



UNIWERSYTET
JAGIELŁOŃSKI
W KRAKOWIE

Zaawansowane Techniki WWW (HTML, CSS i JavaScript)

Dr inż. Marcin Zieliński

Środa 15:30 - 17:00 sala: A-1-04

WYKŁAD 1

Wykład dla kierunku: Informatyka Stosowana II rok

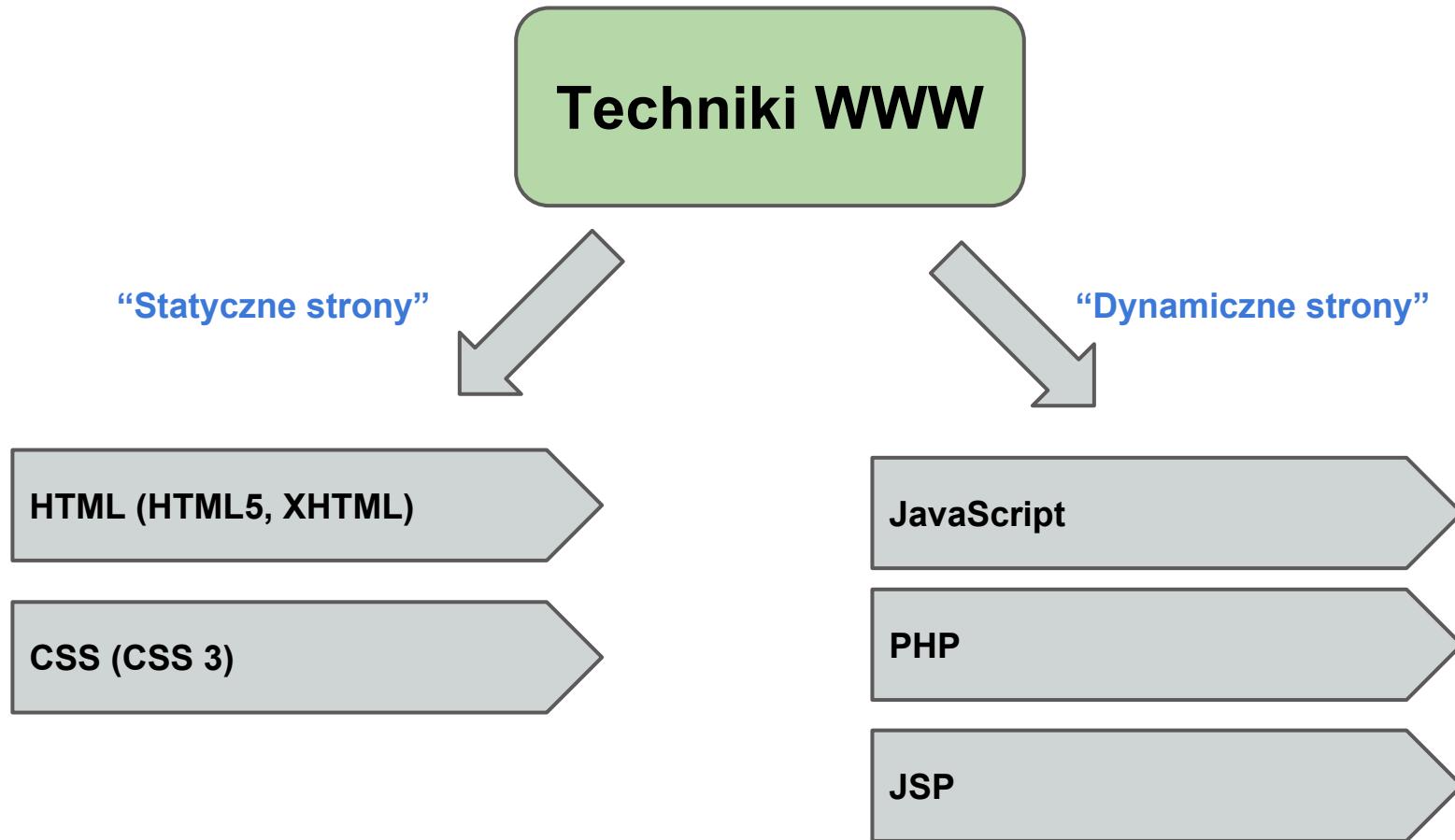
Rok akademicki: 2015/2016 - semestr zimowy



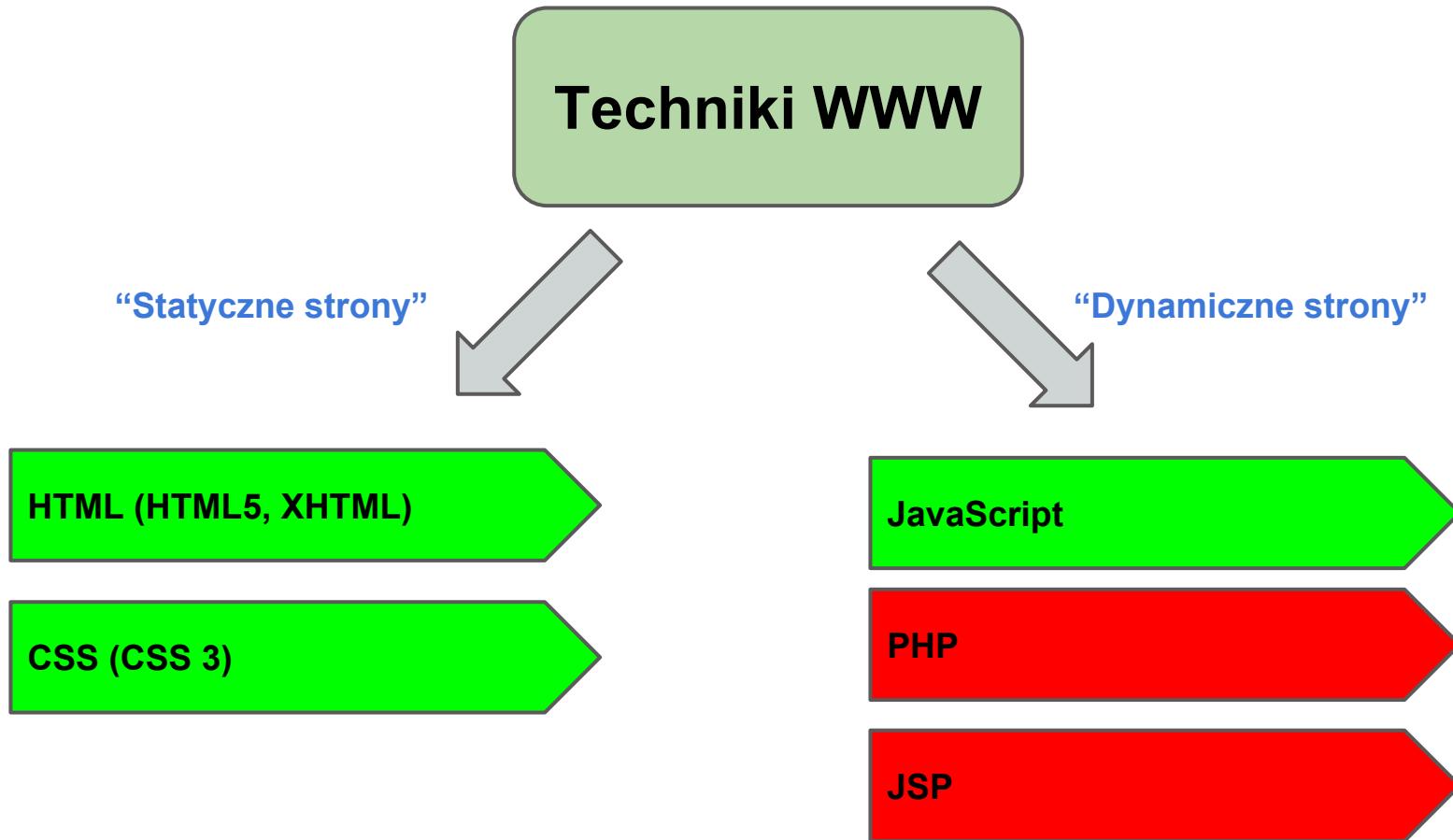
Uwagi ogólne o przedmiocie kursu

Techniki WWW

Uwagi ogólne o przedmiocie kursu



Uwagi ogólne o przedmiocie kursu



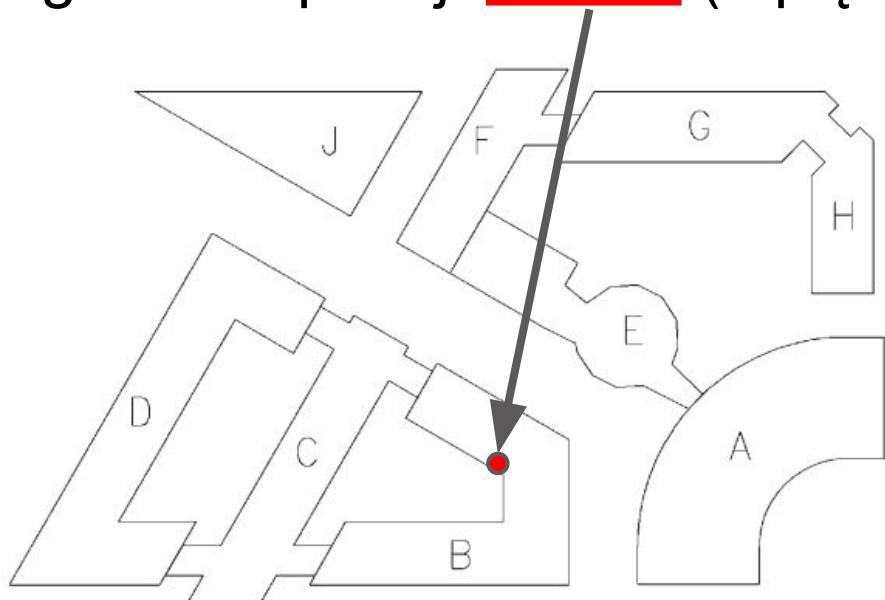
Konsultacje

Zakład Fizyki Jądrowej

Instytut Fizyki

ul. Łojasiewicza 11

Segment B pokój: **B-2-33** (2 piętro)



OZNACZENIE SEGMENTÓW BUDYNKU FIZYKI

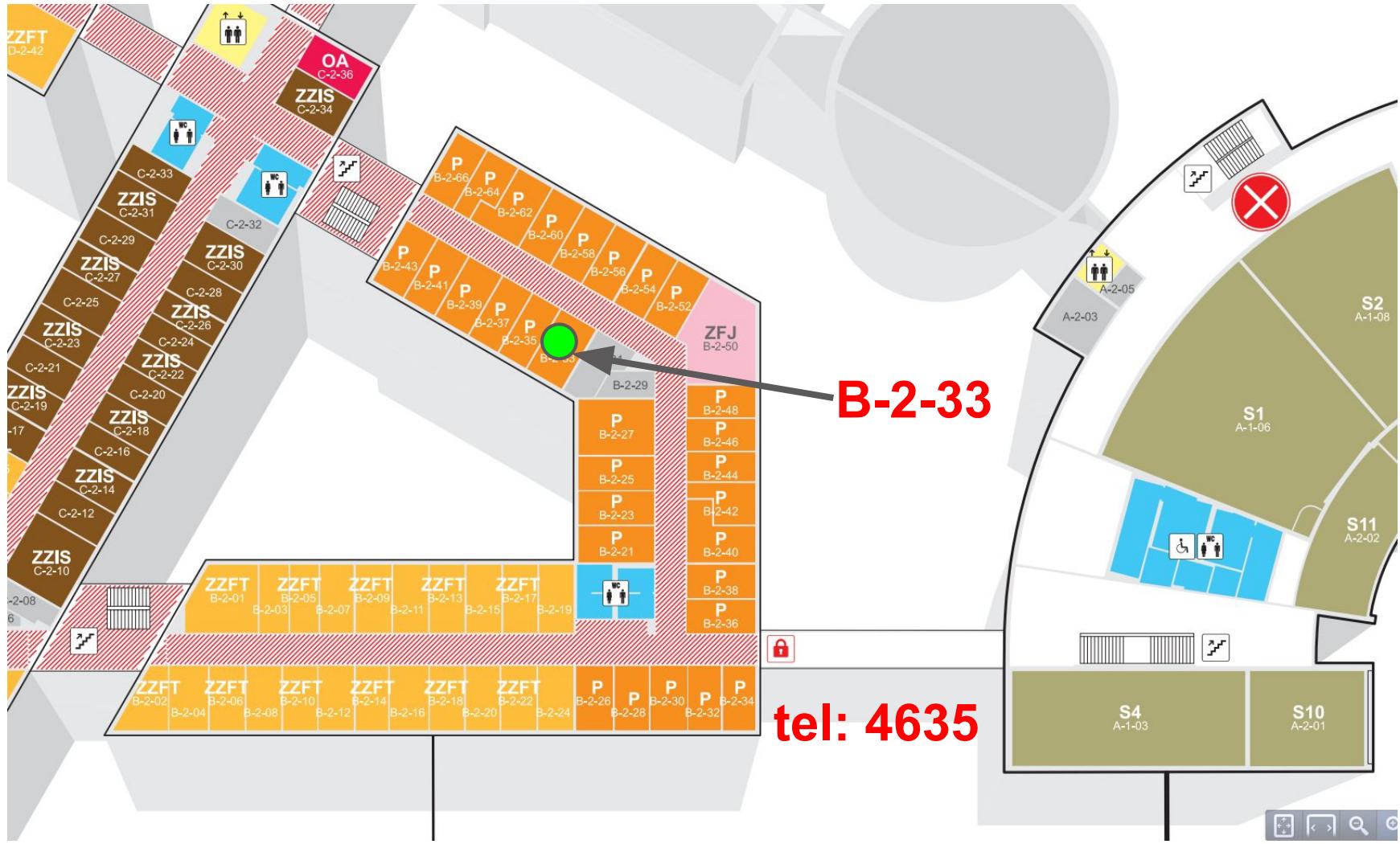


termin: środa 10:00 - 11:00

email: m.zielinski@uj.edu.pl

web: <http://koza.if.uj.edu.pl/~marcin>

Konsultacje





Organizacja kursu

Kurs składa się z wykładu (30h) oraz ćwiczeń praktycznych w pracowni komputerowej (30h)

Wykład:

Środa 15:30 - 17:00 sala: A-1-04

Ćwiczenia:

GRUPA 1: Środa 12:00 - 14:00 sala: G-1-03

Prowadzący: **Prof. dr hab. Jerzy Konior**

GRUPA 2: Czwartek 12:30 - 14:00 sala: G-1-03

Prowadzący: **Dr inż. Marcin Zieliński**

GRUPA 3: Czwartek 12:30 - 14:00 sala: G-1-07

Prowadzący: **Mgr inż. Adam Górski**



Organizacja kursu

Informacje o kursie, zasady zaliczenia oraz materiały do wykładu będą dostępne na stronie kursu pod adresem:

<http://koza.if.uj.edu.pl/~marcin/dydaktyka/twww1516.php>

Warunki zaliczenia

Ćwiczenia:

- Nie więcej niż 2 nieusprawiedliwione nieobecności.
- Oddanie w terminie **3 projektów** zaliczeniowych wykonywanych w trakcie trwania semestru.

Wykład:

- Wykonanie **1 projektu** zaliczeniowego oraz jego prezentacja (omówienie) w trakcie “egzaminu ustnego”.



Warunki zaliczenia - terminy

Ćwiczenia:

termin oddania ostatniego projektu: **28.01.2016**

Wykład:

termin prezentacji projektu zaliczeniowego w formie egzaminu ustnego: **2 tydzień sesji**
(termin do ustalenia)

Literatura analogowa

Elizabeth Castro, *HTML 4*, wydawnictwo Helion, Gliwice 2000.

Matthew MacDonald, *HTML5*, wydawnictwo Helion, 2012.

Arman Danesh, *JavaScript*, wydawnictwo Helion, Gliwice 1997.

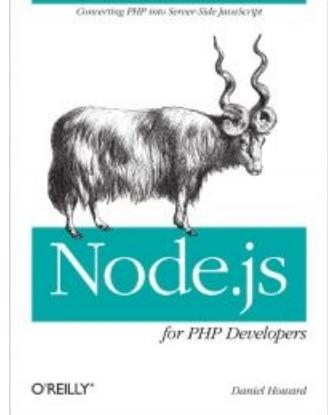
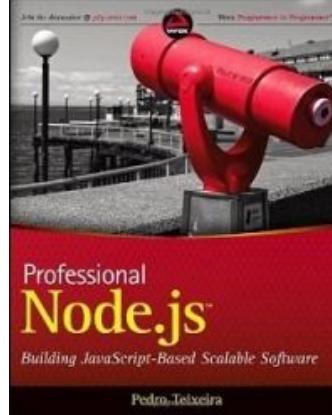
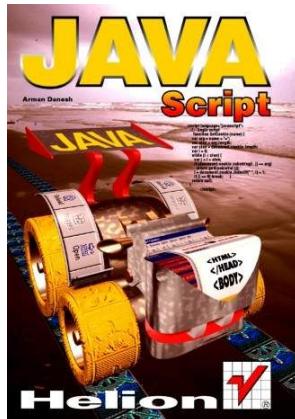
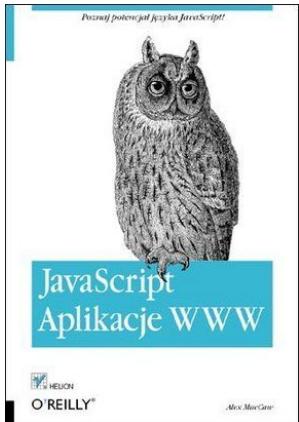
Luke Welling, *PHP i MySQL. Tworzenie stron WWW*, wydawnictwo Helion, Gliwice 2003 i 2005.

Dan Shafer: *Utopia HTML. Projektowanie w CSS bez użycia tabel*, 2005.

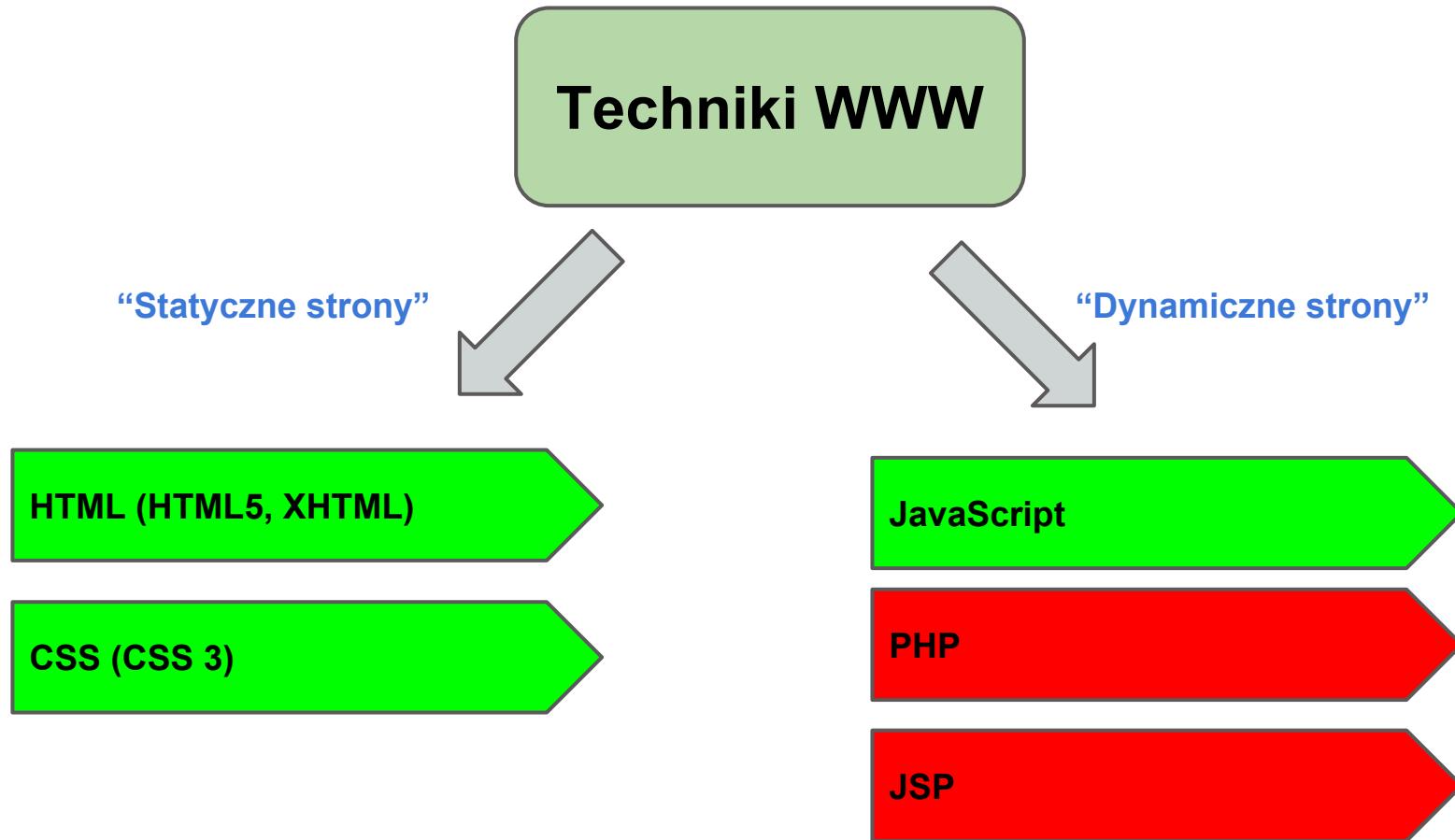
Alex MacCaw, *JavaScript Aplikacje WWW*, ISBN: 978-83-246-3887-1, 2012

Pedro Teixeira, *Professional Node.js*, ISBN: 978-1-1181-8546-9, 2012

Daniel Howard, *Node.js for PHP Developers*, ISBN: 978-1-4493-3360-7, 2012



Ramowy plan wykładu





Ramowy plan wykładu

Część 1:

- Wstęp do technologii projektowania i modelowania stron oraz aplikacji internetowych.
- Podstawy języka znaczników HTML (w wersji 5) oraz kaskadowych arkuszy stylu (CSS3).
- Wprowadzenie do JavaScript jako rozszerzenie funkcjonalności i atrakcyjności stron www.
- Wprowadzenie do bibliotek i środowisk JavaScriptowych: jQuery / BootStrap oraz Angular.js
- Javascript w ujęciu asynchronicznego podejścia do interakcji przeglądarka-serwer.
- Podejście obiektowe do tworzenia aplikacji internetowych z wykorzystaniem JavaScriptu.

Ramowy plan wykładu

Część 2:

- Wprowadzenie do środowiska NODE.JS.
- Środowisko NODE.JS jako serwer www, przykłady tworzenia usług webowych.
- NODE.JS oraz środowisko Express jako narzędzia tworzenia aplikacji w architekturze MVC.
- Usługi bazodanowe w aplikacjach internetowych z wykorzystaniem NODE.JS.
- WebServices w NODE.JS (XML + SOAP) oraz serwery WebSocket i obsługa zdarzeń w oparciu o NODE.JS.
- Komunikacja asynchroniczna - problemy, trudności oraz skalowanie aplikacji.



UNIWERSYTET
JAGIELŁONSKI
W KRAKOWIE

Co to jest INTERNET

?

Co to jest INTERNET ?



To ta niebieska ikonka na pulpicie :)

Co to jest INTERNET ?



Teraz tych ikonek jest więcej !!!

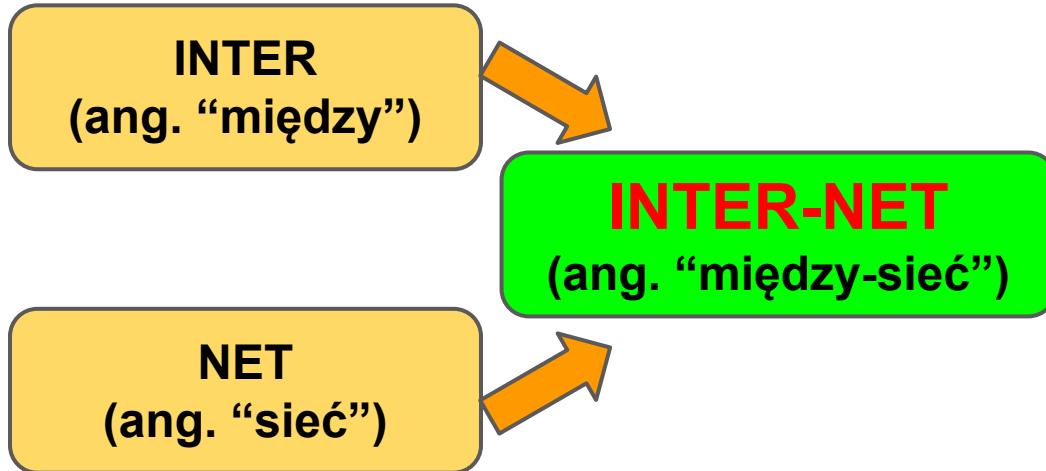


Co to jest INTERNET ?

INTER
(ang. “między”)

NET
(ang. “sieć”)

Co to jest INTERNET ?



- Ogólnoświatowa sieć komputerowa, łącząca miliony komputerów.
- Oparta o protokół TCP/IP w którym każde urządzenie ma swój unikalny identyfikator (adres IP - Internet Protocol).
- Poza komputerami w sieci pracują modemy, routery i switche, które są odpowiedzialne za obsługę ruchu sieciowego.

Skąd się wziął INTERNET ?



ARPANET

Skąd się wziął INTERNET ?



ARPANET

29 września 1969 roku, na Uniwersytecie Kalifornijskim w Los Angeles (UCLA), oraz w trzech innych uniwersytetach zainstalowano w ramach eksperymentu finansowanego przez ARPA (Advanced Research Project Agency, zajmującą się koordynowaniem badań naukowych na potrzeby wojska) pierwsze węzły sieci ARPANET – bezpośredniego “przodka” dzisiejszego Internetu.

Skąd się wziął INTERNET ?



ARPANET

29 września 1969 roku, na Uniwersytecie Kalifornijskim w Los Angeles (UCLA), oraz w trzech innych uniwersytetach zainstalowano w ramach eksperymentu finansowanego przez ARPA (Advanced Research Project Agency, zajmującą się koordynowaniem badań naukowych na potrzeby wojska) pierwsze węzły sieci ARPANET – bezpośredniego “przodka” dzisiejszego Internetu.



CERN-NET

W październiku 1991 roku naukowiec z CERN – Tim Berners-Lee, chcąc podzielić się wynikami swoich badań z innymi ludźmi zajmującymi się fizyką cząstek elementarnych, stworzył podstawy języka **HTML**.



INTERNET w Polsce



19 listopada 1990 roku

20 listopada 1990 roku

30 kwietnia 1991 roku

20 grudnia 1991 roku

kwiecień 1996 roku



INTERNET w Polsce

19 listopada 1990 roku

Nadanie pierwszego w Polsce adresu IP (192.86.14.0) przez departament obrony USA dla komputera w Instytucie Fizyki Jądrowej PAN (Kraków-Bronowice).

20 listopada 1990 roku

<http://popul.ifj.edu.pl/historia/37/zobacz.html>



30 kwietnia 1991 roku

Komputer MikroVAX II
RAM: 16MB,
dysk twardy: 2 x 333 MB

20 grudnia 1991 roku

kwiecień 1996 roku

INTERNET w Polsce



19 listopada 1990 roku

Nadanie pierwszego w Polsce adresu IP (192.86.14.0) przez departament obrony USA dla komputera w Instytucie Fizyki Jądrowej PAN (Kraków-Bronowice).

20 listopada 1990 roku

Odebranie pierwszej wiadomości e-mail wysłanej z CERN do IFJ PAN (wspólnie dr Grzegorz Polok i mgr Paweł Jałocha a odbiorcą w IFJ był mgr inż. Andrzeja Sobala)

30 kwietnia 1991 roku

20 grudnia 1991 roku

kwiecień 1996 roku

<http://popul.ifj.edu.pl/historia/37/zobacz.html>



Komputer MikroVAX II
RAM: 16MB,
dysk twardy: 2 x 333 MB



INTERNET w Polsce

19 listopada 1990 roku

Nadanie pierwszego w Polsce adresu IP (192.86.14.0) przez departament obrony USA dla komputera w Instytucie Fizyki Jądrowej PAN (Kraków-Bronowice).

20 listopada 1990 roku

Odebranie pierwszej wiadomości e-mail wysłanej z CERN do IFJ PAN (wspólnie dr Grzegorz Polok i mgr Paweł Jałocha a odbiorcą w IFJ był mgr inż. Andrzeja Sobala)

30 kwietnia 1991 roku

Uniwersytet w Kopenhadze zarejestrował polską domenę najwyższego poziomu “.pl”

20 grudnia 1991 roku

kwiecień 1996 roku

<http://popul.ifj.edu.pl/historia/37/zobacz.html>



Komputer MikroVAX II
RAM: 16MB,
dysk twardy: 2 x 333 MB





INTERNET w Polsce

19 listopada 1990 roku

Nadanie pierwszego w Polsce adresu IP (192.86.14.0) przez departament obrony USA dla komputera w Instytucie Fizyki Jądrowej PAN (Kraków-Bronowice).

20 listopada 1990 roku

Odebranie pierwszej wiadomości e-mail wysłanej z CERN do IFJ PAN (wspólnie dr Grzegorz Polok i mgr Paweł Jałocha a odbiorcą w IFJ był mgr inż. Andrzeja Sobala)

30 kwietnia 1991 roku

Uniwersytet w Kopenhadze zarejestrował polską domenę najwyższego poziomu “.pl”

20 grudnia 1991 roku

USA zniosły ograniczenia na połączenia z Polską - internet oficjalnie dostępny.

kwiecień 1996 roku

<http://popul.ifj.edu.pl/historia/37/zobacz.html>



Komputer MikroVAX II
RAM: 16MB,
dysk twardy: 2 x 333 MB





INTERNET w Polsce

19 listopada 1990 roku

Nadanie pierwszego w Polsce adresu IP (192.86.14.0) przez departament obrony USA dla komputera w Instytucie Fizyki Jądrowej PAN (Kraków-Bronowice).

20 listopada 1990 roku

Odebranie pierwszej wiadomości e-mail wysłanej z CERN do IFJ PAN (wspólnie dr Grzegorz Polok i mgr Paweł Jałocha a odbiorcą w IFJ był mgr inż. Andrzeja Sobala)

30 kwietnia 1991 roku

Uniwersytet w Kopenhadze zarejestrował polską domenę najwyższego poziomu ".pl"

20 grudnia 1991 roku

USA zniosły ograniczenia na połączenia z Polską - internet oficjalnie dostępny.

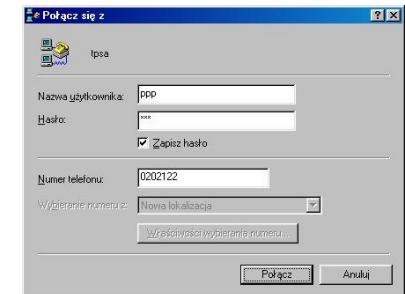
kwiecień 1996 roku

TP S.A. uruchomiła powszechną usługę połączenia zdawanego do sieci przez modem za pomocą ogólnokrajowego nr 020-21-22

<http://popul.ifj.edu.pl/historia/37/zobacz.html>

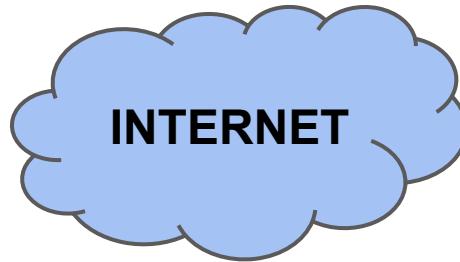


Komputer MikroVAX II
RAM: 16MB,
dysk twardy: 2 x 333 MB



Czy WWW to INTERNET ?

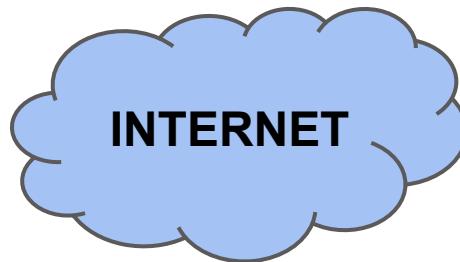
Sieć komputerowa



Działa dzięki wykorzystaniu
protokołu IP (Internet Protocol)

Czy WWW to INTERNET ?

Sieć komputerowa



Działa dzięki wykorzystaniu protokołu IP (Internet Protocol)

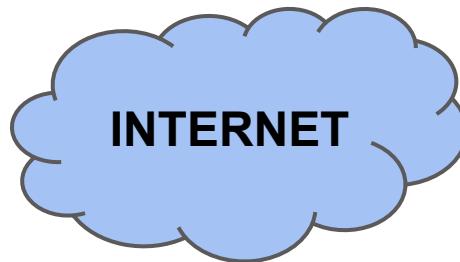
Usługa w sieci



Działa dzięki wykorzystaniu protokołu HTTP (Hyper Text Transfer Protocol)

Czy WWW to INTERNET ?

Sieć komputerowa



Działa dzięki wykorzystaniu protokołu IP (Internet Protocol)

Usługa w sieci



Działa dzięki wykorzystaniu protokołu HTTP (Hyper Text Transfer Protocol)

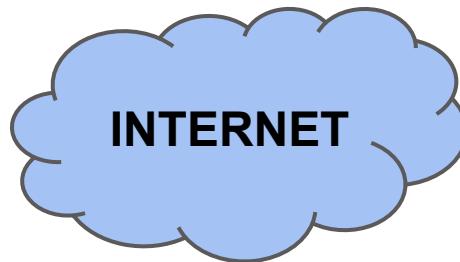
Protokół HTTP działa w wyższej warstwie (warstwie aplikacji) niż IP.
Protokół IP stanowi dla HTTP pewnego rodzaju nośnik.

WWW stanowi miliony dokumentów (głównie tekstowych) znajdujących się w komputerach podłączonych do sieci internet.

Większość z tych dokumentów jest stworzona w języku **HTML (Hyper Text Markup Language)**.

Czy WWW to INTERNET ?

Sieć komputerowa



Działa dzięki wykorzystaniu protokołu IP (Internet Protocol)

Usługa w sieci



Działa dzięki wykorzystaniu protokołu HTTP (Hyper Text Transfer Protocol)

Protokół HTTP działa w wyższej warstwie (warstwie aplikacji) niż IP.
Protokół IP stanowi dla HTTP pewnego rodzaju nośnik.

WWW stanowi miliony dokumentów (głównie tekstowych) znajdujących się w komputerach podłączonych do sieci internet.

Większość z tych dokumentów jest stworzona w języku **HTML (Hyper Text Markup Language)**.

Nie ma WWW bez INTERNETU!



Podstawowe pojęcia: URI, URL, URN

Uniform Resource Identifier (URI) [*Uniwersalny Identyfikator Zasobu*]

RFC 2369
(sierpień 1998)

Uniform Resource Locator (URL) [*Uniwersalny Lokalizator Zasobu*]

RFC 1738
(grudzień 1994)

Uniform Resource Name (URN) [*Uniwersalna Nazwa Zasobu*]

Podstawowe pojęcia: URI, URL, URN

Uniform Resource Identifier (URI) [Uniwersalny Identyfikator Zasobu]

Umożliwia jednoznaczne określenie oraz identyfikację zasobu.

RFC 2369
(sierpień 1998)

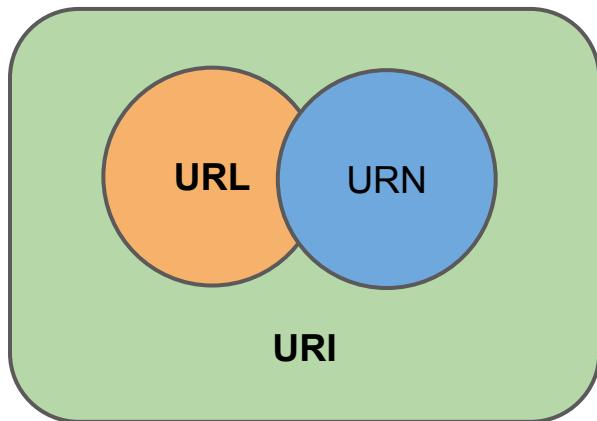
Uniform Resource Locator (URL) [Uniwersalny Lokalizator Zasobu]

Umożliwia jednoznaczne określenie lokalizacji zasobu.
Zasadniczo URL jest to podgrupą identyfikatorów URI.

RFC 1738
(grudzień 1994)

Uniform Resource Name (URN) [Uniwersalna Nazwa Zasobu]

Umożliwia jednoznaczne określenie nazwy dla danego zasobu.
System ten najczęściej jest używany np. do identyfikacji książek



<schemat>:<nazwa-zasobu>

<http://www.uj.edu.pl>

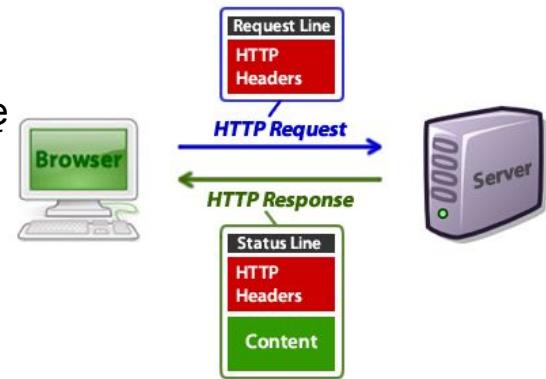
<http://koza.if.uj.edu.pl/~marcin>

<ftp://example.com>

Protokół HTTP

Hyper Text Transfer Protocol (HTTP/1.1) [Protokół przesyłania dokumentów Hiper-Tekstowych]

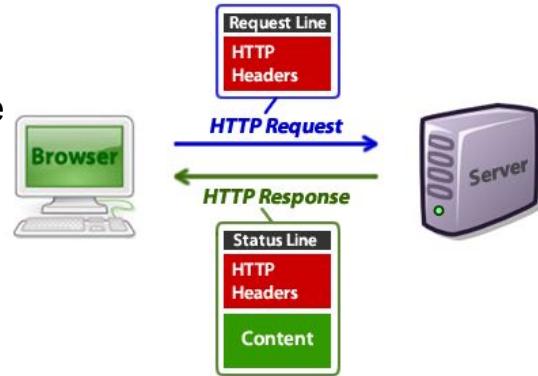
- W ogólności protokół zapewnia komunikację pomiędzy usługodawcą (serwerem) a klientem (hostem).
- W protokole zawarte są metody (*funkcje*) umożliwiające obsługę żądań (*zapytań*) wysyłanych przez klienta oraz odpowiedzi serwera.
- Cechą charakterystyczną protokołu HTTP jest jego “bezstanowcość” (ang. *stateless*), co oznacza że nigdzie nie istnieje zapis stanu poprzednio wykonanych operacji, a kolejne transakcje są wykonywane niezależnie.
- Protokół HTTP operuje standardowo na porcie 80.



Protokół HTTP

Hyper Text Transfer Protocol (HTTP/1.1) [Protokół przesyłania dokumentów Hiper-Tekstowych]

- W ogólności protokół zapewnia komunikację pomiędzy usługodawcą (serwerem) a klientem (hostem).
- W protokole zawarte są metody (*funkcje*) umożliwiające obsługę żądań (*zapytań*) wysyłanych przez klienta oraz odpowiedzi serwera.
- Cechą charakterystyczną protokołu HTTP jest jego “bezstanowcość” (ang. *stateless*), co oznacza że nigdzie nie istnieje zapis stanu poprzednio wykonanych operacji, a kolejne transakcje są wykonywane niezależnie.
- Protokół HTTP operuje standardowo na porcie 80.



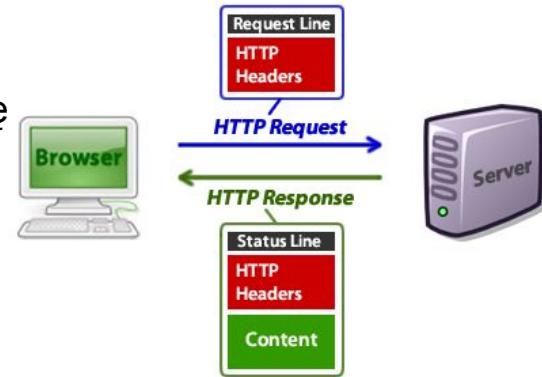
Przykład:

Kliknięcie na *hiperpołączenie* (np. w przeglądarce) powoduje, że komputer-klient (host) wysyła do serwera *żądanie*, po otrzymaniu żądania i po przeszukaniu swoich zasobów, wysyła do klienta – jeśli to możliwe – stosowną stronę WWW (lub w ogólności dowolne zasoby).

Protokół HTTP

Hyper Text Transfer Protocol (HTTP/1.1) [Protokół przesyłania dokumentów Hiper-Tekstowych]

- W ogólności protokół zapewnia komunikację pomiędzy usługodawcą (serwerem) a klientem (hostem).
- W protokole zawarte są metody (*funkcje*) umożliwiające obsługę żądań (*zapytań*) wysyłanych przez klienta oraz odpowiedzi serwera.
- Cechą charakterystyczną protokołu HTTP jest jego “bezstanowcość” (ang. *stateless*), co oznacza że nigdzie nie istnieje zapis stanu poprzednio wykonanych operacji, a kolejne transakcje są wykonywane niezależnie.
- Protokół HTTP operuje standardowo na porcie 80.



Przykład:

Kliknięcie na *hiperpołączenie* (np. w przeglądarce) powoduje, że komputer-klient (host) wysyła do serwera *żądanie*, po otrzymaniu żądania i po przeszukaniu swoich zasobów, wysyła do klienta – jeśli to możliwe – stosowną stronę WWW (lub w ogólności dowolne zasoby).

Definicja i opis standardu HTTP/1.1

(opis standardu HTTP/2.0 - w przygotowaniu)

RFC 2616
(czerwiec 1999)

RFC (Request for Comments)

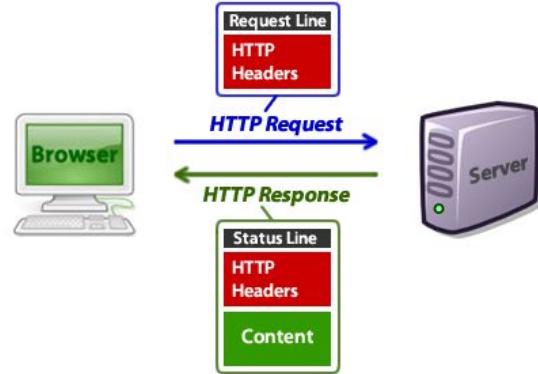
Protokół HTTP

Hyper Text Transfer Protocol (HTTP/1.1) [Protokół przesyłania dokumentów Hiper-Tekstowych]

Najważniejsze metody protokołu HTTP:

- **HEAD** - wysyła żądanie przesłania nagłówka zawierającego meta-dane (informację), bez przesyłania samego zasobu.
- **GET** - wysyła żądanie pobrania konkretnego zasobu URI (np. strony internetowej napisanej w języku HTML).
- **POST** - wysyła żądanie do serwera akceptacji zasobu dołączonego do żądania.

pozostałe: **PUT, DELETE, TRACE, OPTIONS, CONNECT.**



Przykład 1:

Żądanie (klient)

GET HTTP/1.1

Odpowiedź (serwer)

HTTP/1.1 200 OK

Kod odpowiedzi (stanu)



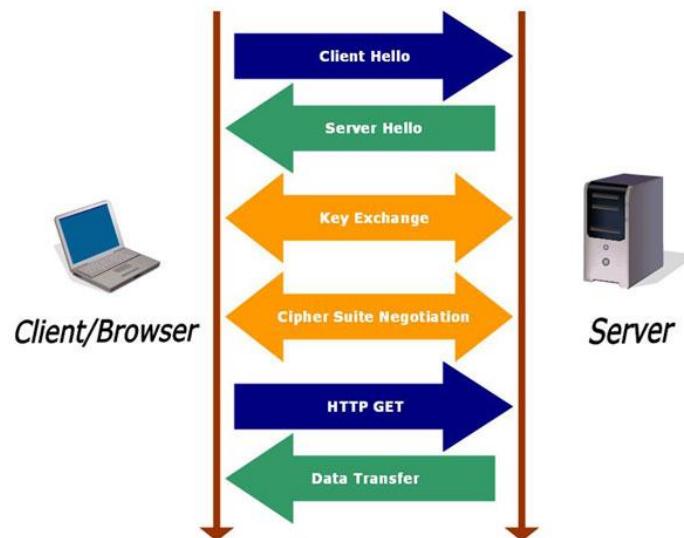
Protokół HTTPS

Hyper Text Transfer Protocol Secure (HTTPS) [Zabezpieczony Protokół przesyłania dokumentów Hiper-Tekstowych]

Protokół HTTPS

Hyper Text Transfer Protocol Secure (HTTPS) [Zabezpieczony Protokół przesyłania dokumentów Hiper-Tekstowych]

- Jest to wersja szyfrowana protokołu HTTP, oparta o protokół szyfrujący SSL (Secure Socket Layer).
- SSL działa warstwę niżej w modelu OSI od samego HTTPS.
- Działanie protokołu HTTPS polega na tym samym co HTTP, natomiast zanim zostanie ustanowiona komunikacja "klient-serwer", wymieniane są klucze szyfrujące SSL.
- Wymiana danych na bazie protokołu HTTPS, zapobiega przechwyceniu oraz zmianie danych w trakcie komunikacji klient-serwer.
- Domyslnym portem na których serwer nasłuchuje żądań HTTPS to 443.



<https://www.extranet.uj.edu.pl>

Protokół HTTP i HTTPS

- Do sprawnego posługiwania się protokołem HTTP/HTTPS konieczna jest znajomość podstawowych kodów oznaczających stany serwera zwracana w odpowiedzi na żądanie klienta.
- Kod stanu (odpowiedź) serwera jest podawana w nagłówku HTTP/HTTPS w postaci liczby trzycyfrowej.



Protokół HTTP i HTTPS

- Do sprawnego posługiwania się protokołem HTTP/HTTPS konieczna jest znajomość podstawowych kodów oznaczających stany serwera zwracana w odpowiedzi na żądanie klienta.
- Kod stanu (odpowiedź) serwera jest podawana w nagłówku HTTP/HTTPS w postaci liczby trzycyfrowej.



Klasyfikacja kodów stanów (odpowiedzi serwera):

Zakres kodów	Znaczenie
100 - 199	Informacyjne.
200 - 299	Żądanie (od klienta) powiodło się
300 - 399	Żądanie klienta zostało przekazane, wymagane są dalsze działania.
400 - 499	Żądanie klienta nie powiodło się.
500 - 599	Błąd serwera.

Protokół HTTP i HTTPS - kody stanu

Kod	Znaczenie
100	Continue: Wstępna część zapytania została odebrana i klient może kontynuować wysyłanie dalszych części zapytania.
101	Switching Protocols: Serwer zastosował się do żądania klienta i przełącza protokół na ten, który określony jest w polu <i>Upgrade</i> w nagłówku zapytania.
200	OK: Zapytanie powiodło się, a odpowiedź wysłana przez serwer zawiera żądane przez klienta dane.
202	Accepted: Zapytanie zostało zaakceptowane, ale serwer nie od razu przystąpił do jego przetwarzania. Nie ma żadnych gwarancji, że serwer przyjmie zapytanie, choć na etapie odbioru i akceptacji nie było do niego żadnych zastrzeżeń.
300	Opcje do wyboru: URI określony w zapytaniu odnosi się do więcej niż jednego zasobu. Na przykład URI może określać dokument, który został przetłumaczony na wiele języków. Wiadomość zwracana przez serwer może zawierać informację, jak poprawnie należy określić zasoby, których potrzebujemy.
301	Przeniesiony: Żądany URI nie jest używany przez serwer i działanie określone w zapytaniu nie zostało wykonane. Nowe miejsce, w którym umieszczony jest wskazywany dokument, określone jest w polu <i>Location</i> . Wszystkie następne zapytania kierowane w sprawie tego dokumentu powinny zawierać nowy URI.
302	Czasowo przeniesiony: Żądany URI został czasowo przeniesiony w inne miejsce. Nowe miejsce, w którym umieszczony jest wskazywany dokument, określone jest w polu <i>Location</i> . Następne zapytanie, wysypane po otrzymaniu tej odpowiedzi, powinno zawierać nowy URI, ale poprzedni URI powinien być używany w zapytaniach wysyłanych w przyszłości.

Protokół HTTP i HTTPS - kody stanu

Kod	Znaczenie
400	Niepoprawne zapytanie: Ten kod odpowiedzi wskazuje, że serwer wykrył błąd w składni zapytania kierowanego przez klienta.
401	Brak autoryzacji: Ten kod jest przesyłany wraz z polem <i>WWW-Authenticate</i> , informując, że zapytanie nie przeszło pomyślnie przez proces uwierzytelniania, więc przy następnym zapytaniu o ten URI klient powinien dołączyć poprawne dane umożliwiające jego uwierzytelnienie.
403	Zakazany: Zapytanie zostało odrzucone przez serwer, ponieważ serwer nie przyjmuje zapytań od tego klienta lub nie może określić, z jakim klientem ma do czynienia.
404	Nie znaleziono: Dokument określony podanym w zapytaniu URI nie istnieje.
405	Niedozwolona metoda: Kod ten jest przekazywany wraz z polem <i>Allow</i> określającym, że metoda używana przez klienta nie jest obsługiwana dla podanego URI.
409	Konflikt: Kod ten wskazuje, że skierowane do serwera zapytanie jest w konflikcie z innym zapytaniem lub z konfiguracją serwera. Dokładniejsze informacje o rodzaju konfliktu powinny być przesłane w zasadniczej części wiadomości.
410	Usunięty: Kod ten wskazuje, że URI umieszczony w zapytaniu już nie istnieje i został ostatecznie usunięty z serwera.

Protokół HTTP i HTTPS - kody stanu

Kod	Znaczenie
500	Wewnętrzny błąd serwera: Kod ten wskazuje, że część serwera (na przykład program CGI) działa niepoprawnie lub wystąpił błąd konfiguracji serwera.
501	Nie używane: Ten kod odpowiedzi wskazuje, że zapytanie klienta dotyczy działań, które nie mogą być wykonane przez serwer.
503	Usługa niedostępna: Ten kod odpowiedzi wskazuje, że usługi serwera są czasowo niedostępne, ale w przyszłości powinny być dostępne. Jeżeli serwer wie, kiedy to nastąpi, do odpowiedzi dołączony może być pole <i>Retry-After</i> .
505	Wersja HTTP nie obsługiwana: Serwer nie obsługuje wersji HTTP użytej w zapytaniu.

