglądarki natrafi w kodzie na to wyrażenie, to wątek zostanie wstrzymany automatycznie. Punkty wstrzymania możemy zatem ustawiać bezpośrednio w kodzie. Oczywiście wyrażenie `debugger` powinno być używane w kodzie tylko i wyłącznie podczas debugowania i nie powinno ono się w nim znajdować po zakończeniu prac, gdyż inne przeglądarki, które mogą nie obsługiwać tego wyrażenia, poinformują o wyjątku – co może doprowadzić do błędnego działania naszej aplikacji.

Warunkowy punkt wstrzymania z użyciem wyrażenia `debugger` z rysunku 1.4 wygląda następująco:

```
function askForAge() {
  return prompt("Your age?");
}
var myAge = askForAge();
if (myAge === "0") {
  debugger;
}
document.write("Age: " + myAge);
```

## Zasoby (*resources*)

Panel zasobów oferuje pogląd wszystkich plików wczytanych do danego dokumentu. Możemy się w nim zapoznać z informacjami dotyczącymi każdego z plików: ich adresu, źródła, wymiarów, typów itd. Znajdziemy tam informacje zarówno na temat plików tekstowych, jak i multimedialnych.




**Rysunek 1.6.** Panel zasobów