KONIEC WYKŁADU 1



UNIWERSYTET
JAGIELŁOŃSKI
W KRAKOWIE

Zaawansowane Techniki WWW (HTML, CSS i JavaScript)

Dr inż. Marcin Zieliński

Środa 15:30 - 17:00 sala: A-1-04

WYKŁAD 2

Wykład dla kierunku: Informatyka Stosowana II rok

Rok akademicki: 2015/2016 - semestr zimowy



Przypomnienie

Wprowadzenie do technologii internetowych

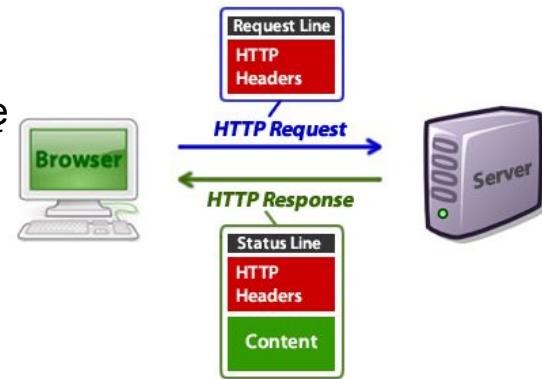
Historia internetu stron WWW

Protokoły komunikacyjne HTTP/HTTPS

Protokół HTTP

Hyper Text Transfer Protocol (HTTP/1.1) [Protokół przesyłania dokumentów Hiper-Tekstowych]

- W ogólności protokół zapewnia komunikację pomiędzy usługodawcą (serwerem) a klientem (hostem).
- W protokole zawarte są metody (*funkcje*) umożliwiające obsługę żądań (*zapytań*) wysyłanych przez klienta oraz odpowiedzi serwera.
- Cechą charakterystyczną protokołu HTTP jest jego “bezstanowcość” (ang. *stateless*), co oznacza że nigdzie nie istnieje zapis stanu poprzednio wykonanych operacji, a kolejne transakcje są wykonywane niezależnie.
- Protokół HTTP operuje standardowo na porcie 80.



Przykład:

Kliknięcie na *hiperpołączenie* (np. w przeglądarce) powoduje, że komputer-klient (host) wysyła do serwera *żądanie*, po otrzymaniu żądania i po przeszukaniu swoich zasobów, wysyła do klienta – jeśli to możliwe – stosowną stronę WWW (lub w ogólności dowolne zasoby).

Definicja i opis standardu HTTP/1.1

(opis standardu HTTP/2.0 - w przygotowaniu)

RFC 2616
(czerwiec 1999)

RFC (Request for Comments)

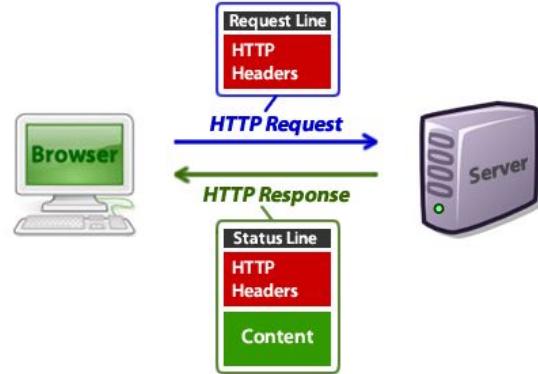
Protokół HTTP

Hyper Text Transfer Protocol (HTTP/1.1) [Protokół przesyłania dokumentów Hiper-Tekstowych]

Najważniejsze metody protokołu HTTP:

- **HEAD** - wysyła żądanie przesłania nagłówka zawierającego meta-dane (informację), bez przesyłania samego zasobu.
- **GET** - wysyła żądanie pobrania konkretnego zasobu URI (np. strony internetowej napisanej w języku HTML).
- **POST** - wysyła żądanie do serwera akceptacji zasobu dołączonego do żądania.

pozostałe: **PUT, DELETE, TRACE, OPTIONS, CONNECT.**



Przykład 1:

Żądanie (klient)

GET HTTP/1.1

Odpowiedź (serwer)

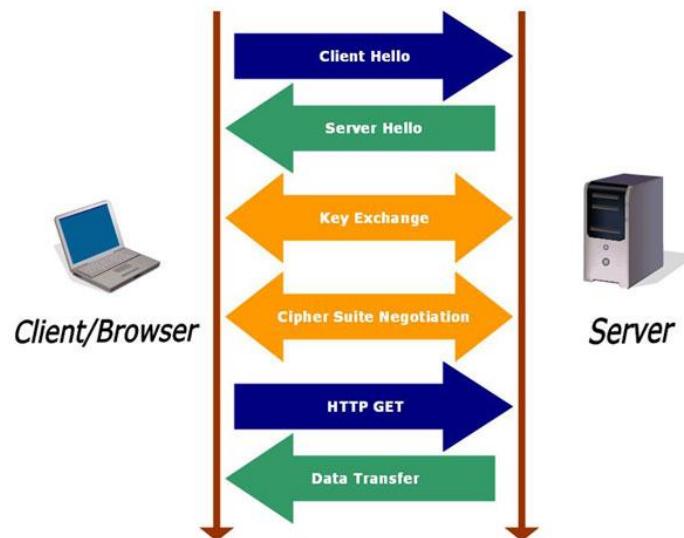
HTTP/1.1 200 OK

Kod odpowiedzi (stanu)

Protokół HTTPS

Hyper Text Transfer Protocol Secure (HTTPS) [Zabezpieczony Protokół przesyłania dokumentów Hiper-Tekstowych]

- Jest to wersja szyfrowana protokołu HTTP, oparta o protokół szyfrujący SSL (Secure Socket Layer).
- SSL działa warstwę niżej w modelu OSI od samego HTTPS.
- Działanie protokołu HTTPS polega na tym samym co HTTP, natomiast zanim zostanie ustanowiona komunikacja "klient-serwer", wymieniane są klucze szyfrujące SSL.
- Wymiana danych na bazie protokołu HTTPS, zapobiega przechwyceniu oraz zmianie danych w trakcie komunikacji klient-serwer.
- Domyslnym portem na których serwer nasłuchuje żądań HTTPS to 443.



<https://www.extranet.uj.edu.pl>

Protokół HTTP i HTTPS

- Do sprawnego posługiwania się protokołem HTTP/HTTPS konieczna jest znajomość podstawowych kodów oznaczających stany serwera zwracana w odpowiedzi na żądanie klienta.
- Kod stanu (odpowiedź) serwera jest podawana w nagłówku HTTP/HTTPS w postaci liczby trzycyfrowej.



Klasyfikacja kodów stanów (odpowiedzi serwera):

Zakres kodów	Znaczenie
100 - 199	Informacyjne.
200 - 299	Żądanie (od klienta) powiodło się
300 - 399	Żądanie klienta zostało przekazane, wymagane są dalsze działania.
400 - 499	Żądanie klienta nie powiodło się.
500 - 599	Błąd serwera.

Serwery obsługujące protokół HTTP/HTTPS (Web-Serwery)

WebServer - komputer (serwer) - a najczęściej klasa komputerowa - obsługujący żądania HTTP/HTTPS, za pomocą odpowiedniego oprogramowania.

Główna a jednocześnie podstawową funkcjonalnością WebServerów jest przechowywanie, przetwarzanie i dostarczanie stron internetowych zapisanych za pomocą języka HTML, oraz obrazów, multimedialnych i skryptów, o które dodatkowo wzbogacone są strony internetowe.

Serwery obsługujące protokół HTTP/HTTPS (Web-Serwery)

WebServer - komputer (serwer), a najczęściej klaster komputerowy, obsługujący żądania HTTP/HTTPS, za pomocą odpowiedniego oprogramowania.

Główna a jednocześnie podstawowa funkcjonalnością WebServerów jest przechowywanie, przetwarzanie i dostarczanie stron internetowych zapisanych za pomocą języka HTML, oraz obrazów, multimedialnych i skryptów, o które dodatkowo wzbogacone są strony internetowe.

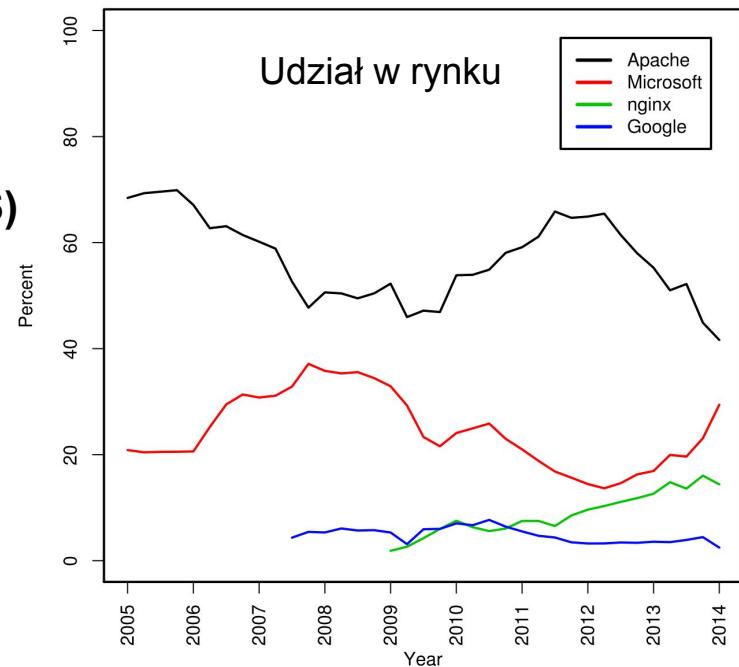
Najpopularniejsze oprogramowanie WebServerowe:

- APACHE
- Microsoft Internet Information Services (IIS)
- Nginx
- Google Web Server (GWS)



NGINX

Google



(dane Netcraft, maj 2014)

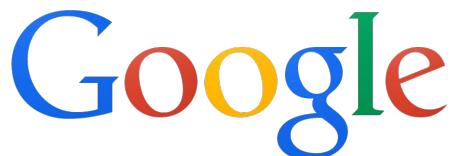
Serwery obsługujące protokół HTTP/HTTPS (Web-Serwery)

WebServer - komputer (serwer), a najczęściej klaster komputerowy, obsługujący żądania HTTP/HTTPS, za pomocą odpowiedniego oprogramowania.

Główna a jednocześnie podstawową funkcjonalnością WebServerów jest przechowywanie, przetwarzanie i dostarczanie stron internetowych zapisanych za pomocą języka HTML, oraz obrazów, multimedialnych i skryptów, o które dodatkowo wzbogacone są strony internetowe.

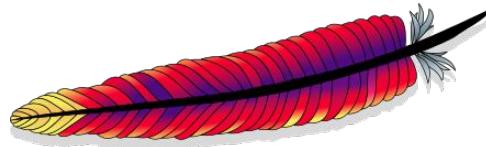
Najpopularniejsze oprogramowanie WebServerowe:

- APACHE
- Microsoft Internet Information Services (IIS)
- Nginx
- Google Web Server (GWS)



WebServer	Udział w rynku
Apache	38%
MS IIS	33%
Nginx	15%
GWS	2%

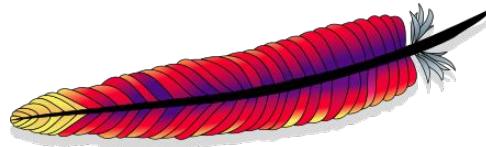
(dane Netcraft, maj 2014)



Serwer APACHE

Apache HTTP Server (Apache Software Foundation) - początki serwera sięgają roku 1995 kiedy w National Center for Supercomputing Applications Robert McCool stworzył aplikację NCSA Httpd.

- Serwer posiada wiele funkcjonalności zaimplementowanych w postaci modułów które rozszerzają jego podstawowe funkcje.
- Zapewnia wsparcie dla wielu języków skryptowych które mogą być wykorzystane przy tworzeniu aplikacji internetowych m.in. : PHP, Perl, Python, TCL.
- Współpracuje z najpopularniejszymi silnikami baz danych: MySQL, PostgreSQL.
- Apache obsługuje protokół HTTP jak i jego szyfrowaną wersję HTTPS (SSL).
- Jedną z zalet Apacha jest moduł "*mod_gzip*" umożliwiający kompresję danych przesyłanych przez protokół HTTP, co znaczco wpływa na prędkość podawania danych.
- W ramach jednego zainstalowanego serwera APACHE, dzięki tzw. "wirtualnym hostom" można obsługiwać wiele stron internetowych alokowanych w ramach różnych domen.
- Jest dostępny bezpłatnie na większość systemów operacyjnych, w szczególności jest dostarczony domyślnie z wieloma dystrybucjami systemu LINUX/UNIX.



Serwer APACHE

Apache HTTP Server (Apache Software Foundation) - początki serwera sięgają roku 1995 kiedy w National Center for Supercomputing Applications Robert McCool stworzył aplikację NCSA Httpd.

- Serwer posiada wiele funkcjonalności zaimplementowanych w postaci modułów które rozszerzają jego podstawowe funkcje.
- Zapewnia wsparcie dla wielu języków skryptowych które mogą być wykorzystane przy tworzeniu aplikacji internetowych m.in. : PHP, Perl, Python, TCL.
- Współpracuje z najpopularniejszymi silnikami baz danych: MySQL, PostgreSQL.
- Apache obsługuje protokół HTTP jak i jego szyfrowaną wersję HTTPS (SSL).
- Jedną z zalet Apacha jest moduł "*mod_gzip*" umożliwiający kompresję danych przesyłanych przez protokół HTTP, co znaczco wpływa na prędkość podawania danych.
- W ramach jednego zainstalowanego serwera APACHE, dzięki tzw. "wirtualnym hostom" można obsługiwać wiele stron internetowych alokowanych w ramach różnych domen.
- Jest dostępny bezpłatnie na większość systemów operacyjnych, w szczególności jest dostarczony domyślnie z wieloma dystrybucjami systemu LINUX/UNIX.

Uwaga:

Na liście procesów (np. w systemie Linux) pracujący serwer Apache oznaczony jest nazwą
httpd (http-daemon)



Serwer APACHE

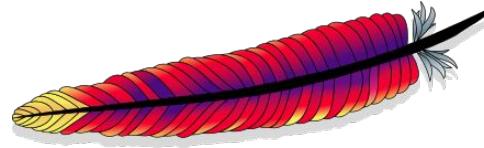
Apache HTTP Server (Apache Software Foundation) - początki serwera sięgają roku 1995 kiedy w National Center for Supercomputing Applications Robert McCool stworzył aplikację NCSA Httpd.

- Serwer posiada wiele funkcjonalności zaimplementowanych w postaci modułów które rozszerzają jego podstawowe funkcje.
- Zapewnia wsparcie dla wielu języków skryptowych które mogą być wykorzystane przy tworzeniu aplikacji internetowych m.in. : PHP, Perl, Python, TCL.
- Współpracuje z najpopularniejszymi silnikami baz danych: MySQL, PostgreSQL.
- Apache obsługuje protokół HTTP jak i jego szyfrowaną wersję HTTPS (SSL).
- Jedną z zalet Apacha jest moduł "*mod_gzip*" umożliwiający kompresję danych przesyłanych przez protokół HTTP, co znaczco wpływa na prędkość podawania danych.
- W ramach jednego zainstalowanego serwera APACHE, dzięki tzw. "wirtualnym hostom" można obsługiwać wiele stron internetowych alokowanych w ramach różnych domen.
- Jest dostępny bezpłatnie na większość systemów operacyjnych, w szczególności jest dostarczony domyślnie z wieloma dystrybucjami systemu LINUX/UNIX.

Uwaga:

Na liście procesów (np. w systemie Linux) pracujący serwer Apache oznaczony jest nazwą
httpd (http-daemon)

Najnowsza wersja: **2.4.10** do pobrania z <https://httpd.apache.org/>



Serwer APACHE

Apache HTTP Server (Apache Software Foundation) - początki serwera sięgają roku 1995 kiedy w National Center for Supercomputing Applications Robert McCool stworzył aplikację NCSA Httpd.

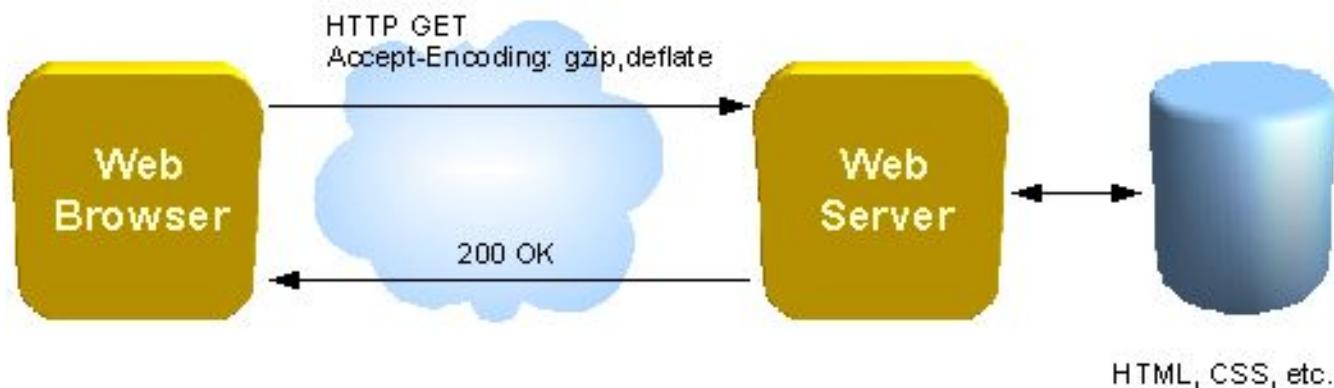
- Serwer posiada wiele funkcjonalności zaimplementowanych w postaci modułów które rozszerzają jego podstawowe funkcje.
- Zapewnia wsparcie dla wielu języków skryptowych które mogą być wykorzystane przy tworzeniu aplikacji internetowych m.in. : PHP, Perl, Python, TCL.
- Współpracuje z najpopularniejszymi silnikami baz danych: MySQL, PostgreSQL.
- Apache obsługuje protokół HTTP jak i jego szyfrowaną wersję HTTPS (SSL).
- Jedną z zalet Apacha jest moduł "*mod_gzip*" umożliwiający kompresję danych przesyłanych przez protokół HTTP, co znaczco wpływa na prędkość podawania danych.
- W ramach jednego zainstalowanego serwera APACHE, dzięki tzw. "wirtualnym hostom" można obsługiwać wiele stron internetowych alokowanych w ramach różnych domen.
- Jest dostępny bezpłatnie na większość systemów operacyjnych, w szczególności jest dostarczony domyślnie z wieloma dystrybucjami systemu LINUX/UNIX.

Uwaga:

Dla wygody użytkowników którzy chcą uruchomić w pełni działający serwer WWW na własnym laptopie polecam rozwiązania integrujące w jednej aplikacji serwer **Apache**, **MySQL**, **PHP**. Jednym z tego typu rozwiązań jest darmowa wersja aplikacji **X-AMP** dostępną na większość popularnych systemów operacyjnych:

<https://www.apachefriends.org/pl/index.html>

Synchroniczne żądanie http

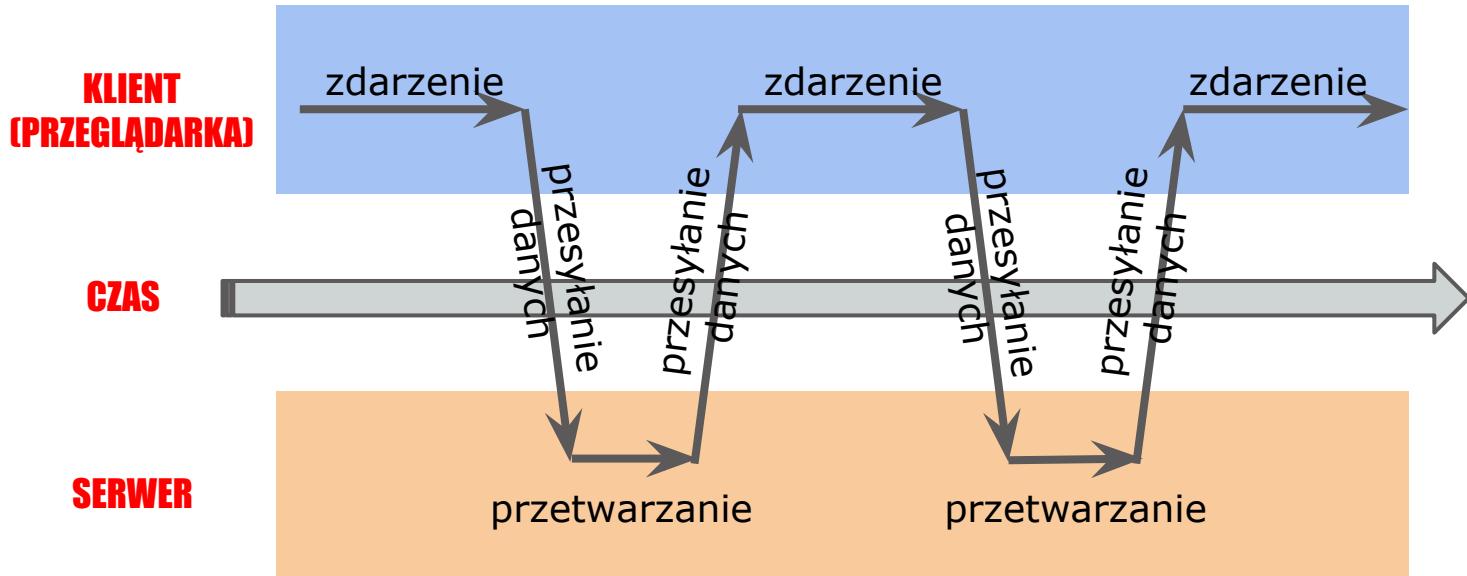


Jest to model synchronicznej komunikacji HTTP, gdzie klient wysyła żądanie, serwer je odbiera następnie przetwarza i na końcu generuje odpowiedź.

W sytuacji takiej klient musi czekać z kolejnym żądaniem do momentu kiedy nie dostanie odpowiedzi od serwera.

W modelu synchronicznym mamy bardzo mały poziom aktywności oraz interaktywności strony, strona musi być przeładowana (pobrać) po każdym żądaniu klienta, jeśli strony są złożone to proces ten jest długi.

Synchroniczne żądanie http

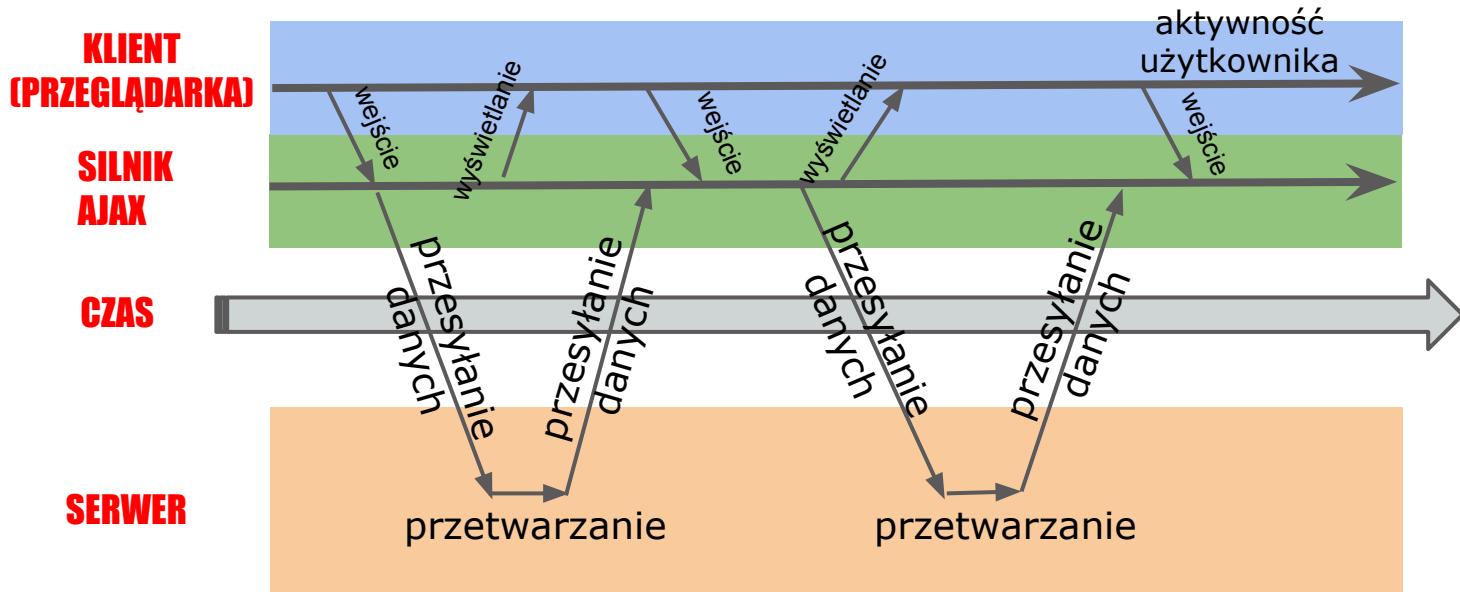


Jest to model synchronicznej komunikacji HTTP, gdzie klient wysyła żądanie, serwer je odbiera następnie przetwarza i na końcu generuje odpowiedź.

W sytuacji takiej klient musi czekać z kolejnym żądaniem do momentu kiedy nie dostanie odpowiedzi od serwera.

W modelu synchronicznym mamy bardzo mały poziom aktywności oraz interaktywności strony, strona musi być przeładowana (pobrać) po każdym żądaniu klienta, jeśli strony są złożone to proces ten jest długi.

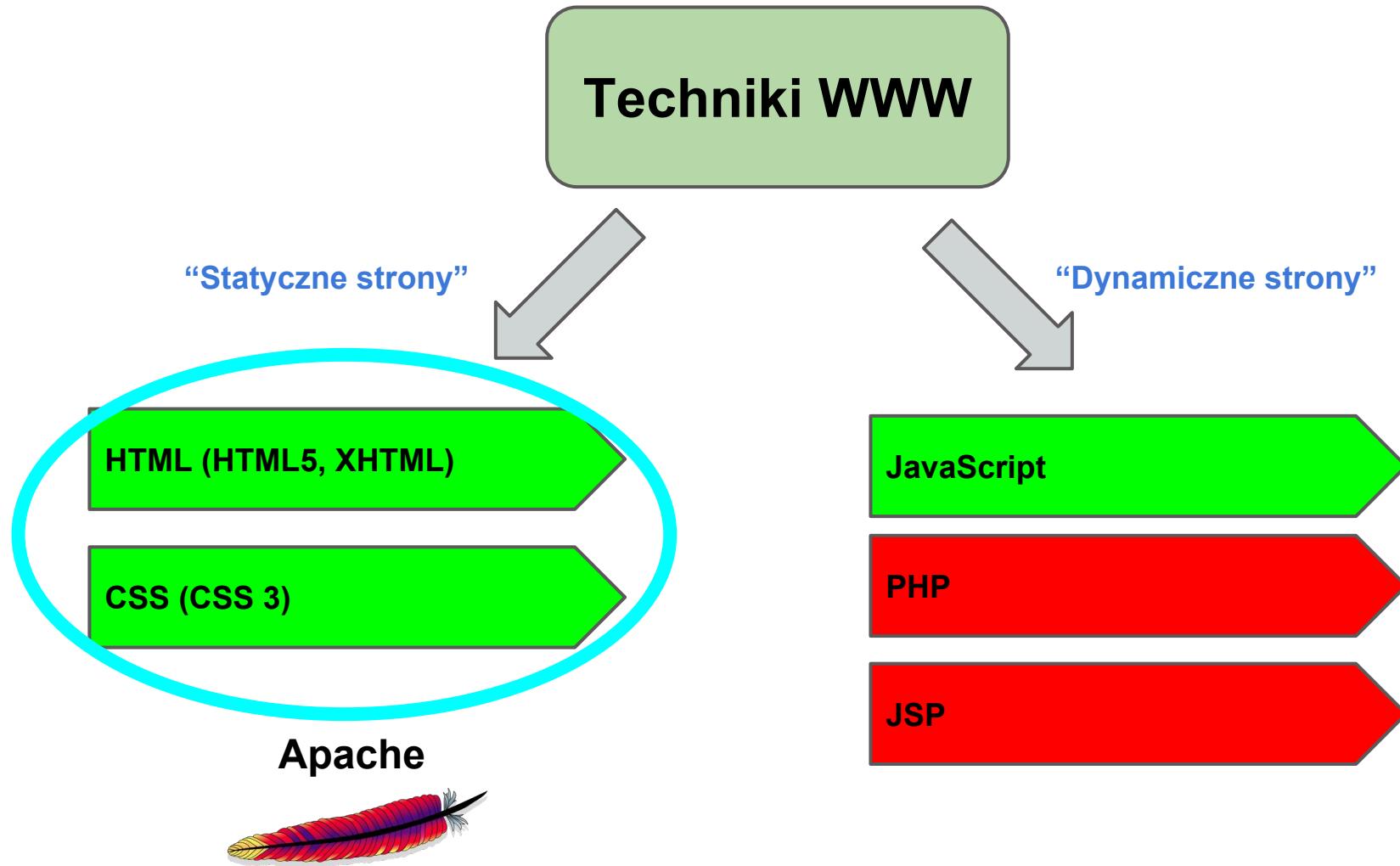
Asynchroniczne żądanie http



Jest to model asynchronicznej komunikacji HTTP, gdzie klient (przeglądarka) nie czeka na przyjście odpowiedzi na żądanie serwera, a wykonuje dalsze żądania.

W takim modelu nie ma konieczności przeładowania strony przy każdej operacji klineta, wystarczy, że zostaną doczytane brakujące dane, a dzięki odpowiednim narzędziom zmodyfikowana zostanie zawartość strony.

Technologie wykorzystane na kursie



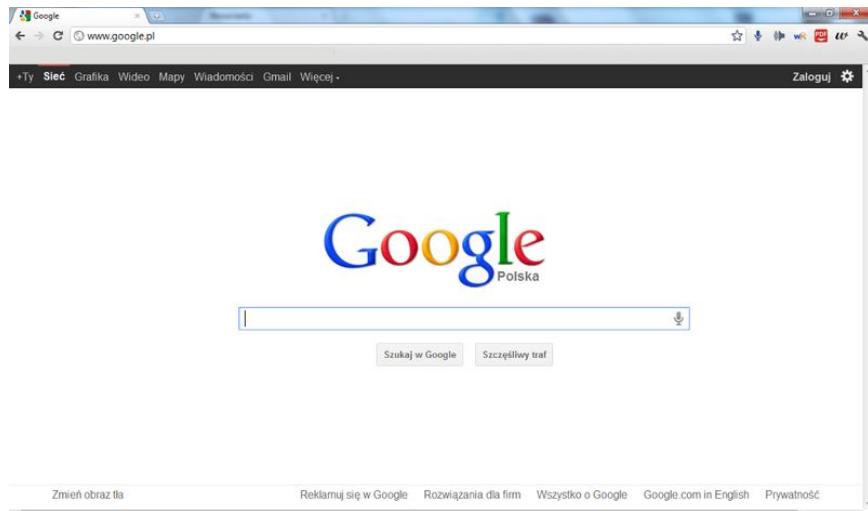


Strona internetowa WWW

Strona Internetowa WWW - kolekcja logicznie połączonych ze sobą zasobów (najczęściej plików napisanych w języku **HTML** oraz multimediarów), znajdujących się na jednym serwerze (w konkretnej kartotece) obsługującym żadania HTTP.

Strona internetowa WWW

Strona Internetowa WWW - kolekcja logicznie połączonych ze sobą zasobów (najczęściej plików napisanych w języku **HTML** oraz multimedii), znajdujących się na jednym serwerze (w konkretnej kartotece) obsługującym żadania HTTP.



<http://www.google.pl>



<http://www.onet.pl>

- Zasoby w ramach jednej strony internetowej są ze sobą połączone za pomocą tzw. hiperłączy które wskazują na URI konkretnego zasobu.
- Pojęcie "*strony internetowej*" (inaczej witryny internetowej) łączy w sobie: dane (treść strony), prezentację (formatowanie) oraz logikę (strukturę).



Strona internetowa WWW

Strona Internetowa WWW - kolekcja logicznie połączonych ze sobą zasobów (najczęściej plików napisanych w języku **HTML** oraz multimedialnych), znajdujących się w konkretnej kartotece na serwerze obsługującym żadania **HTTP**.

```
<!DOCTYPE html>
<html lang="pl">
<head>
<meta charset="utf-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
<meta name="msapplication-starturl" content="http://www.onet.pl/" />
<meta name="msapplication-task" content="name=Wiadomości;action-uri=http://wiadomosci.onet.pl/icon-uri:http://ocdn.eu/static/template-engine/MjU7MDA/_f0d3c1307427ace61799402945edcd72/faviconn2.ico"/>
<meta name="msapplication-task" content="name=Biznes;action-uri=http://biznes.onet.pl/icon-uri:http://ocdn.eu/static/template-engine/MjU7MDA/_f0d3c1307427ace61799402945edcd72/faviconn1.ico"/>
<meta name="msapplication-task" content="name=Sport;action-uri=http://sport.onet.pl/icon-uri:http://ocdn.eu/static/template-engine/MjU7MDA/_f0d3c1307427ace61799402945edcd72/faviconn3.ico"/>
<meta name="msapplication-task" content="name=Film;action-uri=http://film.onet.pl/icon-uri:http://ocdn.eu/static/template-engine/MjU7MDA/_f0d3c1307427ace61799402945edcd72/faviconn4.ico"/>
<meta name="msapplication-task" content="name=Muzyczka;action-uri=http://muzyczka.onet.pl/icon-uri:http://ocdn.eu/static/template-engine/MjU7MDA/_f0d3c1307427ace61799402945edcd72/faviconn5.ico"/>
<meta name="msapplication-task" content="name=VOD;action-uri:http://vod.onet.pl/icon-uri:http://ocdn.eu/static/template-engine/MjU7MDA/_f0d3c1307427ace61799402945edcd72/faviconn6.ico"/>
<meta name="description" content="Dziśaj w Onet.pl: wiadomości z kraju i ze świata; biznes, sport, rozrywka i pogoda. Sprawdź pocztę, bloguj, czatuj i umawiaj się na randki."/>
<title>One informacje, rozrywka, emocje</title>
<link rel="canonical" href="http://www.onet.pl" />
<link rel="dns-prefetch" href="http://ocdn.eu" />
<link rel="dns-prefetch" href="http://m.ocdn.eu" />
<link rel="dns-prefetch" href="http://lib.onet.pl" />
<link rel="dns-prefetch" href="http://csr.onet.pl" />
<link rel="dns-prefetch" href="http://kropka.onet.pl" />
<link rel="dns-prefetch" href="http://google-analytics.com" />
<link rel="dns-prefetch" href="http://reklama.onet.pl" />
<link rel="shortcut icon" href="http://ocdn.eu/static/template-engine/MjU7MDA/_f0d3c1307427ace61799402945edcd72/faviconn.ico" type="image/x-icon" />
<script type="text/javascript">/* <![CDATA[ */
window.suggestmeyes_loaded = true;
/* ]]> */</script>
<link href="http://ocdn.eu/resources/template-engine/mainpagev2templateengine.template-engine/resource/737d8fa66a9b72d454bbda926896b73d/11.55/nsg.css" type="text/css" rel="stylesheet">
<link href="http://ocdn.eu/resources/template-engine/mainpagev2templateengine.template-engine/resource/008f95d4bad5c3563a6389dffaf6ff5/11.55/7.2/menu.css" type="text/css" rel="stylesheet">
<!--[if lt IE 9]>
<link href="http://ocdn.eu/resources/template-engine/mainpagev2templateengine.template-engine/resource/737dfa66a9b72d454bbda926896b73d/11.55/iemq.css" type="text/css" rel="stylesheet">    <![endif]-->
<script src="http://ocdn.eu/resources/template-engine/mainpagev2templateengine.template-engine/resource/none/11.55/nsg-top.min.js" type="text/javascript"></script>
<script type="text/javascript">
    if(NsgStream.requireJS) {
        document.write('<link href="http://ocdn.eu/resources/template-engine/mainpagev2templateengine.template-engine/resource/none/11.55/nsg-stream.css" type="text/css" rel="stylesheet"');
        document.write('<scr'+ipt type="text/javascript" src="http://ocdn.eu/resources/template-engine/mainpagev2templateengine.template-engine/resource/none/11.55/nsg-stream.min.js"></sc'+ript' );
    }
</script>
<!--[if lt IE 9]>
<script src="http://ocdn.eu/static/template-engine/MjU7MDA/_f0d3c1307427ace61799402945edcd72/ie-lt9.js" type="text/javascript"></script>
<![endif]-->
<script type="text/javascript">/* <![CDATA[ */
NSG_AUTOREFRESH = (typeof NSG_AUTOREFRESH == 'undefined' ? false : NSG_AUTOREFRESH);
if (NSG_AUTOREFRESH != true) {
    pp_genius_identifier = 'bPo60Ob5xExczeosfkZZIJa.E.10Rye0gSEhsufRYys3.W7';
}
/* ]]> */</script>
<script type="text/javascript" src="http://ocdn.eu/static/mastx/xgenius.js"></script>
<script type="text/javascript">/* <![CDATA[ */
NSG_KEYWORDS = (typeof NSG_KEYWORDS == 'undefined' ? '' : NSG_KEYWORDS);
NSG_LOCAL = (typeof NSG_LOCAL == 'undefined' ? '' : NSG_LOCAL);
var adSlots = ['flat-search','flat-boks1','flat-link1','flat-link2','flat-link3','flat-link5','flat-link7','flat-link9','flat-link10'];
adSlots.push('flat-sidebarbox');
if(!promoTop && !NsgStream.hashStream) {
    adSlots.unshift('right');
    adSlots.unshift('top');
}
if(!NsgStream.hashStream) {
    adSlots.unshift('flat-boxright5'); adSlots.unshift('flat-boxright6'); adSlots.unshift('flat-boxright9'); adSlots.unshift('flat-boxright10');
    adSlots.unshift('flat-link7'); adSlots.unshift('flat-link8'); adSlots.unshift('flat-link12'); adSlots.unshift('flat-link13'); adSlots.unshift('flat-sidebarbox');
}
onetAds={
```



Technologie teraz i dawniej

Strony internetowe dawniej i obecnie są tworzone z wykorzystaniem języka

HTML (Hyper Text Markup Language)

[początki sięgają 1980 roku]

Z biegiem czasu sam język znaczników HTML przestał spełniać oczekiwania co do tworzenia serwisów WWW i w trakcie ostatnich 20 lat wokół technologii “webowych”, powstały różne narzędzia pozwalające na uatrakcyjnienie wyglądu stron internetowych.

Technologie teraz i dawniej

Strony internetowe dawniej i obecnie są tworzone z wykorzystaniem języka

HTML (Hyper Text Markup Language)

[początki sięgają 1980 roku]

Z biegiem czasu sam język znaczników HTML przestał spełniać oczekiwania co do tworzenia serwisów WWW i w trakcie ostatnich 20 lat wokół technologii “webowych”, powstały różne narzędzia pozwalające na uatrakcyjnienie wyglądu stron internetowych.

1

Pierwszym narzędziem, które pozwalało na zmianę wyglądu strony były Kaskadowe Arkusze Stylu (CSS), które powstały w roku 1994. Umożliwiały nadawanie odpowiednim elementom zapisanym w języku HTML nowych cech (np. zmiana koloru czcionki).

Technologie teraz i dawniej

Strony internetowe dawniej i obecnie są tworzone z wykorzystaniem języka

HTML (Hyper Text Markup Language)

[początki sięgają 1980 roku]

Z biegiem czasu sam język znaczników HTML przestał spełniać oczekiwania co do tworzenia serwisów WWW i w trakcie ostatnich 20 lat wokół technologii “webowych”, powstały różne narzędzia pozwalające na uatrakcyjnienie wyglądu stron internetowych.

- 1** Pierwszym narzędziem, które pozwalało na zmianę wyglądu strony były Kaskadowe Arkusze Stylu (CSS), które powstały w roku 1994. Umożliwiały nadawanie odpowiednim elementom zapisanym w języku HTML nowych cech (np. zmiana koloru czcionki).
- 2** W tym samym czasie, w roku 1994, rozpoczęto pracę nad nowym językiem PHP (Personal Home Page/Forms Interpreter), który pozwalał na generowanie w czasie rzeczywistym kodu w języku HTML bazując na stworzonych przez użytkownika skryptach.

Technologie teraz i dawniej

Strony internetowe dawniej i obecnie są tworzone z wykorzystaniem języka

HTML (Hyper Text Markup Language)

[początki sięgają 1980 roku]

Z biegiem czasu sam język znaczników HTML przestał spełniać oczekiwania co do tworzenia serwisów WWW i w trakcie ostatnich 20 lat wokół technologii “webowych”, powstały różne narzędzia pozwalające na uatrakcyjnienie wyglądu stron internetowych.

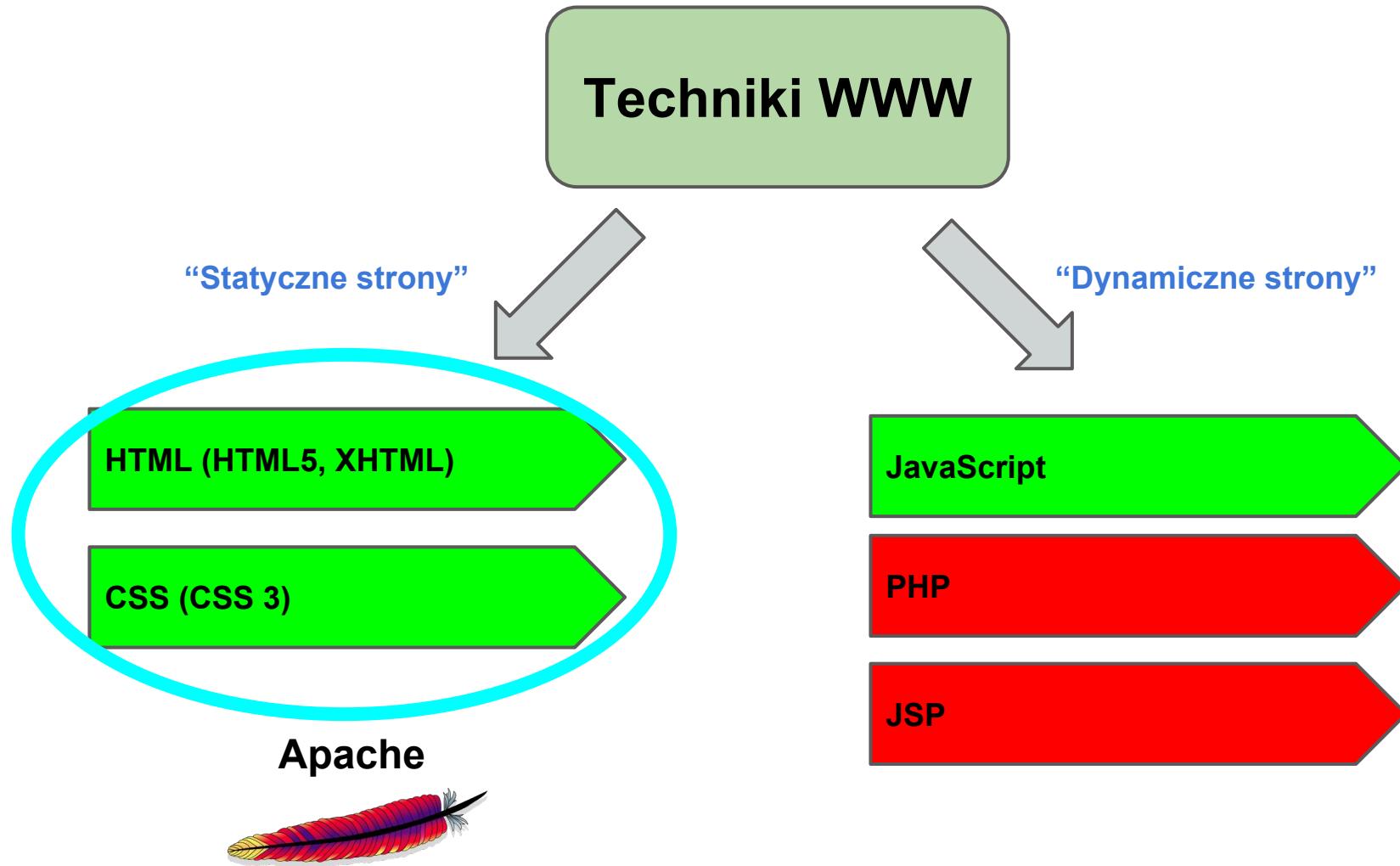
- 1** Pierwszym narzędziem, które pozwalało na zmianę wyglądu strony były Kaskadowe Arkusze Stylu (CSS), które powstały w roku 1994. Umożliwiały nadawanie odpowiednim elementom zapisanym w języku HTML nowych cech (np. zmiana koloru czcionki).
- 2** W tym samym czasie, w roku 1994, rozpoczęto pracę nad nowym językiem PHP (Personal Home Page/Forms Interpreter), który pozwalał na generowanie w czasie rzeczywistym kodu w języku HTML bazując na stworzonych przez użytkownika skryptach.
- 3** Rok później firma Netscape rozpoczęła pracę nad językiem skryptowym JavaScript, który miał umożliwiać podniesienie atrakcyjności strony oraz wprowadzenie do niej interaktywności.

Strony statyczne vs. dynamiczne

Statyczna strona WWW - strona taka zawiera w kodzie dane, które są wyświetlane w przeglądarce internetowej. Zawartość strony nie zmienia się pod wpływem interakcji z użytkownikiem, zawsze wyświetlane są te same treści. Każda zmiana danych (treści strony) wymaga ingerencji programisty w kod strony. Użytkownik nie posiadający wiedzy na temat struktury strony oraz języka HTML nie będzie w stanie zmieniać zawartości strony.

Dynamiczna strona WWW - strona dynamiczna jest generowana przez serwer HTTP pod wpływem żądań przychodzących od klienta na podstawie przesyłanych parametrów i zmiennych. Strony takie dostosowują swoją zawartość w zależności od działań użytkownika w przeglądarce. W takim wypadku zmiany stanu strony mogą być wykonywane po stronie użytkownika dzięki wykorzystanie języków skryptowych np. JavaScript lub po stronie serwera wykorzystując do tego celu języki programowania takie jak PHP, Perl, Python. W nowoczesnych serwisach internetowych wykorzystywane są obie metody. Dodatkowo najczęściej dane (czyli zawartość strony) jest przechowywana w bazach danych (np. SQL), z których są pobierane w zależności od wywoływanych przez użytkownika żądań.

Technologie wykorzystane na kursie



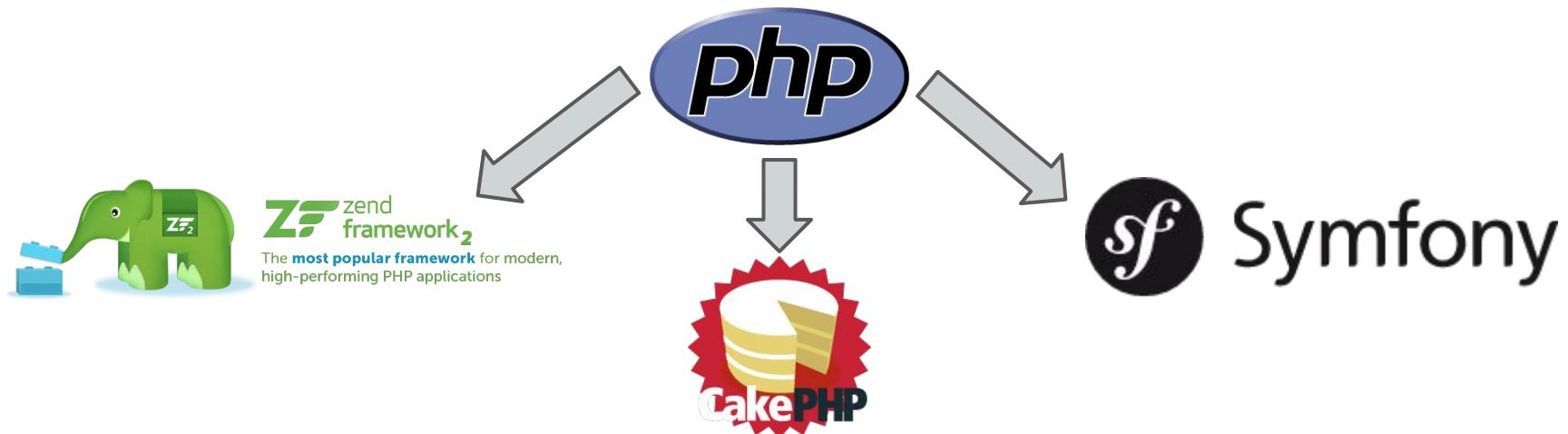
Przegląd dostępnych rozwiązań

Najpopularniejsze środowiska programistyczne:



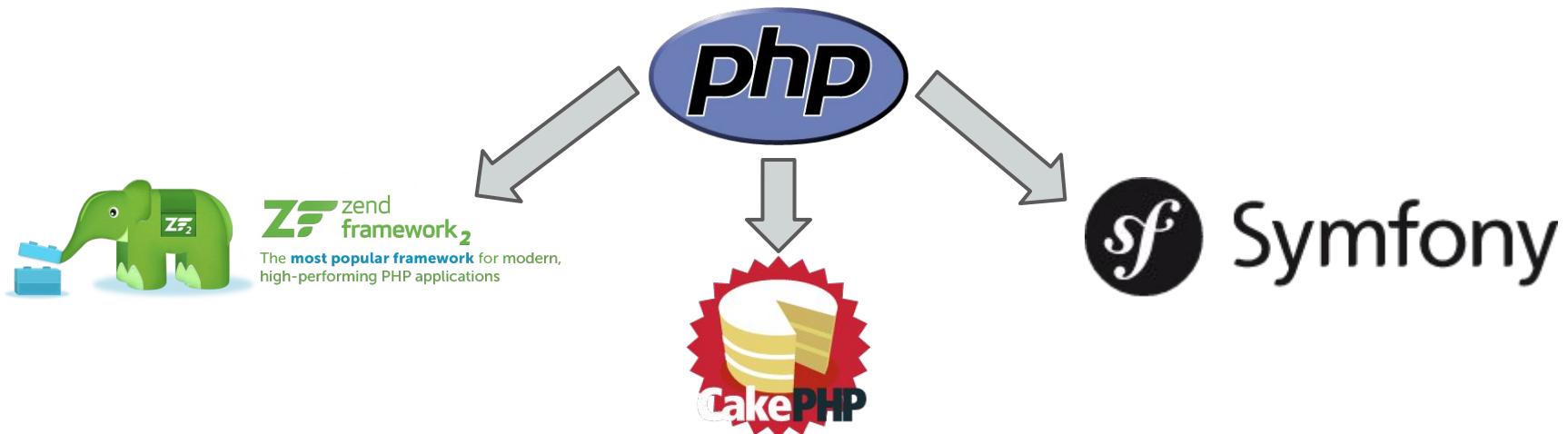
Przegląd dostępnych rozwiązań

Najpopularniejsze środowiska programistyczne:



Przegląd dostępnych rozwiązań

Najpopularniejsze środowiska programistyczne:



oraz systemy CMS (Content Manager System):

Drupal™

PHP FUSION

Joomla!™

WORDPRESS

PHP nuke

MediaWiki
Because ideas want to be free.



UNIWERSYTET
JAGIELŁOŃSKI
W KRAKOWIE

Dlaczego NODE.JS a nie PHP

Dlaczego NODE.JS a nie PHP



vs.



Dlaczego NODE.JS a nie PHP



vs.



Obecnie PHP jest wykorzystywane na około 75% wszystkich stron internetowych.

Dlaczego NODE.JS a nie PHP



vs.



Obecnie PHP jest wykorzystywane na około 75% wszystkich stron internetowych.

Ale większość nowych projektów odchodzi od PHP na rzecz NODE.JS jako rozwiązania bardziej wydajnego, elastycznego i niezależnego.

Node.JS jest środowiskiem programistycznym bazującym na języku JavaScript, instalowanym na serwerze. Jego serce stanowi interpreter (silnik) V8 stworzony przez firmę Google.

Daje możliwości tworzenia łatwo skalowalnych aplikacji internetowych, których działanie jest sterowane zdarzeniowo wykorzystując żądania asynchroniczne czyli bez konieczności przeładowania danego zasobu.

Dlaczego NODE.JS a nie PHP

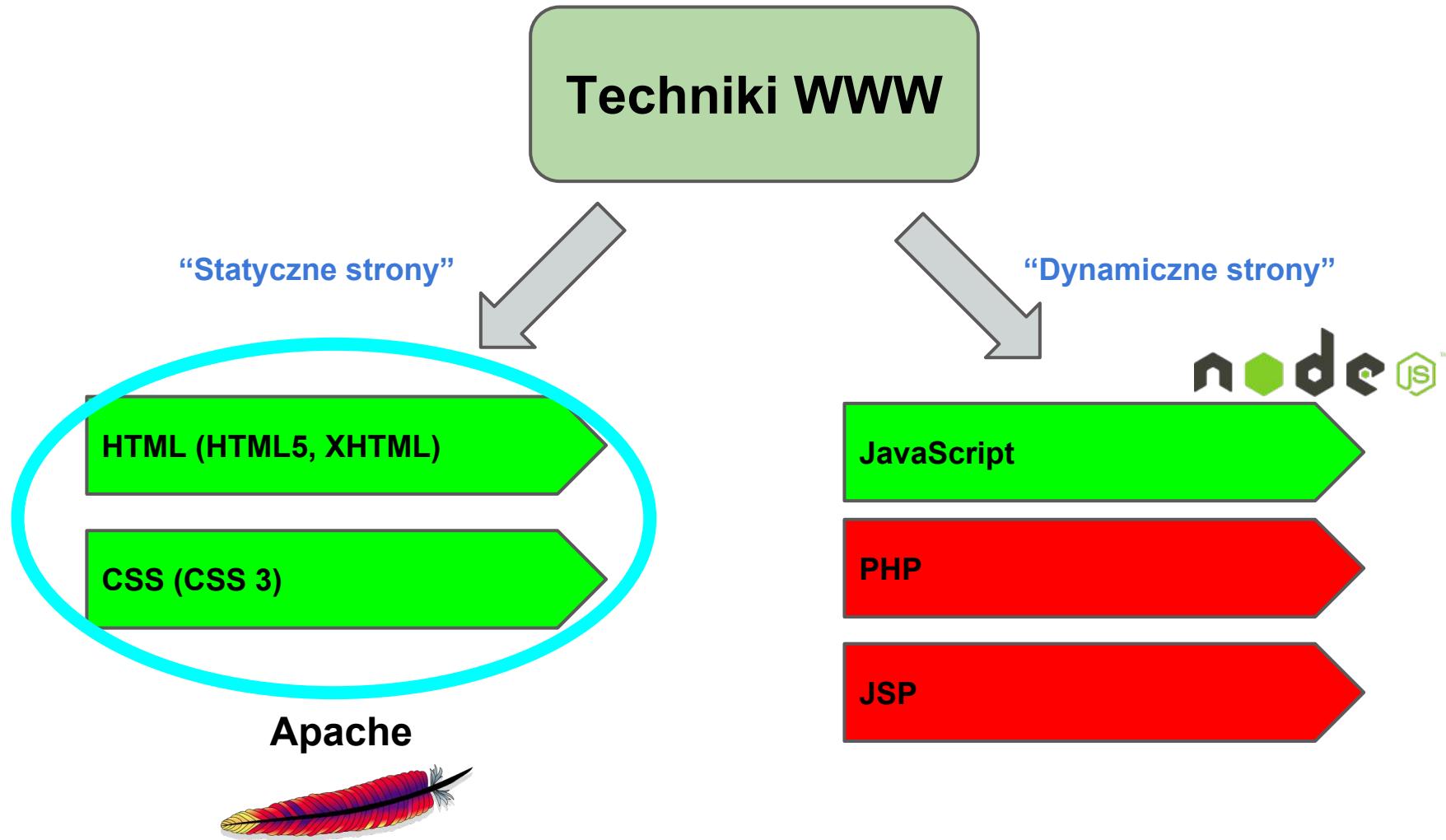


vs.

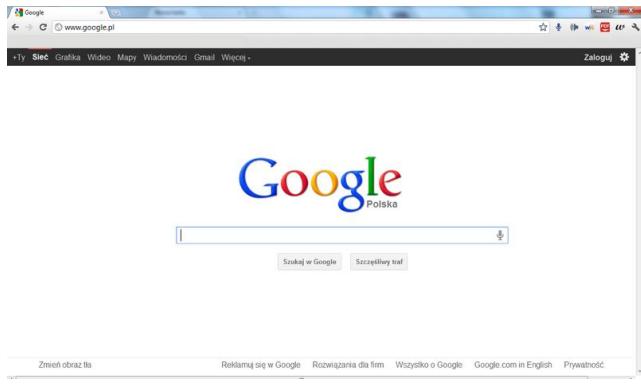


Więcej szczegółów o NODE.JS na kolejnych wykładach

Technologie wykorzystane na kursie



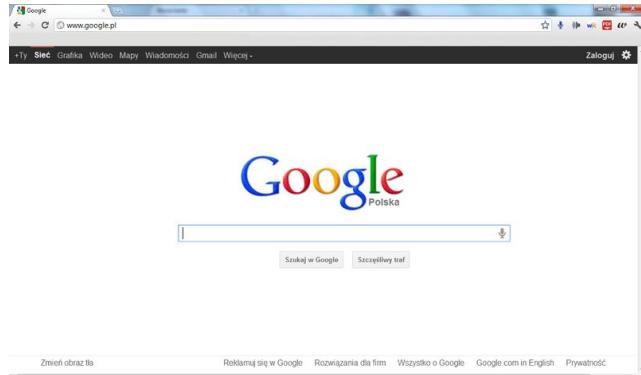
Front-end a Back-end



FRONT-END - pojęciowo odnosi się w technologiach internetowych do kodu **wykonywanego po stronie użytkownika**. Wogólności do tej kategorii można zaliczyć HTML, CSS oraz JavaScript.



Front-end a Back-end



FRONT-END - pojęciowo odnosi się w technologiach internetowych do kodu **wykonywanego po stronie użytkownika**. Wogólności do tej kategorii można zaliczyć HTML, CSS oraz JavaScript.



BACK-END - pojęciowo odnosi się w technologiach internetowych do kodu **wykonywanego po stronie serwera**. W ogólności do tej kategorii można zaliczyć PHP, NODE.JS, Perl, CGI, Ruby





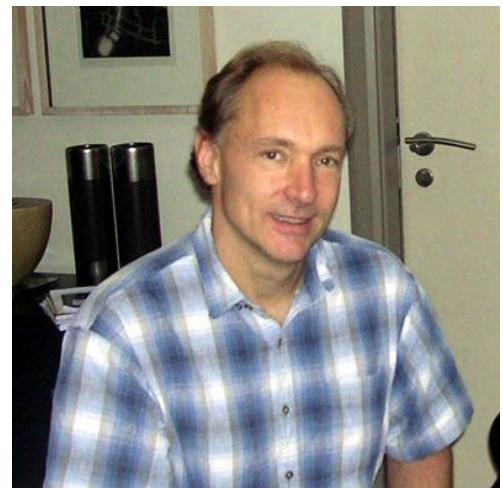
HTML - początek

HTML (Hyper Text Markup Language) - powstał na bazie zasad języka SGML (Standard Generalized Markup Language), który jest pewnego rodzaju “meta-językiem” służącym do definiowania znaczników i zasad ich iżytowania.

HTML - początek

HTML (Hyper Text Markup Language) - powstał na bazie zasad języka SGML (Standard Generalized Markup Language), który jest pewnego rodzaju “meta-językiem” służącym do definiowania znaczników i zasad ich iżytkowania.

Pierwsza wersja HTML została opracowana przez Tim'a Bernersa-Lee fizyka pracującego w CERN i została opublikowana w 1991 roku.
Pierwsza propozycja zawierała 22 znaczniki.



Przewodniczący konsorcjum
W3C
(www.w3c.org)



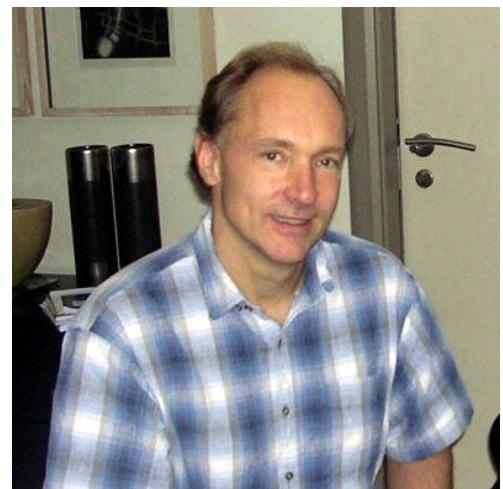
HTML - początek

HTML (Hyper Text Markup Language) - powstał na bazie zasad języka SGML (Standard Generalized Markup Language), który jest pewnego rodzaju “meta-językiem” służącym do definiowania znaczników i zasad ich iżytkowania.

Pierwsza wersja HTML została opracowana przez Tima Bernersa-Lee fizyka pracującego w CERN i została opublikowana w 1991 roku.
Pierwsza propozycja zawierała 22 znaczniki.

Podstawowe cechy standardu HTML:

- **hipertekstowość** - za pomocą znaczników można tworzyć hiperpołączenia (hiperlinki), które wskazują na inne zasoby w obrębie tej samej witryny lub dowolnej innej zamieszczone w sieci. Dzięki hiperpołączeniom można w łatwy sposób poruszać się po stronach.
- **uniwersalność** - dokument zapisany w standardzie HTML jest zwykłym plikiem tekstowym (ASCII), więc może być stworzony na dowolnej platformie systemowej bez żadnego specjalistycznego i dedykowanego oprogramowania.



Przewodniczący konsorcjum
W3C
(www.w3c.org)





HTML - początek

HTML (Hyper Text Markup Language) - powstał na bazie zasad języka SGML (Standard Generalized Markup Language), który jest pewnego rodzaju “meta-językiem” służącym do definiowania znaczników i zasad ich iżytkowania.

Ostatni obowiązujący standard języka HTML opublikowany przez konsorcjum W3, oznaczony jest numerem 4.01 i powstał w 1999 roku. W wersji tej zostały uporządkowane i ujednolicone dotychczas niekompatybilne ze sobą wersje HTML w oparciu o standard zapisu SGML.

HTML - początek

HTML (Hyper Text Markup Language) - powstał na bazie zasad języka SGML (Standard Generalized Markup Language), który jest pewnego rodzaju “meta-językiem” służącym do definiowania znaczników i zasad ich iżytkowania.

Ostatni obowiązujący standard języka HTML opublikowany przez konsorcjum W3, oznaczony jest numerem 4.01 i powstał w 1999 roku. W wersji tej zostały uporządkowane i ujednolicone dotychczas niekompatybilne ze sobą wersje HTML w oparciu o standard zapisu SGML.

Drugim obowiązującym standardem skonstruowanym na bazie HTML 4.01 oraz w oparciu o zasady języka XML (Extensible Markup Language) jest standard XHTML 1.1. Standard ten łączy zalety HTML oraz XML dając ostatecznie specyfikę o funkcjonalnościach XML. Jedną z zalet takiego rozwiązania jest możliwość łączenia XHTML z innymi językami opartymi o XML oraz można do ich przetwarzania używać transformatora XSLT (*Extensible Stylesheet Language Transformations*).

HTML - początek

HTML (Hyper Text Markup Language) - powstał na bazie zasad języka SGML (Standard Generalized Markup Language), który jest pewnego rodzaju “meta-językiem” służącym do definiowania znaczników i zasad ich iżytkowania.

Ostatni obowiązujący standard języka HTML opublikowany przez konsorcjum W3, oznaczony jest numerem 4.01 i powstał w 1999 roku. W wersji tej zostały uporządkowane i ujednolicone dotychczas niekompatybilne ze sobą wersje HTML w oparciu o standard zapisu SGML.

Drugim obowiązującym standardem skonstruowanym na bazie HTML 4.01 oraz w oparciu o zasady języka XML (Extensible Markup Language) jest standard XHTML 1.1. Standard ten łączy zalety HTML oraz XML dając ostatecznie specyfikę o funkcjonalnościach XML. Jedną z zalet takiego rozwiązania jest możliwość łączenia XHTML z innymi językami opartymi o XML oraz można do ich przetwarzania używać transformatora XSLT (*Extensible Stylesheet Language Transformations*).

Różnica pomiędzy HTML i XHTML jest z pozoru jest nieznaczna, natomiast istotnie standardy te różnią się z punktu widzenia systematyzacji informacji, np. strona niepoprawnie przygotowana w języku XHTML nie powinna być wyświetlona podobie jak to jest z XML-em. Natomiast HTML zostanie wyświetlony mimo błędów.



Znacznik

Znacznik (tag)- pojęcie w informatyce odnoszące się do klasyfikowania dowolnych informacji, ale najczęściej tekstu. Znaczniki nie są elementami żadnego języka programowania, określają działania które mają zostać wykonane w stosunku do zasobu jaki kwalifikują. Najczęściej są to działania które informują jak dana informacja ma zostać zaprezentowana.

W dziedzinie tworzenia stron internetowych w standardach HTML i XHTML znacznik to tekst umieszczony pomiędzy znakiem większości i znakiem mniejszości:

<znacznik>



Znacznik

Znacznik (tag)- pojęcie w informatyce odnoszące się do klasyfikowania dowolnych informacji, ale najczęściej tekstu. Znaczniki nie są elementami żadnego języka programowania, określają działania które mają zostać wykonane w stosunku do zasobu jaki kwalifikują. Najczęściej są to działania które informują jak dana informacja ma zostać zaprezentowana.

W dziedzinie tworzenia stron internetowych w standardach HTML i XHTML znacznik to tekst umieszczony pomiędzy znakiem większości i znakiem mniejszości:

<znacznik>

Znaczniki w innych dziedzinach np.: LaTeX:

```
\begin{center}  
\huge{Tytuł dokumentu}  
\end{center}
```



Znacznik

<znacznik>

Najczęściej znacznik służy do kwalifikacji obszaru tekstu (w oogołości dowolnej informacji). Kwalifikacja polega na umieszczeniu tekstu pomiędzy znacznikiem otwierającym i znacznikiem zamykającym:

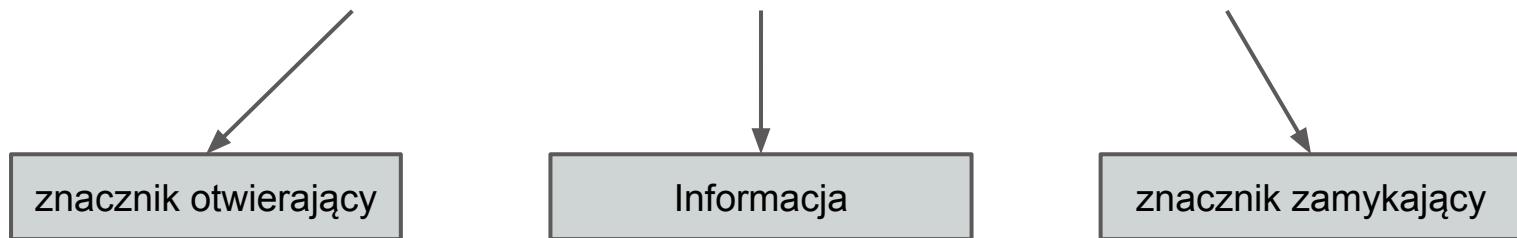
<znacznik> Tekst </znacznik>

Znacznik

<znacznik>

Najczęściej znacznik służy do kwalifikacji obszaru tekstu (w ogólności dowolnej informacji). Kwalifikacja polega na umieszczeniu tekstu pomiędzy znacznikiem otwierającym i znacznikiem zamykającym:

<znacznik> Tekst </znacznik>





Znacznik

<znacznik>

Czasem znacznik spełnia tylko rolę określenia układu w dokumencie i wtedy nie obejmuje swoim zasięgiem żadnej informacji, jest to tzw. znacznik “pusty”

<znacznik/>

Znacznik

<znacznik>

Czasem znacznik spełnia tylko rolę określenia układu w dokumencie i wtedy nie obejmuje swoim zasięgiem żadnej informacji, jest to tzw. znacznik "pusty"

<znacznik/>



Znacznik pusty

Uwaga:

- według standardu HTML 4.01 nie trzeba pisać "ukośnika w znaczniku pustym.
- w XHTML znacznik musi być zamknięty, inaczej jest to błąd składniowy.
- ze względu na wstępczą kompatybilność zaleca się przed ukośnikiem w znaku pustym stawiać spację
 .

Znacznik

<znacznik>

Znacznik poza słowem kluczowym definiującym jego cechę, może zawierać różne atrybuty modyfikujące jego podstawowe własności. Atrybuty są definiowane w znaczniku otwierającym w następujący sposób:

```
<znacznik atrybut1="wartość">  
Tekst  
</znacznik>
```

Znacznik

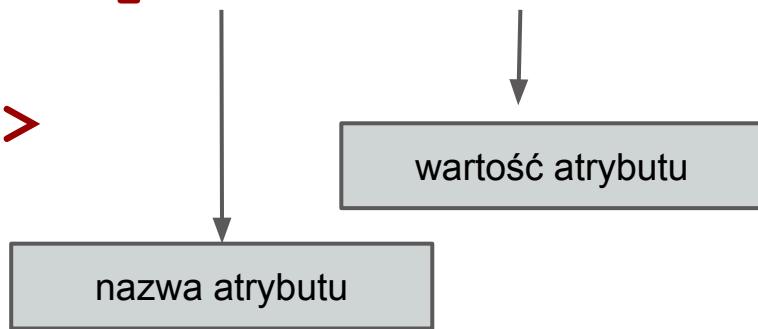
<znacznik>

Znacznik poza słowem kluczowym definiującym jego cechę, może zawierać różne atrybuty modyfikujące jego podstawowe własności. Atrybuty są definiowane w znaczniku otwierającym w następujący sposób:

<znacznik atrybut1="wartość">

Tekst

</znacznik>



Uwaga:

- nazwy elementów (znaczników) oraz atrybutów według standardu XHTML powinny być pisane małymi literami.

Nazwy plików

Jeśli tworzymy dokument hipertekstowy to dla poprawnej interpretacji jego zawartości przez przeglądarki oraz dla identyfikacji pliku przez serwer i system operacyjny zaleca się aby plikom nadawać rozszerzenia “[.html](#)” lub “[htm](#)”.

Dodatkowo jako, że pliki takie stają się częścią zasobów internetowych i są identyfikowane przez adresy URL, zaleca się aby konsekwentnie w nazwach plików używać **małych liter**.



Nazwy plików

Jeśli tworzymy dokument hipertekstowy to dla poprawnej interpretacji jego zawartości przez przeglądarki oraz dla identyfikacji pliku przez serwer i system operacyjny zaleca się aby plikom nadawać rozszerzenia “**.html**” lub “**htm**”.

Dodatkowo jako, że pliki takie stają się częścią zasobów internetowych i są identyfikowane przez adresy URL, zaleca się aby konsekwentnie w nazwach plików używać **małych liter**.



index.html

index.htm

mojastrona.html

twojastrona.htm



GLOWNA.HTML

PODSTRONA.html

podpodstrona.HTML



Uwaga:

- gdy przygotowujemy dokument tekstowy, dobrą praktyką jest robienie “wcięć” w kodzie,
- przeglądarki ignorują zbędne spacje oraz przejścia do nowej linii robione za pomocą klawisza ENTER.

Znaczniki podstawowe

Nazwy elementów zapisanych w znacznikach w języku HTML i XHTML wywodzą się od słów z języka angielskiego.

Wszystkie dokumenty hipertekstowe nizależnie od używanego standardu , posiadają trzy podstawowe znaczniki:

- <html> </html>** - znacznik wskazujący na dokument hipertekstowy (otacza wszystkie pozostałe znaczniki)
- <head> </head>** - znacznik wskazuje na nagłówek dokumentu hipertekstowego (zasadniczo znajduje się na jego początku).
- <body> </body>** - znacznik wskazuje na część główną dokumentu hipertekstowego, w niej znajdują się wszystkie inne znaczniki związane z prezentowaniem danych.



Znaczniki podstawowe

Nazwy elementów zapisanych w znacznikach w języku HTML i XHTML wywodzą się od słów z języka angielskiego.

Wszystkie dokumenty hipertekstowe nizależnie od używanego standardu , posiadają trzy podstawowe znaczniki:

- `<html> </html>` - znacznik wskazujący na dokument hipertekstowy (otacza wszystkie pozostałe znaczniki)
- `<head> </head>` - znacznik wskazuje na nagłówek dokumentu hipertekstowego (zasadniczo znajduje się na jego początku).
- `<body> </body>` - znacznik wskazuje na część główną dokumentu hipertekstowego, w niej znajdują się wszystkie inne znaczniki związane z prezentowaniem danych.



Wałąną częścią każdego języka jest możliwość dodawania komentarzy. W języku HTML, komentarz jest zaznaczany w następujący sposób:

`<!-- Komentarz -->`

W komentarzu możemy umieścić zwykły tekst lub inne znaczniki. Zakomentowane znaczniki nie będą wyświetlane przez przeglądarkę w trakcie wczytywania strony.

Znaczniki podstawowe

Nazwy elementów zapisanych w znacznikach w języku HTML i XHTML wywodzą się od słów z języka angielskiego.

Wszystkie dokumenty hipertekstowe nizależnie od używanego standardu , posiadają trzy podstawowe znaczniki:

- | | |
|---|---|
| <code><html> </html></code> | - znacznik wskazujący na dokument hipertekstowy (otacza wszystkie pozostałe znaczniki) |
| <code><head> </head></code> | - znacznik wskazuje na nagłówek dokumentu hipertekstowego (zasadniczo znajduje się na jego początku). |
| <code><body> </body></code> | - znacznik wskazuje na część główną dokumentu hipertekstowego, w niej znajdują się wszystkie inne znaczniki związane z prezentowaniem danych. |



Ważną częścią każdego języka jest możliwość dodawania komentarzy. W języku HTML, komentarz jest zaznaczany w następujący sposób:

`<!-- Komentarz -->`

UWAGA: Ale komentarz cały czas jest widoczny w kodzie strony i można go odczytać mimo nie wyświetlania go przez przeglądarkę.

Podstawowa struktura dokumentu

<!DOCTYPE ... >	deklaracja typu dokumentu
<html>	początek dokumentu hipertekstowego
<head>	sekcja nagłówkowa
<title> Tytuł strony </title>	tytuł strony
</head>	
<body> Treść strony </body>	sekcja zawartości strony
</html>	koniec dokumentu hipertekstowego

Struktura hierarchiczna z zagnieżdżeniami



UNIWERSYTET
JAGIELŁOŃSKI
W KRAKOWIE

Zaawansowane Techniki WWW (HTML, CSS i JavaScript)

Dr inż. Marcin Zieliński

Środa 15:30 - 17:00 sala: A-1-04

WYKŁAD 3

Wykład dla kierunku: Informatyka Stosowana II rok

Rok akademicki: 2015/2016 - semestr zimowy



Przypomnienie

Serwery WWW (Apache, Nginex, MII, GWS)

Wprowadzenie do HTML

Elementy strony WWW

Znacznik

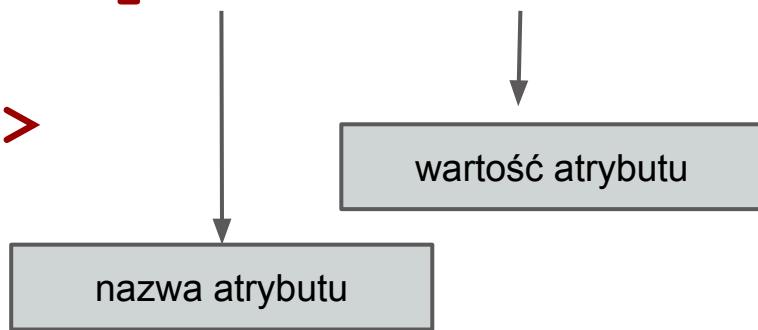
<znacznik>

Znacznik poza słowem kluczowym definiującym jego cechę, może zawierać różne atrybuty modyfikujące jego podstawowe własności. Atrybuty są definiowane w znaczniku otwierającym w następujący sposób:

<znacznik atrybut1="wartość">

Tekst

</znacznik>



Uwaga:

- nazwy elementów (znaczników) oraz atrybutów według standardu XHTML powinny być pisane małymi literami.

Nazwy plików

Jeśli tworzymy dokument hipertekstowy to dla poprawnej interpretacji jego zawartości przez przeglądarki oraz dla identyfikacji pliku przez serwer i system operacyjny zaleca się aby plikom nadawać rozszerzenia “[.html](#)” lub “[htm](#)”.

Dodatkowo jako, że pliki takie stają się częścią zasobów internetowych i są identyfikowane przez adresy URL, zaleca się aby konsekwentnie w nazwach plików używać **małych liter**.



Znaczniki podstawowe

Nazwy elementów zapisanych w znacznikach w języku HTML i XHTML wywodzą się od słów z języka angielskiego.

Wszystkie dokumenty hipertekstowe nizależnie od używanego standardu , posiadają trzy podstawowe znaczniki:

- | | |
|---|---|
| <code><html> </html></code> | - znacznik wskazujący na dokument hipertekstowy (otacza wszystkie pozostałe znaczniki) |
| <code><head> </head></code> | - znacznik wskazuje na nagłówek dokumentu hipertekstowego (zasadniczo znajduje się na jego początku). |
| <code><body> </body></code> | - znacznik wskazuje na część główną dokumentu hipertekstowego, w niej znajdują się wszystkie inne znaczniki związane z prezentowaniem danych. |



Znaczniki podstawowe

Nazwy elementów zapisanych w znacznikach w języku HTML i XHTML wywodzą się od słów z języka angielskiego.

Wszystkie dokumenty hipertekstowe nizależnie od używanego standardu , posiadają trzy podstawowe znaczniki:

- `<html> </html>` - znacznik wskazujący na dokument hipertekstowy (otacza wszystkie pozostałe znaczniki)
- `<head> </head>` - znacznik wskazuje na nagłówek dokumentu hipertekstowego (zasadniczo znajduje się na jego początku).
- `<body> </body>` - znacznik wskazuje na część główną dokumentu hipertekstowego, w niej znajdują się wszystkie inne znaczniki związane z prezentowaniem danych.



Wałąną częścią każdego języka jest możliwość dodawania komentarzy. W języku HTML, komentarz jest zaznaczany w następujący sposób:

`<!-- Komentarz -->`

W komentarzu możemy umieścić zwykły tekst lub inne znaczniki. Zakomentowane znaczniki nie będą wyświetlane przez przeglądarkę w trakcie wczytywania strony.

Znaczniki podstawowe

Nazwy elementów zapisanych w znacznikach w języku HTML i XHTML wywodzą się od słów z języka angielskiego.

Wszystkie dokumenty hipertekstowe nizależnie od używanego standardu , posiadają trzy podstawowe znaczniki:

- | | |
|---|---|
| <code><html> </html></code> | - znacznik wskazujący na dokument hipertekstowy (otacza wszystkie pozostałe znaczniki) |
| <code><head> </head></code> | - znacznik wskazuje na nagłówek dokumentu hipertekstowego (zasadniczo znajduje się na jego początku). |
| <code><body> </body></code> | - znacznik wskazuje na część główną dokumentu hipertekstowego, w niej znajdują się wszystkie inne znaczniki związane z prezentowaniem danych. |



Ważną częścią każdego języka jest możliwość dodawania komentarzy. W języku HTML, komentarz jest zaznaczany w następujący sposób:

`<!-- Komentarz -->`

UWAGA: Ale komentarz cały czas jest widoczny w kodzie strony i można go odczytać mimo nie wyświetlania go przez przeglądarkę.

Podstawowa struktura dokumentu

<!DOCTYPE ... >	deklaracja typu dokumentu
<html>	początek dokumentu hipertekstowego
<head>	sekcja nagłówkowa
<title> Tytuł strony </title>	tytuł strony
</head>	
<body> Treść strony </body>	sekcja zawartości strony
</html>	koniec dokumentu hipertekstowego

Struktura hierarchiczna z zagnieżdżeniami

Deklaracja typu dokumentu

<!DOCTYPE ... >

- Zazwyczaj w początkowym etapie prac projektowych, ten atrybut dokumentu hipertekstowego jest pomijany. Chodzi o pierwszą deklarację dokumentu, określana znacznikiem:
- **<!DOCTYPE ...>**
- i definiującą, którą wersją języka **XHTML** jest w danym dokumencie użyta. Pełna nazwa tej deklaracji brzmi „Document type Definition” (w skrócie DTD lub Doctype).
- **Uwaga! Deklaracja typu dokumentu poprzedza inne znaczniki.**
- Ta informacja jest niezbędna i podaje, którą wersję języka **XHTML** dokument używa tak, by przeglądarka i pozostałe elementy oprogramowania mogły właściwie zinterpretować cały dokument hipertekstowy. Użycie tej deklaracji zwiększa również szybkość ładowania strony WWW. Więcej na ten temat można przeczytać na stronie WWW Konsorcjum W3C: <http://www.w3.org/QA/Tips/Doctype>.
- Dobre edytory kodu (np. HTML Kit) generują szablon strony, łącznie z definicją typu dokumentu w różnych wersjach.

Deklaracja typu dokumentu

<!DOCTYPE ... >

- Zazwyczaj w początkowym etapie prac projektowych, ten atrybut dokumentu hipertekstowego jest pomijany. Chodzi o pierwszą deklarację dokumentu, określana znacznikiem:
- **<!DOCTYPE ...>**
- i definiującą, którą wersją języka **XHTML** jest w danym dokumencie użyta. Pełna nazwa tej deklaracji brzmi „Document type Definition” (w skrócie DTD lub Doctype).
- **Uwaga! Deklaracja typu dokumentu poprzedza inne znaczniki.**
- Ta informacja jest niezbędna i podaje, którą wersję języka **XHTML** dokument używa tak, by przeglądarka i pozostałe elementy oprogramowania mogły właściwie zinterpretować cały dokument hipertekstowy. Użycie tej deklaracji zwiększa również szybkość ładowania strony WWW. Więcej na ten temat można przeczytać na stronie WWW Konsorcjum W3C: <http://www.w3.org/QA/Tips/Doctype>.
- Dobre edytory kodu (np. HTML Kit) generują szablon strony, łącznie z definicją typu dokumentu w różnych wersjach.

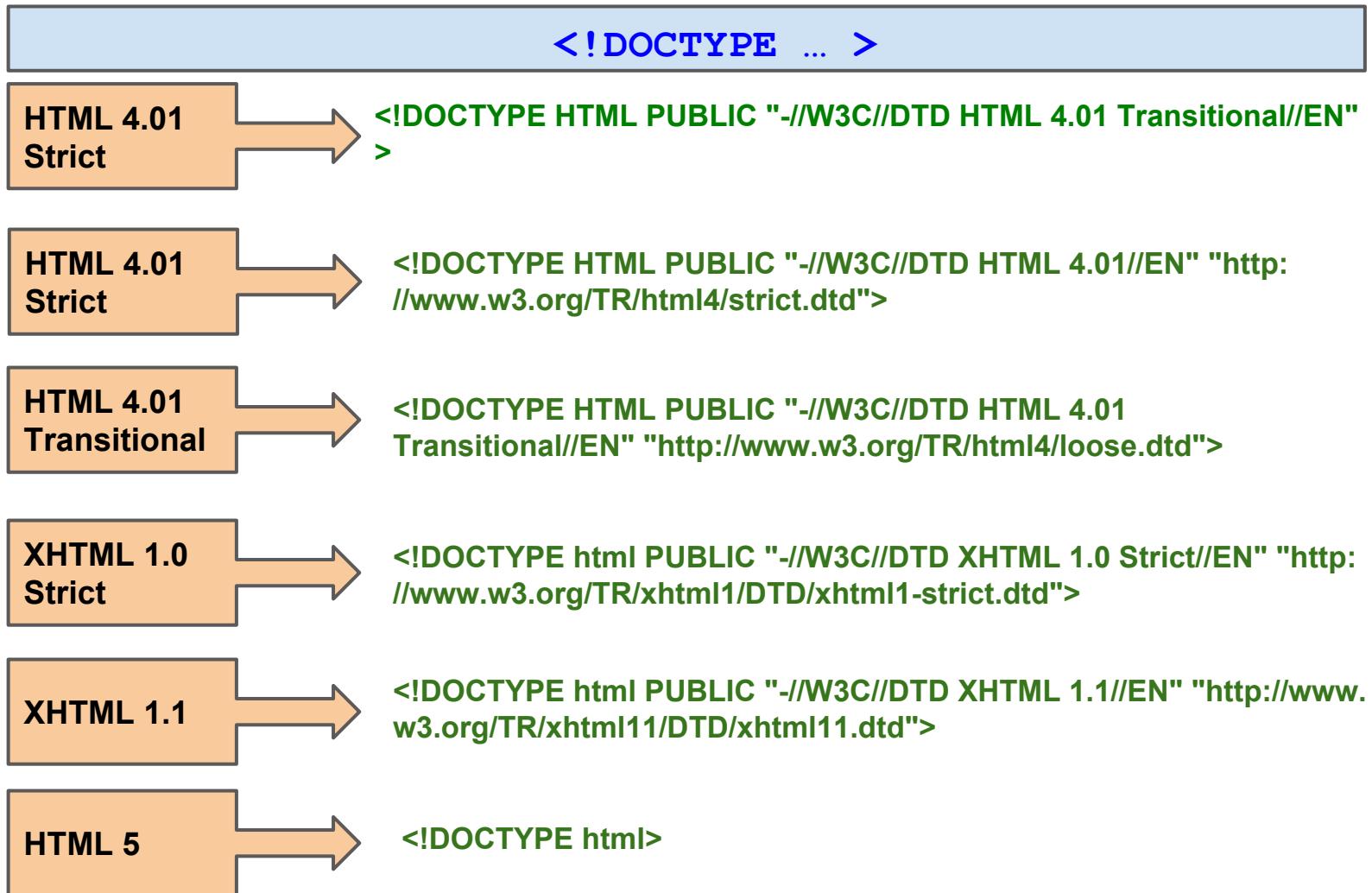
HTML 4.01

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<!DOCTYPE html>

HTML 5

Deklaracja typu dokumentu



Deklaracja typu dokumentu

<!DOCTYPE ... >

HTML 4.01
Strict

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

HTML 4.01
Strict

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

HTML 4.01
Transitional

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

XHTML 1.0
Strict

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

XHTML 1.1

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

HTML 5

<!DOCTYPE html>



Podstawowa struktura dokumentu

```
<!DOCTYPE ... >
<html>
  <head>
    <title>
      Tytuł strony
    </title>
  </head>
  <body>
    Treść strony
  </body>
</html>
```

sekcja nagłówkowa



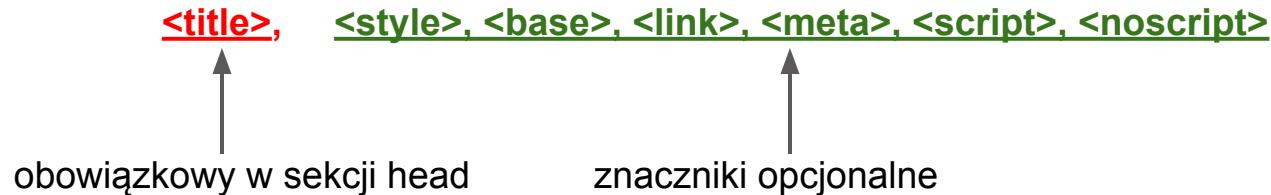
Nagłówek <head> ... </head>

Znacznik **head** otacza nagłówek dokumentu **HTML**. Zawiera informacje o tytule dokumentu, meta-informacje, a w większości przypadków także dołączone skrypty. W ramach tego znacznika należy zagnieździć znacznik **<title>**, który zawiera tekst, wyświetlany przez przeglądarkę jako tytuł strony. Następujące znaczniki mogą znaleźć się w sekcji nagłówka:

<title>, **<style>**, **<base>**, **<link>**, **<meta>**, **<script>**, **<noscript>**

Nagłówek <head> ... </head>

Znacznik **head** otacza nagłówek dokumentu **HTML**. Zawiera informacje o tytule dokumentu, meta-informacje, a w większości przypadków także dołączone skrypty. W ramach tego znacznika należy zagnieździć znacznik **<title>**, który zawiera tekst, wyświetlany przez przeglądarkę jako tytuł strony. Następujące znaczniki mogą znaleźć się w sekcji nagłówka:



Nagłówek <head> ... </head>

Znacznik **head** otacza nagłówek dokumentu **HTML**. Zawiera informacje o tytule dokumentu, meta-informacje, a w większości przypadków także dołączone skrypty. W ramach tego znacznika należy zagnieździć znacznik **<title>**, który zawiera tekst, wyświetlany przez przeglądarkę jako tytuł strony. Następujące znaczniki mogą znaleźć się w sekcji nagłówka:



Przykład:

```
<head>
  <link rel="stylesheet" type="text/css" href="styles.css" />
  <title>Tytuł dokumentu</title>
  <meta name="description" content="Przykładowa strona" />
  <meta name="keywords" content="przykład, nagłówek, strona" />
  <script type="text/javascript" src="skrypt.js">
  </script>
</head>
```



Meta-znaczniki <meta>

Meta-znaczniki, są specjalnymi elementami umieszczonymi w sekcji <head> strony internetowej, które opisują informację na niej zawartą. W szczególności są to słowa kluczowe, dane twórców strony, język kodowania strony i jej opis. Są niewidoczne dla użytkownika w przeglądarce, natomiast są bardzo przydatne dla “robotów” indeksujących strony na potrzeby pozycjonowania witryn w wyszukiwarkach internetowych.

Element <meta> jest znacznikiem pustym (nie przenosi danych). Należy zwracać uwagę na różnicę w deklaracji meta-znaczników pomiędzy XHTML i HTML:

W XHTML : <meta />

w HTML : <meta >



Meta-znaczniki <meta>

Meta-znaczniki, są specjalnymi elementami umieszczonymi w sekcji <head> strony internetowej, które opisują informację na niej zawartą. W szczególności są to słowa kluczowe, dane twórców strony, język kodowania strony i jej opis. Są niewidoczne dla użytkownika w przeglądarce, natomiast są bardzo przydatne dla “robotów” indeksujących strony na potrzeby pozycjonowania witryn w wyszukiwarkach internetowych.

Element <meta> jest znacznikiem pustym (nie przenosi danych). Należy zwracać uwagę na różnicę w deklaracji meta-znaczników pomiędzy XHTML i HTML:

W XHTML : <meta />

w HTML : <meta >

Przykład:

```
■ <head>
■   <meta charset="UTF-8">
■   <meta name="description" content="Opis twojej strony">
■   <meta name="keywords" content="Słowa kluczowe opisujące
■     twoją stronę">
■   <meta name="author" content="Autor strony">
■ </head>
```

Meta-znaczniki <meta>

Meta-znaczniki, są specjalnymi elementami umieszczonymi w sekcji <head> strony internetowej, które opisują informację na niej zawartą. W szczególności są to słowa kluczowe, dane twórców strony, język kodowania strony i jej opis. Są niewidoczne dla użytkownika w przeglądarce, natomiast są bardzo przydatne dla “robotów” indeksujących strony na potrzeby pozycjonowania witryn w wyszukiwarkach internetowych.

Drugim zadaniem meta-znaczników jest sterowanie zachowaniem przeglądarki na podstawie zwróconej odpowiedzi od serwera HTTP. Serwer interpretuje znaczniki <meta> i automatycznie tłumaczy je na nagłówek HTTP odpowiedzi serwera.

```
<meta http-equiv="Content-Language" content="pl">  
  
<meta http-equiv="Content Type" content="text/html; charset=iso-8859-2">
```

HTML 4.01 :

```
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
```

HTML 5 :

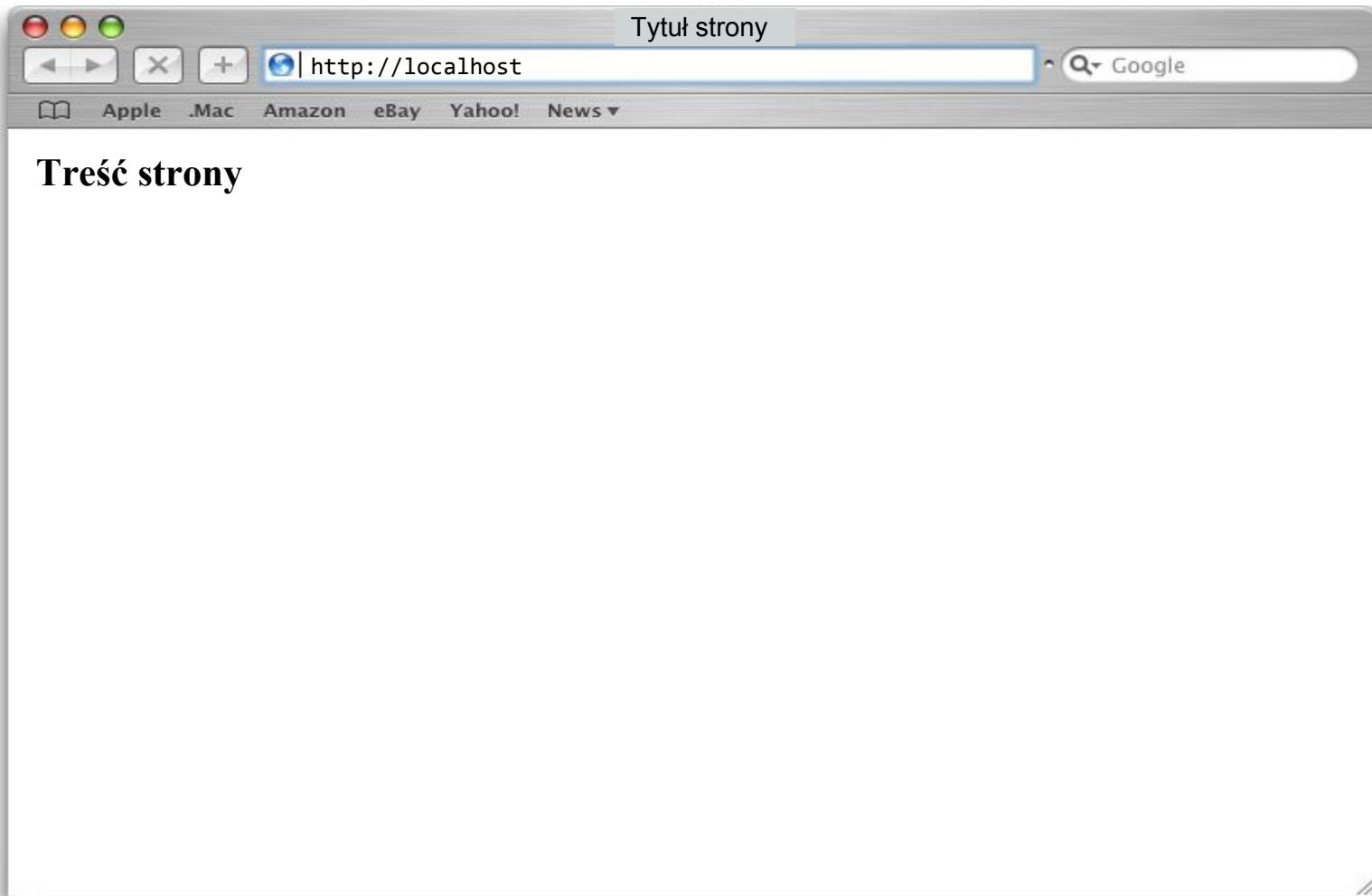
```
<meta charset="UTF-8">
```



Podstawowa struktura dokumentu

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="description" content="Opis strony">
    <meta name="keywords" content="Słowa kluczowe">
    <meta name="author" content="Autor strony">
    <title>
      Tytuł strony
    </title>
  </head>
  <body>
    Treść strony
  </body>
</html>
```

Podstawowa struktura dokumentu



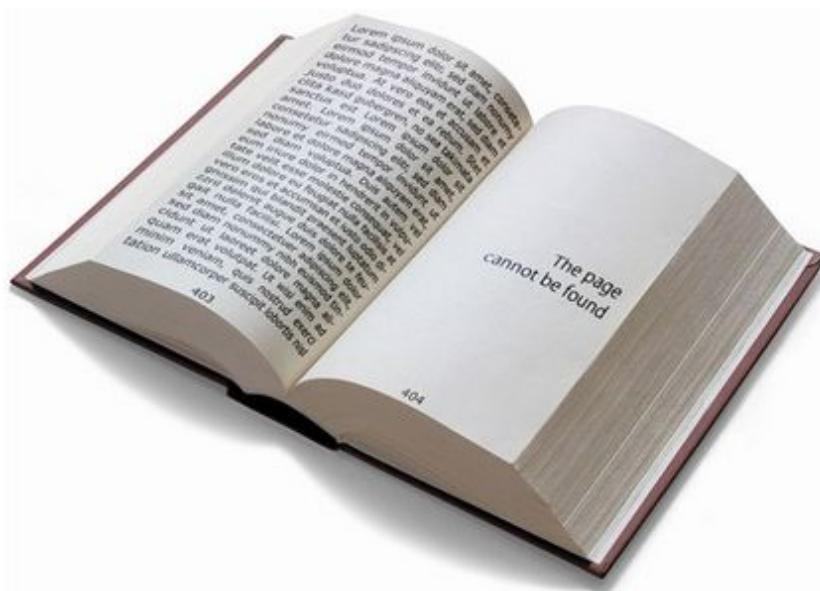
Podstawowa struktura dokumentu

```
<!DOCTYPE ... >
<html>
  <head>
    <title>
      Tytuł strony
    </title>
  </head>
  <body>
    Treść strony
  </body>
</html>
```

The code illustrates the basic structure of an HTML document. It starts with a doctype declaration, followed by the opening and closing tags for the html element. Inside the html block, there is one head element containing a title element with the text "Tytuł strony". Below the head is a body element containing the text "Treść strony". The entire body content is highlighted with a light green background, and the text "sekcja zawartości strony" is placed to its right, indicating the section of the page where the body content resides.

Budowanie struktury strony

Na początek warto wyobrazić sobie stronę internetową jak książkę z rozdziałami, podrozdziałami, pod-pod-rozdziałami, akapitami, listami, ilustracjami



Budowanie struktury strony

Na początek warto wyobrazić sobie stronę internetową jak książkę z rozdziałami, podrozdziałami, pod-podrozdziałami, akapitami, listami, ilustracjami



Co zrobić aby nasza strona wyglądała np. tak ??



Znaczniki proste

Aby zacząć tworzyć stronę należy poznać kilkanaście znaczników prostych (zasadniczych), które umożliwią zbudowanie struktury danych strony internetowej.

1. **< p >** **Znacznik akapitu** (paragrafu) - służy do określania miejsca występowania danej części tekstu na stronie internetowej.

```
-----  
: <p>  
: Pachołkowie rozwiązali go zaraz, lecz on, mając członki zdrętwiałe,  
: upadł na ziemię, a gdy go podniesiono, począł mdleć raz po razu,  
: gdyż był i srodze poturbowan. Próżno z rozkazu Zbyszka zawiedziono  
: go przed ognisko, dano jeść i pić, natarto sadłem, a następnie  
: okryto ciepło skórami;  
: </p>  
-----
```

Znaczniki proste

Aby zacząć tworzyć stronę należy poznać kilkanaście znaczników prostych (zasadniczych), które umożliwiają zbudowanie struktury danych strony internetowej.

1. **< p >** **Znacznik akapitu** (paragrafu) - służy do określania miejsca występowania danej części tekstu na stronie internetowej.



Znaczniki proste

-
2. <hn> **Znacznik nagłówków** (tytułów rozdziałów) - służy do określania tytułu nagłówka, może posiadać kilka poziomów określonych liczbami od n = 1 do n = 9: <hn> ... </hn>

```
:-----  
<h1> Księga I - Gospodarstwo </h1>
```

```
:-----  
<h1> Księga II - Zamek </h1>
```

```
:-----  
<h2> Pod-Księga 1 </h2>
```

```
:-----  
<h2> Pod-Księga 2 </h2>
```

```
:-----  
<h3> Pod-Pod-Księga 1 </h3>
```

```
:-----  
<h3> Pod-Pod-Księga 2 </h3>
```

Znaczniki proste

2. <hn> **Znacznik nagłówków** (tytułów rozdziałów) - służy do określania tytułu nagłówka, może posiadać kilka poziomów określonych liczbami od n = 1 do n = 9: <hn> ... </hn>



Znaczniki proste

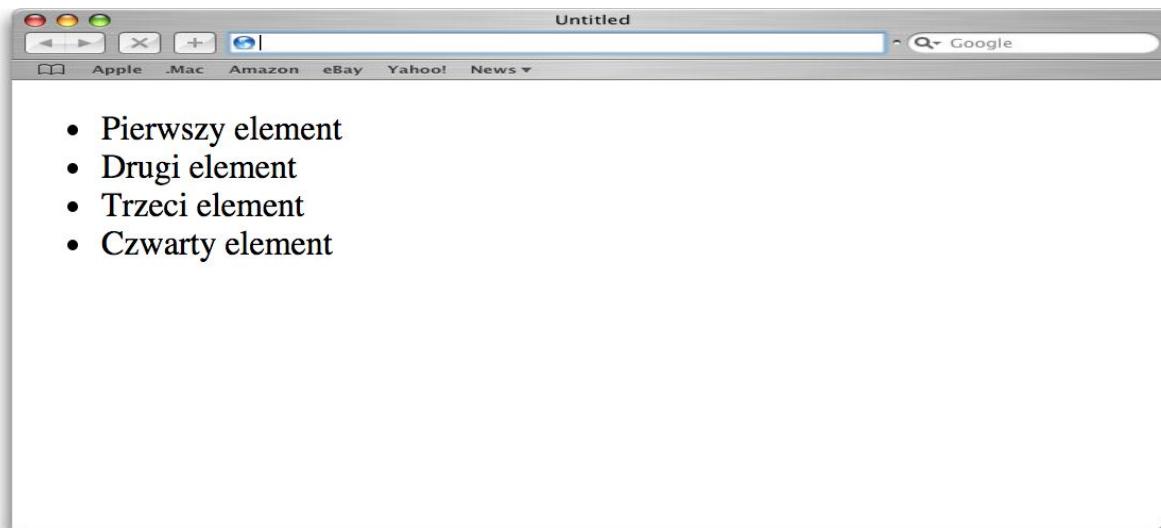
3. ** Lista uporządkowana nienumerowana** - służy do tworzenia list uporządkowanych nienumerowanych tzn. każdy element listy będzie oznaczony symbolem kropki.

```
<ul>
  <li> Pierwszy element </li>
  <li> Drugi element </li>
  <li> Trzeci element </li>
  <li> Czwarty element </li>
</ul>
```

Znaczniki proste

3. **** Lista uporządkowana nienumerowana - służy do tworzenia list uporządkowanych nienumerowanych tzn. każdy element listy będzie oznaczony symbolem kropki.

```
<ul>
  <li> Pierwszy element </li>
  <li> Drugi element </li>
  <li> Trzeci element </li>
  <li> Czwarty element </li>
</ul>
```



Znaczniki proste

3. **** Lista uporządkowana nienumerowana - służy do tworzenia list uporządkowanych nienumerowanych tzn. każdy element listy będzie oznaczony symbolem kropki.

```
<ul>
  <li> Pierwszy element </li>
  <li> Drugi element </li>
  <li> Trzeci element </li>
  <li> Czwarty element </li>
</ul>
```



Uwaga: Znaczniki listy są bardzo ważne w kontekście tworzenia na stronach internetowych poprawnych semantycznie menu służących do nawigacji po stronie.

Znaczniki proste

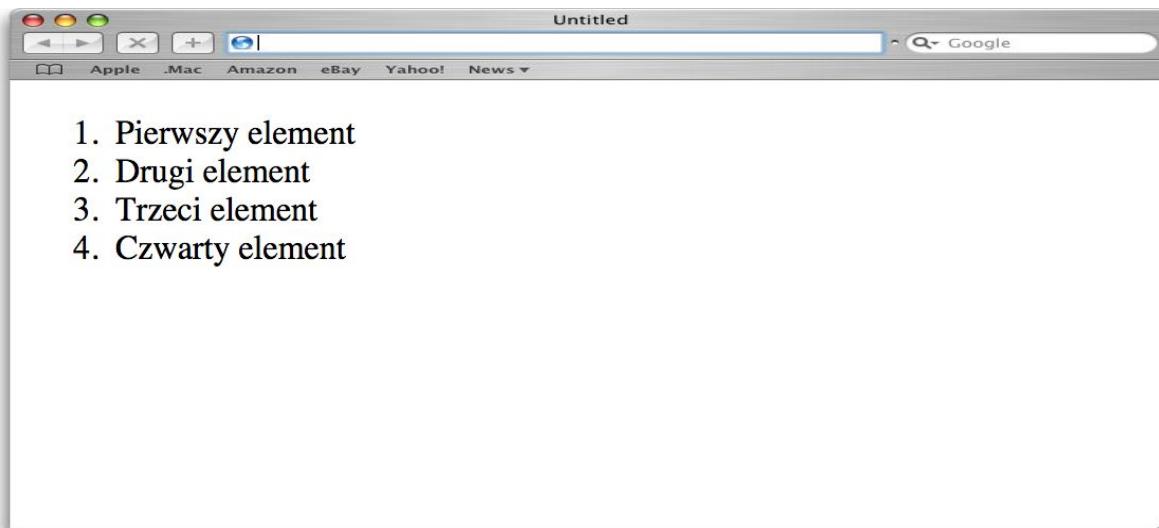
-
- 4. **** Lista uporządkowana numerowana - służy do tworzenia list uporządkowanych numerowanych tzn. każdy element listy będzie oznaczony kolejnym numerem porządkowym od 1 do n.

```
<ol>
  <li> Pierwszy element </li>
  <li> Drugi element </li>
  <li> Trzeci element </li>
  <li> Czwarty element </li>
</ol>
```

Znaczniki proste

4. **** Lista uporządkowana numerowana - służy do tworzenia list uporządkowanych numerowanych tzn. każdy element listy będzie oznaczony kolejnym numerem porządkowym od 1 do n.

```
<ol>
  <li> Pierwszy element </li>
  <li> Drugi element </li>
  <li> Trzeci element </li>
  <li> Czwarty element </li>
</ol>
```



Znaczniki proste

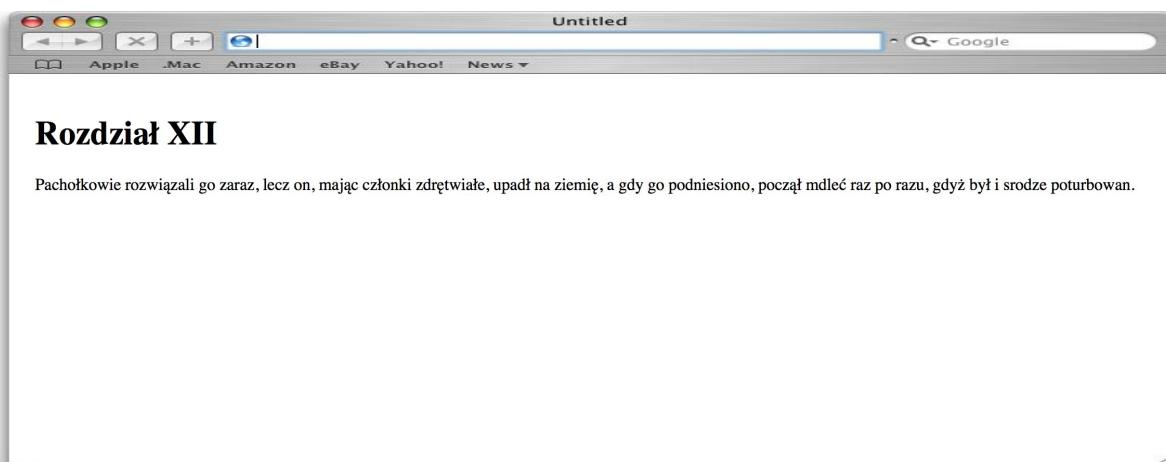
-
- 5. **<div>** Znacznik grupowania blokowego (znacznik blokowy) - służy do określenia bloku strony internetowej o odpowiednich właściwościach (np formatowaniu). W obrębie znacznika div mogą znaleźć się inne znaczniki.

```
-----  
  <div>  
    <h1> Rozdział XII </h1>  
    <p>  
      Pachołkowie rozwiązały go zaraz, lecz on, mając członki  
      zdrętwiałe, upadł na ziemię, a gdy go podniesiono, począł mdleć  
      raz po razu, gdyż był i srodze poturbowany.  
    </p>  
  </div>  
-----
```

Znaczniki proste

5. **<div>** Znacznik grupowania blokowego (znacznik blokowy) - służy do określenia bloku strony internetowej o odpowiednich właściwościach (np formatowaniu). W obrębie znacznika div mogą znaleźć się inne znaczniki.

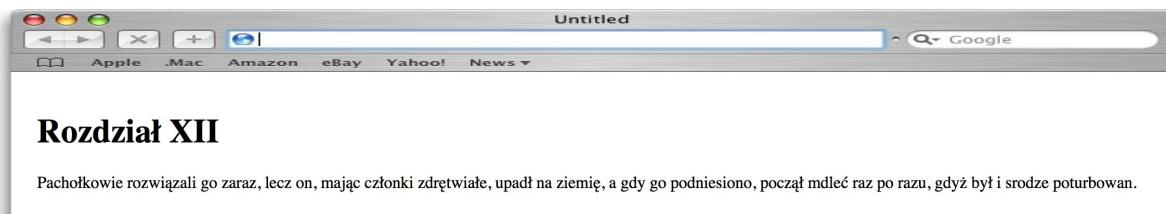
```
<div>
  <h1> Rozdział XII </h1>
  <p>
    Pachołkowie rozwiązały go zaraz, lecz on, mając członki
    zdrętwiałe, upadł na ziemię, a gdy go podniesiono, począł mdleć
    raz po razu, gdyż był i srodze poturbowan.
  </p>
</div>
```



Znaczniki proste

5. **<div>** Znacznik grupowania blokowego (znacznik blokowy) - służy do określenia bloku strony internetowej o odpowiednich właściwościach (np formatowaniu). W obrębie znacznika div mogą znaleźć się inne znaczniki.

```
<div>
  <h1> Rozdział XII </h1>
  <p>
    Pachołkowie rozwijała go zaraz, lecz on, mając członki
    zdrętwiałe, upadł na ziemię, a gdy go podniesiono, począł mdleć
    raz po razu, gdyż był i srodze poturbowan.
  </p>
</div>
```



Uwaga: Znacznik **<div>** nie jest widoczny dla użytkownika służy tylko do zgrupowania innych elementów w jednym bloku.

Obecnie struktury większości stron internetowych są oparte o znaczniki **<div>** które pozwalają na logiczne grupowanie ich zawartości.

Znaczniki proste

-
- 6. **** Znacznik grupowania liniowego (znacznik liniowy) - służy do grupowania innych znaczników lub tekstu.

```
-----  
|<span> Tekst  
|<p>  
| Pachołkowie rozwiązali go zaraz, lecz on, mając członki  
| zdrętwiałe, upadł na ziemię, a gdy go podniesiono, począł mdleć  
| raz po razu, gdyż był i srodze poturbowan.  
|</p>  
|<span> Tekst 2 </span>  
|</span>  
-----
```

Znaczniki proste

-
- 6. **** Znacznik grupowania liniowego (znacznik liniowy) - służy do grupowania innych znaczników lub tekstu.

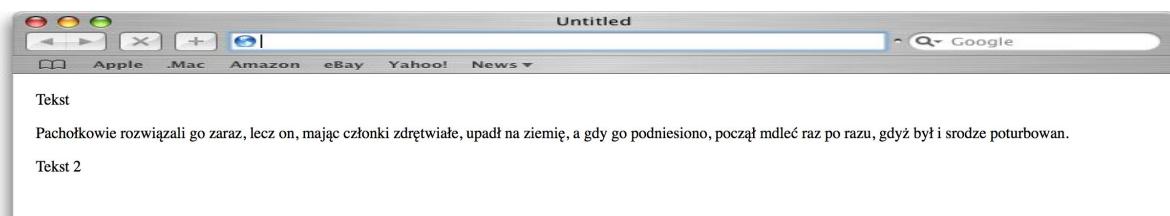
```
-----  
|<span> Tekst  
| <p>  
| Pachołkowie rozwiązały go zaraz, lecz on, mając członki  
| zdrętwiałe, upadł na ziemię, a gdy go podniesiono, począł mdleć  
| raz po razu, gdyż był i srodze poturbowan.  
| </p>  
| <span> Tekst 2 </span>  
</span>  
-----
```



Znaczniki proste

-
- 6. **** Znacznik grupowania liniowego (znacznik liniowy) - służy do grupowania innych znaczników lub tekstu.

```
-----  
|<span> Tekst  
|<p>  
| Pachołkowie rozwiązały go zaraz, lecz on, mając członki  
| zdrętwiałe, upadł na ziemię, a gdy go podniesiono, począł mdleć  
| raz po razu, gdyż był i srodze poturbowan.  
|</p>  
|<span> Tekst 2 </span>  
</span>
```



Uwaga: Znacznik **** jest często wykorzystywany do nadawania odpowiedniego formatowania tekstu lub pojedynczych wyrazów za pomocą kaskadowych arkuszy stylu.



Znaczniki proste

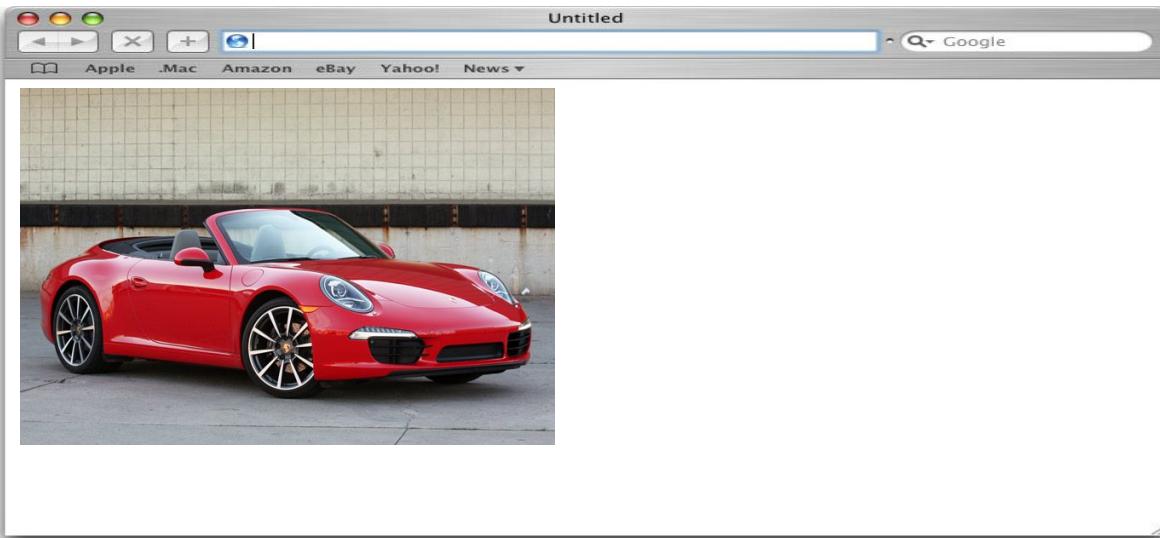
6. **** **Znacznik osadzenia grafiki** - służy do umieszczania na stronie obrazu w postaci plików w formacie: *jpg, png, svg, bmp, gif, tiff* itp.

```
:
:
:<div>
:  
:</div>
```

Znaczniki proste

-
- 6. **** Znacznik osadzenia grafiki - służy do umieszczania na stronie obrazu w postaci plików w formacie: *jpg, png, svg, bmp, gif, tiff* itp.

```
:  
:  
:  
:<div>  
:    
:</div>  
:
```



Znaczniki proste

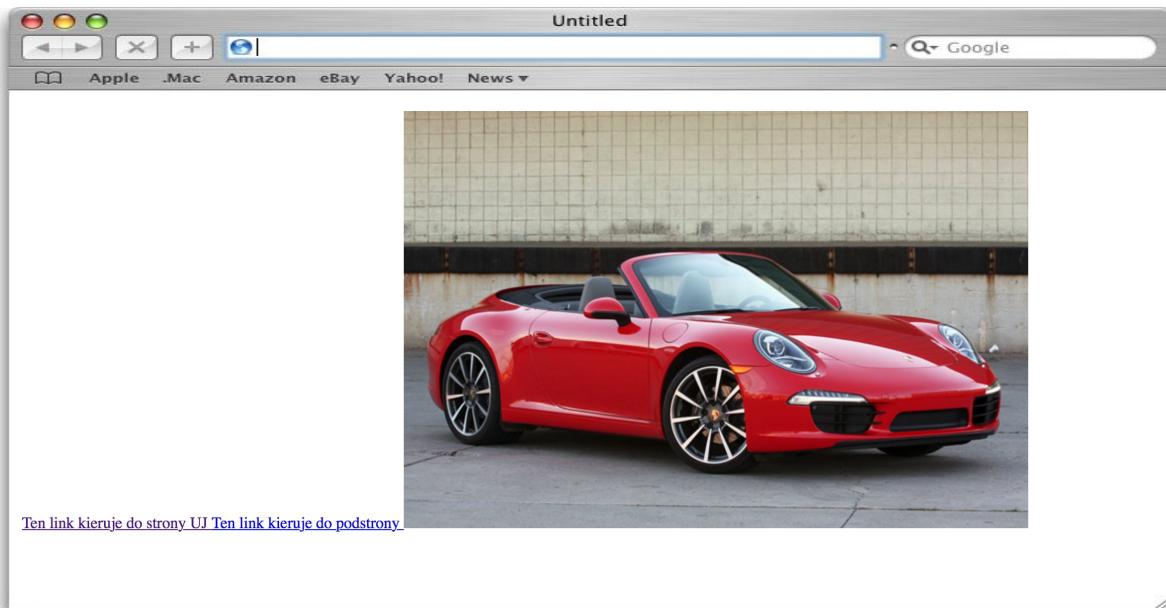
-
- 7. **< a >** **Znacznik hiperłącza (linku)** - pozwala utworzyć odnośnik do zasobu. Może być ustawiony na tekście, obrazie lub innym elemencie strony. Odsyłacze pozwalają na tworzenie logicznej struktury jaką na stronie internetowej w celu

```
-----  
: <a href="http://www.uj.edu.pl"> Ten link kieruje do strony UJ </a>  
:  
: <a href="podstrona1.html"> Ten link kieruje do podstrony </a>  
:  
: <a href="http://www.uj.edu.pl">  </a>  
-----
```

Znaczniki proste

7. **< a >** **Znacznik hiperłącza (linku)** - pozwala utworzyć odnośnik do zasobu. Może być ustawiony na tekście, obrazie lub innym elemencie strony. Odsyłacze pozwalają na tworzenie logicznej struktury jaką na stronie internetowej w celu

```
-----  
• <a href="http://www.uj.edu.pl"> Ten link kieruje do strony UJ </a>  
• <a href="podstrona1.html"> Ten link kieruje do podstrony </a>  
• <a href="http://www.uj.edu.pl">  </a>  
-----
```



Znaczniki proste

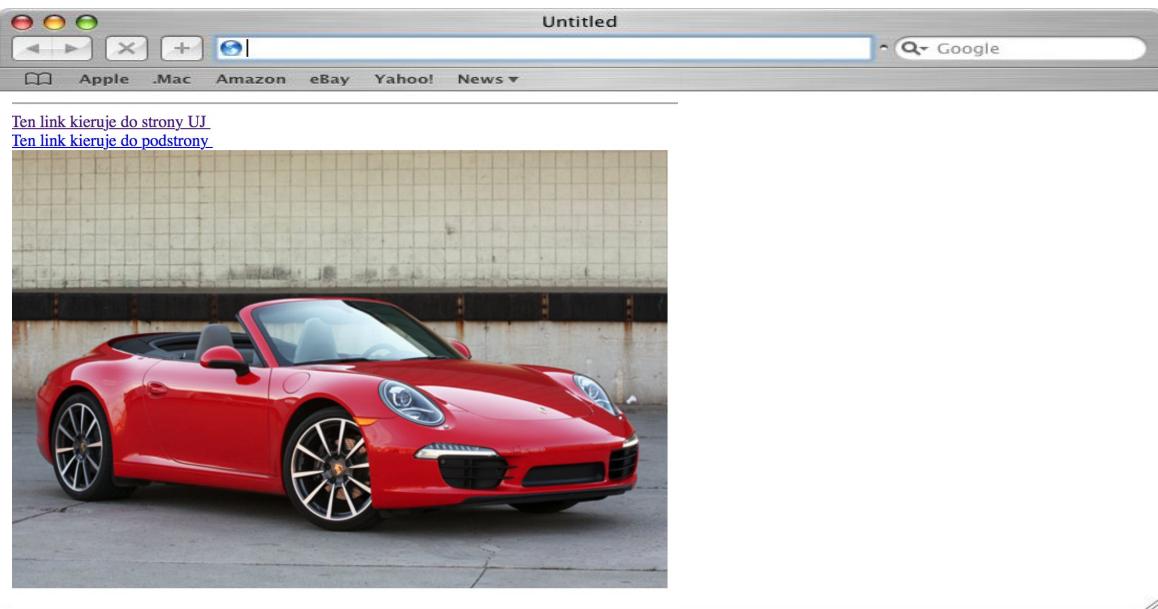
-
- 8. **
 Znacznik znacznik przejścia do nowej linii** - wymusza na przeglądarce przejście do nowej linii ponieważ domyślnie przeglądarki interpretując kod HTML nie zwracają uwagi na odstępy oraz przejścia do nowej linii wykonane za pomocą przycisku ENTER.

```
-----  
▪ <a href="http://www.uj.edu.pl"> Ten link kieruje do strony UJ </a>  
▪ <br/>  
▪ <a href="podstrona1.html"> Ten link kieruje do podstrony </a>  
▪ <br/>  
▪ <a href="http://www.uj.edu.pl">  </a>  
-----
```

Znaczniki proste

8. **
 Znacznik znacznik przejścia do nowej linii** - wymusza na przeglądarce przejście do nowej linii ponieważ domyślnie przeglądarki interpretując kod HTML nie zwracają uwagi na odstępy oraz przejścia do nowej linii wykonane za pomocą przycisku ENTER.

```
-----  
▪ <a href="http://www.uj.edu.pl"> Ten link kieruje do strony UJ </a>  
▪ <br/>  
▪ <a href="podstrona1.html"> Ten link kieruje do podstrony </a>  
▪ <br/>  
▪ <a href="http://www.uj.edu.pl">  </a>  
-----
```



Znaczniki proste

8. **
 Znacznik znacznik przejścia do nowej linii** - wymusza na przeglądarce przejście do nowej linii ponieważ domyślnie przeglądarki interpretując kod HTML nie zwracają uwagi na odstępy oraz przejścia do nowej linii wykonane za pomocą przycisku ENTER.

```
-----  
▪ <a href="http://www.uj.edu.pl"> Ten link kieruje do strony UJ </a>  
▪ <br/>  
▪ <a href="podstrona1.html"> Ten link kieruje do podstrony </a>  
▪ <br/>  
▪ <a href="http://www.uj.edu.pl">  </a>  
-----
```



Uwaga:

Zapis znacznika br w różnych standardach języka HTML jest inny:

HTML 4.01 :

XHTML :
 (pomiędzy br a ukośnikiem jest spacja!)

HTML 5 :

Znaczniki proste

9. **Encje HTML (znaki specjalne)** - pozwalają na użycie w tekście strony znaków specjalnych lub zastrzeżonych dla funkcji języka HTML (np. znak większości / mniejszości). Encja ma postać:

&_ _ _ _ ;

gdzie w miejsce kresek najczęściej wpisujemy oznaczenie liczbowe lub literowe danej encji.

Znaczniki proste

9. **Encje HTML (znaki specjalne)** - pozwalają na użycie w tekście strony znaków specjalnych lub zastrzeżonych dla funkcji języka HTML (np. znak większości / mniejszości). Encja ma postać:

& _____ ;

gdzie w miejsce kresek najczęściej wpisujemy oznaczenie liczbowe lub literowe danej encji.

&nbsp = ** ** – twarda spacja.

" = **"** – cudzysłów informatyczny: " .

< = **<** – symbol mniejszy niż: < (ostry lewy nawias).

> = **>** – symbol większy niż: > (ostry prawy nawias).

© = **©** – znak praw autorskich: ©.

& = **&** – znak: &.

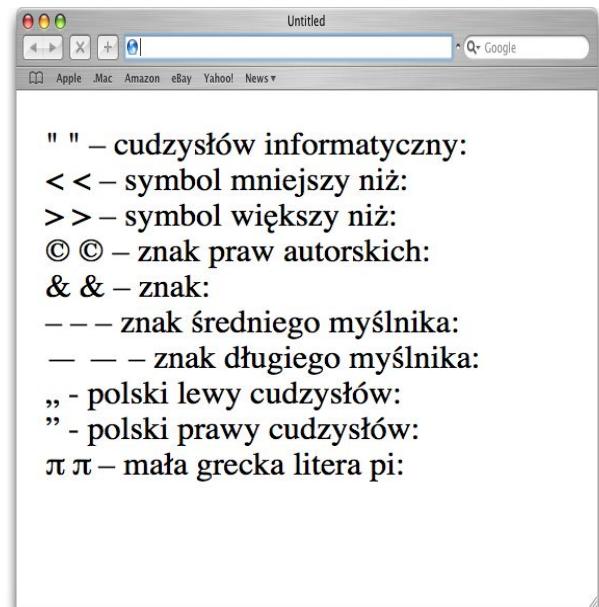
– = **–** – znak średniego myślnika: – .

— = **—** – znak długiego myślnika: — .

„ – polski lewy cudzysłów: „ .

” – polski prawy cudzysłów: ” .

π = **π** – mała grecka litera pi: π .



Znaczniki proste

9. **Encje HTML (znaki specjalne)** - pozwalają na użycie w tekście strony znaków specjalnych lub zastrzeżonych dla funkcji języka HTML (np. znak większości / mniejszości). Encja ma postać:

&_ _ _ _ ;

gdzie w miejsce kresek najczęściej wpisujemy oznaczenie liczbowe lub literowe danej encji.

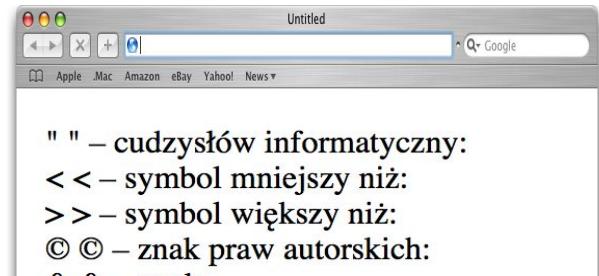
 = – twarda spacja.

" = " – cudzysłów informatyczny: " .

< = < – symbol mniejszy niż: < (ostry lewy nawias).

> = > – symbol większy niż: > (ostry prawy nawias).

© = © – znak praw autorskich: ©.



Uwaga:

Prezentowane encje stanowią tylko najczęściej używane symbole, pełne zestawy zapisu dowolnego znaku można znaleźć na wielu stronach internetowych np.:

http://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references

π = π – mała grecka litera pi: π.

HTML5 - Historia w skrócie

HTML5 narodził się z potrzeby stworzenia bardziej nowoczesnego, a jednocześnie kompatybilnego wstecz standardu języka mającego zalety HTML4.01 oraz XHTML.

Powstał niejako w wyniku porażki przy próbie stworzenia standardu XHTML 2.0 (który miał zastąpić XHTML1.0). Niestety ze względu na daleko idące zmiany jakie wprowadzał (np. likwidacja znaczników h1, h2) oraz konieczność dostosowania do niego już istniejących witryn, pomysł został zarzucony.

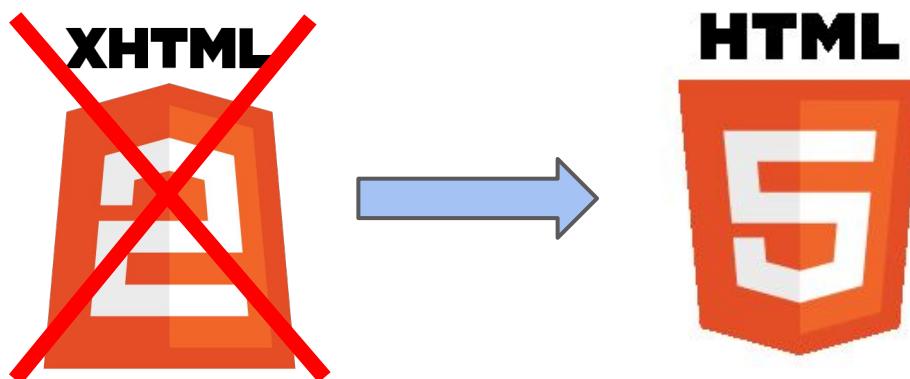
XHTML



HTML5 - Historia w skrócie

HTML5 narodził się z potrzeby stworzenia bardziej nowoczesnego, a jednocześnie kompatybilnego wstecz standardu języka mającego zalety HTML4.01 oraz XHTML.

Powstał niejako w wyniku porażki przy próbie stworzenia standardu XHTML 2.0 (który miał zastąpić XHTML1.0). Niestety ze względu na daleko idące zmiany jakie wprowadzał (np. likwidacja znaczników h1, h2) oraz konieczność dostosowania do niego już istniejących witryn, pomysł został zarzucony.



W 2004 grupa entuzjastów oraz specjalści z Opera, Firefox, Apple stworzyli WHATWG (Web Hypertext Application Technology Working Group), która opracowała jednolity system nazwany później HTML5. Cyfra 5 oznacza bezpośrednią kontynuację standardu HTML4.01.

HTML5 - Ogólne zasady

1. Wszystko co było przed wprowadzeniem HTML5 działa w niezaburzony sposób oraz jest obsługiwane.
2. Standaryzacja dotychczasowych technik oraz dostosowanie ich do istniejących przeglądarek.
3. Praktyczność - oznaczająca, że wszelkie zmiany w standardzie języka powinny być uzasadnione praktycznością ich wprowadzania.



HTML5 - Ogólne zasady

1. Wszystko co było przed wprowadzeniem HTML5 działa w niezaburzony sposób oraz jest obsługiwane.
2. Standaryzacja dotychczasowych technik oraz dostosowanie ich do istniejących przeglądarek.
3. Praktyczność - oznaczająca, że wszelkie zmiany w standardzie języka powinny być uzasadnione praktycznością ich wprowadzania.



Ogólna specyfikacja języka:



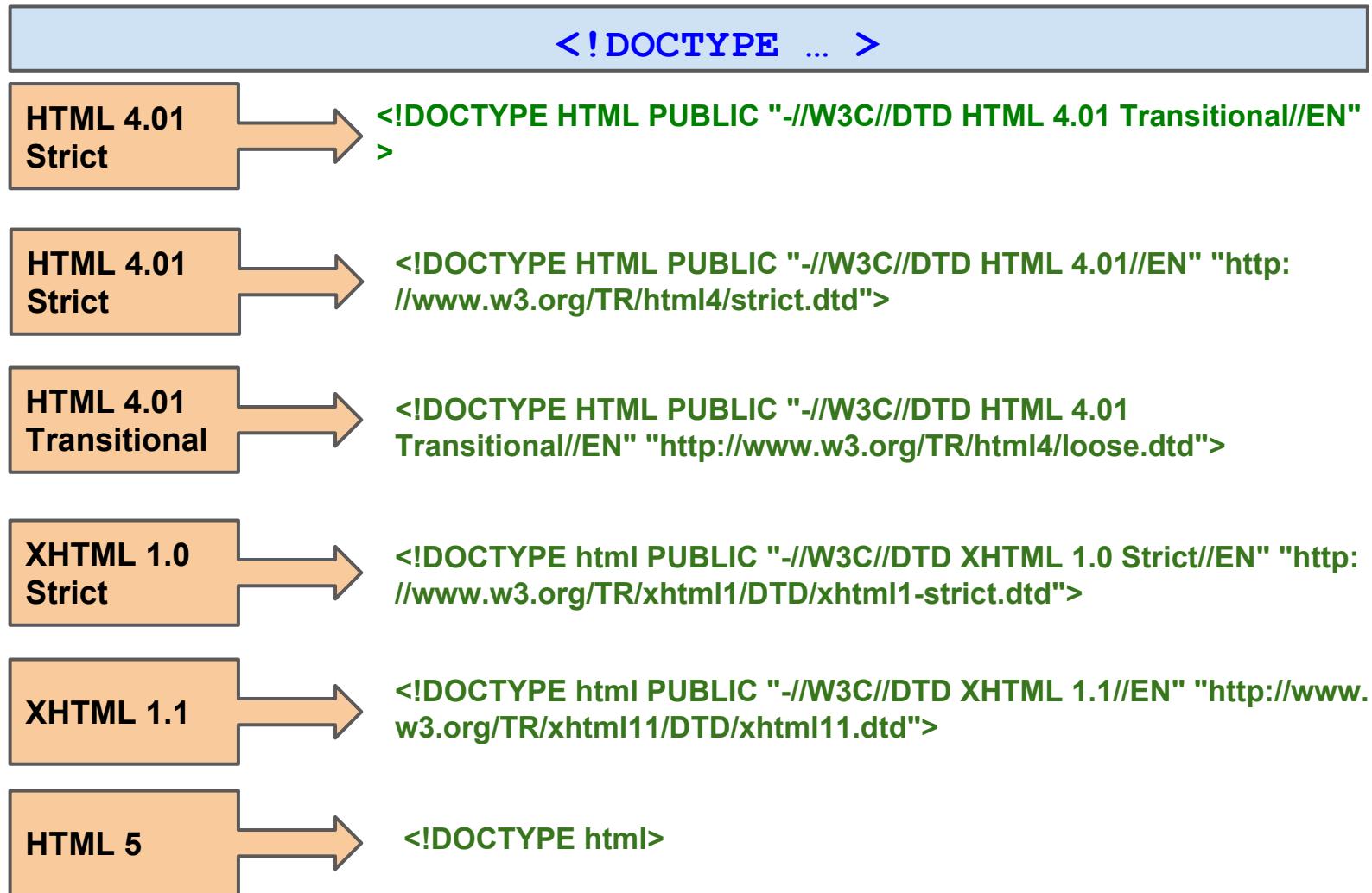
www.w3.org/TR/html5

Najnowsze zmiany języka:



<https://html.spec.whatwg.org/multipage/>

DTD dla HTML5



DTD dla HTML5

<!DOCTYPE ... >

HTML 4.01
Strict → <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >

HTML 4.01
Strict → <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

HTML 4.01
Transitional → <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

XHTML 1.0
Strict → <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

XHTML 1.1 → <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

HTML 5 → <!DOCTYPE html>



Struktura dokumentu w HTML5

<!DOCTYPE html>	deklaracja typu dokumentu
<html>	początek dokumentu hipertekstowego
<head>	sekcja nagłówkowa
<title> Tytuł strony </title>	tytuł strony
</head>	
<body> <p>Treść strony</p> </body>	sekcja zawartości strony
</html>	koniec dokumentu hipertekstowego

Struktura dokumentu w HTML5

<!DOCTYPE html>	deklaracja typu dokumentu
<html>	początek dokumentu hipertekstowego
<head>	sekcja nagłówkowa
<title> Tytuł strony </title>	tytuł strony
</head>	
<body> <p>Treść strony</p> </body>	sekcja zawartości strony
</html>	koniec dokumentu hipertekstowego

MODEL PUDEŁKOWY



Struktura dokumentu w HTML5

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      Tytuł strony
    </title>
  </head>
  <body>
    <p>Treść strony</p>
  </body>
</html>
```

Struktura dokumentu w HTML5

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      Tytuł strony
    </title>
  </head>
  <body>
    <p>Treść strony</p>
  </body>
</html>
```

W zasadzie cały kod można jeszcze skrócić ponieważ można pominąć znaczniki zamykające np. </p>. Przeglądarki same na końcu dokumentu zamykają wszystkie znaczniki.

Struktura dokumentu w HTML5

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      Tytuł strony
    </title>
  </head>
  <body>
    <p>Treść strony</p>
  </body>
</html>
```

W zasadzie cały kod można jeszcze skrócić ponieważ można pominąć znaczniki zamykające np. </p>. Przeglądarki same na końcu dokumentu zamykają wszystkie znaczniki.

Jednak lepiej dbać o zamykanie wszystkich znaczników oraz robienie odstępów dla uniknięcia błędów oraz dla przejrzystości kodu.

Kodowanie znaków i język strony

Kodowanie jest standardem według którego komputery konwertują tekst na sekwencję bajtów przy zapisie i odczytanie danych z pliku. Jest wiele rodzajów kodowania jednak w ostatnich latach ze względu na szybkość działania i obsługę dużej liczby znaków najczęściej wykorzystuję się standard UTF-8.

W HTML5 kodowanie znaków jest bardzo ułatwione i sprowadza się do jednej dyrektywy meta w części head naszej strony internetowej:

Kodowanie:

```
<head>
  <meta charset="utf-8">
  <title> Gżegżółka </title>
</head>
```

Kodowanie znaków i język strony

Kodowanie jest standardem według którego komputery konwertują tekst na sekwencję bajtów przy zapisie i odczytanie danych z pliku. Jest wiele rodzajów kodowania jednak w ostatnich latach ze względu na szybkość działania i obsługę dużej liczby znaków najczęściej wykorzystuję się standard UTF-8.

W HTML5 kodowanie znaków jest bardzo ułatwione i sprowadza się do jednej dyrektywy meta w części head naszej strony internetowej:

Kodowanie:

```
<head>
  <meta charset="utf-8">
  <title> Gżegżolka </title>
</head>
```

Dodatkowo dobrą praktyką i oznaką profesjonalizmu jest wskazanie języka strony, co pozwala np. wyszukiwarką na odfiltrowywanie treści pasujących tylko do języka wyszukującego. Atrybut wskazujący na język można dodać do dowolnego znacznika jednak najczęściej umieszcza się go w znaczniku html na początku strony:

```
<html lang="pl">
```



Dodawanie arkusza stylu i skryptów

W ostatecznym kształcie strona internetowa korzysta z arkusza sylu (CSS) który nadaje jej odpowiedni wygląd oraz ze skryptu, który obsługuje różne zdarzenia na stronie. Informacje o stylach i skryptach są najczęściej umieszczone w osobnych plikach. Aby dołączyć je do naszej strony musimy zapisać odpowiednie dyrektywy w sekcji head:

Style CSS:

```
-----  
|<head>  
|  <link href="styl.css" rel="stylesheet">  
|</head>  
-----
```

Dodawanie arkusza stylu i skryptów

W ostatecznym kształcie strona internetowa korzysta z arkusza sylu (CSS) który nadaje jej odpowiedni wygląd oraz ze skryptu, który obsługuje różne zdarzenia na stronie. Informacje o stylach i skryptach są najczęściej umieszczone w osobnych plikach. Aby dołączyć je do naszej strony musimy zapisać odpowiednie dyrektywy w sekcji head:

Style CSS:

```
-----  
|<head>  
|  <link href="styl.css" rel="stylesheet">  
|</head>  
-----
```



W porównaniu z
HTML4 i XHTML
nie ma atrybutu
`type="text/css"`

Dodawanie arkusza stylu i skryptów

W ostatecznym kształcie strona internetowa korzysta z arkusza sylu (CSS) który nadaje jej odpowiedni wygląd oraz ze skryptu, który obsługuje różne zdarzenia na stronie. Informacje o stylach i skryptach są najczęściej umieszczone w osobnych plikach. Aby dołączyć je do naszej strony musimy zapisać odpowiednie dyrektywy w sekcji head:

Style CSS:

```
<head>
  <link href="styl.css" rel="stylesheet">
</head>
```

W porównaniu z
HTML4 i XHTML
nie ma atrybutu
type="text/css"



Skrypty:

```
<head>
  <script src="skrypt.js"></script>
</head>
```

Uwaga:
Ten znacznik musi
być zapisany
dokładnie w ten
sposób, a nie
<script/>

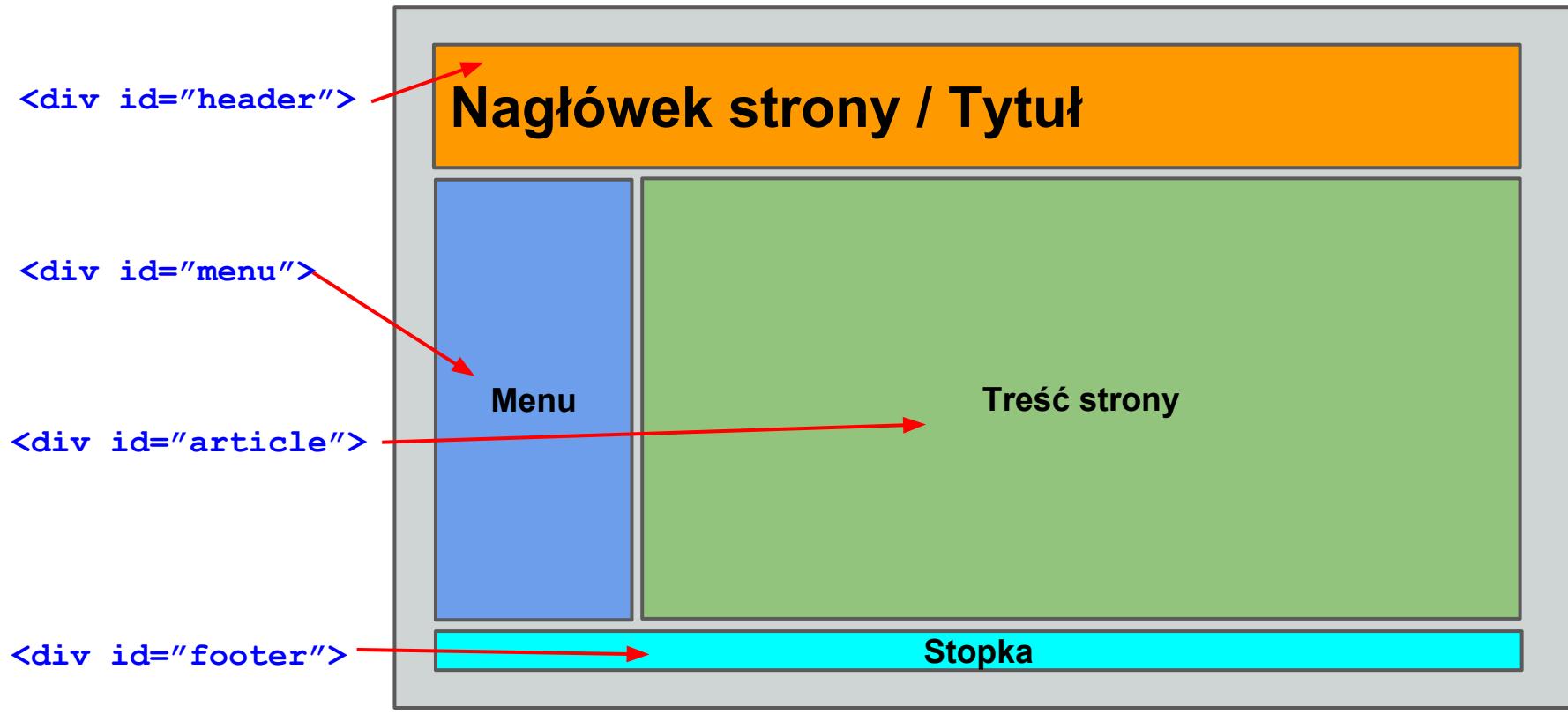


Ostateczna struktura

```
<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="utf-8">
    <link href="styl.css" rel="stylesheet">
    <title>Tytuł strony</title>
    <script scr="skrypt.js"></script>
  </head>
  <body>
    <p>Treść strony</p>
  </body>
</html>
```

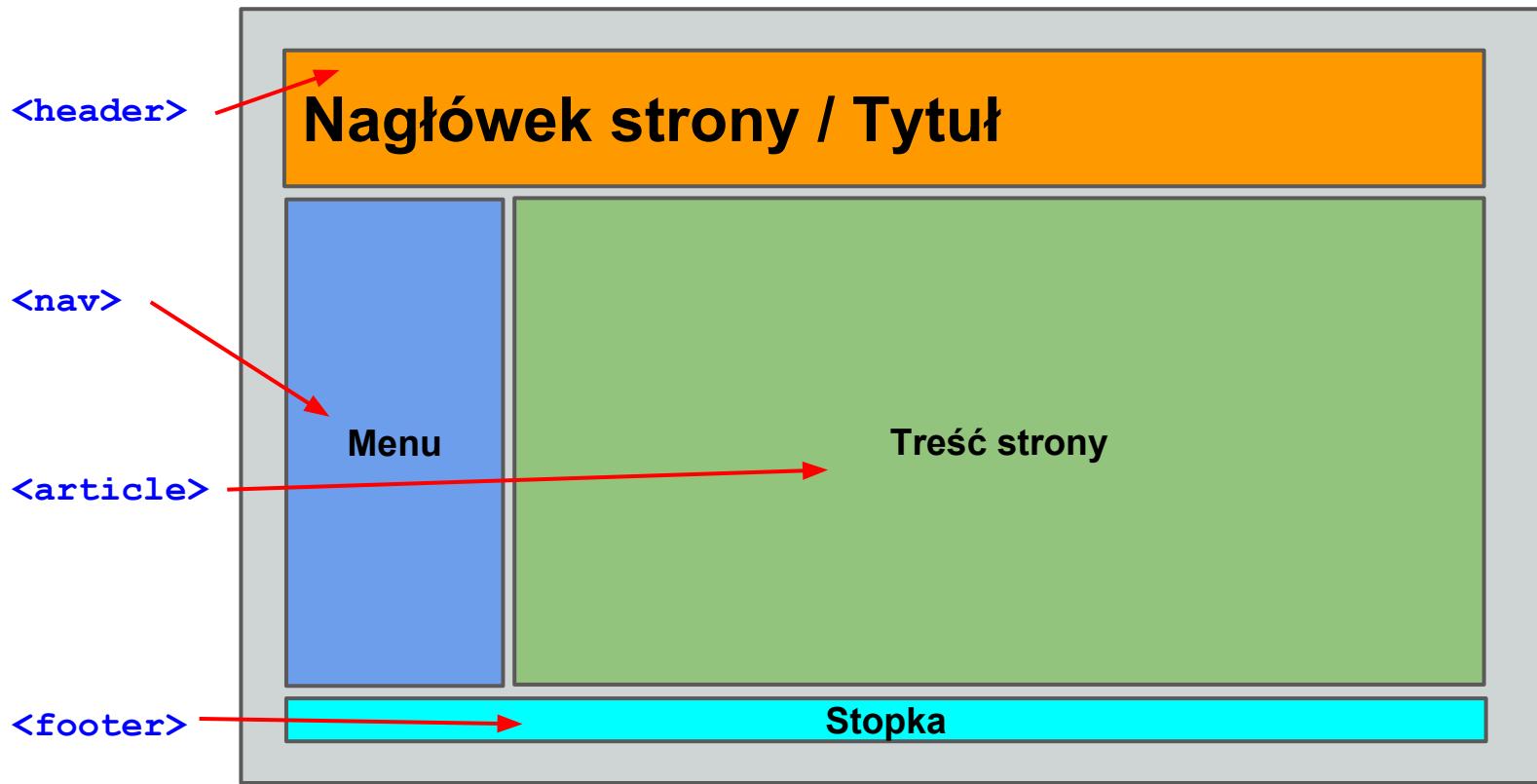
HTML5 i semantyka

Standard HTML5 wprowadza kilkanaście nowych znaczników oraz nowe i lepsze metody organizacji treści na stronie internetowej. Do tej pory strony były zbudowane najczęściej w oparciu o elementy podziału (elementy blokowe) <div>, które umożliwiały w miarę logiczne rozdzielenie treści na stronie.



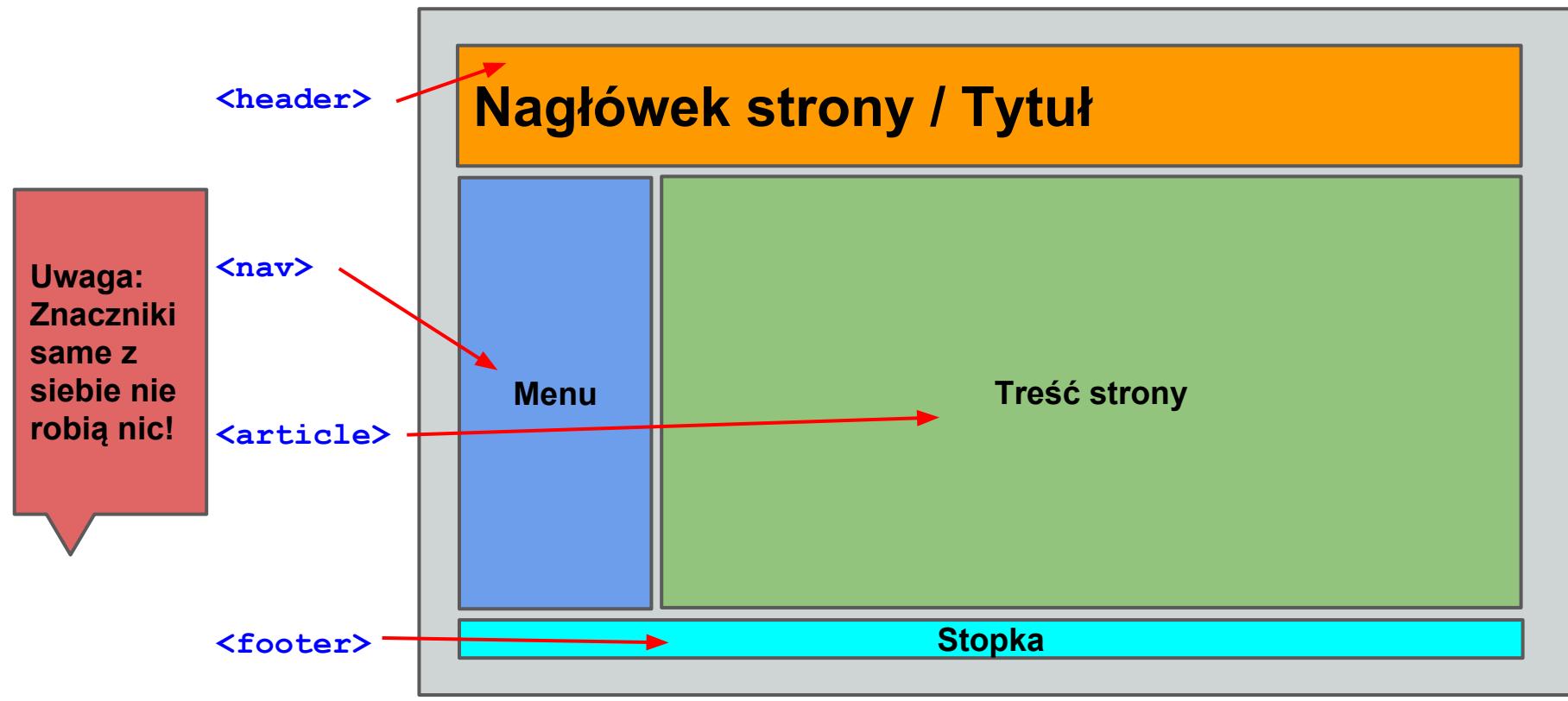
HTML5 i semantyka

Dzięki HTML5 mamy nowe znaczniki które organizują treści na stronie w sposób logiczny oraz dzięki temu są bardziej zoptymalizowane pod kątem wyszukiwarek. Dodatkowo pozwalają na przyjazną organizację strony i czynią stronę bardziej dostępną np. dla czytników ekranów ułatwiających przeglądanie stron osobą niedowidzącym.



HTML5 i semantyka

Dzięki HTML5 mamy nowe znaczniki które organizują treści na stronie w sposób logiczny oraz dzięki temu są bardziej zoptymalizowane pod kątem wyszukiwarek. Dodatkowo pozwalają na przyjazną organizację strony i czynią stronę bardziej dostępną np. dla czytników ekranów ułatwiających przeglądanie stron osobą niedowidzącym.



Podstawowe znaczniki w HTML5

Standard HTML wprowadza kilkanaście nowych znaczników ułatwiających organizację kodu strony internetowej i wprowadza nowe możliwości dla lepszego przeszukiwania stron przez roboty wyszukiwarek internetowych.

Artykuł / paragraf :

```
<article>
    Tekst rozdziału bardzo podobny znacznik do paragrafu p!
</article>
```

Nawigacja (menu):

```
<nav>
    <ul>
        <li> Menu 1 </li>
        <li> Menu 2 </li>
        <li> Menu 3 </li>
    </ul>
</nav>
```



Podstawowe znaczniki w HTML5

Standard HTML wprowadza kilkanaście nowych znaczników ułatwiających organizację kodu strony internetowej i wprowadza nowe możliwości dla lepszego przeszukiwania stron przez roboty wyszukiwarek internetowych.

Nagłówek w stronie internetowej (np tytuł:

```
<header>
  <h1> Tu jest wyświetlany tytuł strony </h1>
</header>
```

Stopka strony:

```
<footer>
  <h4> tu jest stopka mojej strony </h4>
</footer>
```

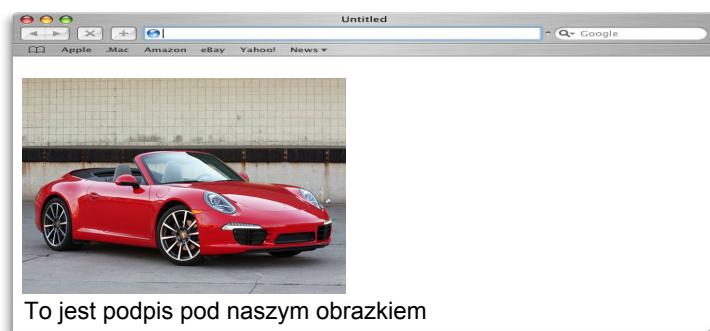
Podstawowe znaczniki w HTML5

HTML5 wprowadza również pewną nowość ułatwiającą osadzanie ilustracji. Wcześniej aby osadzić na stronie obraz wraz z podpisem musiliśmy:

```
<div>
  
  <p>To jest podpis pod naszym obrazem</p>
</div>
```

Standard HTML oferuje nam rozwiązanie semantyczne `<figure>` dzięki któremu możemy w sposób zorganizowany w ramach jednego bloku osadzać obrazy:

```
<figure>
  
  <figcaption>To jest podpis pod naszym obrazem</figcaption>
</figure>
```



W obu przypadkach efekt jest ten sam.

KONIEC WYKŁADU 3



UNIWERSYTET
JAGIELŁOŃSKI
W KRAKOWIE

Zaawansowane Techniki WWW (HTML, CSS i JavaScript)

Dr inż. Marcin Zieliński

Środa 15:30 - 17:00 sala: A-1-04

WYKŁAD 4

Wykład dla kierunku: Informatyka Stosowana II rok

Rok akademicki: 2015/2016 - semestr zimowy



Przypomnienie

Podstawowe znaczniki HTML

Semantyczny HTML 5

Budowa serwisu internetowego - model pudełkowy

HTML5 - Ogólne zasady

1. Wszystko co było przed wprowadzeniem HTML5 działa w niezaburzony sposób oraz jest obsługiwane.
2. Standaryzacja dotychczasowych technik oraz dostosowanie ich do istniejących przeglądarek.
3. Praktyczność - oznaczająca, że wszelkie zmiany w standardzie języka powinny być uzasadnione praktycznością ich wprowadzania.



Ogólna specyfikacja języka:



www.w3.org/TR/html5

Najnowsze zmiany języka:



<https://html.spec.whatwg.org/multipage/>

DTD dla HTML5

<!DOCTYPE ... >

HTML 4.01
Strict → <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >

HTML 4.01
Strict → <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

HTML 4.01
Transitional → <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

XHTML 1.0
Strict → <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

XHTML 1.1 → <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

HTML 5 → <!DOCTYPE html>



Struktura dokumentu w HTML5

<!DOCTYPE html>	deklaracja typu dokumentu
<html>	początek dokumentu hipertekstowego
<head>	sekcja nagłówkowa
<title> Tytuł strony </title>	tytuł strony
</head>	
<body> <p>Treść strony</p> </body>	sekcja zawartości strony
</html>	koniec dokumentu hipertekstowego

MODEL PUDEŁKOWY

Kodowanie znaków i język strony

Kodowanie jest standardem według którego komputery konwertują tekst na sekwencję bajtów przy zapisie i odczytanie danych z pliku. Jest wiele rodzajów kodowania jednak w ostatnich latach ze względu na szybkość działania i obsługę dużej liczby znaków najczęściej wykorzystuję się standard UTF-8.

W HTML5 kodowanie znaków jest bardzo ułatwione i sprowadza się do jednej dyrektywy meta w części head naszej strony internetowej:

Kodowanie:

```
<head>
  <meta charset="utf-8">
  <title> Gżegżolka </title>
</head>
```

Kodowanie znaków i język strony

Kodowanie jest standardem według którego komputery konwertują tekst na sekwencję bajtów przy zapisie i odczytanie danych z pliku. Jest wiele rodzajów kodowania jednak w ostatnich latach ze względu na szybkość działania i obsługę dużej liczby znaków najczęściej wykorzystuję się standard UTF-8.

W HTML5 kodowanie znaków jest bardzo ułatwione i sprowadza się do jednej dyrektywy meta w części head naszej strony internetowej:

Kodowanie:

```
<head>
  <meta charset="utf-8">
  <title> Gżegżolka </title>
</head>
```

Dodatkowo dobrą praktyką i oznaką profesjonalizmu jest wskazanie języka strony, co pozwala np. wyszukiwarką na odfiltrowywanie treści pasujących tylko do języka wyszukującego. Atrybut wskazujący na język można dodać do dowolnego znacznika jednak najczęściej umieszcza się go w znaczniku html na początku strony:

```
<html lang="pl">
```



Dodawanie arkusza stylu i skryptów

W ostatecznym kształcie strona internetowa korzysta z arkusza sylu (CSS) który nadaje jej odpowiedni wygląd oraz ze skryptu, który obsługuje różne zdarzenia na stronie. Informacje o stylach i skryptach są najczęściej umieszczone w osobnych plikach. Aby dołączyć je do naszej strony musimy zapisać odpowiednie dyrektywy w sekcji head:

Style CSS:

```
-----  
|<head>  
|  <link href="styl.css" rel="stylesheet">  
|</head>  
-----
```

Dodawanie arkusza stylu i skryptów

W ostatecznym kształcie strona internetowa korzysta z arkusza sylu (CSS) który nadaje jej odpowiedni wygląd oraz ze skryptu, który obsługuje różne zdarzenia na stronie. Informacje o stylach i skryptach są najczęściej umieszczone w osobnych plikach. Aby dołączyć je do naszej strony musimy zapisać odpowiednie dyrektywy w sekcji head:

Style CSS:

```
-----  
|<head>  
|  <link href="styl.css" rel="stylesheet">  
|</head>  
-----
```



W porównaniu z
HTML4 i XHTML
nie ma atrybutu
`type="text/css"`

Dodawanie arkusza stylu i skryptów

W ostatecznym kształcie strona internetowa korzysta z arkusza sylu (CSS) który nadaje jej odpowiedni wygląd oraz ze skryptu, który obsługuje różne zdarzenia na stronie. Informacje o stylach i skryptach są najczęściej umieszczone w osobnych plikach. Aby dołączyć je do naszej strony musimy zapisać odpowiednie dyrektywy w sekcji head:

Style CSS:

```
<head>
  <link href="styl.css" rel="stylesheet">
</head>
```

W porównaniu z
HTML4 i XHTML
nie ma atrybutu
type="text/css"



Skrypty:

```
<head>
  <script src="skrypt.js"></script>
</head>
```

Uwaga:
Ten znacznik musi
być zapisany
dokładnie w ten
sposób, a nie
<script/>

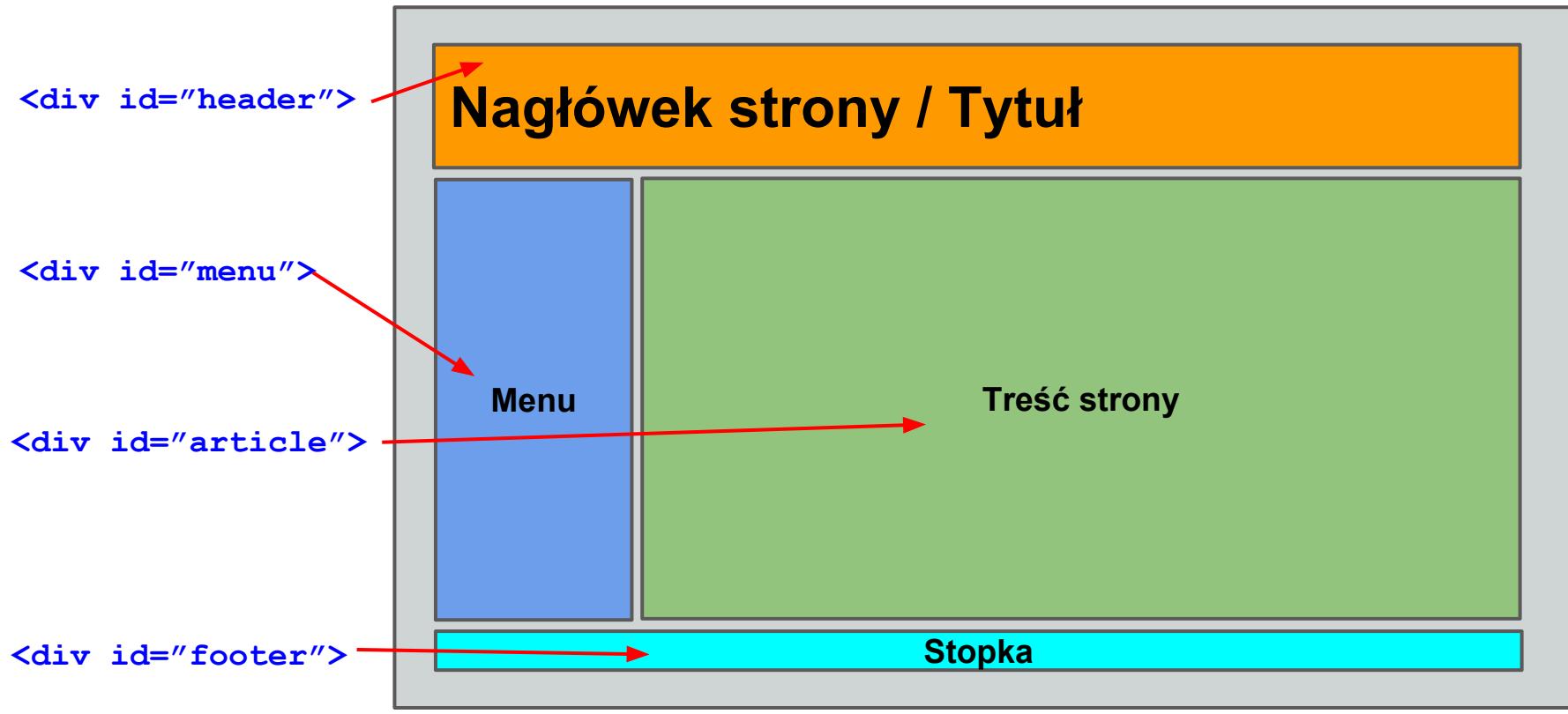


Ostateczna struktura

```
<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="utf-8">
    <link href="styl.css" rel="stylesheet">
    <title>Tytuł strony</title>
    <script scr="skrypt.js"></script>
  </head>
  <body>
    <p>Treść strony</p>
  </body>
</html>
```

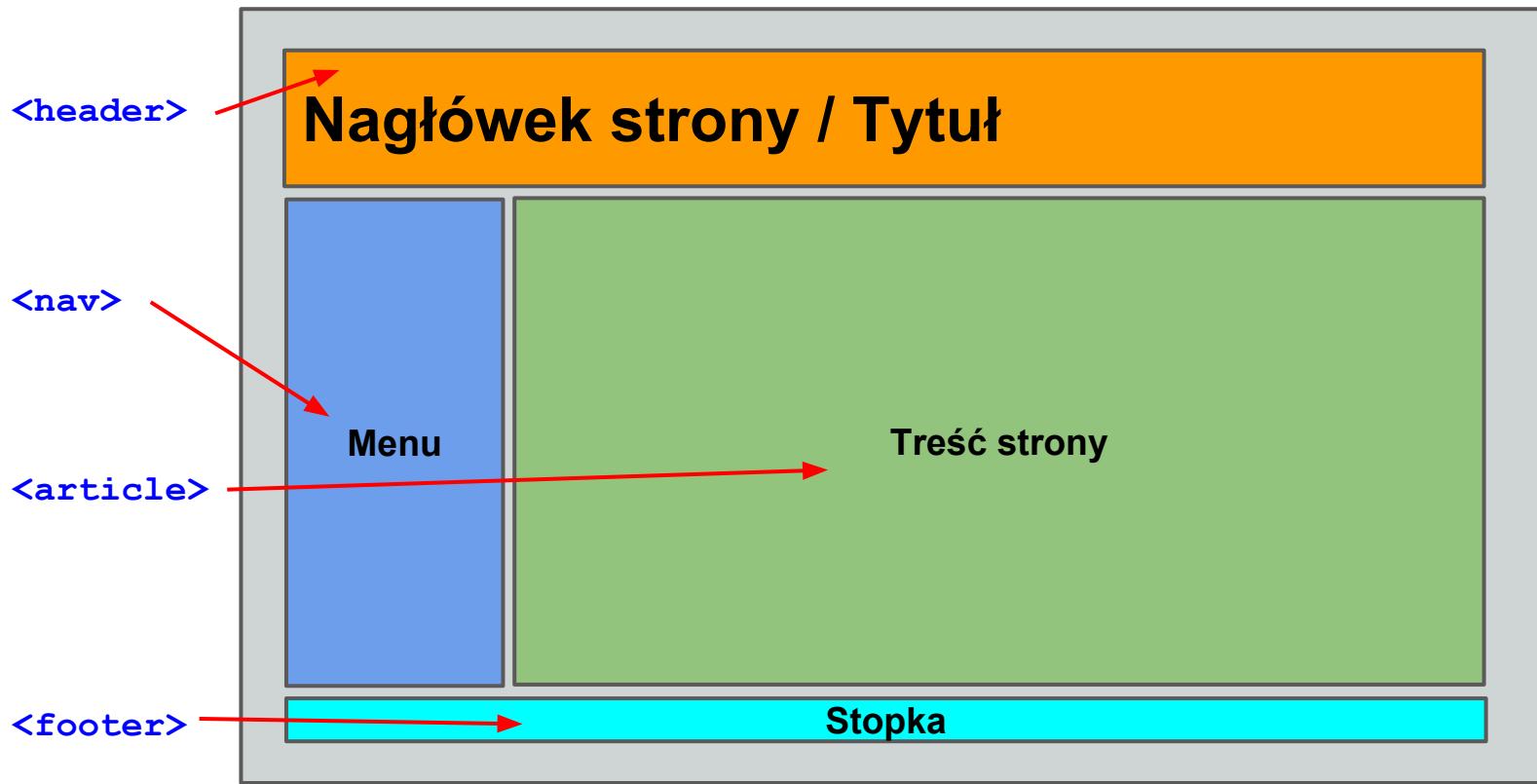
HTML5 i semantyka

Standard HTML5 wprowadza kilkanaście nowych znaczników oraz nowe i lepsze metody organizacji treści na stronie internetowej. Do tej pory strony były zbudowane najczęściej w oparciu o elementy podziału (elementy blokowe) <div>, które umożliwiały w miarę logiczne rozdzielenie treści na stronie.



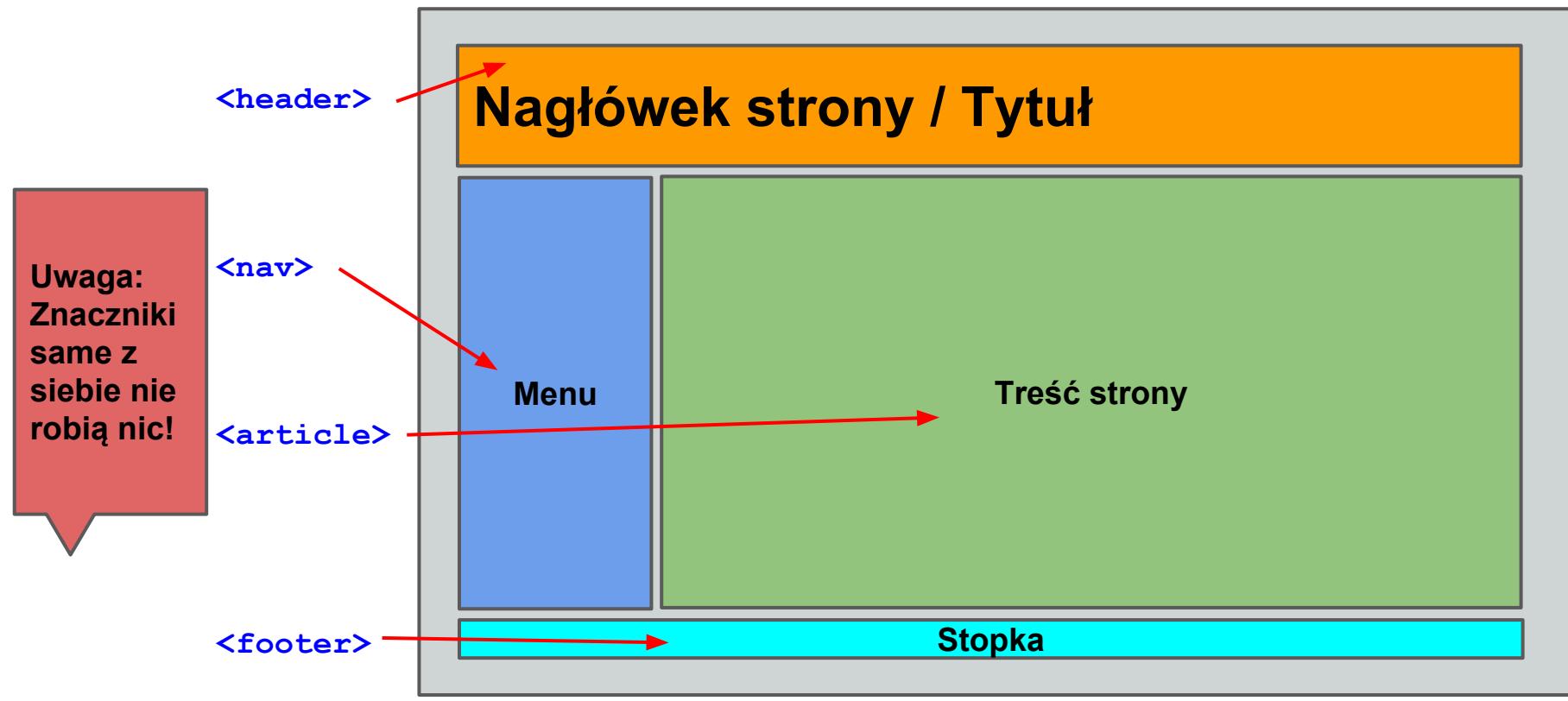
HTML5 i semantyka

Dzięki HTML5 mamy nowe znaczniki które organizują treści na stronie w sposób logiczny oraz dzięki temu są bardziej zoptymalizowane pod kątem wyszukiwarek. Dodatkowo pozwalają na przyjazną organizację strony i czynią stronę bardziej dostępną np. dla czytników ekranów ułatwiających przeglądanie stron osobą niedowidzącym.



HTML5 i semantyka

Dzięki HTML5 mamy nowe znaczniki które organizują treści na stronie w sposób logiczny oraz dzięki temu są bardziej zoptymalizowane pod kątem wyszukiwarek. Dodatkowo pozwalają na przyjazną organizację strony i czynią stronę bardziej dostępną np. dla czytników ekranów ułatwiających przeglądanie stron osobą niedowidzącym.



CSS (Kaskadowe Arkusze Stylu)

CSS (Cascading Style Sheets) [Kaskadowe Arkusze Stylu] - pozwalają na przeniesienie a jednocześnie oddzielenie warstwy formatowania (prezentacji) od treści/danych. Dzięki temu kod strony jest jasny prosty i przejrzysty, a streść strony jest niezależna od jej wyglądu. Same arkusze CSS nie wprowadzają nowych elementów w języku HTML natomiast pozwalają na jego optymalizację i lepsze go wykorzystanie.

Standard CSS został opracowany przez organizację W3C w 1996 roku, jednak jego początki sięgają roku 1994 kiedy został zaproponowany przez Håkon Wium Lie.



Håkon Wium Lie

CSS (Kaskadowe Arkusze Stylu)

CSS (Cascading Style Sheets) [Kaskadowe Arkusze Stylu] - pozwalają na przeniesienie a jednocześnie oddzielenie warstwy formatowania (prezentacji) od treści/danych. Dzięki temu kod strony jest jasny prosty i przejrzysty, a streść strony jest niezależna od jej wyglądu. Same arkusze CSS nie wprowadzają nowych elementów w języku HTML natomiast pozwalają na jego optymalizację i lepsze go wykorzystanie.

Standard CSS został opracowany przez organizację W3C w 1996 roku, jednak jego początki sięgają roku 1994 kiedy został zaproponowany przez Håkon Wium Lie.



Strona projektu:

<http://www.w3.org/Style/CSS/>

CSS 2.1 wersja rekomendowana od 7 lipca 2011



Håkon Wium Lie

CSS (Kaskadowe Arkusze Stylu)

CSS (Cascading Style Sheets) [Kaskadowe Arkusze Stylu] - pozwalają na przeniesienie a jednocześnie oddzielenie warstwy formatowania (prezentacji) od treści/danych. Dzięki temu kod strony jest jasny prosty i przejrzysty, a streść strony jest niezależna od jej wyglądu. Same arkusze CSS nie wprowadzają nowych elementów w języku HTML natomiast pozwalają na jego optymalizację i lepsze go wykorzystanie.

Standard CSS został opracowany przez organizację W3C w 1996 roku, jednak jego początki sięgają roku 1994 kiedy został zaproponowany przez Håkon Wium Lie.



Strona projektu:

<http://www.w3.org/Style/CSS/>

CSS 2.1 wersja rekomendowana od 7 lipca 2011

CSS 3.0 w trakcie standaryzacji



Håkon Wium Lie

CSS (Kaskadowe Arkusze Stylu)

Kaskadowe arkusze stylu dzielimy na trzy grupy ze względu na miejsce wystąpienia:

1. **Wbudowane (inline-style)** - są zapisane w miejscu ich działania, tzn. w znaczniku, któremu mają nadawać specyficzne cechy.
2. **Osadzone (embedded-style)** - są zapisane za pomocą zacznika `<style>` w sekcji head dokumentu hipertekstowego.
3. **Dołączone (linked-style)** - są zapisane w osobnych plikach o rozszerzeniu .css.



CSS (Kaskadowe Arkusze Stylu)

Kaskadowe arkusze stylu dzielimy na trzy grupy ze względu na miejsce wystąpienia:

1. **Wbudowane (inline-style)** - są zapisane w miejscu ich działania, tzn. w znaczniku, któremu mają nadawać specyficzne cechy.
2. **Osadzone (embedded-style)** - są zapisane za pomocą zacznika <style> w sekcji head dokumentu hipertekstowego.
3. **Dołączone (linked-style)** - są zapisane w osobnych plikach o rozszerzeniu .css.

Która z metod zapisywania CSS ma największy priorytet ?

CSS (Kaskadowe Arkusze Stylu)

Kaskadowe arkusze stylu dzielimy na trzy grupy ze względu na miejsce wystąpienia:

1. **Wbudowane (inline-style)** - są zapisane w miejscu ich działania, tzn. w znaczniku, któremu mają nadawać specyficzne cechy.
2. **Osadzone (embedded-style)** - są zapisane za pomocą zacznika `<style>` w sekcji head dokumentu hipertekstowego.
3. **Dołączone (linked-style)** - są zapisane w osobnych plikach o rozszerzeniu .css.

Która z metod zapisywania CSS ma największy priorytet ?

KASKADOWOŚĆ



CSS (Kaskadowe Akusze Stylu)

**KASKADOWOŚĆ
(DZIEDZICZENIE)**

CSS (Kaskadowe Akusze Stylu)

KASKADOWOŚĆ (DZIEDZICZENIE)

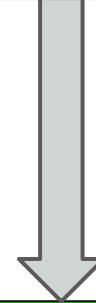


Kolejność oddziaływania cech zapisanych w regułach CSS na elementy strony.

CSS (Kaskadowe Akusze Stylu)

KASKADOWOŚĆ (DZIEDZICZENIE)

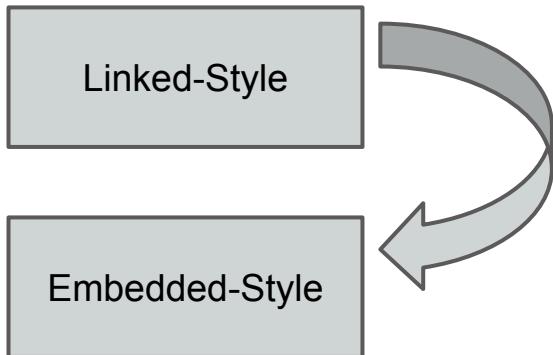
Linked-Style



Kolejność oddziaływanie cech zapisanych w regułach CSS na elementy strony.

CSS (Kaskadowe Akusze Stylu)

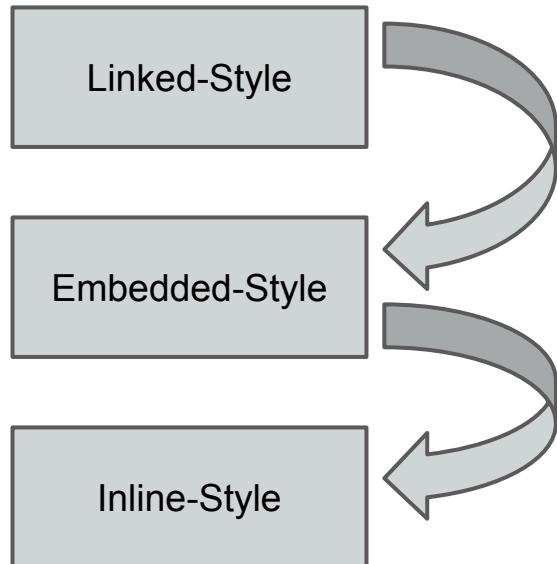
KASKADOWOŚĆ (DZIEDZICZENIE)



Kolejność oddziaływanie cech zapisanych w regułach CSS na elementy strony.

CSS (Kaskadowe Akusze Stylu)

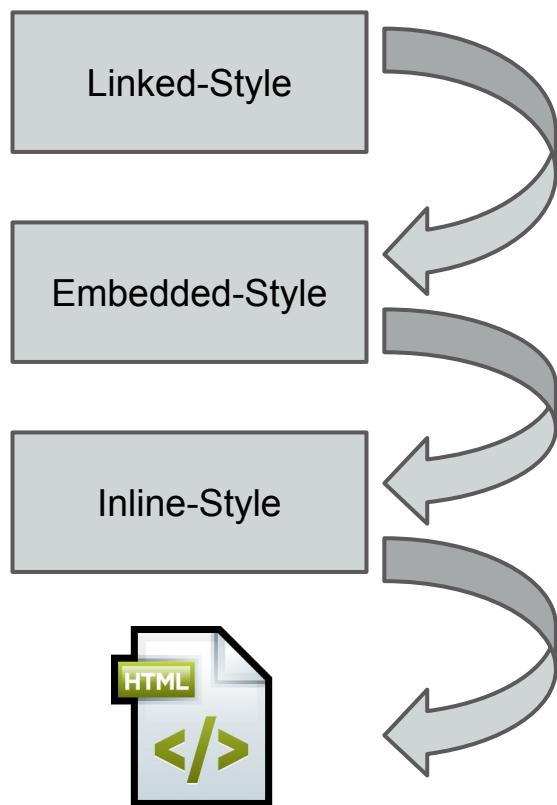
KASKADOWOŚĆ (DZIEDZICZENIE)



Kolejność oddziaływania cech zapisanych w regułach CSS na elementy strony.

CSS (Kaskadowe Akusze Stylu)

KASKADOWOŚĆ (DZIEDZICZENIE)



Kolejność oddziaływania cech zapisanych w regułach CSS na elementy strony.

CSS (Kaskadowe Akusze Stylu)

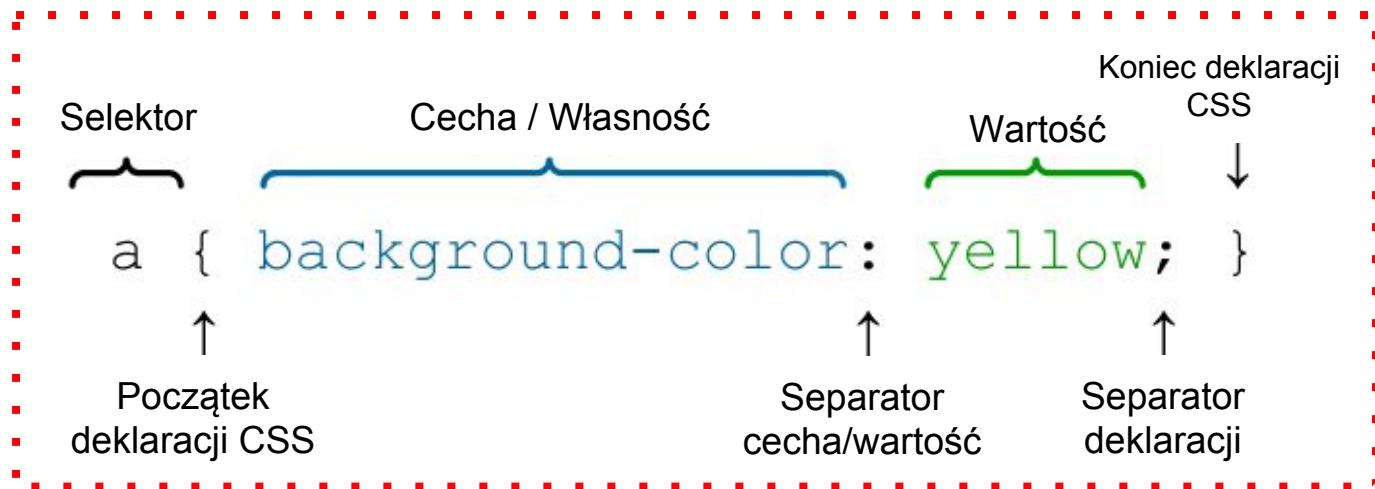
Kaskadowy arkusz stylu zawiera definicję zestawu specyficznych właściwości jakie mają zostać nadane przez przeglądarkę wybranemu znacznikowi HTML, a następnie wyświetcone użytkownikowi. Reguły w CSS polegają na określeniu dla jakiego znacznika ma zostać nadana specyficzna właściwość oraz jaką ta właściwość ma być. W języku CSS znacznik jest określany mianem "selekторa" natomiast właściwość mianem "cechy". Ogólny zapis formatu CSS wygląda następująco:

```
.....  
:selektor1, selektor2, ... { cecha1: wartość;  
                            cecha2: wartość;  
                            cecha3: wartość;  
                            ... ;  
}
```

W ogólności liczba selektorów oraz cech jest dowolna.

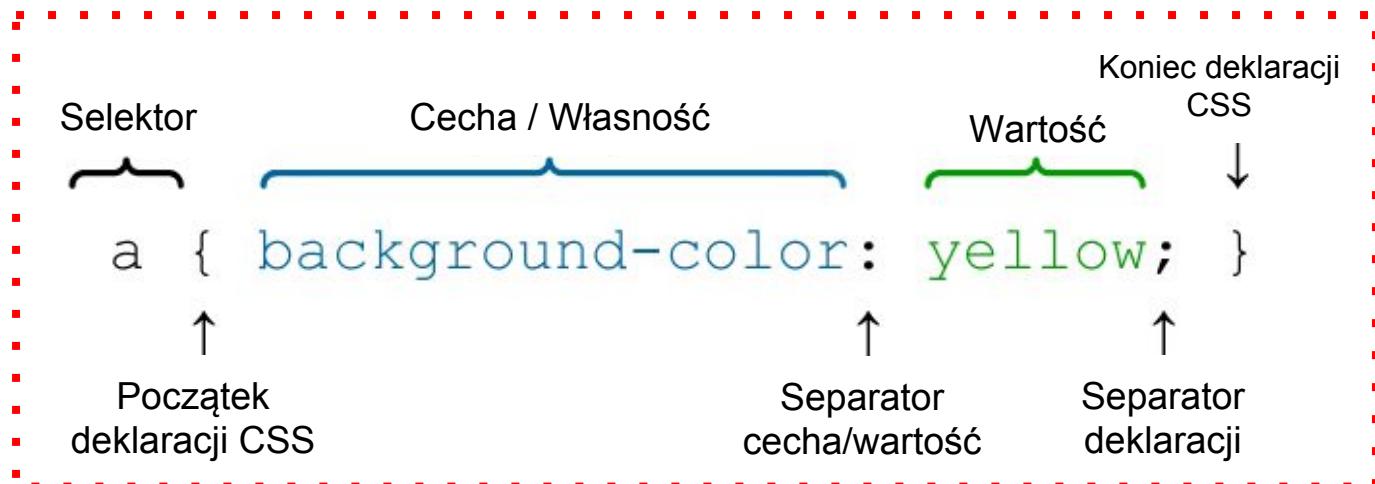
CSS (Kaskadowe Akusze Stylu)

Przykład:



CSS (Kaskadowe Akusze Stylu)

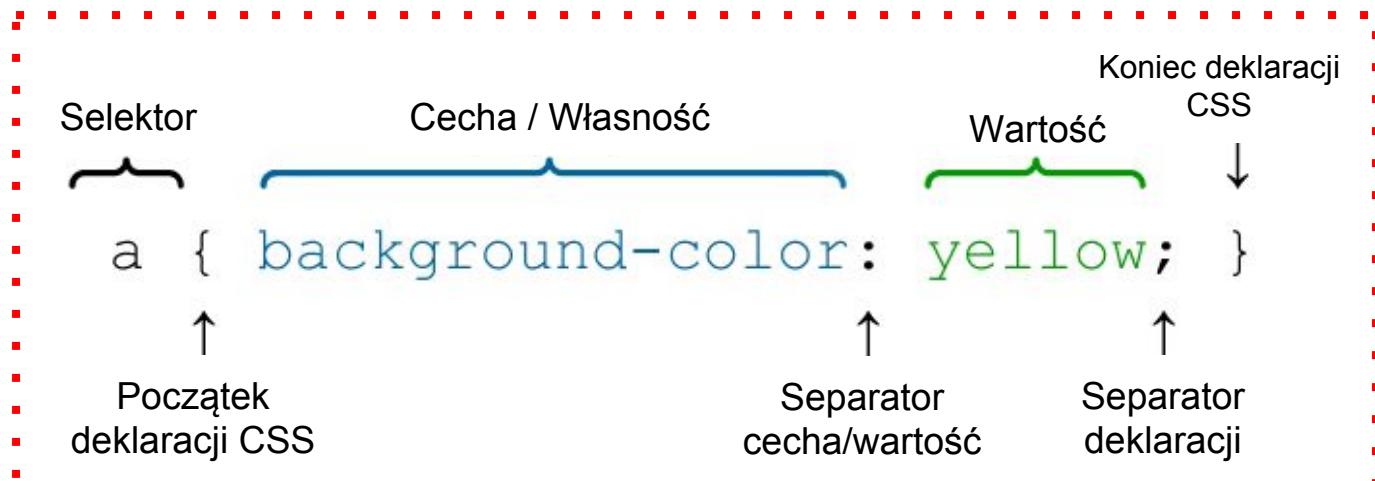
Przykład:



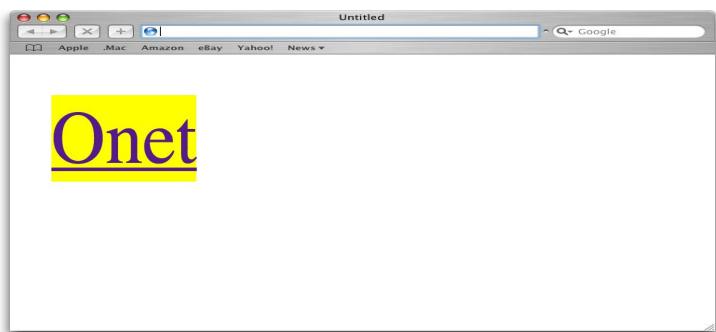
Przykładowa deklaracja powoduje ustawienie koloru tła dla wszystkich hiperłączy na kolor żółty.

CSS (Kaskadowe Akusze Stylu)

Przykład:



Przykładowa deklaracja powoduje ustawienie koloru tła dla wszystkich hiperłączy na kolor żółty.



```
<a href="http://www.onet.pl">onet</a>
```

CSS (Kaskadowe Akusze Stylu)

Linked-Style

```
<body>
  <a href="http://www.onet.pl">
    Onet
  </a>
</body>
```

Plik
.html

```
a { background-color:
      yellow;
}
```

Plik
.css

CSS (Kaskadowe Akusze Stylu)

Linked-Style

```
<body>
  <a href="http://www.onet.pl">
    Onet
  </a>
</body>
```

Plik
.html

```
a { background-color:
      yellow;
    }
```

Plik
.css

Embedded-Style

```
<head>
<style>
  a { background-color: yellow;
    }
</style>
<body>
  <a href="http://www.onet.pl">onet</a>
</body>
```

Plik
.html

CSS (Kaskadowe Akusze Stylu)

Linked-Style

```
<body>
  <a href="http://www.onet.pl">
    Onet
  </a>
</body>
```

Plik
.html

```
a { background-color:
      yellow;
    }
```

Plik
.css

Embedded-Style

```
<head>
<style>
  a { background-color: yellow;
    }
</style>
<body>
  <a href="http://www.onet.pl">onet</a>
</body>
```

Plik
.html

Inline-Style

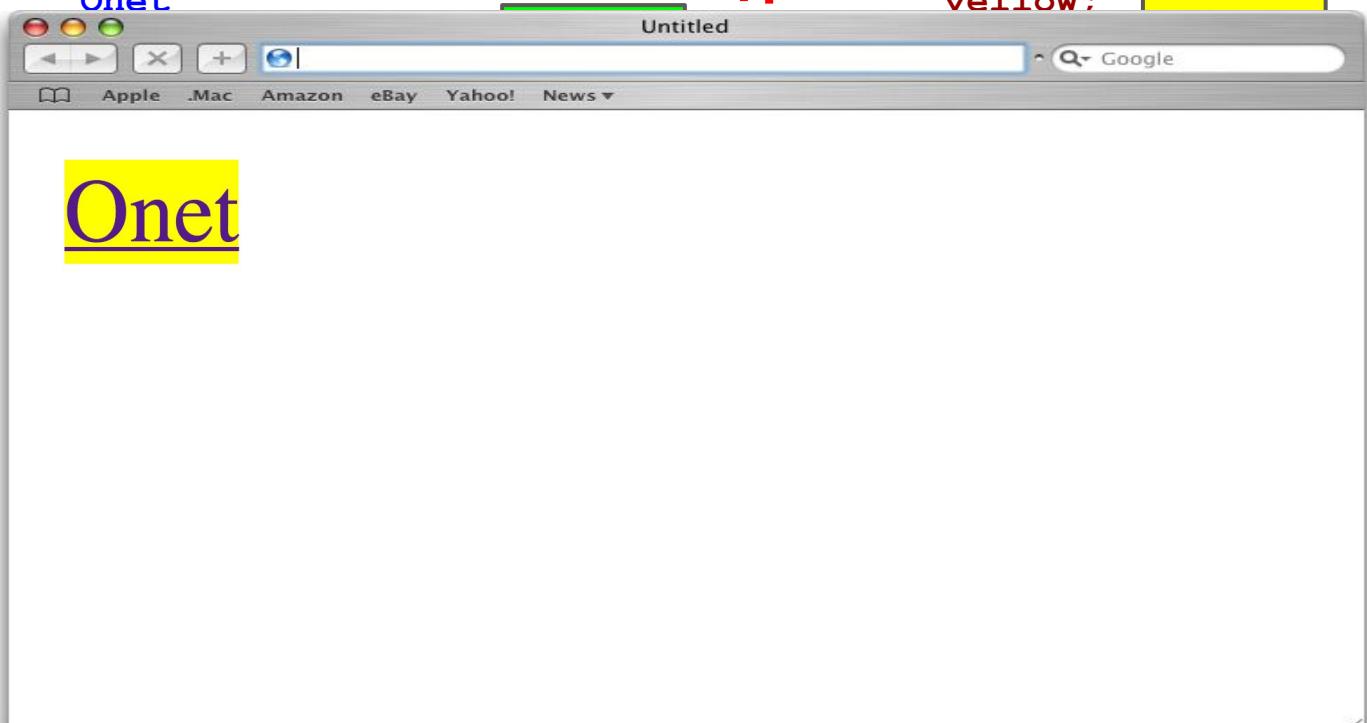
```
<a style="background-color:yellow;" href="http://www.onet.pl">
  Onet
</a>
```

Plik
.html

CSS (Kaskadowe Akusze Stylu)

Linked-Style

```
<body>
  <a href="http://www.onet.pl">Onet</a>
    a { background-color: yellow; }
```



Embedded-Style

```
<html>
  <a href="http://www.onet.pl">
    Onet
  </a>
```

.html

Inline-Style

CSS (Kaskadowe Akusze Stylu)

Linked-Style

```
<body>
  <a href="http://www.onet.pl">
    Onet
  </a>
</body>
```

a { background-color: green; }

Plik
.html

Plik
.css

CSS (Kaskadowe Akusze Stylu)

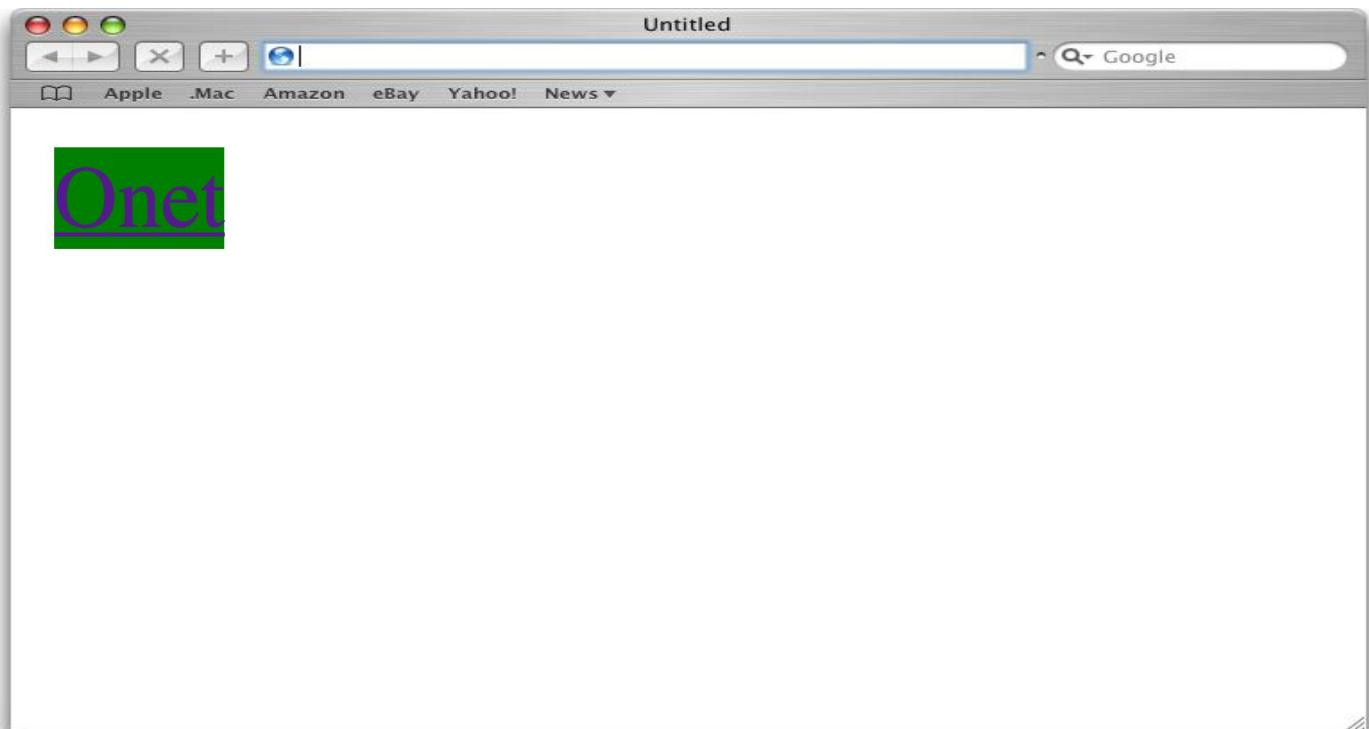
Linked-Style

```
<body>
  <a href="http://www.onet.pl">
    Onet
  </a>
</body>
```

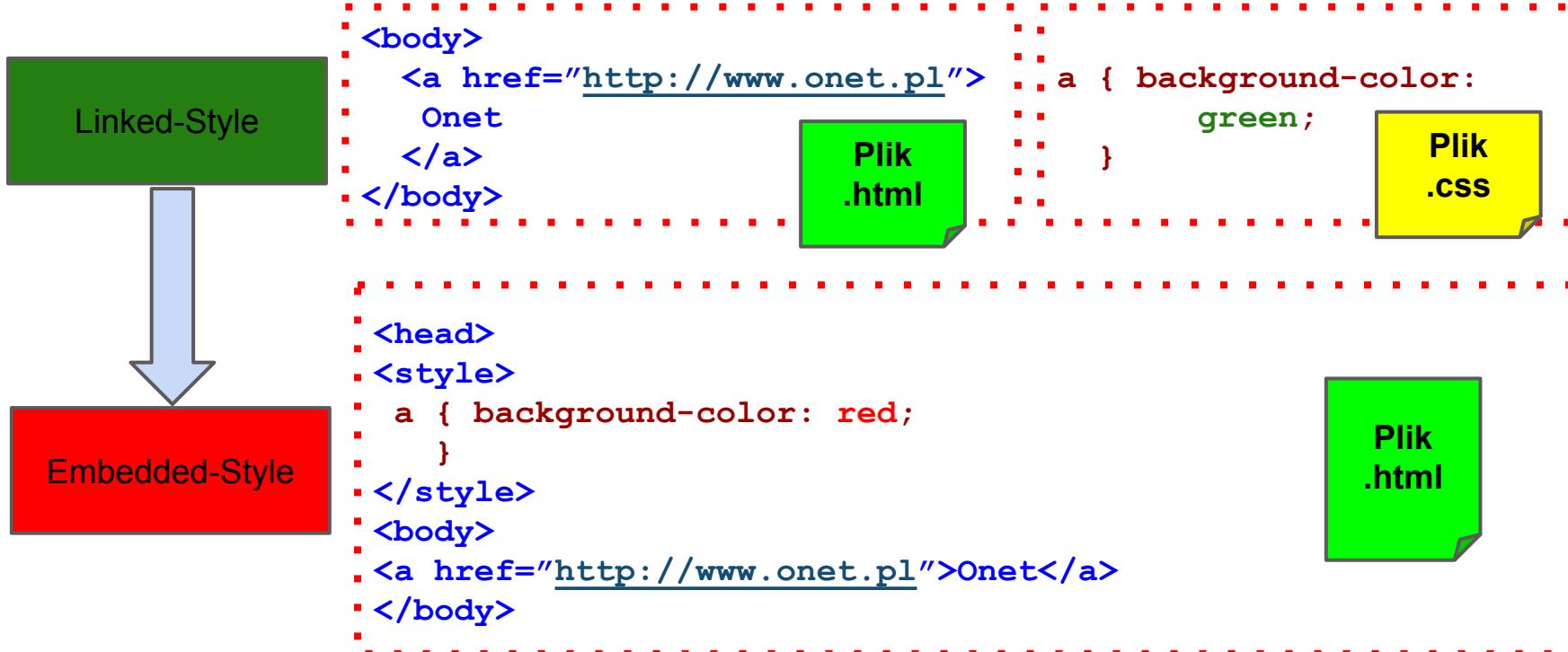
Plik
.html

```
a { background-color:  
      green;  
}
```

Plik
.css

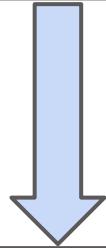


CSS (Kaskadowe Akusze Stylu)



CSS (Kaskadowe Akusze Stylu)

Linked-Style



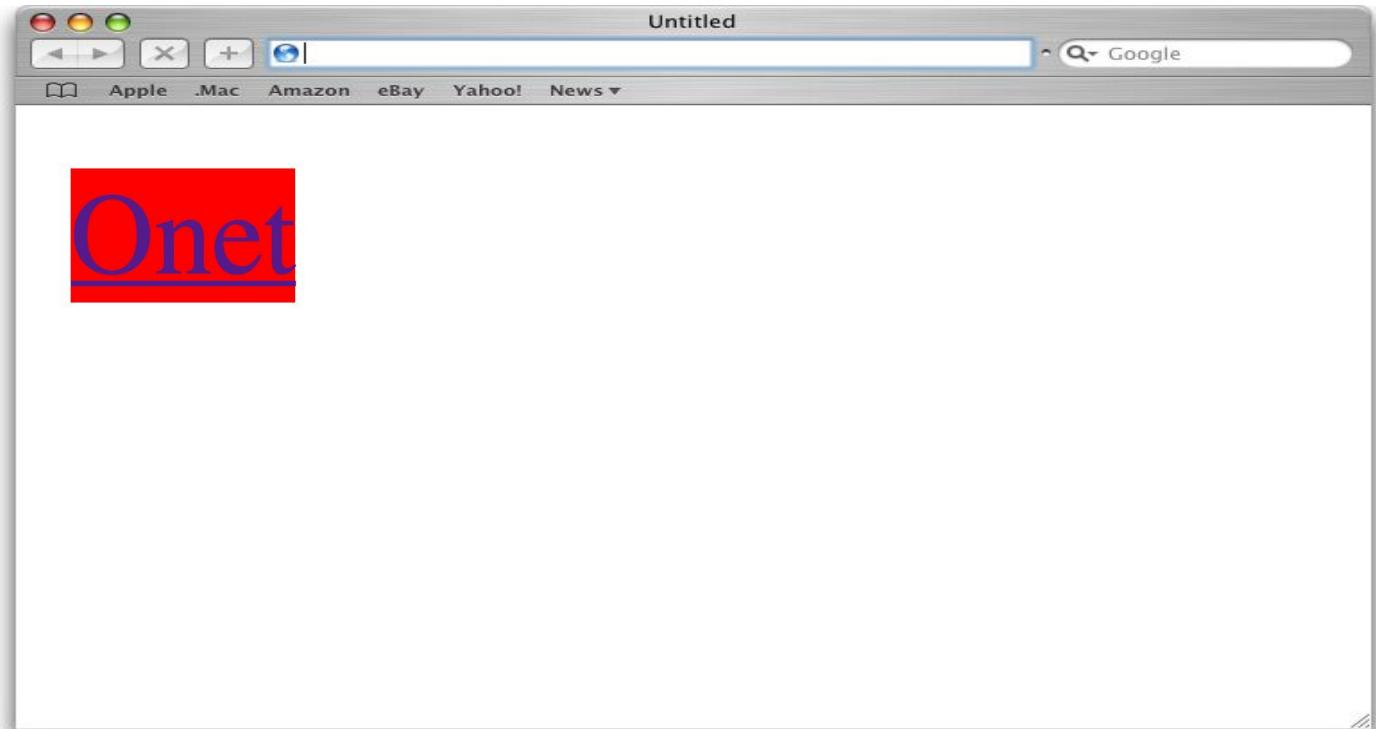
Embedded-Style

```
<body>
  <a href="http://www.onet.pl">
    Onet
  </a>
</body>
```

Plik
.html

```
a { background-color:
      green;
}
```

Plik
.css

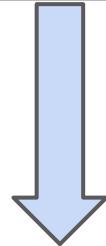


CSS (Kaskadowe Akusze Stylu)

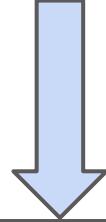


CSS (Kaskadowe Akusze Stylu)

Linked-Style



Embedded-Style



Inline-Style

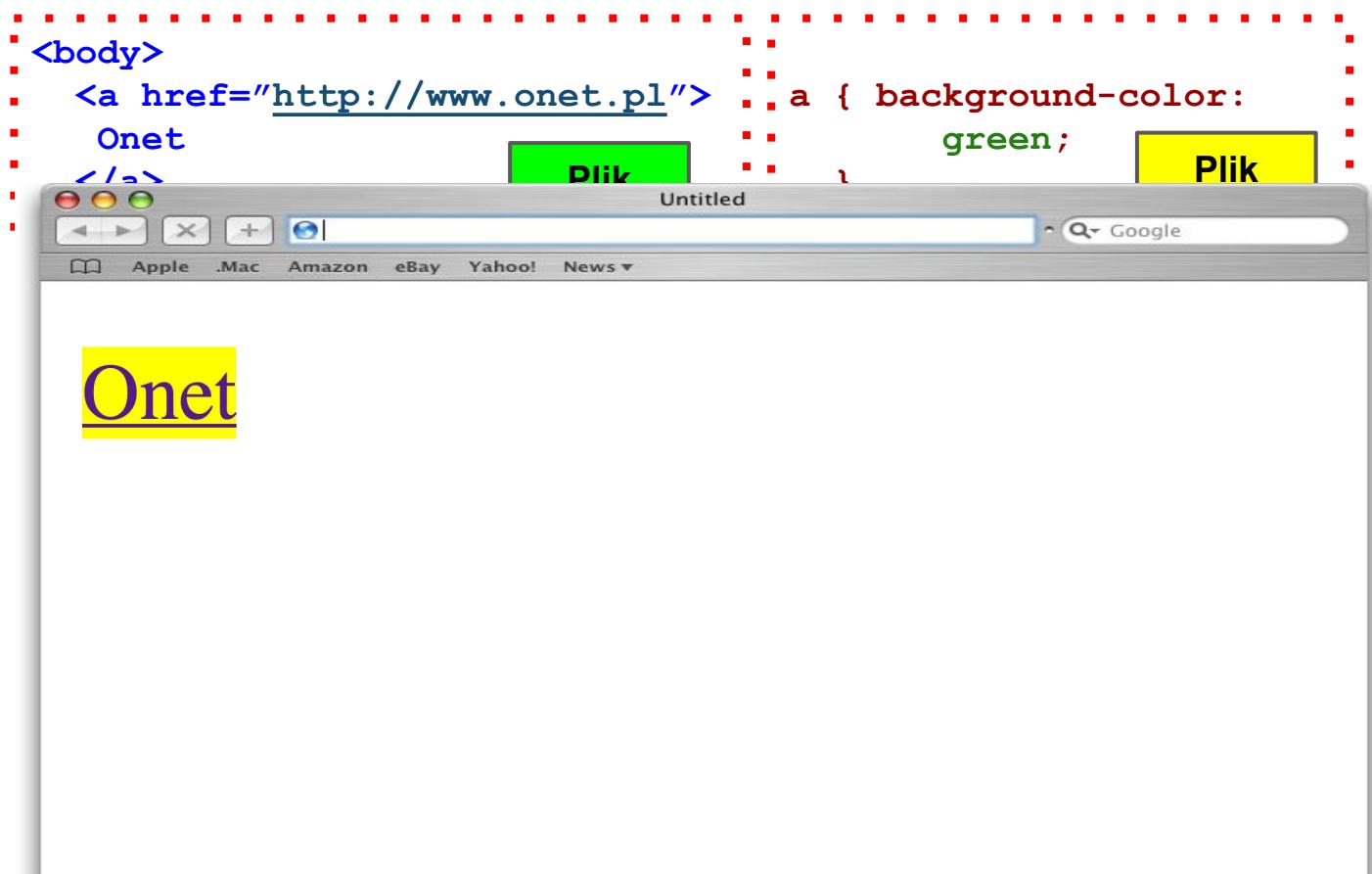
```
<body>
  <a href="http://www.onet.pl">onet
</a>
```

Plik

Untitled

```
a { background-color:
      green; }
```

Plik



onet

```
</a>
```



CSS (Kaskadowe Akusze Stylu)

Dobre praktyki w pisaniu CSS:

1. **Stosujmy podobnie jak w html odstępy i wcięcia co pozwala na utrzymanie przejrzystości kodu.**
2. **Starajmy się zapisywać style CSS w dołączonych plikach tak aby oddzielić warstwę formatowania od danych.**
3. **Dobrą praktyką jest nie nazywanie dołączonych plików CSS np. "styl.css".**



Klasy i identyfikatory

Jaka jest różnica pomiędzy “klasą” a “identyfikatorem” ?

Klasy i identyfikatory

Jaka jest różnica pomiędzy “klasą” a “identyfikatorem” ?

Na jednej stronie (jednym pliku) może wystąpić **WIELE** elementów należących do danej klasy (np. znaczniki p/div/span):

```
<p class="czerwony">
    Tekst rozdziału bardzo podobny znacznik do paragrafu p!
</p>
<h1 class="czerwony">
    Czerwony tytuł nagłówka
</h1>
```

Na jednej stronie (jednym pliku) może wystąpić **TYLKO 1 RAZ** element oznaczony wybranym identyfikatorem:

```
<h2 id="zielony">
    Ten tytuł jest zielony
</h2>
```

Klasy i identyfikatory

W ogólności selektorem nie musi być sam znacznik! HTML umożliwia wprowadzenie oznaczeń dla znaczników za pomocą tzw. "klas" oraz "identyfikatorów", które pozwalają na odróżnienie od siebie elementów strony:

Klasa:

```
<div class="content">
  <p> Tekst rozdziału bardzo podobny znacznik do paragrafu p! </p>
</div>
<div class="foot">
  <p> prawa zastrzeżone </p>
</div>
```

Identyfikator:

```
<div id="main">
  Tekst rozdziału bardzo podobny znacznik do paragrafu p!
</div>
```

Jaka jest różnica pomiędzy "klasą" a "identyfikatorem" ?

Klasy i identyfikatory

Jak klasy i identyfikatory pomagają w definicji stylu strony ?

Możemy za pomocą klasy lub identyfikatora konkretnie wybrać element(y) dla którego mają być nadane specyficzne właściwości i cechy.

Selektor klasy:

```
:selektor.nazwaklasy { cecha1: wartość;  
                        cecha2: wartość;  
                        ...;  
}
```

↑
Separator klasy

Selektor identyfikatora:

```
:selektor#nazwaklasy { cecha1: wartość;  
                        cecha2: wartość;  
                        ...;  
}
```

↑
Separator
identyfikatora

Klasy i identyfikatory

W ogólności reguły CSS nie muszą być związane z konkretnym znacznikiem (selektorem znacznikowym), ale wystarczy aby odnosiły się do konkretnej klasy lub identyfikatora:

Selektor klasy:

```
nazwaklasy { cecha1: wartość;  
             cecha2: wartość;  
             ...;  
 }
```

Separator klasy

Selektor identyfikatora:

```
#nazwaklasy { cecha1: wartość;  
                cecha2: wartość;  
                ...;  
 }
```

Separator
identyfikatora

Klasy i identyfikatory

Selektor klasy:

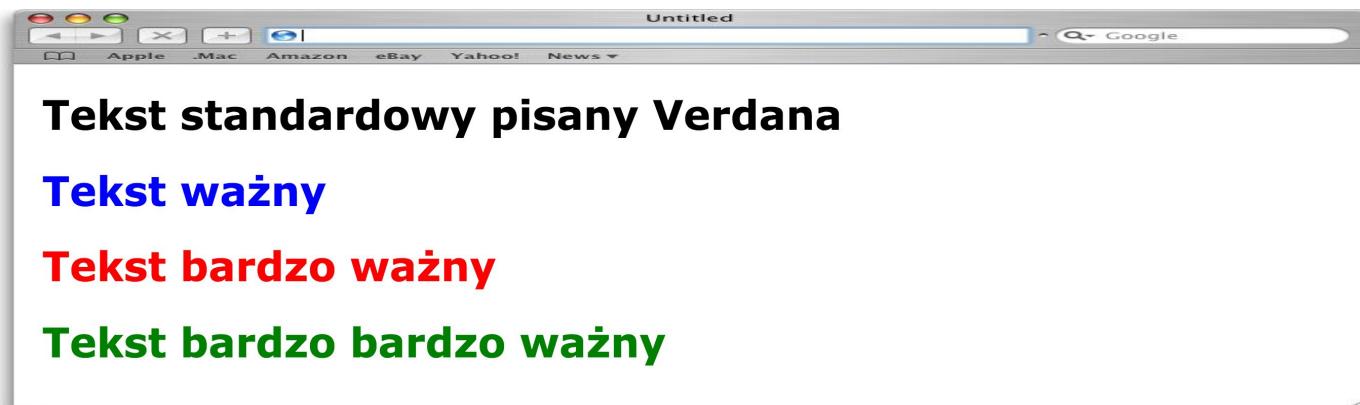
```
h1 { font-family: Verdana; }

h1.wazny { color: blue; }

h1.bwazny { color: red; }

h1.bbwazny {color: green;}

<h1> Tekst standardowy pisany Verdana </h1>
<h1 class="wazny"> Tekst ważny </h1>
<h1 class="bwazny"> Tekst bardzo ważny </h1>
<h1 class="bbwazny"> Tekst bardzo bardzo ważny </h1>
```

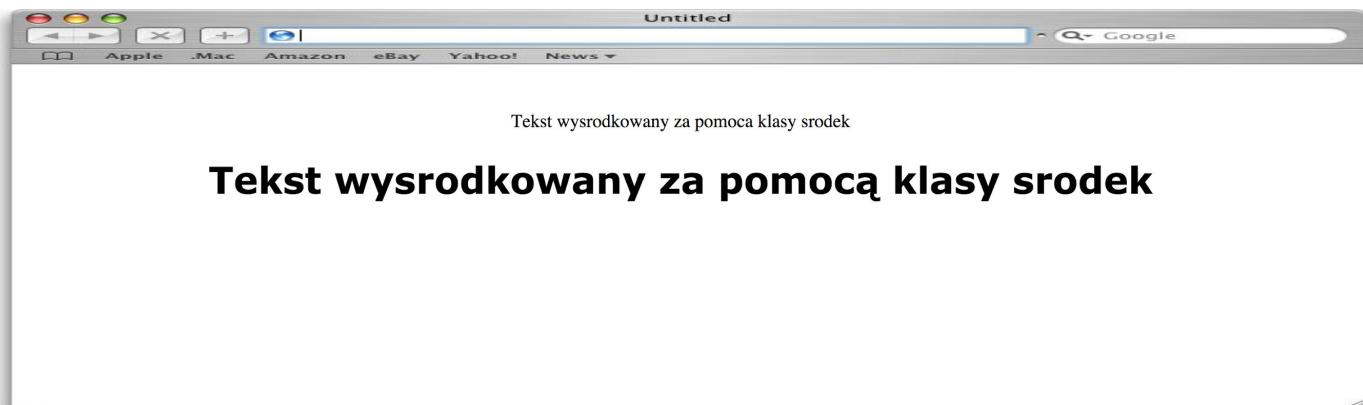


Klasy i identyfikatory

Selektor klasy:

```
.srodek {  
    text-align: center;  
}
```

```
<p class="srodek"> Tekst wysrodkowany za pomoca klasy srodek </p>  
<h1 class="srodek"> Tekst wysrodkowany za pomocą klasy srodek </h1>
```

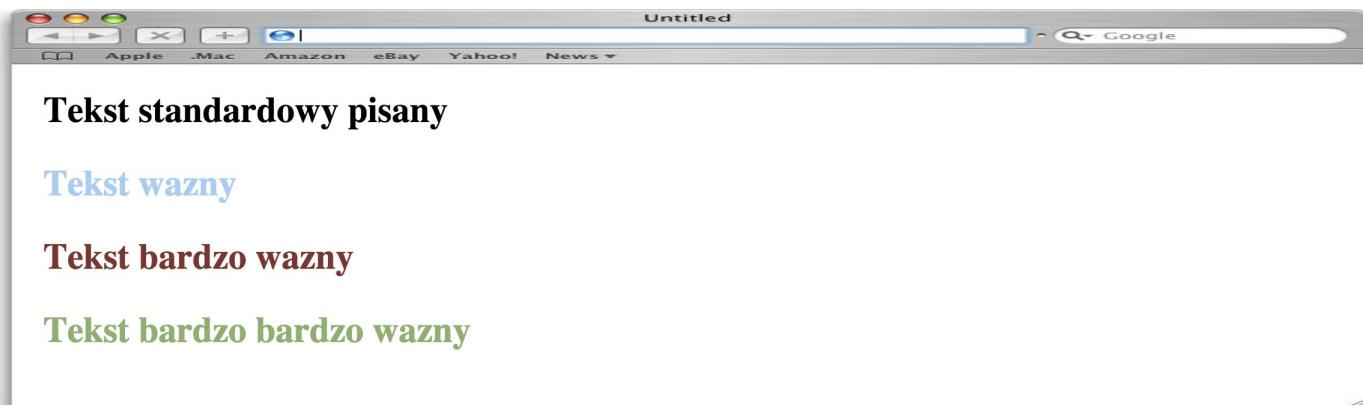


Klasy i identyfikatory

Selektor identyfikatora:

```
h2 { font-family: Times; }  
h2#wazny {color: #ABCDEF; }  
h2#bwazny {color: #793732; }  
h2#bbwazny {color: #91AF72; }  
  
<h2> Tekst standardowy pisany </h2>  
<h2 id="wazny"> Tekst wzny </h2>  
<h2 id="bwazny"> Tekst bardzo wzny </h2>  
<h2 id="bbwazny"> Tekst bardzo bardzo wzny </h2>
```

Kolory mogą być definiowane z całej palety 24 bitowej za pomocą identyfikatorów HEX.



Najczęściej używane cechy CSS

	Własności
Barwy	color background-color
Przestrzeń i odstępy	margin padding margin-left, margin-right, margin-top, margin-bottom padding-left, padding-right, padding-top, padding-bottom
Obramowanie	border-width border-style border-color border (ustawia grubość, styl i barwę za jednym zamachem)
Wyrównanie tekstu	text-align text-indent word-spacing letter-spacing line-height white-space

Najczęściej używane cechy CSS

	Własności
Fonty	font-family font-size font-weight font-style font-variant text-decoration @font-face
Wielkość	width height
Układ	position left, right float, clear
Grafika	background-image background-repeat background-position

Selektor uniwersalny

Selektor uniwersalny odnosi się do wszystkich elementów zapisanych w dokumencie hipertekstowym i pozwala na nadanie odpowiednich cech i własności wszystkim znacznikom. Selektor uniwersalny jest zapisywany za pomocą gwiazdki:

```
* { cecha1: wartość;  
    cecha2: wartość;  
    ...;  
}
```

Selektor uniwersalny

Selektor uniwersalny odnosi się do wszystkich elementów zapisanych w dokumencie hipertekstowym i pozwala na nadanie odpowiednich cech i własności wszystkim znacznikom. Selektor uniwersalny jest zapisywany za pomocą gwiazdki:

```
* { cecha1: wartość;  
    cecha2: wartość;  
    ...;  
}
```

W praktyce najczęściej stosuje się je do wyzerowania marginesów i odstępów we wszystkich elementach strony np.:

```
* { margin:0 ;  
    padding: 0;  
}
```

Selektor potomka

Selektor potomka stosuje regułę CSS tylko wtedy kiedy dany selektor znajdzie się wewnątrz innego zdefiniowanego selektora na dowolnym poziomie zagnieżdżenia (w dowolnym pokoleniu). Selektor potomka jest oznaczony w regule CSS pisząc bez żadnego separatora nazwy dwóch selektorów:

```
:selektor-rodzica selektor-potomka ... { cecha1: wartość;  
                                         cecha2: wartość;  
                                         ...;  
                                         }  
                                         
```

Przykład:

```
div span { color:red; }
```

```
<div>  
    <span> Text </span>  
</div>
```

```
<div>  
    <p>  
        <span> Text </span>  
    </p>  
</div>
```

W obu przypadkach
reguła CSS będzie
zastosowana.

Selektor “dziecka”

Selektor dziecka stosuje regułę CSS tylko wtedy kiedy dany selektor znajdzie się bezpośrednio wewnątrz innego zdefiniowanego selektora. Selektor dziecka jest oznaczony w regule CSS pisząc pomiędzy nazwami dwóch selektorów znak “>” (większości):

```
: selektor-rodzica > selektor-dziecka ... { cecha1: wartość;  
                                              cecha2: wartość;  
                                              ...;  
                                              }
```

Przykład:

```
div > span { color:red; }
```

W tym przypadku
reguła CSS będzie
zastosowana.

```
<div>  
  <span> Text </span>  
</div>
```

W tym przypadku
reguła CSS **NIE**
BĘDZIE zastosowana.

```
<div>  
  <p>  
    <span> Text </span>  
  </p>  
</div>
```

Selektor “brata”

Selektor brata stosuje regułę CSS tylko wtedy kiedy dany selektor znajdzie się bezpośrednio **ZA** innym zdefiniowanym selektorem. Selektor brata jest oznaczony w regule CSS pisząc pomiędzy nazwami dwóch selektorów znak “+” (plus):

```
selektor + selektor-brata ... { cecha1: wartość;  
                                  cecha2: wartość;  
                                  ...;  
                                  }  
;
```

Przykład

1:

```
div + span { color:blue; }  
;  
  
<div>  
  <h1> Tytuł </h1>  
</div>  
<span> Text </span>  
;
```

Selektor “brata”

Selektor brata stosuje regułę CSS tylko wtedy kiedy dany selektor znajdzie się bezpośrednio **ZA** innym zdefiniowanym selektorem. Selektor brata jest oznaczony w regule CSS pisząc pomiędzy nazwami dwóch selektorów znak “+” (plus):

```
selektor + selektor-brata ... { cecha1: wartość;  
                                  cecha2: wartość;  
                                  ...;  
                                  }  
;
```

Przykład

2:

```
div + p + span { color:blue; }  
;  
;  
;  
;  
;  
;  
;  
;  
;  
;  
;  
;  
;  
;  
;  
;  
;  
;
```

```
<div>  
  <h1> Tytuł </h1>  
  </div>  
  <p> Inny tekst </p>  
  <span> Text </span>  
;
```

Selektor “braci”

Selektor braci stosuje regułę CSS do wszystkich selektorów znajdujących się bezpośrednio **ZA** danym zdefiniowanym selektorem. Selektor braci jest oznaczony w regule CSS pisząc pomiędzy nazwami dwóch selektorów znak “~” (tyldy):

```
selektor ~ selektor-brata ... { cecha1: wartość;  
                                  cecha2: wartość;  
                                  ...;  
                                  }  
;
```

Przykład:

```
div ~ span { color:blue; }  
  
<div> </div>  
      <span> czerwony </span>  
      <span> czerwony</span>  
      <span>czerwony</span>  
      <div> <span> czarny </span></div>  
      <span> czerwony </span>  
      <div> <p> <span> czarny </span></p> </div>  
;
```



Selektor atrybutu

Reguły CSS można stosować również w zależności jakie atrybuty przyjmuje dany znacznik w kodzie dokumentu hipertekstowego:

```
selektor[nazwa-atrybutu] { cecha: wartość; }
```

Przykład:

```
<style>
    div[title] {color:red;}
</style>
...
<div title="www"> czerwony </div>
<hr>
<p title="www"> czarny </p>
```

Selektor atrybutu z wartością

Regułę można skonstruować tak aby wybierała elementy, których atrybuty mają konkretną wartość:

```
: selektor[nazwa-atrybutu="wartość"] { cecha: wartość; }
```

Przykład 1:

```
<style>
    div[title="www"] {color:red;}
</style>
...
<div title="www"> czerwony </div>
<hr>
<p title="www"> czarny </p>
```

Selektor atrybutu z wartością

Regułę można skonstruować tak aby wybierała elementy, których atrybuty mają konkretną wartość:

```
: selektor[nazwa-atrybutu="wartość"] { cecha: wartość; }
```

Przykład 2:

```
<style>
  [title="www"] {color:red;}
</style>
...
<div title="www"> czerwony </div>
<hr>
<p title="www"> czarny </p>
```

Reguła ma zastosowanie do wszystkich elementów zawierających atrybut "title" którego wartość wynosi "www".

Selektor atrybutu z wartością

Inne specyficzne selektory dla atrybutów będą pojawiać się jeszcze w trakcie trwania kursu.

Stylizowanie hiperłączy

Nadanie odpowiedniego stylu hiperłącza (linkowi / odsyłaczowi) stanowi odrębny temat. Hiperłącza charakteryzują się tym że w zależności od wykonanych czynności przez użytkownika w przeglądarce przyjmują różne stany:

- **link** – podstawowy odsyłacz (*brak działań użytkownika*),
- **hover** – odsyłacz, nad którym zatrzymano kurSOR myszy,
- **focus** – odsyłacz z tzw. "fokusem", czyli miejscem zaznaczonym przy poruszaniu się po dokumencie przy użyciu klawisza tabulacji. Naciśnięcie klawisza Enter – spowoduje taką samą reakcję przeglądarki, jakby dany odsyłacz został kliknięty,
- **active** – aktywny odsyłacz (*kliknięty*).
- **visited** – odsyłacz, który prowadzi do wcześniej już odwiedzonego dokumentu HTML, tj. takiego, który znajduje się w historii przeglądarki.

Jak ustawić regułę w zależności od stanu hiperłącza ?

Stylizowanie hiperłączy

Nadanie odpowiedniego stylu hiperłącza (linkowi / odsyłaczowi) stanowi odrębny temat. Hiperłącza charakteryzują się tym że w zależności od wykonanych czynności przez użytkownika w przeglądarce przyjmują różne stany:

- **link** – podstawowy odsyłacz (*brak działań użytkownika*),
- **hover** – odsyłacz, nad którym zatrzymano kurSOR myszy,
- **focus** – odsyłacz z tzw. "fokusem", czyli miejscem zaznaczonym przy poruszaniu się po dokumencie przy użyciu klawisza tabulacji. Naciśnięcie klawisza Enter – spowoduje taką samą reakcję przeglądarki, jakby dany odsyłacz został kliknięty,
- **active** – aktywny odsyłacz (*kliknięty*).
- **visited** – odsyłacz, który prowadzi do wcześniej już odwiedzonego dokumentu HTML, tj. takiego, który znajduje się w historii przeglądarki.

Jak ustawić regułę w zależności od stanu hiperłącza ?



Używamy pseudo-klas odpowiadającym konkretnym stanom.

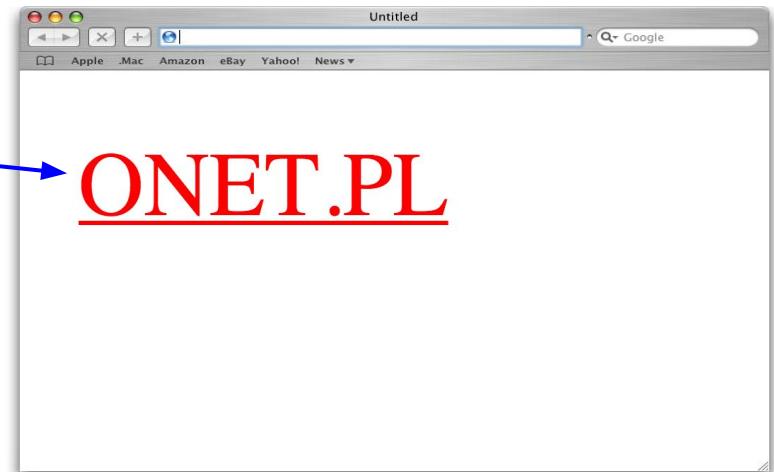
Stylizowanie hiperłączy

Pseudo-klasy w regułach CSS są zaznaczane za pomocą dwukropka pomiędzy nazwą selektora, a nazwą pseudoklasy:

```
selektor:nazwa-pseudoklasy { cecha: wartość; }
```

```
<style>
    a:link {color: red;}
    a:visited {color: #9AB676;}
    a:hover {color: yellow;}
</style>

<a href="http://www.onet.pl">ONET.PL</a>
```



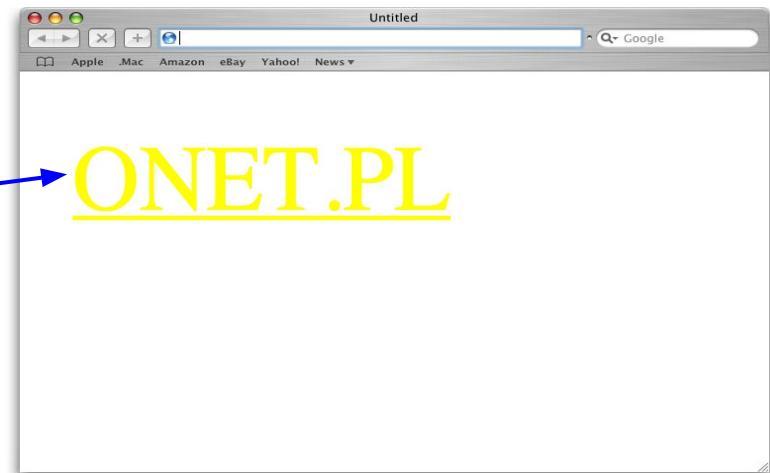
Stylizowanie hiperłączy

Pseudo-klasy w regułach CSS są zaznaczane za pomocą dwukropka pomiędzy nazwą selektora, a nazwą pseudoklasy:

```
selektor:nazwa-pseudoklasy { cecha: wartość; }
```

```
<style>
    a:link {color: red;}
    a:visited {color: #9AB676;}
    a:hover {color: yellow;}
</style>

<a href="http://www.onet.pl">ONET.PL</a>
```



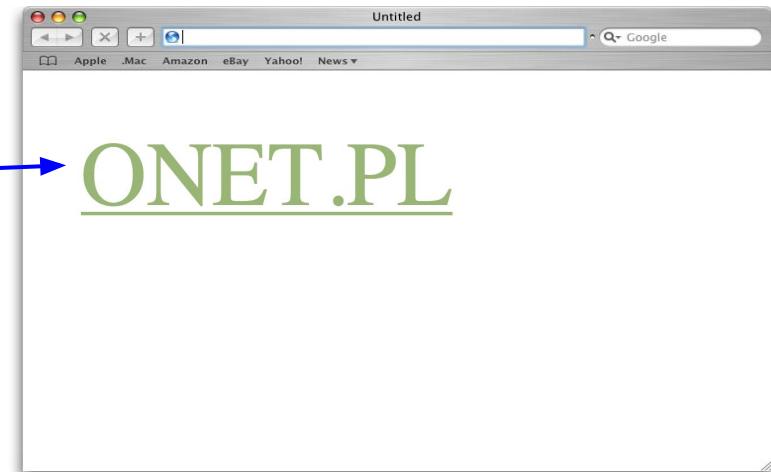
Stylizowanie hiperłączy

Pseudo-klasy w regułach CSS są zaznaczane za pomocą dwukropka pomiędzy nazwą selektora, a nazwą pseudoklasy:

```
selektor:nazwa-pseudoklasy { cecha: wartość; }
```

```
<style>
    a:link {color: red;}
    a:visited {color: #9AB676;}
    a:hover {color: yellow;}
</style>

<a href="http://www.onet.pl">ONET.PL</a>
```



Uwaga: Definicje pseudo-klas można łączyć również z klasami i identyfikatorami: np.:
a:link#uj { color: red;} lub **a:link.uj { color: red;}**

Strony dzisiaj ...

Obecnie internet to nie tylko sieć połączonych ze sobą komputerów, ale także telefonów komórkowych, smartfonów, tabletów, lodówek, pralek etc.



Strony dzisiaj ...

Obecnie internet to nie tylko sieć połączonych ze sobą komputerów, ale także telefonów komórkowych, smartfonów, tabletów, lodówek, pralek etc.



Czy będzie poprawnie wyświetlana ?

Czy będzie reagować na działania użytkownika w taki sam sposób ?

Czy będzie przekazywać wszystkie treści ?

Strony dzisiaj ...

Ale dla jakiej rozdzielczości napisać stronę ? Różne modele smartfonów mają różne rozmiary ekranu!



Strony dzisiaj ...

Problemem nie są tylko smartfony ale również telewizory ...

1920 px



1920 px

Strony dzisiaj ...

Obecnie internet to nie tylko sieć połączonych ze sobą komputerów, ale także telefonów komórkowych, smartfonów, tabletów, lodówek, pralek etc.



Czy będzie poprawnie wyświetlana ?

Czy będzie reagować na działania użytkownika w taki sam sposób ?

Czy będzie przekazywać wszystkie treści ?

3 x TAK
=
RWD



RWD (Responsive Web Design)

RWD Responsive Web Design [Reaktywne Projektowanie Stron] - jest to nowoczesne podejście do projektowania stron internetowych, w którym programista zapewnia, że niezależnie od tego na jakie urządzenie strona zostanie podana wyświetli się ona prawidłowo. W tym kontekście programista zapewnia dostosowanie się wyglądu strony, rozmiaru oraz układu strony do rozmiaru okna przeglądarki w którym będzie wyświetlana.

Jak to osiągnąć ?

RWD (Responsive Web Design)

RWD Responsive Web Design [Reaktywne Projektowanie Stron] - jest to nowoczesne podejście do projektowania stron internetowych, w którym programista zapewnia, że niezależnie od tego na jakie urządzenie strona zostanie podana wyświetli się ona prawidłowo. W tym kontekście programista zapewnia dostosowanie się wyglądu strony, rozmiaru oraz układu strony do rozmiaru okna przeglądarki w którym będzie wyświetlana.

Jak to osiągnąć ?



CSS3 &
media queries

CSS3 i Mediaqueries



Oficjalna rekomendacja W3C:

<http://www.w3.org/TR/css3-mediaqueries>

CSS3 jest standardem wciąż ulegającym ciągłej ewolucji, jednak metodyka tworzenia tej technologii oparta jest o system modułowy, w którym kolejne elementy są udostępniane użytkownikom po ich opracowaniu dalejego nawet kiedy brak jest całego standardu CSS3 są dostępne jego moduły. Jednym z elementów który został zaprojektowany z myślą o urządzeniach mobilnych i wyświetlaniu na nich stron jest moduł “**media-queries**”, dla którego rekomendacja została zatwierdzona przez W3C w lipcu 2012 roku.

Pierwsza propozycja dla media-queries pojawiła się już w CSS 1.0 (1994 rok), na długo przed powstaniem pierwszych nowoczesnych smartfonów.

CSS3 i Mediaqueries



UNIWERSYTET
JAGIELLOŃSKI
W KRAKOWIE



Pack is the beautiful new home for your dog online. We believe there's something magical about dogs and the people who love them. We're building something different just for you. Right now you can only join Pack by invitation only. Maybe you can see if anyone out there has one to give you!

RESERVE YOUR SPOT!

Pack is the beautiful new home for your dog online. We believe there's something magical about dogs and the people who love them. We're building something different just for you. Right now you can only join Pack by invitation only. Maybe you can see if anyone out there has one to give you!

RESERVE YOUR SPOT!

Pack is the beautiful new home for your dog online. We believe there's something magical about dogs and the people who love them. We're building something different just for you. Right now you can only join Pack by invitation only. Maybe you can see if anyone out there has one to give you!

RESERVE YOUR SPOT!

UC San Diego

Prospective Students Current Students Parents & Families Faculty & Staff Alumni Friends & Visitors

Heat and Cold Damage Corals in Their Own Ways, Scripps Study Shows

February 1-February 29 Black History Month

February 8 Annual Equal Opportunity/Affirmative Action Awards

February 14 Greenovation Forum: Coastal Resiliency, Energy Efficiency, The Ecological Economic, and Governance Principles for Healthy Cities

January 18 to March 14 50 Years: Rethinking Science, Winter 2013

UC San Diego

Prospective Students Current Students Parents & Families Faculty & Staff Alumni Friends & Visitors

Heat and Cold Damage Corals in Their Own Ways, Scripps Study Shows

February 1-February 29 Black History Month

February 8 Annual Equal Opportunity/Affirmative Action Awards

February 14 Greenovation Forum: Coastal Resiliency, Energy Efficiency, The Ecological Economic, and Governance Principles for Healthy Cities

January 18 to March 14 50 Years: Rethinking Science, Winter 2013

UC San Diego

Prospective Students Current Students Parents & Families Faculty & Staff Alumni Friends & Visitors

Heat and Cold Damage Corals in Their Own Ways, Scripps Study Shows

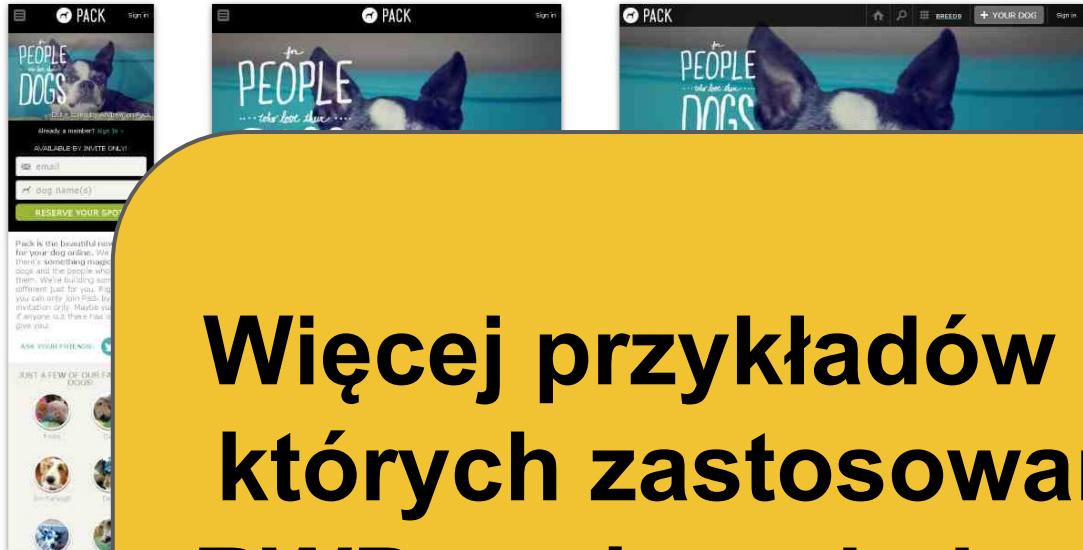
February 1-February 29 Black History Month

February 8 Annual Equal Opportunity/Affirmative Action Awards

February 14 Greenovation Forum: Coastal Resiliency, Energy Efficiency, The Ecological Economic, and Governance Principles for Healthy Cities

January 18 to March 14 50 Years: Rethinking Science, Winter 2013

CSS3 i Mediaqueries



Więcej przykładów projektów na których zastosowano podejście RWD można obejrzeć na stronie:
<http://mediaqueri.es/>

The screenshots show the UC San Diego homepage and various internal pages like 'Google Earth Ocean Terrain Receives Major Update' and 'San Diego Festival of Science & Engineering'. The layout and content are optimized for mobile, tablet, and desktop devices.

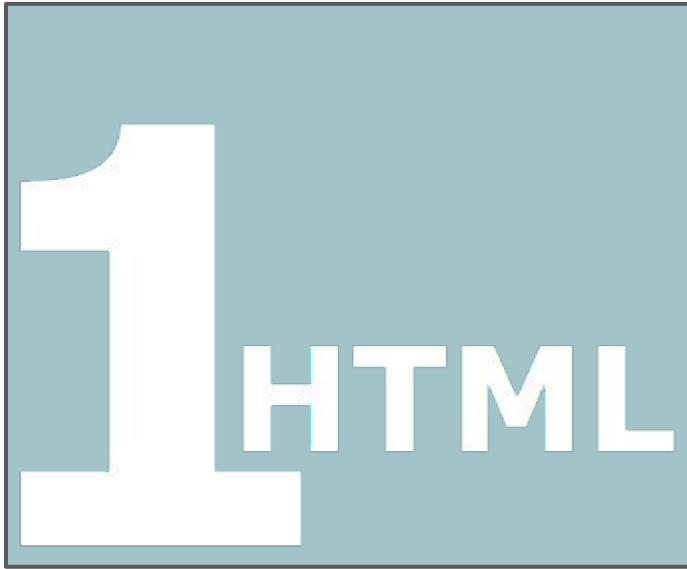


Jak stosować Mediaqueries?

ZASADA:

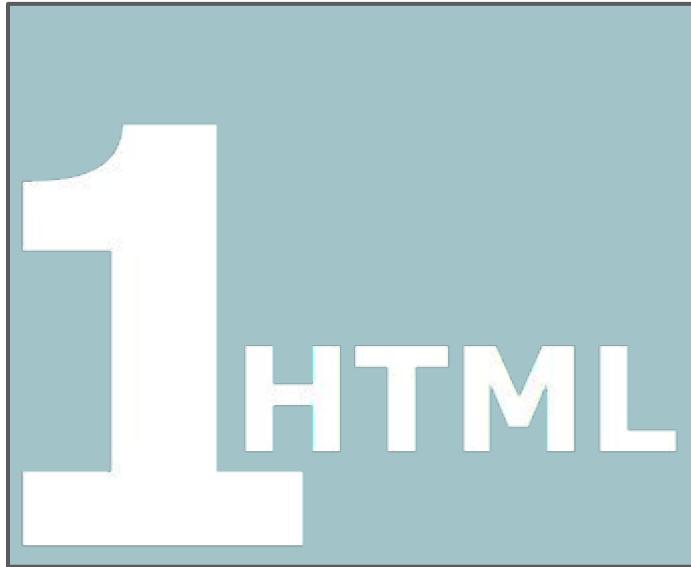


Jak stosować Mediaqueries?



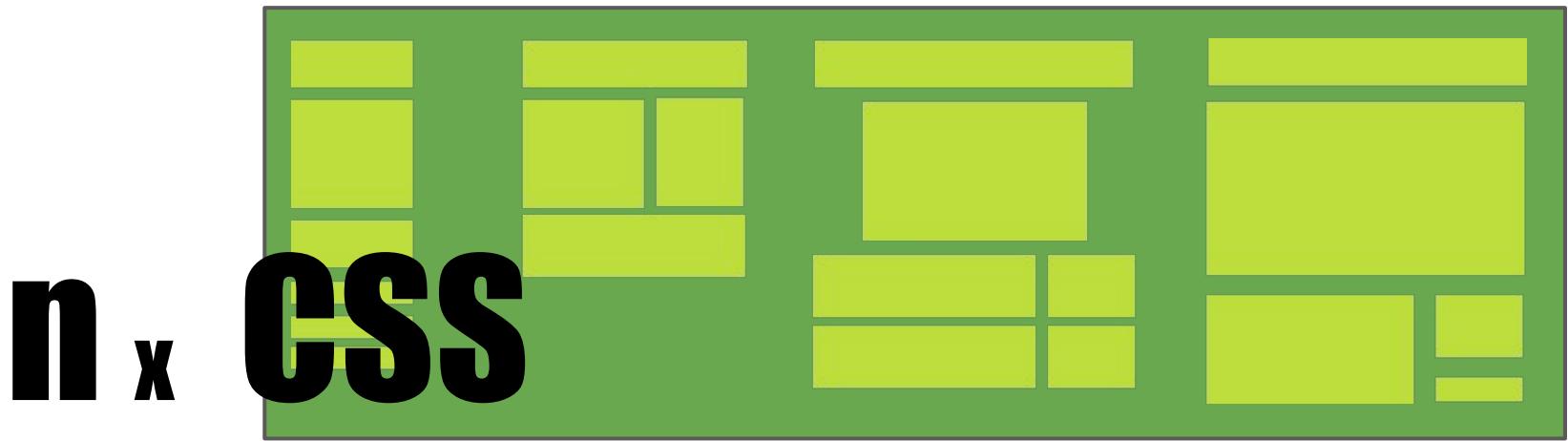
ZASADA:

Jak stosować Mediaqueries?



ZASADA:

W ogólności zasada jaka powinna przyświecać tworzeniu stron zgodnie z metodologią RWD jest tworzenie jednego pliku HTML, a dla formatowania jego wygądu wiele “layoutów” w CSS które będą zawierać odpowiednie reguły wyświetlania strony w zależności od rozdzielczości ekranu.



Czy reguły Mediaqueries zadziałają wszędzie ?

W ogólności zasada Mediaqueries są obecnie wspierane przez większość przeglądarek. W przypadku obecnych smartfonów nie ma z nimi najmniejszego problemu.

mobile

safari iOs, Opera Mini, Opera Mobile,
Android Browser, BlackBerry Browser, Chrome, FF

FF 3.5+

IE 9+

Opera 9.5+

Chrome 4+

Safari 4+

Czy reguły Mediaqueries zadziałają wszędzie ?

W ogólności zasada Mediaqueries są obecnie wspierane przez większość przeglądarek. W przypadku obecnych smartfonów nie ma z nimi najmniejszego problemu.

W przypadku “starszych” przeglądarek stacjonarnych jest trochę gorzej.

W przypadku przeglądarek IE<9, FF<3.5 czy Opery <9.5 - niestety reguły te nie są interpretowane. Dlatego trzeba skorzystać z alternatywnych rozwiązań:

Czy reguły Mediaqueries zadziałają wszędzie ?

W ogólności zasada Mediaqueries są obecnie wspierane przez większość przeglądarek. W przypadku obecnych smartfonów nie ma z nimi najmniejszego problemu.

W przypadku “starszych” przeglądarek stacjonarnych jest trochę gorzej.

W przypadku przeglądarek IE<9, FF<3.5 czy Opery <9.5 - niestety reguły te nie są interpretowane. Dlatego trzeba skorzystać z alternatywnych rozwiązań:

```
<!--[if lt IE 9]>
<script src="html5shiv.js"></script>
<![endif]-->
```

```
<!--[if lt IE 9]>
<script src="css3-mediaqueries.js"></script>
<![endif]-->
```

Biblioteki JS takie jak: *html5shiv.js* czy *css3-mediaqueries.js*, które pozwalają za pomocą JavaScriptu podmieniać wartości CSS poszczególnych elementów strony.

CSS3 i Mediaqueries

Aby zdefiniować reguły CSS które będą adaptować stronę w zależności od tego na jakim urządzeniu jest ona w danym momencie wyświetlana musimy skorzystać z “selekторa”:

@media

CSS3 i Mediaqueries

Aby zdefiniować reguły CSS które będą adaptować stronę w zależności od tego na jakim urządzeniu jest ona w danym momencie wyświetlana musimy skorzystać z “selekторa”:

@media

Selektor ten odnosi się bezpośrednio do określenia typu urządzenia dla jakiego ma zostać zastosowana reguła CSS. Możliwe typy to np.:

all	Dla wszystkich urządzeń.
aural	Dla syntezy mowy i dźwięku.
braille	Dla urządzeń którymi posługują się niewidomi w celu przeglądania stron internetowych.
embossed	Dla drukarek breilla.
handheld	Dla urządzeń ręcznych bezprzewodowych .
print	Dla drukarek / materiałów drukowanych.
projection	Dla projektorów sciennych.
screen	Dla komputerów z ekranem.
tty	Dla terminali z ograniczonymi możliwościami wyświetlania.
tv	Dla telewizorów.

CSS3 i Mediaqueries

Określenie reguły odbywa się w następujący sposób:

```
-----  
| @media media-type {  
|   selector {  
|     cecha: wartość;  
|   }  
| }  
|  
|-----
```

gdzie “*media-type*” oznacza określenie typu urządzenia z poprzedniej tabeli.

```
-----  
|<style>  
|   @media screen {  
|     div {  
|       background-color:red;  
|     }  
|   }  
|</style>  
|-----
```



CSS3 i Mediaqueries

Przykład dla dwóch urządzeń:

```
-----  
■ <style>  
■   div {  
■     background-color:blue;  
■   }  
■   @media tv, handheld {  
■     div {  
■       background-color:red;  
■     }  
■   }  
■ </style>  
-----
```

W tym wypadku kiedy strona zostanie wyświetlona na telewizorze lub smartfonie kolor tła elementów div powinien przybrać kolor czerwony, we wszystkich pozostałych przypadkach kolor niebieski.

CSS3 i Mediaqueries

W zależności od tego na jakim urządzeniu wyświetlamy stronę wczytujemy dedykowany plik z regułami CSS:

```
<head>
    <link media="tv,handheld" href="media.css">
</head>
```

lub możemy użyć dyrektywy @import:

```
<style>
    @import url('styl-media3.css') screen and (color);
</style>
```

CSS3 i Mediaqueries

Dla lepszego sterowania regułami media możemy posługiwać się atrybutami:

Oznaczenie	Przeznaczenie
<u>width</u>	określenie wartości szerokości okna przeglądarki internetowej
<u>height</u>	określenie wartości wysokości okna przeglądarki internetowej
<u>device-width</u>	określenie wartości rozdzielczości ekranu urządzenia (szerokość)
<u>device-height</u>	określenie wartości rozdzielczości ekranu urządzenia (wysokość)
<u>color</u>	określenie liczby bitów na kolor lub określenie czy urządzenie posiada kolorowy ekran
<u>color-index</u>	określenie wartości głębi kolorów, które obsługuje dane urządzenie
<u>aspect-ratio</u>	określenie wartości proporcji szerokości do wysokości okna przeglądarki internetowej

CSS3 i Mediaqueries

Dla lepszego sterowania regułami media możemy posługiwać się atrybutami:

[device-aspect-ratio](#)

określenie wartości proporcji szerokości do wysokości rozdzielczości ekranu urządzenia

[grid](#)

określenie urządzenia z ograniczonymi możliwościami wyświetlania

[monochrome](#)

określenie liczby bitów na piksel w urządzeniach monochromatycznych, jednokolorowych

[orientation](#)

określenie orientacji pionowej lub poziomej urządzenia

[resolution](#)

określenie wartości gęstości pikseli dla danego urządzenia

[scan](#)

określenie czy urządzenie posiada skanowanie obrazu progresywne czy międzyliniowe

CSS3 i Mediaqueries

Oraz operatory logiczne:

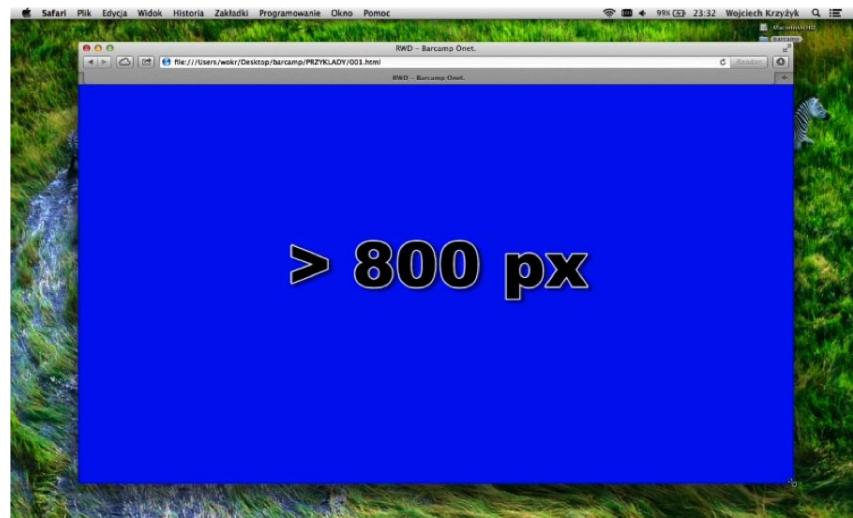
Oznaczenie	Przeznaczenie
<u>and</u>	operator " i ", służy do tworzenia bardziej precyzyjnych warunków w regule @media
<u>przecinek</u>	operator " lub ", służy do tworzenia bardziej precyzyjnych warunków w regule @media
<u>not</u>	operator " negacji ", służy do tworzenia bardziej precyzyjnych warunków w regule @media
<u>only</u>	operator przeznaczony dla starszych przeglądarek internetowych

CSS3 i Mediaqueries

Dodatkowo do poza określaniem konkretnej wartości atrybutów można zadać zakresy dolny i górny:

Reguły ograniczające
z góry i z dołu:

{ min-
max- }



```
@media screen and ( min-width:  
                      max-width
```



CSS3 i Mediaqueries

Przykład zastosowania:

```
<style>
  @media all and (min-width:480px) and (max-width:800px) {
    div {
      background-color:green;
    }
  }
</style>
```

Definicja określona powyżej mówi przeglądarce że tło elementu <div> ma zostać ustawione na kolor zielony na wszystkich typach urządzeń tylko wtedy kiedy szerokość okna przeglądarki będzie zawierać się w przedziale od 480px od 800px.



CSS3 i Mediaqueries

Przykład zastosowania:

```
<style>
  @media all and (min-color-index:256) {
    div {
      background-color:green;
    }
  }
</style>
```

W przypadku tej reguły kolor tła elementu <div> zostanie zmieniony na zielony tylko wtedy kiedy wartość głębi kolorów możliwych do wyświetlenia na danym urządzeniu nie jest mniejsza od 256.

CSS3 i Mediaqueries

Zastosowanie operatora “only” nie ma on wpływu na definicję reguły, ale zapewnia kompatybilność ze starszymi przeglądarkami, które nie obsługują MediaQueries. Operator był dostępny w wersji 2.1 CSS - ograniczał reguły css dla danego typu "media". Ponieważ starsze przeglądarki nie rozpoznają reguł - nie zinterpretują prawidłowo cssów z media.

```
<style>
@media only screen (min-width:480px) and (max-width:800px) {
  div {
    background-color:red;
  }
}
</style>
```



UNIWERSYTET
JAGIELŁOŃSKI
W KRAKOWIE

KONIEC WYKŁADU 4



UNIWERSYTET
JAGIELŁOŃSKI
W KRAKOWIE

Zaawansowane Techniki WWW (HTML, CSS i JavaScript)

Dr inż. Marcin Zieliński

Środa 15:30 - 17:00 sala: A-1-04

WYKŁAD 5

Wykład dla kierunku: Informatyka Stosowana II rok

Rok akademicki: 2015/2016 - semestr zimowy

Przypomnienie z poprzedniego wykładu

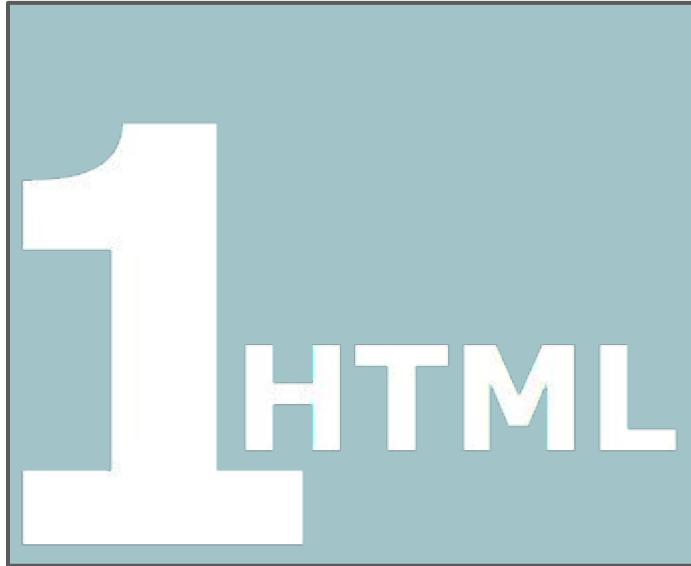
Strony na urządzeniach mobilnych.

Responsive Web Design (RWD)

Metodologia Mobile-First

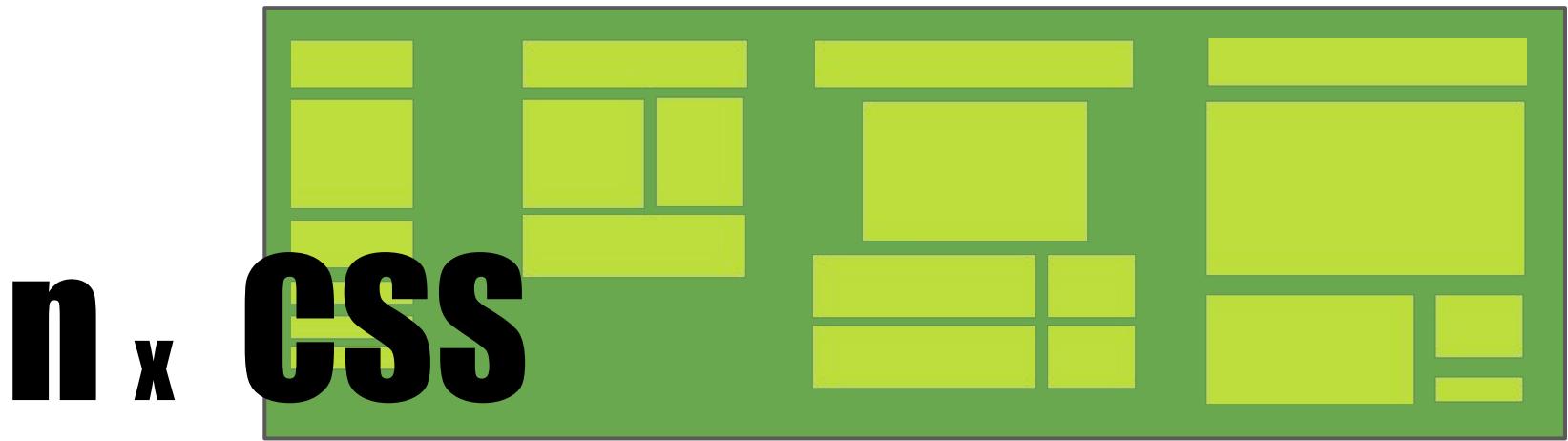
CSS3 i Mediaqueries

Jak stosować Mediaqueries?



ZASADA:

W ogólności zasada jaka powinna przyświecać tworzeniu stron zgodnie z metodologią RWD jest tworzenie jednego pliku HTML, a dla formatowania jego wygądu wiele “layoutów” w CSS które będą zawierać odpowiednie reguły wyświetlania strony w zależności od rozdzielczości ekranu.

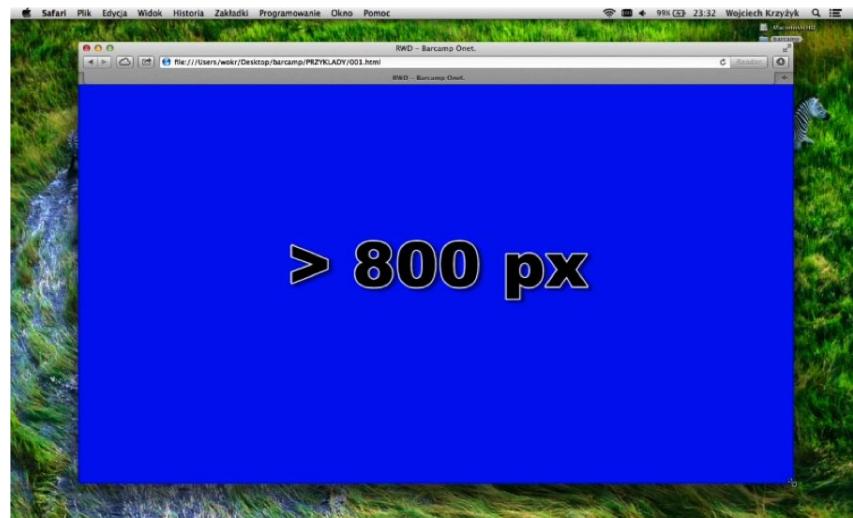


CSS3 i Mediaqueries

Dodatkowo do poza określaniem konkretnej wartości atrybutów można zadać zakresy dolny i górny:

Reguły ograniczające
z góry i z dołu:

{ min-
max- }



```
@media screen and ( min-width:  
                      max-width
```



CSS3 i Mediaqueries

Przykład zastosowania:

```
<style>
  @media all and (min-width:480px) and (max-width:800px) {
    div {
      background-color:green;
    }
  }
</style>
```

Definicja określona powyżej mówi przeglądarce że tło elementu <div> ma zostać ustawione na kolor zielony na wszystkich typach urządzeń tylko wtedy kiedy szerokość okna przeglądarki będzie zawierać się w przedziale od 480px od 800px.



CSS3 i Mediaqueries

Przykład zastosowania:

```
<style>
  @media all and (min-color-index:256) {
    div {
      background-color:green;
    }
  }
</style>
```

W przypadku tej reguły kolor tła elementu <div> zostanie zmieniony na zielony tylko wtedy kiedy wartość głębi kolorów możliwych do wyświetlenia na danym urządzeniu nie jest mniejsza od 256.

CSS3 i Mediaqueries

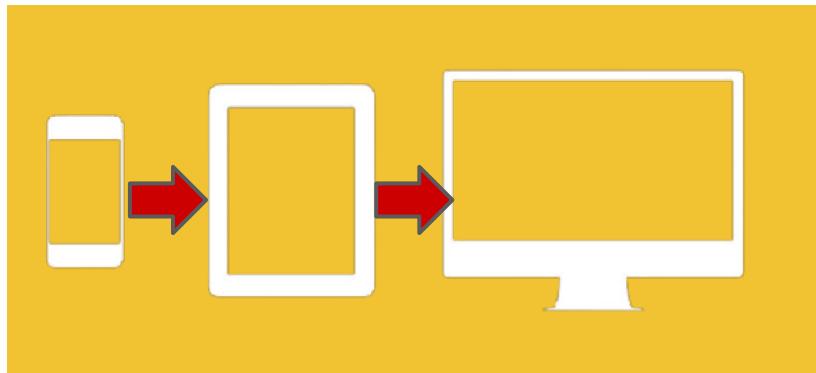
Zastosowanie operatora “only” nie ma on wpływu na definicję reguły, ale zapewnia kompatybilność ze starszymi przeglądarkami, które nie obsługują MediaQueries. Operator był dostępny w wersji 2.1 CSS - ograniczał reguły css dla danego typu "media". Ponieważ starsze przeglądarki nie rozpoznają reguł - nie zinterpretują prawidłowo cssów z media.

```
<style>
@media only screen (min-width:480px) and (max-width:800px) {
  div {
    background-color:red;
  }
}
</style>
```



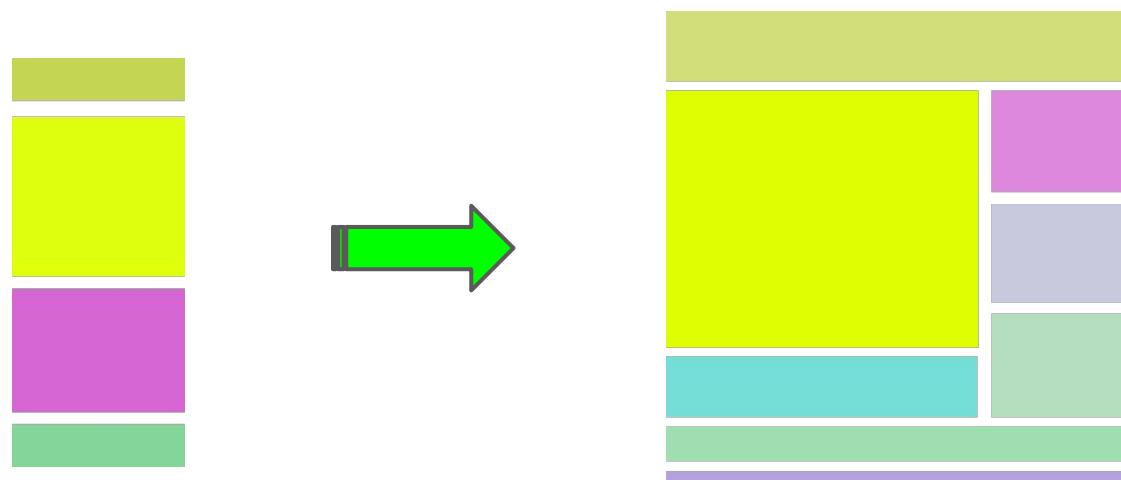
Które CSSy tworzyć najpierw ?

Które CSSy tworzyć najpierw ?



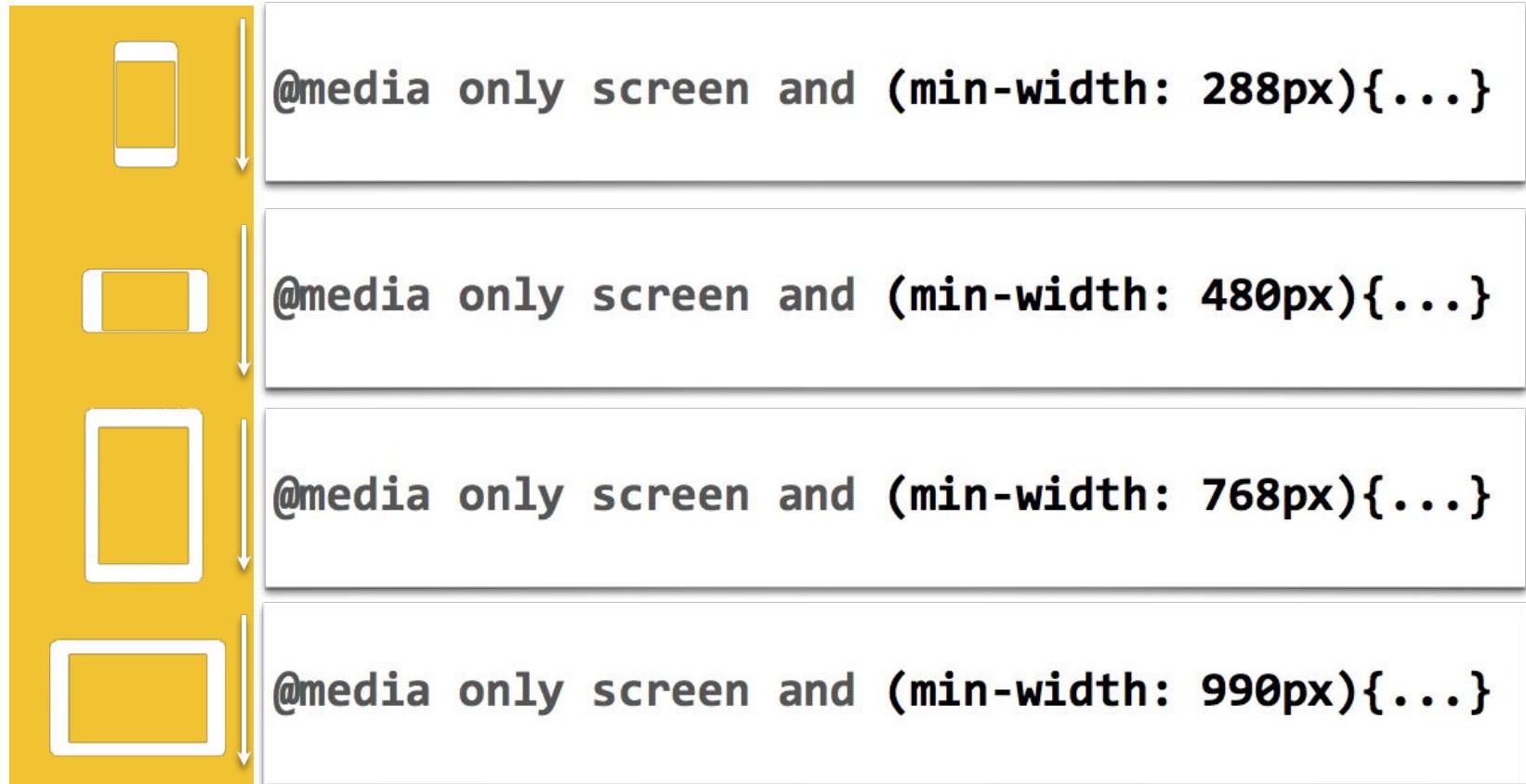
**Filozofia
Mobile-First**

Zgodnie z filozofią MobileFirst tworzenie strony powinno być rozpoczęte od najmniejszych ekranów czyli dla urządzeń mobilnych.

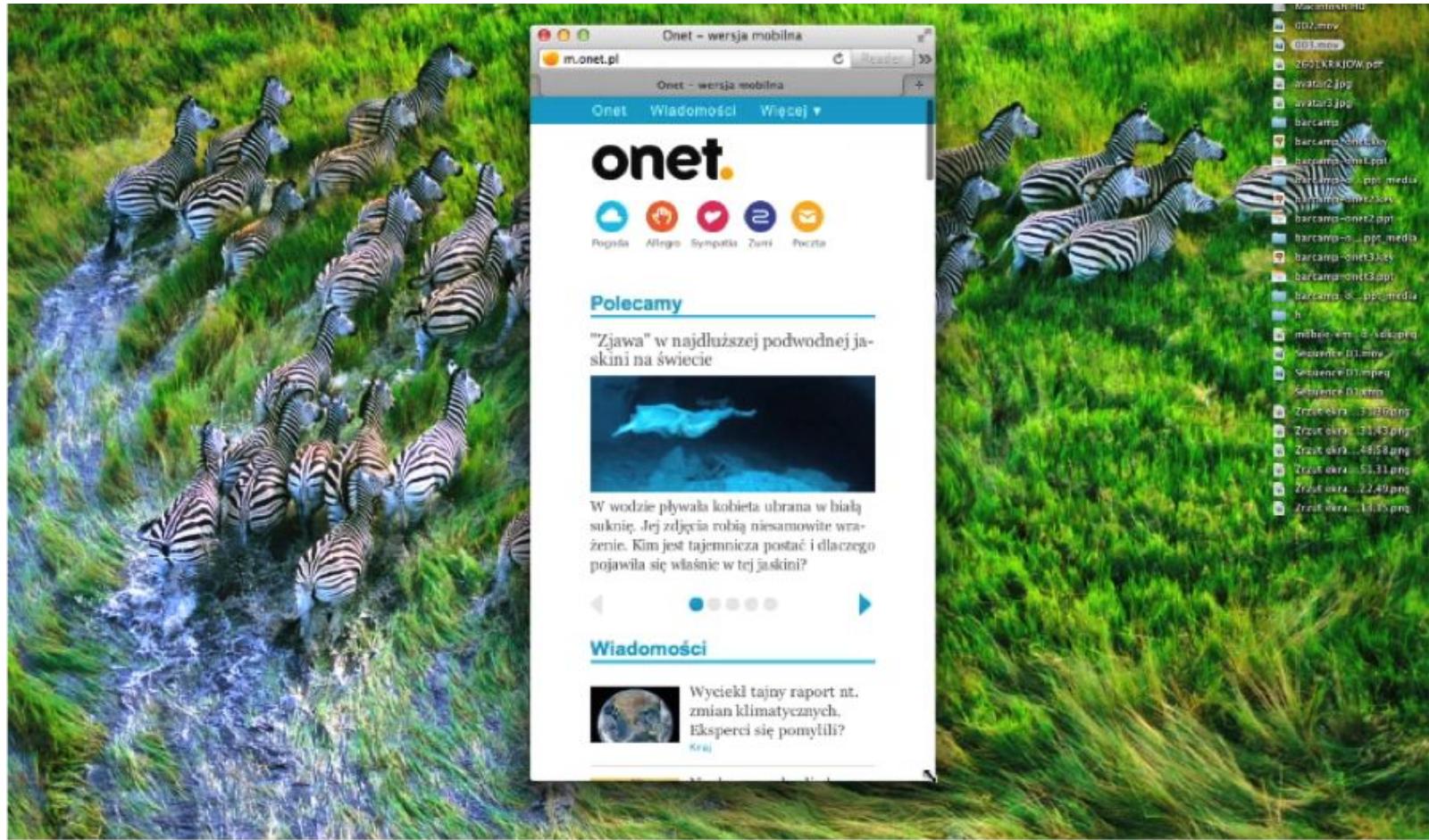


Mobile First

Zgodnie z tymi zasadami CSSy powinniśmy układać mniej więcej wg takiego schematu:



Mobile First



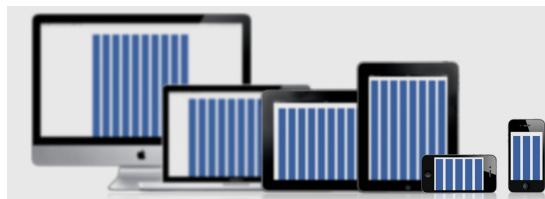


Narzędzia ułatwiające pracę

Pracę z Mediaqueries oraz z projektowaniem widoków ułatwiają frameworki m.in.
Lessframework, Cssgrid, Columnal czy Foundation.

Narzędzia ułatwiające pracę

Pracę z Mediaqueries oraz z projektowaniem widoków ułatwiają frameworki m.in.
Lessframework, Cssgrid, Columnal czy Foundation.



Less Framework 4
An adaptive CSS grid system.

<http://lessframework.com/>



<http://www.1140px.com/>



<http://www.columnal.com/>



<http://foundation.zurb.com/>

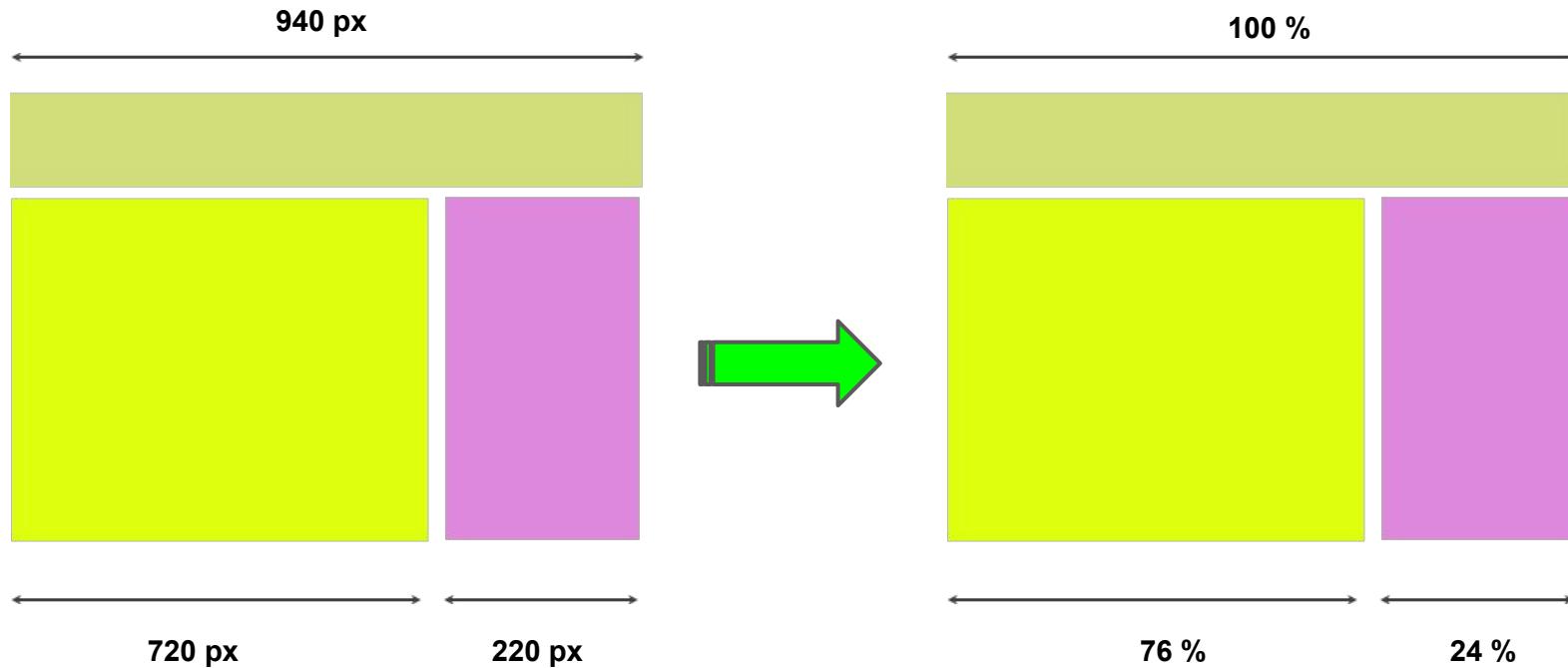
Frameworki te dostarczają zdefiniowanych reguł CSS/Mediaqueries dla zdefiniowanych szerokości ekranów. Oczywiście domyślne wartości możemy dowolnie modyfikować. Ułatwiają nam budowanie layoutów responsywnych stron, a niektóre jak np Foundation oferują już gotowe prototypy.

Nowe podejście do definiowania rozmiarów elementów

Chcąc zaprojektować "elastyczną" stronę dopasowującą się do całego ekranu powinniśmy zrezygnować z jednostek bezwzględnych wyrażanych np. w px i zastąpić je jednostkami względnymi w procentach %.

Pozwoli to uzyskać dopasowany układ do różnych szerokości ekranów.

Atrybuty max-width powinny być używane często zwłaszcza w przypadku elementów, które mogą nam "rozpychać layout" tj. obrazków, osadzonych odtwarzaczy video etc.



Funkcja calc()

CSS został wyposażony w specjalną funkcję o nazwie “calc()”, która pozwala wykonywać nieskomplikowane obliczenia matematyczne do określenia wartości danej cechy jeśli jest ona liczbą. Do dyspozycji mamy następujące działania: +, - , * (mnożenie), / (dzielenie). Z wykorzystaniem tej funkcji możemy określić za pomocą równiania matematycznego jak ma się dopasować do ekranu dana cecha.

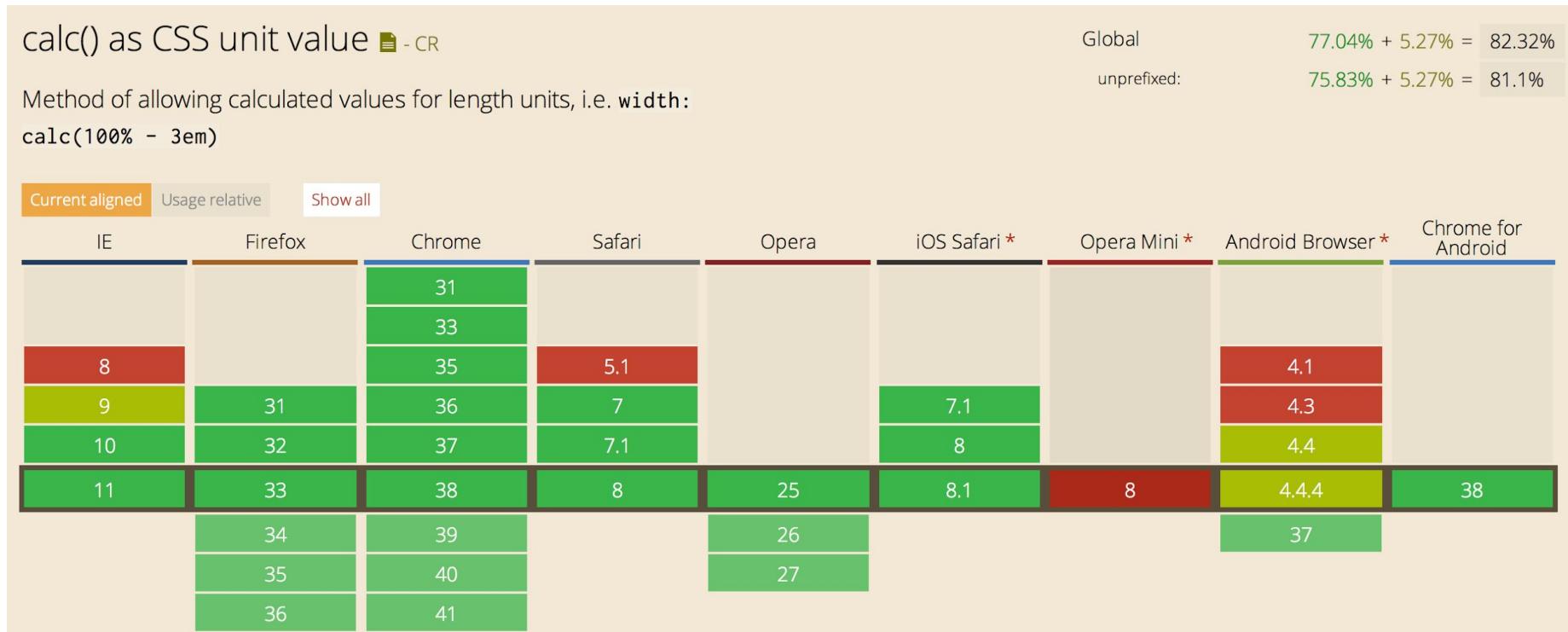
Funkcja calc()

CSS został wyposażony w specjalną funkcję o nazwie “calc()”, która pozwala wykonywać nieskomplikowane obliczenia matematyczne do określenia wartości danej cechy jeśli jest ona liczbą. Do dyspozycji mamy następujące działania: +, - , * (mnożenie), / (dzielenie). Z wykorzystaniem tej funkcji możemy określić za pomocą równiania matematycznego jak ma się dopasować do ekranu dana cecha.

```
<style>
  @media only screen (max-width:800px) {
    .calc {
      /* Firefox */
      width: -moz-calc(75% - 100px);
      /* WebKit: Safari, Chrome */
      width: -webkit-calc(75% - 100px);
      /* Opera */
      width: -o-calc(75% - 100px);
      /* Standard */
      width: calc(75% - 100px);
    }
  }
</style>
```

Funkcja calc()

Zgodność z przeglądarkami:



Jak testować Mediaqueries ?

Jak sprawdzić czy przygotowana strona działa wszędzie poprawnie??

Na rynku mamy niezliczoną liczbę urządzeń, mnogość systemów operacyjnych, na każdym jest po kilka różnych przeglądarek. Nie mamy dostępu do wszystkich tych urządzeń, a problemów jakich możemy się spodziewać jest cały szereg: brak obsługi Javascriptu, brak obsługi Mediaqueries, problemy z layoutem etc...



 Windows Phone

 **htc**
quietly brilliant



ZTE 中兴



XPERIA
Sony Smartphone

NOKIA



Samsung
GALAXY S III

Jak testować Mediaqueries ?

Jak sprawdzić czy przygotowana strona działa wszędzie poprawnie??

Na rynku mamy niezliczoną liczbę urządzeń, mnogość systemów operacyjnych, na każdym jest po kilka różnych przeglądarek. Nie mamy dostępu do wszystkich tych urządzeń, a problemów jakich możemy się spodziewać jest cały szereg: brak obsługi Javascriptu, brak obsługi Mediaqueries, problemy z layoutem etc...



Jak poradzić sobie testowaniem ?





Jak testować Mediaqueries ?

Metoda nr 1

Skalowanie okna przeglądarki

Jak testować Mediaqueries ?

Metoda nr 1

Skalowanie okna przeglądarki

Metoda nr 2

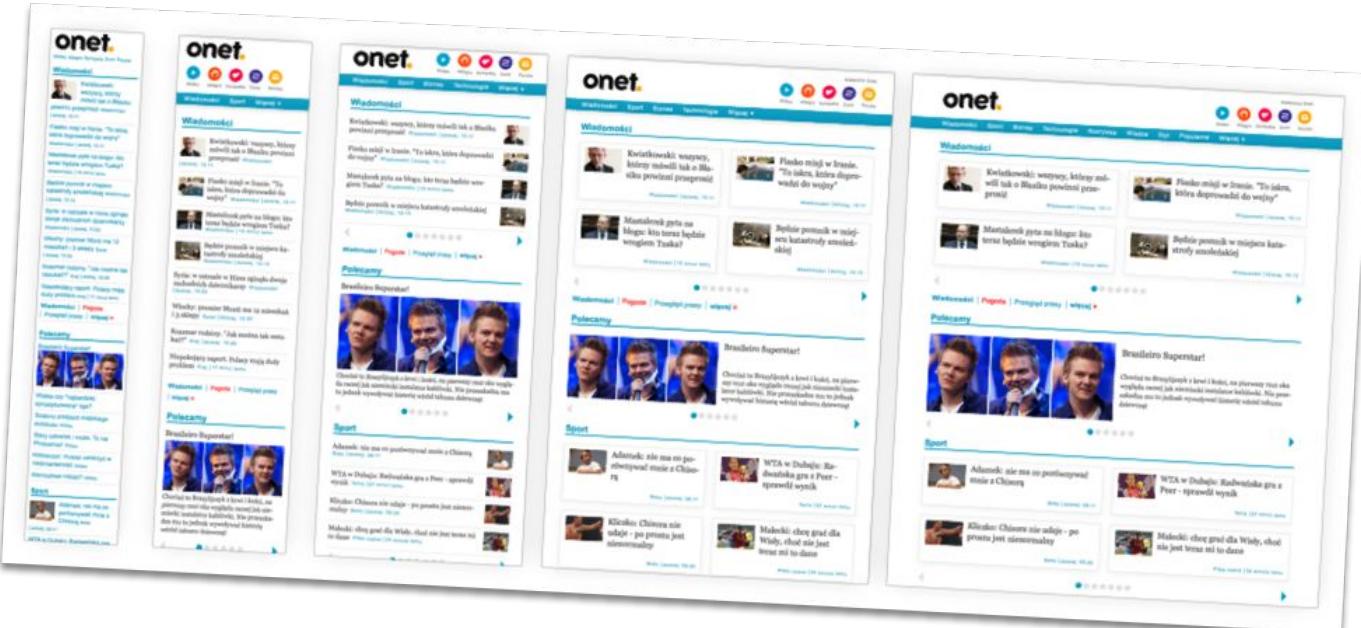
Narzędzia deweloperski w
przeglądarkach

Jak testować Mediaqueries ?

Metoda nr 1

Skalowanie okna przeglądarki

Zawsze najlepiej zacząć od najprostszej metody - skalowania okna przeglądarki. Wydaje się banalne, ale jest to najszybsza metoda wyłapania wszystkich niedoskonałości.

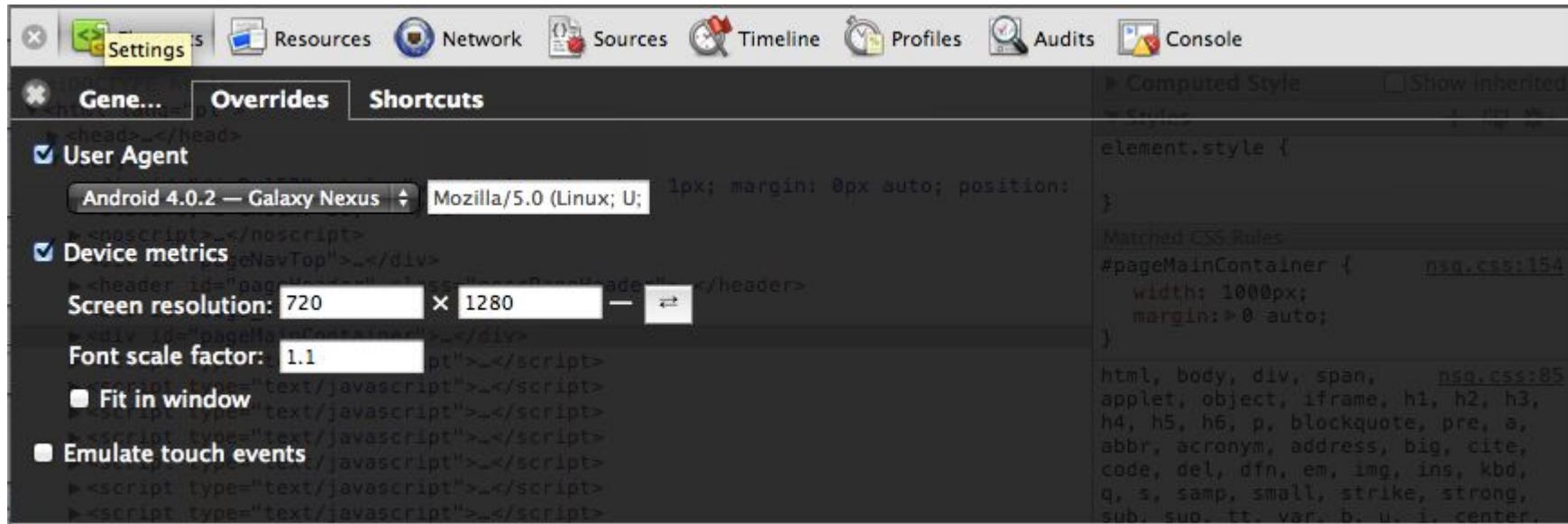


Jak testować Mediaqueries ?

Metoda nr 2

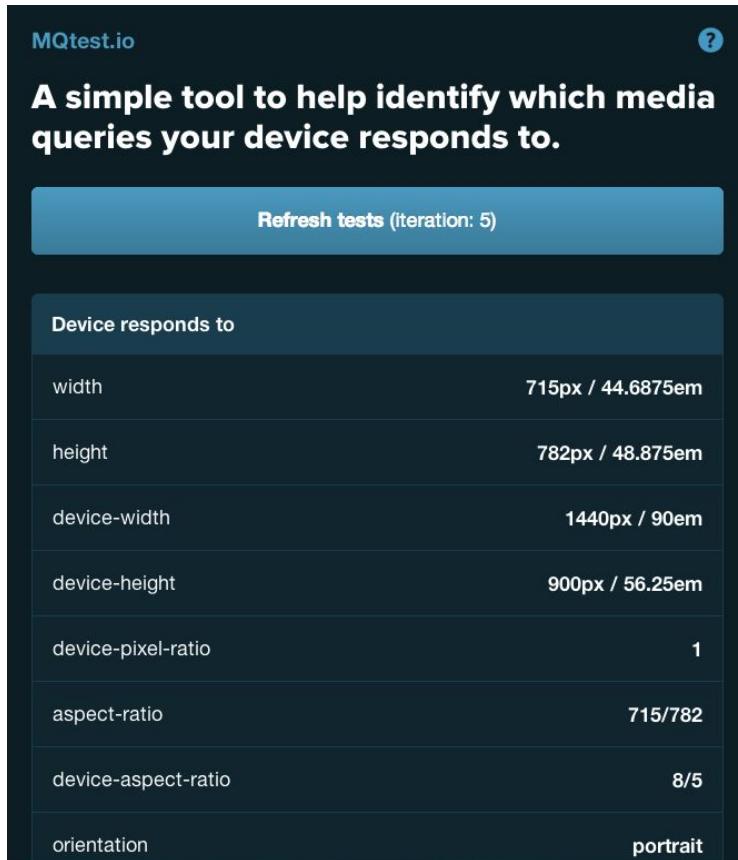
Narzędzia deweloperski w przeglądarkach

Wiele nowoczesnych przeglądarek posiada specjalne rozszerzenia pozwalające używać tzw. Trybu Debugowania, pozwalającego też na zasymulować różnych wymiarów ekranu.



Jak testować Mediaqueries ?

Dodatkowo można również za pomocą strony www.mqtest.io poznać wszelkie informacje o regułach jakie są obsługiwane na urządzeniu z którego ją odwiedziliśmy, a które możemy wykorzystać w mediaqueries.

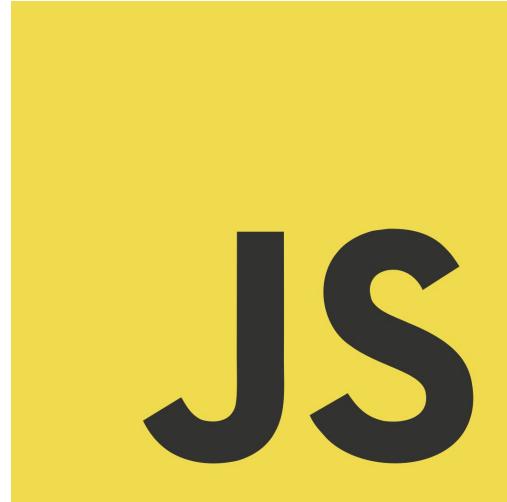


The screenshot shows the MQtest.io interface. At the top, it says "A simple tool to help identify which media queries your device responds to." Below that is a blue button labeled "Refresh tests (iteration: 5)". The main area is titled "Device responds to" and lists the following media queries with their values:

Media Query	Value
width	715px / 44.6875em
height	782px / 48.875em
device-width	1440px / 90em
device-height	900px / 56.25em
device-pixel-ratio	1
aspect-ratio	715/782
device-aspect-ratio	8/5
orientation	portrait



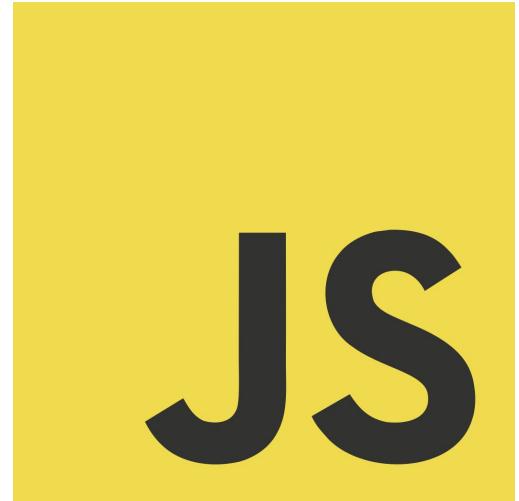
Wstęp do JavaScript



Wstęp do JavaScript

W latach 1994 - 1995 podstawy języka JavaScript zostały stworzone przez firmę Netscape, a następnie w kolejnym etapie był rozwijany we współpracy z firmą SUN. Twórcą standardu jest Brendan Eich. Język ten pozwala na rozszerzenie funkcji oferowanych przez HTML oraz CSS w celu zwiększenia funkcjonalności stron internetowych.

Java script posiada wszystkie podstawowe elementy poprawnego języka programowania: zmienne, instrukcje warunkowe, pętle, instrukcje wejścia/wyjścia, tablice, funkcje, a zwłaszcza obiekty. Język ten jest oparty na obiektach (ang. *object-based*) i jest sterowany zdarzeniami (ang. *event-driven*).





Wstęp do JavaScript

Cechy języka JavaScript:

- Zapewnia obsługę DOM (Document Object Model),
- Brak statycznej kontroli typów zmiennych (trudności z wykryciem błędów).
- Współpraca z formatem JSON,
- Wbudowane dziedziczenie prototypowe,
- Brak klas typowych z innych języków programowania.

Opis standardu języka:

<http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>

Obecnie najnowszą rekomendowaną wersją JavaScript jest wersja 1.8.5.

Jak stosować kod w języku JS ?

Załączenie do strony internetowej kodu napisanego w języku JavaScript (w skrócie skryptu) można dokonać na dwa sposoby:

- osadzić skrypt w kodzie HTML strony (**embedded-script**),
- umieścić w osobnym pliku (**linked-script**).



HTML5

Embedded-Script

```
<body>
<script>
  alert("Hello World!");
</script>
</body>
```

Linked-Script

```
<head>
<script src="skrypt.js">
</script>
</head>
```



Plik
.html

```
  alert("Hello World!");
```



Plik
.js

Jak stosować kod w języku JS ?

Załączenie do strony internetowej kodu napisanego w języku JavaScript (w skrócie skryptu) można dokonać na dwa sposoby:

- osadzić skrypt w kodzie HTML strony (**embedded-script**),
- umieścić w osobnym pliku (**linked-script**).

HTML4 /
XHTML

Embedded-Script

```
<body>
<script type="text/javascript">
  alert("Hello World!");
</script>
</body>
```

Linked-Script

```
<head>
<script
  type="text/javascript"
  src="skrypt.js">
</script>
</head>
```

alert("Hello World!");

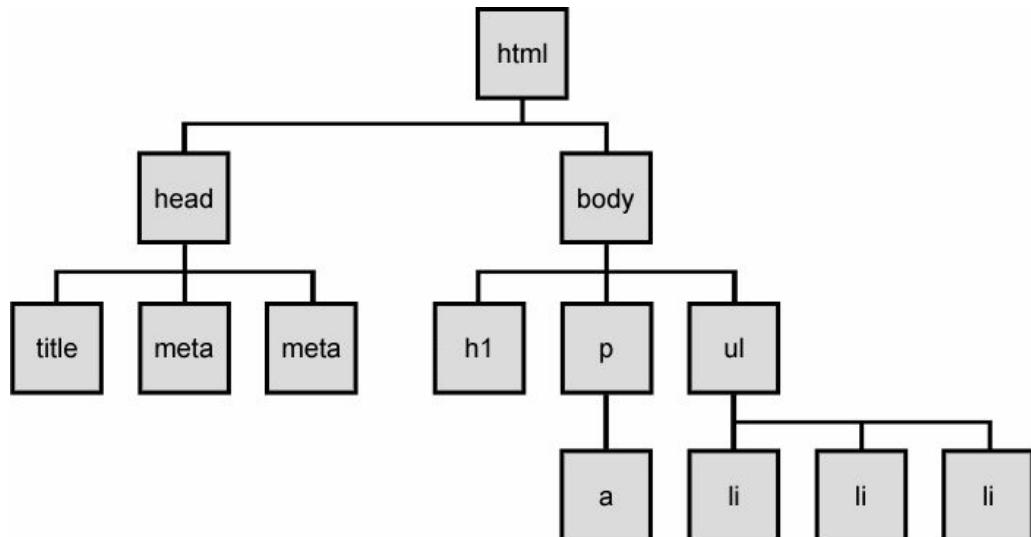
Plik
.html

Plik
.js

DOM (Document Object Model)

DOM (Document Object Model) [Obiektowy Model Dokumentu] - jest to model reprezentujący budowę dokumentów XML, HTML w postaci hierarchicznej struktury obiektów i elementów. Model DOM jest niezależny od platformy i środowiska programistycznego. Pozwala na dostęp do elementów dokumentu (jego struktury) oraz wykonywanie czynności jak dodawanie, usuwanie i modyfikacja zawartości poszczególnych elementów.

Przykład modelu DOM dla prostego dokumentu hipertekstowego



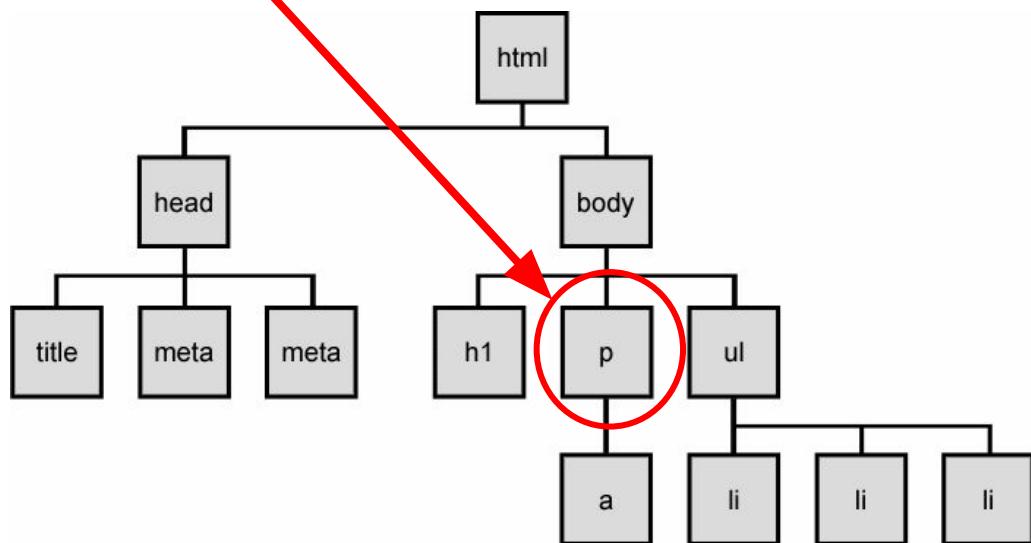
DOM (Document Object Model)

Dzięki zdefiniowanym w modelu metodom i klasą możliwy jest dostęp do dowolnego elementu dokumentu tzw. węzła (node), np.:

```
document.getElementsByTagName("p");
```

Przykład modelu DOM dla prostego dokumentu hipertekstowego

<http://www.w3.org/DOM/>



Jak stosować kod w języku JS ?

Dobre praktyki

W poprzednich przykładach, które pokazywały sposoby osadzenia kodu używaliśmy funkcji wbudowanej `alert()`, która umożliwia wyświetlanie komunikatów na stronie w formie wyskakującego okienka. Jednak taki zapis powoduje znów mieszanie kodu HTML i JS, dlatego zalecaną metodą jest stosowanie funkcji definiowanych przez użytkownika:

```
<head>
  <script>
    function Komunikat() {
      alert("Hello World!");
    }
  </script>
</head>
<body>
  <script>
    Komunikat();
  </script>
</body>
```

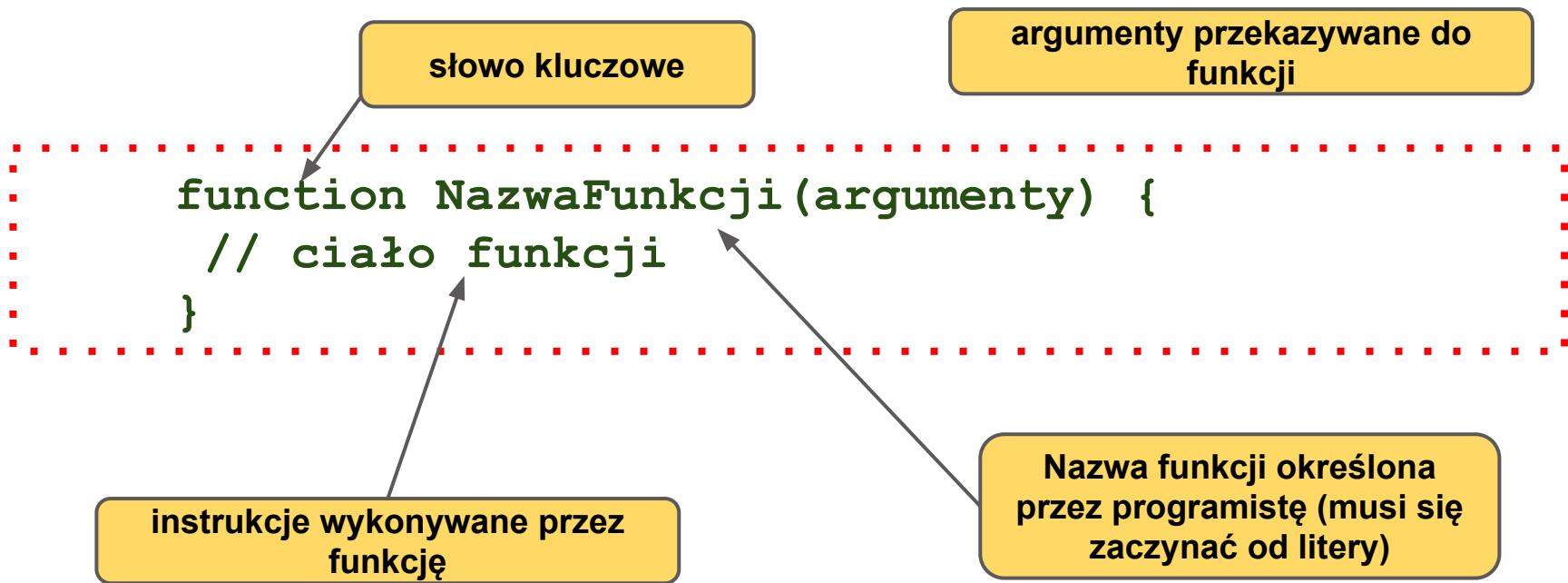
Definicja funkcji w sekcji HEAD (lub osobnym pliku js)

Wywołanie funkcji

Zacznijmy od funkcji ...

Zacznijmy od zdefiniowania (narazie) jak tworzyć funkcje, które są podstawowymi narzędziami i jednostkami modularnymi wykorzystywanymi przez programistę JavaScriptu:

Funkcja nazwana:



Zacznijmy od funkcji ...

Zacznijmy od zdefiniowania (narazie) jak tworzyć funkcje, które są podstawowymi narzędziami i jednostkami modularnymi wykorzystywanymi przez programistę JavaScriptu:

Funkcja nazwana:

słowo kluczowe

argumenty przekazywane do funkcji

```
function NazwaFunkcji(argumenty) {
```

W języku JavaScript funkcje są obiektami!!

instrukcje wykonywane przez funkcję

przez programistę (musi się zaczynać od litery)

Kiedy przeglądarka nie obsługuje JavaScriptu

W przypadku kiedy przeglądarka nie obsługuje języka JavaScript lub jest on celowo wyłączony przez użytkownika, operacje zawarte w skrypcie nie zostaną wykonane. Dlatego istotnym jest poinformowanie użytkownika że jego przeglądarka nie wspiera obsługi JavaScript. Można to uczynić stosując specjalny znacznik <noscript>, za pomocą którego możemy użytkownikowi wystawić jasny komunikat o zaistniałej sytuacji.

Kiedy przeglądarka nie obsługuje JavaScriptu

W przypadku kiedy przeglądarka nie obsługuje języka JavaScript lub jest on celowo wyłączony przez użytkownika, operacje zawarte w skrypcie nie zostaną wykonane. Dlatego istotnym jest poinformowanie użytkownika że jego przeglądarka nie wspiera obsługi JavaScript. Można to uczynić stosując specjalny znacznik <noscript>, za pomocą którego możemy użytkownikowi wystawić jasny komunikat o zaistniałej sytuacji.

```
-----  
  <script type="text/javascript">  
  document.write("Witaj świecie!")  
  </script>  
  <noscript>  
  Twoja przeglądarka nie obsługuje języka Javascript.  
  </noscript>  
-----
```

Znacznik ten definiuje treść alternatywną, która będzie wyświetlona, gdy przeglądarka nie obsługuje języka JavaScript.

Zmienne

JavaScript jest językiem który nie obsługuje statycznych typów zmiennych, a więc nie możemy zadeklarować że dana zmienna jest np. typu całkowitego, znakowego lub zmiennoprzecinkowego. Może to powodować liczne błędy ale jako że JS nie jest językiem kompilowalnym nie powoduje tak dużych trudności. W języku JavaScript zmienne deklaruje się przez słowo kluczowe “var”:

```
var moja_zmienna_1; // brak typu
var moja_zmienna_2 = 2; // typ całkowity
var moja_zmienna_3 = 'a'; // typ znakowy
var moja_zmienna_4 = 'Jan'; // typ łańcuchowy
var moja_zmienna_5 = '3.1415'; //
```

Typ zmiennej jest ustalany w momencie przypisania wartości.



Zmienne

Natomiast nie nie stoi na przeszkodzie aby zrobić tak:

```
// definiujemy zmienną bez typu
var zmienna;
zmienna = 2; //ustalamy typ na całkowity
zmienna = 3.4 // typ zmiennoprzecinkowy
zmienna = 'Jan'; // typ łańcuchowy
```

W dowolnym momencie kodu możemy zmienić typ zmiennej przez przypisanie wartości!! Może to powodować pewne niebezpieczne sytuacje, dalego należby bardzo rozważnie przypisywać wartości zmiennym.



Zakresy zmiennych

Typowo dla każdego języka programowania zmienne dzielimy na globalne i lokalne. Zmienne globalne oddziałują w obszarze całego skryptu JS i z dowolnego miejsca mamy do nich dostęp. Natomiast zmienne lokalne działają np. tylko w obrębie jednej funkcji:

```
var zmienna_globalna = 5;
function zrobCos() {
    var zmienna_lokalna = 7;
}
alert(zmienna_globalna);
alert(zmienna_lokalna); // to nie zadziała!!!
```

Zakresy zmiennych

Typowo dla każdego języka programowania zmienne dzielimy na globalne i lokalne. Zmienne globalne oddziałują w obszarze całego skryptu JS i z dowolnego miejsca mamy do nich dostęp. Natomiast zmienne lokalne działają np. tylko w obrębie jednej funkcji:

```
var zmienna_globalna = 5;
function zrobCos() {
    var zmienna_lokalna = 7;
}
alert(zmienna_globalna);
alert(zmienna_lokalna); // to nie zadziała!!!
```

Uwaga:

W środowisku programistów JS obowiązuje zasada, że należy używać zmiennych lokalnych (jak najczęściej funkcji), chyba że z pewnych względów ta sama zmienna musi być wykorzystana w wielu funkcjach wtedy może być globalna. Wynika to z tego że jeśli będzie za dużo zmiennych globalnych kod staje się nieczytelny i trudno jest je zorganizować oraz mamy ograniczoną przestrzeń nazw.

Operacje na zmiennych

Typowym działaniem w JS podobnie jak w innych językach programowania jest wykonywanie operacji na zmiennych. Można do tego celu użyć operatorów artemetycznych: dodawania, odejmowania, mnożenia, dzielenia.

```
var a = 5;  
var b = 6;  
var c = 9;  
var wynik = (a + b) / (c + 1);
```

Operacje na zmiennych

Typowym działaniem w JS podobnie jak w innych językach programowania jest wykonywanie operacji na zmiennych. Można do tego celu użyć operatorów artemetycznych: dodawania, odejmowania, mnożenia, dzielenia.

```
var a = 5;  
var b = 6;  
var c = 9;  
var wynik = (a + b) / (c + 1);
```

Operacji można używać również do łączeniałańcuchów w jeden ciąg:

```
var imie = "Marcin";  
var nazwisko = "Zielinski";  
var imieInazwisko = imie + " " + nazwisko;
```

Operatory logiczne

W języku JavaScript wszystkie instrukcje warunkowe zaczynają się od warunku, czyli wyrażenia które może zrównać wartość logiczną prawda lub fałsz. Aby utworzyć warunek można posłużyć się operatorami logicznymi:

Operator	Opis
<code>==</code>	jest równe
<code>!=</code>	nie jest równe
<code>></code>	jest większe
<code><</code>	jest mniejsze
<code>>=</code>	większe lub równe
<code><=</code>	mniejsze lub równe
<code>&&</code>	i
<code> </code>	lub
<code>!</code>	zaprzeczenie

Instrukcje warunkowe

Najprostszą i jednocześnie najpopularniejszą instrukcją warunkową jest instrukcja “if”, która po obliczeniu wartości wyrażenia wykonuje dany ciąg instrukcji lub nie:

```
var liczba = 7;  
if( liczba < 6) { // warunek niespełniony  
    alert(liczba);  
}  
  
var liczba = 7;  
if( liczba > 6) { // warunek spełniony  
    alert(liczba);  
}  
  
var liczba = 7;  
if( liczba > 6) alert(liczba);
```

Instrukcje warunkowe

Rozbudowana wersja instrukcji “if” z wieloma warunkami:

```
var liczba = 7;
if( liczba < 6) {
    alert("Liczba jest mniejsza od 6");
}
else if (liczba >=6 && liczba < 100) {
    alert("liczba z zakresu 6 do 99");
}
else {
    alert("liczba poza zakresem");
}
```

Pętle

Podobnie jak w językach kompilowalnych do wykonania wielokrotnie tej samej czynności można zastosować pętle. W JavaScript pętla ma postać identyczną jak w języku C++:

```
for( var i=0; i < 5; i++) {  
    alert("To jest komunikat nr" + i);  
}
```

Wynik:

```
To jest komunikat nr 0  
To jest komunikat nr 1  
To jest komunikat nr 2  
To jest komunikat nr 3  
To jest komunikat nr 4
```

DOM (Document Object Model)

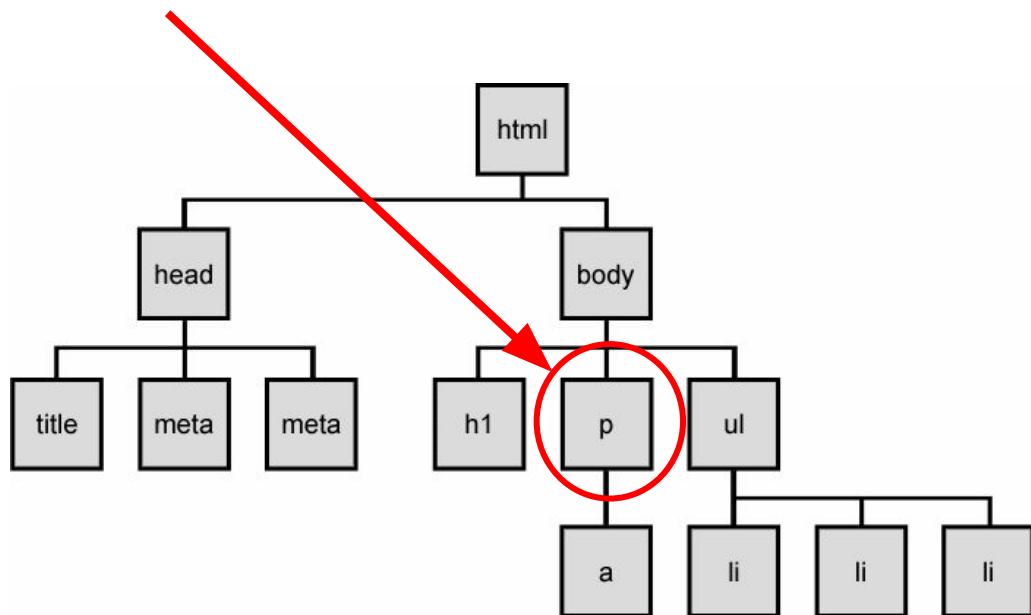
Elementy dokumentu HTML tworzą strukturę drzewiastą w której każdy znacznik pełni rolę węzła (node).

Dzięki zdefiniowanym w modelu DOM metodom i klasą możliwy jest dostęp do dowolnego węzła:

```
document.getElementsByTagName("p");
```

Przykład modelu DOM dla prostego dokumentu hipertekstowego

<http://www.w3.org/DOM/>



Tablice

Tablice są bardzo wygodne do przechowywania różnego rodzaju danych.

W JavaScript mamy pełną swobodę tworzenia tablic:

```
var kolory= [];  
kolory.push("zielony");  
kolory.push("czerwony");  
kolory.push("zolty");  
  
// możemy też bezpośrednio dodawać elementy  
tablicy  
kolory[3] = "niebieski";  
// wyciągamy teraz kolory  
for (var i=0; i< kolory.length; i++) {  
    alert("kolor: " + kolory[i]);  
}
```

Tablice

W JavaScript tablice najlepiej jest definiować od razu jako utworzenie nowego obiektu typu Array:

```
.....  
// pusta tablica - niezadeklarowana  
var tablica = new Array();  
// tablica 5 elementów bez zadeklar. wartości  
var tablica = new Array(5);  
// tablica 3 elementów zadeklarowanych  
var tablica = new Array(2,3,5);  
var tablica = new Array("a","b","c");  
var tablica = new Array(2,"a",3.14);  
// dodawanie kolejnych elementów tablicy  
tablica.push("b",10,12.2);  
// dostęp do elementu nr 2 tablicy  
alert(tablica[2]); // na ekranie wartość ???  
.....
```

Tablice

W JavaScript tablice najlepiej jest definiować od razu jako utworzenie nowego obiektu typu Array:

```
.....  
// pusta tablica - niezadeklarowana  
var tablica = new Array();  
// tablica 5 elementów bez zadeklar. wartości  
var tablica = new Array(5);  
// tablica 3 elementów zadeklarowanych  
var tablica = new Array(2,3,5);  
var tablica = new Array("a","b","c");  
var tablica = new Array(2,"a",3.14);  
// dodawanie kolejnych elementów tablicy  
tablica.push("b",10,12.2);  
// dostęp do elementu nr 2 tablicy  
alert(tablica[2]); // na ekranie wartość
```



3.14

Tablice i pętla “for-in”

Bardzo często koniecznym jest przeiterowanie przez wszystkie elementy tablicy w tym celu możemy użyć pętli “for”, jednak wygodniejszą jej postacią jest pętla “for-in”

```
var tablica = new Array(3,4,5,6);
for ( k in tablica) {
    alert(tablica[k]);
}
```

```
var tablica = new Array(3,4,5,6);
var suma = 0;
for ( k in tablica) {
    suma += tablica[k];
}
```

Tablice i pętla “for-in”

Bardzo często koniecznym jest przeiterowanie przez wszystkie elementy tablicy w tym celu możemy użyć pętli “for”, jednak wygodniejszą jej postacią jest pętla “for-in”

```
var tablica = new Array(3,4,5,6);
for ( k in tablica) {
    alert(tablica[k]);
}
```

```
var tablica = new Array(3,4,5,6);
var suma = 0;
for ( k in tablica) {
    suma += tablica[k];
}
```

suma = 18

Przydatne metody dla obiektu Array

Standardowe obiekty JavaScriptowe dostarczają w pakuiecie przydatne metody pozwalające na proste operacje na danym obiekcie. Dla obiektu Array najważniejsze to:

```
// metoda .concat()  
  
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
  
var tab3 = tab1.concat(tab2);
```

Przydatne metody dla obiektu Array

Standardowe obiekty JavaScriptowe dostarczają w pakuie przydatne metody pozwalające na proste operacje na danym obiekcie. Dla obiektu Array najważniejsze to:

```
// metoda .concat()  
  
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
  
var tab3 = tab1.concat(tab2);
```

Metoda ta kopiuje "tab1" i dodaje do niej elementy "tab2".

Przydatne metody dla obiektu Array

Standardowe obiekty JavaScriptowe dostarczają w pakuie przydatne metody pozwalające na proste operacje na danym obiekcie. Dla obiektu Array najważniejsze to:

```
-----  
// metoda .concat()  
  
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
  
var tab3 = tab1.concat(tab2);  
  
-----  
// metoda .join()  
  
var tab1 = new Array("a", "b", "c");  
var string = tab1.join('|');
```

Metoda ta kopiuje "tab1" i dodaje do niej elementy "tab2". Wynikiem jest nowa tablica "tab3" zawierająca: [a,b,c,d,e,f].

Przydatne metody dla obiektu Array

Standardowe obiekty JavaScriptowe dostarczają w pakuie przydatne metody pozwalające na proste operacje na danym obiekcie. Dla obiektu Array najważniejsze to:

```
-----  
// metoda .concat()  
  
var tab1 = new Array("a", "b", "c") ;  
var tab2 = new Array("d", "e", "f") ;  
  
var tab3 = tab1.concat(tab2) ;  
-----
```

Metoda ta kopiuje "tab1" i dołącza do niej elementy "tab2". Wynikiem jest nowa tablica "tab3" zawierająca: [a,b,c,d,e,f].

```
-----  
// metoda .join()  
  
var tab1 = new Array("a", "b", "c") ;  
var string = tab1.join(' | ') ;  
-----
```

Tworzy łańcuch tekstowy z elementów tablicy z separatorem podanym jak argument metody.

Przydatne metody dla obiektu Array

```
// metoda .pop()  
  
var tab1 = new Array("a", "b", "c");  
var ostatni = tab1.pop();
```

Przydatne metody dla obiektu Array

```
// metoda .pop()
```

Usuwa ostatni element tablicy i jednocześnie zwraca jego wartość.

```
var tab1 = new Array("a", "b", "c");  
var ostatni = tab1.pop();
```

Przydatne metody dla obiektu Array

```
// metoda .pop()
```

Usuwa ostatni element tablicy i jednocześnie zwraca jego wartość.

```
var tab1 = new Array("a", "b", "c");  
var ostatni = tab1.pop();
```

```
// metoda .push()
```

```
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
var tab3 = tab1.push(tab2);
```

Przydatne metody dla obiektu Array

```
// metoda .pop()
```

Usuwa ostatni element tablicy i jednocześnie zwraca jego wartość.

```
var tab1 = new Array("a", "b", "c");  
var ostatni = tab1.pop();
```

```
// metoda .push()
```

Dodaje na końcu tablicy elementy zapisane jako argument metody. Zwraca długość tablicy po modyfikacji.

```
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
var tab3 = tab1.push(tab2);
```

Przydatne metody dla obiektu Array

```
// metoda .pop()
```

Usuwa ostatni element tablicy i jednocześnie zwraca jego wartość.

```
var tab1 = new Array("a", "b", "c");  
var ostatni = tab1.pop();
```

```
// metoda .push()
```

Dodaje na końcu tablicy elementy zapisane jako argument metody. Zwraca długość tablicy po modyfikacji.

```
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
var tab3 = tab1.push(tab2);
```

```
// metoda .reverse()
```

```
var tab1 = new Array("a", "b", "c");  
var tab3 = tab1.reverse();
```

Przydatne metody dla obiektu Array

```
// metoda .pop()
```

Usuwa ostatni element tablicy i jednocześnie zwraca jego wartość.

```
var tab1 = new Array("a", "b", "c");  
var ostatni = tab1.pop();
```

```
// metoda .push()
```

Dodaje na końcu tablicy elementy zapisane jako argument metody. Zwraca długość tablicy po modyfikacji.

```
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
var tab3 = tab1.push(tab2);
```

```
// metoda .reverse()
```

Odwraca kolejność elementów tablicy.

```
var tab1 = new Array("a", "b", "c");  
var tab3 = tab1.reverse();
```

Przydatne metody dla obiektu Array

```
// metoda .shift()  
  
var tab1 = new Array("a", "b", "c");  
var pierwszy = tab1.shift();
```

Przydatne metody dla obiektu Array

```
// metoda .shift()  
  
var tab1 = new Array("a", "b", "c");  
var pierwszy = tab1.shift();
```

Usuwa pierwszy element tablicy i jednocześnie zwraca jego wartość.

Przydatne metody dla obiektu Array

```
// metoda .shift()
```

Usuwa pierwszy element tablicy i jednocześnie zwraca jego wartość.

```
var tab1 = new Array("a", "b", "c");  
var pierwszy = tab1.shift();
```

```
// metoda .unshift()
```

```
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
var tab3 = tab1.push(tab2);
```

Przydatne metody dla obiektu Array

```
// metoda .shift()
```

Usuwa pierwszy element tablicy i jednocześnie zwraca jego wartość.

```
var tab1 = new Array("a", "b", "c");  
var pierwszy = tab1.shift();
```

```
// metoda .unshift()
```

Dodaje na początku tablicy elementy zapisane jako argument metody. Zwraca długość tablicy po modyfikacji.

```
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
var tab3 = tab1.push(tab2);
```

Przydatne metody dla obiektu Array

```
// metoda .shift()
```

Usuwa pierwszy element tablicy i jednocześnie zwraca jego wartość.

```
var tab1 = new Array("a", "b", "c");  
var pierwszy = tab1.shift();
```

```
// metoda .unshift()
```

Dodaje na początku tablicy elementy zapisane jako argument metody. Zwraca długość tablicy po modyfikacji.

```
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
var tab3 = tab1.push(tab2);
```

```
// własność .length
```

```
var tab1 = new Array("a", "b", "c");  
var pierwszy = tab1.length;
```

Przydatne metody dla obiektu Array

```
// metoda .shift()
```

Usuwa pierwszy element tablicy i jednocześnie zwraca jego wartość.

```
var tab1 = new Array("a", "b", "c");  
var pierwszy = tab1.shift();
```

```
// metoda .unshift()
```

Dodaje na początku tablicy elementy zapisane jako argument metody. Zwraca długość tablicy po modyfikacji.

```
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
var tab3 = tab1.push(tab2);
```

```
// własność .length
```

To jest własność która zwraca długość tablicy.

```
var tab1 = new Array("a", "b", "c");  
var pierwszy = tab1.length;
```



Obiekty

Poznaliśmy już jeden typ obiektów w języku JavaScript jakim są tablice. W języku JavaScript wszystkie wartości poza prostymi typami liczbowymi, łańcuchowymi lub logicznymi są obiektami.



Obiekty

Poznaliśmy już jeden typ obiektów w języku JavaScript jakim są tablice. W języku JavaScript wszystkie wartości poza prostymi typami liczbowymi, łańcuchowymi lub logicznymi są obiektami.

Typy proste

Obiekty

Obiekty

Poznaliśmy już jeden typ obiektów w języku JavaScript jakim są tablice. W języku JavaScript wszystkie wartości poza prostymi typami liczbowymi, łańcuchowymi lub logicznymi są obiektami.

Typy proste

Są to liczby, łańcuchy oraz wartości logiczne, posiadające metody, ale są niezmienne.

VS.

Obiekty

Są to asocjacyjne kolekcje klucz-wartość, które można dowolnie modyfikować.

Obiekty

Poznaliśmy już jeden typ obiektów w języku JavaScript jakim są tablice. W języku JavaScript wszystkie wartości poza prostymi typami liczbowymi, łańcuchowymi lub logicznymi są obiektami.

Typy proste

Są to liczby, łańcuchy oraz wartości logiczne, posiadające metody, ale są **niezmienne**.

VS.

Obiekty

Są to asocjacyjne kolekcje klucz-wartość, które można dowolnie **modyfikować**.

W języku JavaScript:

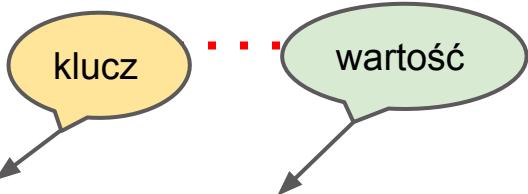
- tablice są obiektami,
- funkcje są obiektami,
- wyrażenia regularne są obiektami,
- obiekty są obiektami.

Tworzenie obiektów

Jedną z podstawowych umiejętności w JavaScript jest tworzenie własnych obiektów reprezentujących logicznie powiązane kolekcje danych.

Obiekt a w zasadzie “literał obiektowy” jest zapisywany za pomocą nawiasów klamrowych który zawiera zero lub więcej par klucz (nazwa)-wartość:

```
var obiekt1 = {};  
var obiekt2 = {  
    imię: "Marcin",  
    nazwisko: "Zielinski"  
};
```



Uwaga:

Nazwa własności może być dowolnym słowem, jednak kiedy jest to np. słowo zastrzeżone dla składni języka JavaScript powinno być pisane w cudzysłowie.

Tworzenie obiektów

Inny przykład:

```
var flight = {  
    carrier: "LOT",  
    number: 3911,  
    origin: "WAW",  
    destination: "KRK",  
    schedule: {  
        departure_time: "22:45",  
        arrival_time: "23:35"  
    }  
};
```

Tworzenie obiektów

Różnica w pisaniu nazw własności (kluczy) :

```
var flight = {  
    carrier: "LOT",  
    number: 3911,  
    origin: "WAW",  
    destination: "KRK",  
    schedule: {  
        "departure-time": "22:45",  
        "arrival-time": "23:35"  
    }  
};
```

Uwaga:

W przypadku wybierania nazw własności (kluczy) w których separatorem jest puza (-) obowiązkowo należy pisać te nazwy w cudzysłowach.

Dostęp do pól obiektu

Mamy dwa obiekty:

```
var flight = {  
    carrier: "LOT",  
    number: 3911,  
    origin: "WAW",  
    destination: "KRK",  
    schedule:{  
        "departure-time": "22:45",  
        "arrival-time": "23:35"  
    }  
};
```

```
var flight = {  
    carrier: "LOT",  
    number: 3911,  
    origin: "WAW",  
    destination: "KRK",  
    schedule:{  
        departure_time: "22:45",  
        arrival_time: "23:35"  
    }  
};
```

Dostęp do pól obiektu

Mamy dwa obiekty:

```
var flight = {  
    carrier: "LOT",  
    number: 3911,  
    origin: "WAW",  
    destination: "KRK",  
    schedule:{  
        "departure-time": "22:45",  
        "arrival-time": "23:35"  
    }  
};
```

```
var flight = {  
    carrier: "LOT",  
    number: 3911,  
    origin: "WAW",  
    destination: "KRK",  
    schedule:{  
        departure_time: "22:45",  
        arrival_time: "23:35"  
    }  
};
```

Aby pobrać dane z konkretnego pola w obiekcie najczęściej stosuje się notację z “kropką” która jest separatorem np.:

```
var numer_lotu = flight.number;  
alert(flight.origin);  
alert(flight.schedule["arrival-time"]);
```

Dostęp do pól obiektu

Mamy dwa obiekty:

```
var flight = {  
    carrier: "LOT",  
    number: 3911,  
    origin: "WAW",  
    destination: "KRK",  
    schedule:{  
        "departure-time": "22:45",  
        "arrival-time": "23:35"  
    }  
};
```

```
var flight = {  
    carrier: "LOT",  
    number: 3911,  
    origin: "WAW",  
    destination: "KRK",  
    schedule:{  
        departure_time: "22:45",  
        arrival_time: "23:35"  
    }  
};
```

Aby pobrać dane z konkretnego pola w obiekcie najczęściej stosuje się notację z “kropką” która jest separatorem np.:

```
var numer_lotu = flight.number;  
alert(flight.origin);  
alert(flight.schedule["arrival-time"]);  
alert(flight.schedule.arrival_time); // dla przykładu po  
// prawej może być tak
```

Modyfikacja własności obiektu

Obiekt można zmodyfikować przez przypisanie. W przypadku gdy dana własność istnieje zostanie ona nadpisana nową wartością, w przypadku kiedy jeszcze nie istnieje zostanie dodana do obiektu:

```
flight.number = 3913; // zmiana istniejącej wartości
flight.equipment = "388";

console.log(flight); // polecenie dla konsoli np. w
                     // Firebugu
```

Modyfikacja własności obiektu

Obiekt można zmodyfikować przez przypisanie. W przypadku gdy dana własność istnieje zostanie ona nadpisana nową wartością, w przypadku kiedy jeszcze nie istnieje zostanie dodana do obiektu:

```
flight.number = 3913; // zmiana istniejącej wartości
flight.equipment = "388";

console.log(flight); // polecenie dla konsoli np. w
                      // Firebugu

var flight = {
    carrier: "LOT",
    number: 3913,
    equipment: "388",
    origin: "WAW",
    destination: "KRK",
    schedule: {
        departure_time: "22:45",
        arrival_time: "23:35"
    }
};
```

Usuwanie własności obiektu

Ostatnią istotną operacją jaką można wykonać na obiekcie jest usuwanie własności. Do tego celu służy operator “delete”:

```
delete flight.number; // usuwamy własność number
delete flight.carrier; // usuwamy nazwę przewoźnika
```

Usuwanie własności obiektu

Ostatnią istotną operacją jaką można wykonać na obiekcie jest usuwanie własności. Do tego celu służy operator “delete”:

```
delete flight.number; // usuwamy własność number
delete flight.carrier; // usuwamy nazwę przewoźnika
```

Postać obiektu po usunięciu dwóch własności:

```
var flight = {
    equipment = "388",
    origin: "WAW",
    destination: "KRK",
    schedule:{
        departure_time: "22:45",
        arrival_time: "23:35"
    }
};
```

Zmienne globalne

W JavaScript możemy przechowywać całą zawartość aplikacji w zmiennych globalnych. Gdy chcemy zastosować zmienne globalne możemy:

```
var globalObj = {} // tworzymy nowy pusty obiekt który
                    // będzie kontenerem dla naszej aplikacji
globalObj.pax = {
    first_name: "Marcin",
    last_name: "Zielinski"
};
globalObj.flight = {
    carrier: "LOT",
    number: 3913,
    equipment: "388",
    origin: "WAW",
    destination: "KRK",
    schedule:{
        departure_time: "22:45",
        arrival_time: "23:35"
    }
};
```

Zmienne globalne

W JavaScript możemy przechowywać całą zawartość aplikacji w zmiennych globalnych. Gdy chcemy zastosować zmienne globalne możemy:

```
var globalObj = {} // tworzymy nowy pusty obiekt który
                    // będzie kontenerem dla naszej aplikacji
globalObj.pax = {
    first_name: "Marcin",
    last_name: "Zielinski"
};
globalObj.flight = {
    carrier: "LOT",
    number: 3913,
    equipment: "388",
    origin: "WAW",
    destination: "KRK",
    schedule:{
        departure_time: "22:45",
        arrival_time: "23:35"
    }
};
```

Jednak zmienne globalne zmniejszają elastyczność aplikacji i raczej się powinno ich unikać.

Zmienne globalne

```
-----  
| console.log(globalObj);  
|  
|-----  
  
globalObj = {  
    pax:{  
        first_name: "Marcin",  
        last_name: "Zielinski"  
    },  
    flight: {  
        carrier: "LOT",  
        number: 3913,  
        equipment: "388",  
        origin: "WAW",  
        destination: "KRK",  
        schedule:{  
            departure_time: "22:45",  
            arrival_time: "23:35"  
        }  
    }  
};
```

Funkcje w obiektach

Składnikiem obiektu może być również funkcja wykonująca dowolne operacje na elementach naszego obiektu:

```
var flight = {  
    carrier: "LOT",  
    number: 3913,  
    equipment: "388",  
    origin: "WAW",  
    destination: "KRK",  
    schedule:{  
        departure_time: "22:45",  
        arrival_time: "23:35"  
    }  
};
```

Nasz pierwotny obiekt do którego chcemy dodać metodę.

Funkcje w obiektach

Składnikiem obiektu może być również funkcja wykonująca dowolne operacje na elementach naszego obiektu:

```
var flight = {  
    carrier: "LOT",  
    number: 3913,  
    equipment: "388",  
    origin: "WAW",  
    destination: "KRK",  
    schedule:{  
        departure_time: "22:45",  
        arrival_time: "23:35"  
    },  
    showFlightNumber: function(){  
        alert(this.number);  
    }  
};
```

Stworzyliśmy
wewnątrz
obiektu funkcję
nienazwaną!

Funkcje w obiektach

Składnikiem obiektu może być również funkcja wykonująca dowolne operacje na elementach naszego obiektu:

```
var flight = {  
    carrier: "LOT",  
    number: 3913,  
    equipment = "388",  
    origin: "WAW",  
    destination: "KRK",  
    schedule:{  
        departure_time: "22:45",  
        arrival_time: "23:35"  
    },  
    showFlightNumber: function(){  
        alert(this.number);  
    }  
};
```

Stworzyliśmy
wewnątrz
obiektu funkcję
nienazwaną!

Odwołanie do obiektu z wnętrza
samego siebie !

Funkcje w obiektach

Składnikiem obiektu może być również funkcja wykonująca dowolne operacje na elementach naszego obiektu:

```
var flight = {  
    carrier: "LOT",  
    number: 3913,  
    equipment = "388",  
    origin: "WAW",  
    destination: "KRK",  
    schedule:{  
        departure_time: "22:45",  
        arrival_time: "23:35"  
    },  
    showFlightNumber: function(){  
        alert(this.number);  
    }  
};  
  
// wywołanie funkcji będącej częścią obiektu  
flight.showFlightNumber();
```

Stworzyliśmy
wewnętrz
obiektu funkcję
nienazwaną!

Odwołanie do obiektu z wnętrza
samego siebie !



Informacja o typie pola

Czasami zdarza się że konieczny jest znajomość typu danego pola obiektu. Można to zrobić korzystając z funkcji wybudowanej “typeof” która pobiera daną wartość i sprawdza jej typ:

Informacja o typie pola

Czasami zdarza się że konieczny jest znajomość typu danego pola obiektu. Można to zrobić korzystając z funkcji wybudowanej “typeof” która pobiera daną wartość i sprawdza jej typ:

```
typeof flight.number;      // liczba całkowita bo 3912
typeof flight.equipment; // łańcuch bo "388"

typeof flight.schedule; // obiekt

typeof flight.showFlightNumber; // funkcja
```

Informacja o typie pola

Czasami zdarza się że konieczny jest znajomość typu danego pola obiektu. Można to zrobić korzystając z funkcji wybudowanej “typeof” która pobiera daną wartość i sprawdza jej typ:

```
typeof flight.number;      // liczba całkowita bo 3912
typeof flight.equipment; // łańcuch bo "388"

typeof flight.schedule; // obiekt

typeof flight.showFlightNumber; // funkcja
```

Aby sprawdzić czy obiekt posiada interesującą nas własność możemy zastosować metodę “hasOwnProperty()”, która zwraca wartość logiczną true/false w zależności czy obiekt posiada interesującą nas własność:

```
flight.hasOwnProperty("number"); // true !!
flight.hasOwnProperty("showEquipment"); // false !!
```



KONIEC WYKŁADU 5



UNIWERSYTET
JAGIELŁOŃSKI
W KRAKOWIE

Zaawansowane Techniki WWW (HTML, CSS i JavaScript)

Dr inż. Marcin Zieliński

Środa 15:30 - 17:00 sala: A-1-04

WYKŁAD 6

Wykład dla kierunku: Informatyka Stosowana II rok

Rok akademicki: 2015/2016 - semestr zimowy

Przypomnienie z poprzedniego wykładu

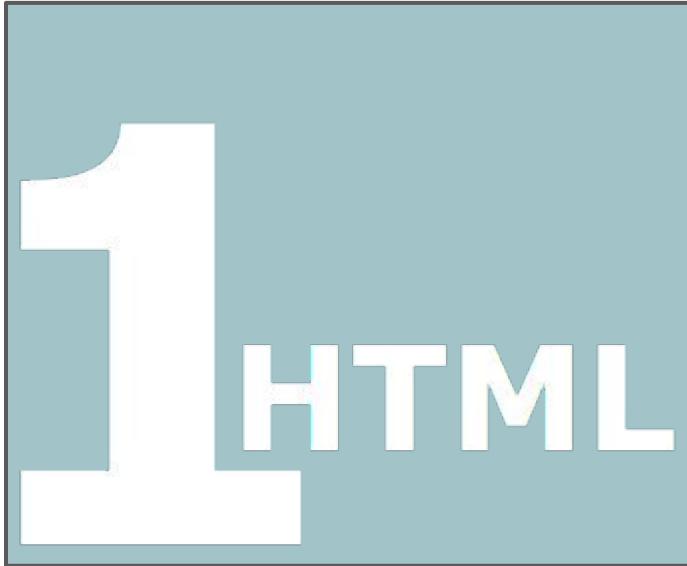
RWD, Mobile-First, Mediaqueries

Wprowadzenie do języka JavaScript

DOM - Document Object Model

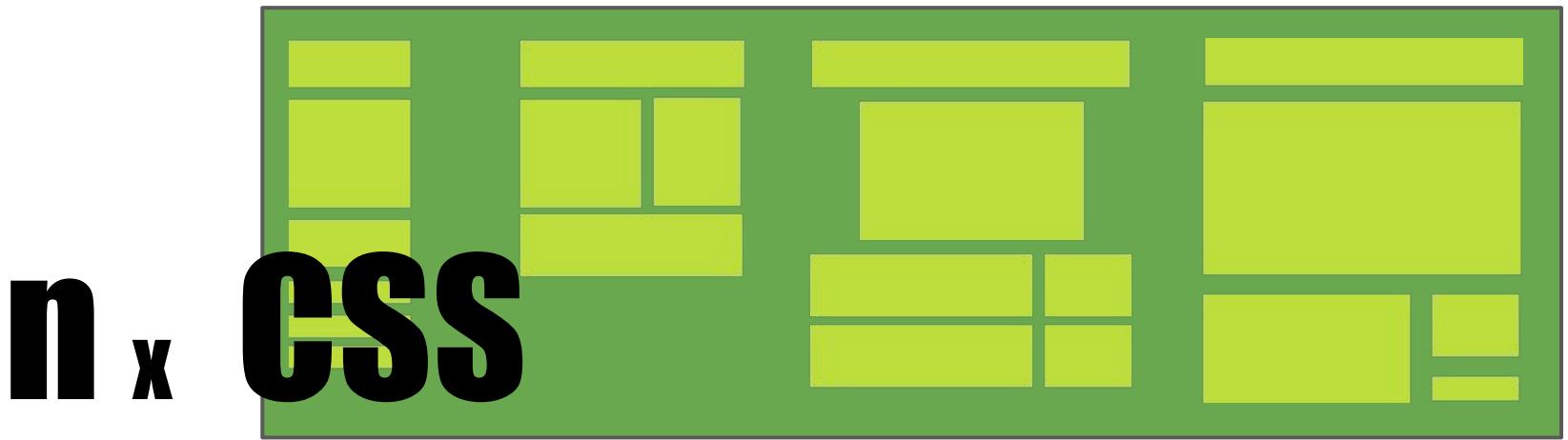
Obiekty w JavaScript

Jak stosować Mediaqueries?

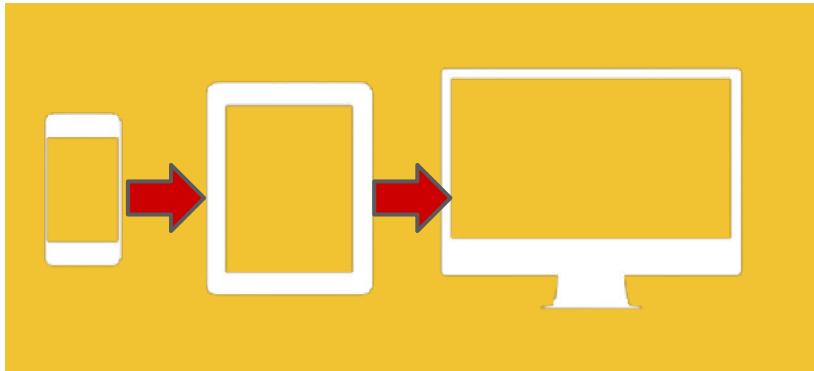


ZASADA:

W ogólności zasada jaka powinna przyświecać tworzeniu stron zgodnie z metodologią RWD jest tworzenie jednego pliku HTML, a dla formatowania jego wygądu wiele “layoutów” w CSS które będą zawierać odpowiednie reguły wyświetlania strony w zależności od rozdzielczości ekranu.

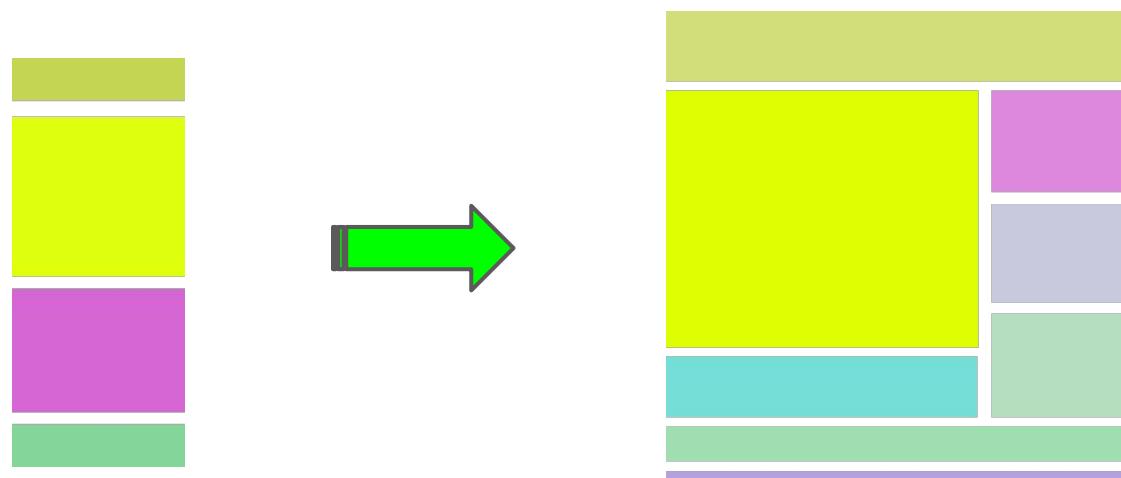


Które CSSy tworzyć najpierw ?



**Filozofia
Mobile-First**

Zgodnie z filozofią MobileFirst tworzenie strony powinno być rozpoczęte od najmniejszych ekranów czyli dla urządzeń mobilnych.



Jak stosować kod w języku JS ?

Załączenie do strony internetowej kodu napisanego w języku JavaScript (w skrócie skryptu) można dokonać na dwa sposoby:

- osadzić skrypt w kodzie HTML strony (**embedded-script**),
- umieścić w osobnym pliku (**linked-script**).



HTML5

Embedded-Script

```
<body>
<script>
  alert("Hello World!");
</script>
</body>
```

Linked-Script

```
<head>
<script src="skrypt.js">
</script>
</head>
```



Plik
.html

```
  alert("Hello World!");
```



Plik
.js

Obiekty

Poznaliśmy już jeden typ obiektów w języku JavaScript jakim są tablice. W języku JavaScript wszystkie wartości poza prostymi typami liczbowymi, łańcuchowymi lub logicznymi są obiektami.

Typy proste

Są to liczby, łańcuchy oraz wartości logiczne, posiadające metody, ale są **niezmienne**.

VS.

Obiekty

Są to asocjacyjne kolekcje klucz-wartość, które można dowolnie **modyfikować**.

W języku JavaScript:

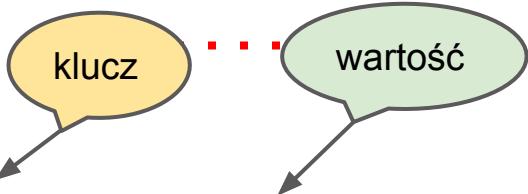
- tablice są obiektami,
- funkcje są obiektami,
- wyrażenia regularne są obiektami,
- obiekty są obiektami.

Tworzenie obiektów

Jedną z podstawowych umiejętności w JavaScript jest tworzenie własnych obiektów reprezentujących logicznie powiązane kolekcje danych.

Obiekt a w zasadzie “literał obiektowy” jest zapisywany za pomocą nawiasów klamrowych który zawiera zero lub więcej par klucz (nazwa)-wartość:

```
var obiekt1 = {};  
var obiekt2 = {  
    imię: "Marcin",  
    nazwisko: "Zielinski"  
};
```



Uwaga:

Nazwa własności może być dowolnym słowem, jednak kiedy jest to np. słowo zastrzeżone dla składni języka JavaScript powinno być pisane w cudzysłowie.

Funkcje w obiektach

Składnikiem obiektu może być również funkcja wykonująca dowolne operacje na elementach naszego obiektu:

```
var flight = {  
    carrier: "LOT",  
    number: 3913,  
    equipment = "388",  
    origin: "WAW",  
    destination: "KRK",  
    schedule:{  
        departure_time: "22:45",  
        arrival_time: "23:35"  
    },  
    showFlightNumber: function(){  
        alert(this.number);  
    }  
};  
  
// wywołanie funkcji będącej częścią obiektu  
flight.showFlightNumber();
```

Stworzyliśmy
wewnętrz
obiektu funkcję
nienazwaną!

Odwołanie do obiektu z wnętrza
samego siebie !



JavaScript i aplikacje WWW

Jak już wspomnieliśmy język JavaScript powstał niejako z potrzeby podniesienia atrakcyjności stron www oraz nadania im odpowiedniej dynamiki oraz nowych funkcjonalności. Jedną z jego naważniejszych cech jest to że jest całkowicie obiektowy i operuje na zdarzeniach.



JavaScript i aplikacje WWW

Jak już wspomnieliśmy język JavaScript powstał niejako z potrzeby podniesienia atrakcyjności stron www oraz nadania im odpowiedniej dynamiki oraz nowych funkcjonalności. Jedną z jego naważniejszych cech jest to że jest całkowicie obiektowy i operuje na zdarzeniach.

Ale co to w zasadzie znaczy ??

JavaScript i aplikacje WWW

Jak już wspomnieliśmy język JavaScript powstał niejako z potrzeby podniesienia atrakcyjności stron www oraz nadania im odpowiedniej dynamiki oraz nowych funkcjonalności. Jedną z jego naważniejszych cech jest to że jest całkowicie obiektowy i operuje na zdarzeniach.

Ale co to w zasadzie znaczy ??

Obiektemi są wszystkie elementy strony jak i sama strona w całości. Jest to konsekwencją zastosowanie modelu DOM. Te obiekty można modyfikować za pomocą metod.

Zdarzenie to określona czynność, która została wykonana przez użytkownika strony WWW podczas odwiedzania danej strony. Przykładowe zdarzenia to m.in. przesunięcie wskaźnika myszki nad obrazkiem, zaznaczenie zawartości obiektu czy wysłanie zawartości formularza.

W języku JavaScript zdarzenia są obsługiwane za pomocą specjalnych poleceń, **zwanych funkcjami obsługi zdarzeń**. Oznacza to, że określona czynność, wykonana przez użytkownika strony WWW, wywoła przypisaną do tej czynności (do tego zdarzenia) funkcję obsługi zdarzenia.



Obiekty i zdarzenia na stronie

Obiektem jest cały dokument hipertekstowy html. Obiektem który go reprezentuje jest “document”. Obiekt ten posiada własności oraz kilka wbudowanych metod które pozwalają na modyfikację zawartości strony lub uzyskanie informacji. Przykładowo gdy chcemy dodać



Obiekty i zdarzenia na stronie

Obiektem jest cały dokument hipertekstowy html. Obiektem który go reprezentuje jest “document”. Obiekt ten posiada własności oraz kilka wbudowanych metod które pozwalają na modyfikację zawartości strony lub uzyskanie informacji. Przykładowo gdy chcemy dodać

```
// właściwość obiektu document
document.lastModified;

// metoda dla obiektu document
document.write("Modyfikujemy zawartość dokumentu");
```

Obiekty i zdarzenia na stronie

Obiektem jest cały dokument hipertekstowy html. Obiektem który go reprezentuje jest “document”. Obiekt ten posiada własności oraz kilka wbudowanych metod które pozwalają na modyfikację zawartości strony lub uzyskanie informacji. Przykładowo gdy chcemy dodać

```
// właściwość obiektu document
document.lastModified;

// metoda dla obiektu document
document.write("Modyfikujemy zawartość dokumentu");
```

Zdarzeniem jest np. naciśnięcie klawisza myszy. Funkcja obsługi zdarzenia odpowiedzialna dla takiego przypadku to “onClick()”, która spowoduje wykonanie określonej czynności jaką jej przypisano w sytuacji zajścia danego zdarzenia:

Obiekty i zdarzenia na stronie

Obiektem jest cały dokument hipertekstowy html. Obiektem który go reprezentuje jest “document”. Obiekt ten posiada własności oraz kilka wbudowanych metod które pozwalają na modyfikację zawartości strony lub uzyskanie informacji. Przykładowo gdy chcemy dodać

```
// właściwość obiektu document
document.lastModified;

// metoda dla obiektu document
document.write("Modyfikujemy zawartość dokumentu");
```

Zdarzeniem jest np. naciśnięcie klawisza myszy. Funkcja obsługi zdarzenia odpowiedzialna dla takiego przypadku to “onClick()”, która spowoduje wykonanie określonej czynności jaką jej przypisano w sytuacji zajścia danego zdarzenia:

```
document.getElementById("button").onclick = function() {
    alert("Nacisnęłaś przycisk");
}
```

Obiekty i zdarzenia na stronie

Pozostałe typy zdarzeń które mogą zostać obsłużone:

onabort

Zaniechano ładowania strony

onblur

Obiekt przestał być aktywny (wybrany)

onchange

Obiekt zmienił swój stan

onclick

Kliknięto obiekt

onerror

Błąd w skrypcie

onfocus

Obiekt został uaktywniony (wybrany)

onload

Obiekt (strona) został załadowany
(zakończono jego ładowanie)

onmouseover

Obiekt został wskazany myszką



Obiekty i zdarzenia na stronie

Pozostałe typy zdarzeń które mogą zostać obsłużone:

onmouseout

Obiekt przestał być wskazywany myszką

onselect

Zawartość obiektu została zaznaczona

onsubmit

Zawartość formularza została przesłana
do serwera

onunload

Zmieniono wyświetlana stronę

Obiekty i zdarzenia na stronie

Przykład:

```
<script>
var name="NN"
function hello()
{
name=prompt("Podaj swoje imię:",name);
alert("Witaj "+name+" na naszej stronie!");
}
function goodbye()
{
alert("Do widzenia "+name+" !");
}
</script>

<body onload="hello(); onunload=goodbye();">
```

Obiekty i zdarzenia na stronie

Przykład:

```
<script>
var name="NN"
function hello()
{
name=prompt("Podaj swoje imię:",name);
alert("Witaj "+name+" na naszej stronie!");
}
function goodbye()
{
alert("Do widzenia "+name+" !");
}
</script>

<body onload="hello(); onunload=goodbye();">
```

Procedura obsługi **onload** gwarantuje, iż cała zawartość strony zostanie wyświetlona przed wykonaniem tej procedury. Podobnie, procedura obsługi **onunload** pozwala na wykonanie skryptu przed załadowaniem nowej strony.

Obiekty i zdarzenia na stronie

Przykład:

```
function listaURL(a,b,c,d,e)
{
  this[0]=a;
  this[1]=b;
  this[2]=c;
  this[3]=d;
  this[4]=e;
}

function wyborStrony(lista)
{
  var dzis = new Date();
  var numer = dzis.getSeconds() % 5;
  window.open(lista[numer] , "oknoLosowe")
}

listaAdresow = new listaURL("http://www.onet.pl",
"http://www.wp.pl.pl",
"http://www.interia..pl",
"http://www.uj.edu.pl",
"http://www.gazeta.pl");
```

```
<body onload="wyborStrony(listaAdresow);">
```

Obiekty i zdarzenia na stronie

Przykład:

```
function listaURL(a,b,c,d,e)
{
  this[0]=a;
  this[1]=b;
  this[2]=c;
  this[3]=d;
  this[4]=e;
}

function wyborStrony(lista)
{
  var dzis = new Date();
  var numer = dzis.getSeconds() % 5;
  window.open(lista[numer] , "oknoLosowe")
}

listaAdresow = new listaURL("http://www.onet.pl",
"http://www.wp.pl.pl",
"http://www.interia..pl",
"http://www.uj.edu.pl",
"http://www.gazeta.pl");
```

```
<body onload="wyborStrony(listaAdresow);">
```

W kodzie JavaScriptu wszystkie odwołania do procedur obsługi zdarzeń należy zapisywać za pomocą małych liter, np. **onclik**. Wewnątrz etykiet języka HTML można zamiennie używać małych i dużych liter, pisząc np. **onClick**.

- ale nie ma to żadnego znaczenia !!!

Obiekty i zdarzenia na stronie

Emulowanie zdarzeń:

blur()	Metoda JavaScriptu.	Usuwa miejsce wprowadzania z aktywnego pola.
click()	Metoda JavaScriptu.	Emuluje kliknięcie myszą na obiekcie.
focus()	Metoda JavaScriptu.	Emuluje uaktywnienie danego obiektu.
reset()	Metoda JavaScriptu.	Emuluje kliknięcie na przycisku Reset przez użytkownika.
submit()	Metoda JavaScriptu.	Emuluje kliknięcie na przycisku Submit przez użytkownika.
select()	Metoda JavaScriptu.	Powoduje wybranie określonego pola formularza.

Obiekty i zdarzenia na stronie

Emulowanie zdarzeń:

blur()	Metoda JavaScriptu.	Usuwa miejsce wprowadzania z aktywnego pola.
click()	Metoda JavaScriptu.	Emuluje kliknięcie myszą na obiekcie.
focus()	Metoda JavaScriptu.	Emuluje uaktywnienie danego obiektu.
reset()	Metoda JavaScriptu.	Emuluje kliknięcie na przycisku Reset przez użytkownika.
submit()	Metoda JavaScriptu.	Emuluje kliknięcie na przycisku Submit przez użytkownika.
select()	Metoda JavaScriptu.	Powoduje wybranie określonego pola formularza.

Emulowanie zdarzeń może się okazać przydatne, gdy np. trzeba wysłać formularz bez konieczności proszenia użytkownika o kliknięcie na przycisku **Submit lub gdy trzeba np. zmienić miejsce wprowadzania informacji w zależności od czynności podejmowanych przez użytkownika.**

Obiekty i zdarzenia na stronie

Emulowanie zdarzeń:

blur()	Metoda JavaScriptu.	Usuwa miejsce wprowadzania z aktywnego pola.
click()	Metoda JavaScriptu.	Emuluje kliknięcie myszą na obiekcie.
focus()	Metoda JavaScriptu.	Emuluje uaktywnienie danego obiektu.
reset()	Metoda JavaScriptu.	Emuluje kliknięcie na przycisku Reset przez użytkownika.
submit()	Metoda JavaScriptu.	Emuluje kliknięcie na przycisku Submit przez użytkownika.
select()	Metoda JavaScriptu.	Powoduje wybranie określonego pola formularza.

Emulowanie zdarzeń może się okazać przydatne, gdy np. trzeba wysłać formularz bez konieczności proszenia użytkownika o kliknięcie na przycisku **Submit lub gdy trzeba np. zmienić miejsce wprowadzania informacji w zależności od czynności podejmowanych przez użytkownika.**

Przykład:

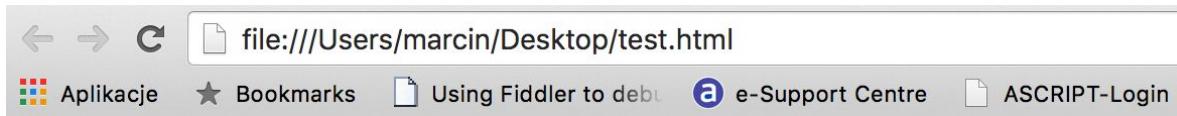
W przypadku formularza który chcemy zwalidować przed wysłaniem, możemy wykonać odpowiedni skrypt sprawdzający po naciśnięciu przycisku wyslij, a dopiero gdy wynik sprawdzenia jest pozytywny skrypt za pomocą zdarzenia **submit()** przesyła dane na serwer.

Obiekty i zdarzenia na stronie

Przykład emulowania zdarzenia "click":

```
<form>
  <input type="checkbox" id="myCheck" onmouseover="myFunction()" onclick="
    alert('click event occurred')">
</form>

<script>
function myFunction() {
  document.getElementById("myCheck").click();
}
</script>
```



Testuje emulacje zdarzenia "click" po najechaniu na "checkboxa" kursem myszy:..



Obiekty i zdarzenia na stronie

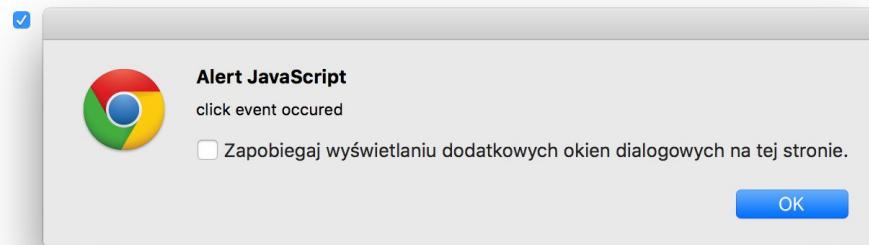
Przykład emulowania zdarzenia "click":

```
<form>
  <input type="checkbox" id="myCheck" onmouseover="myFunction()" onclick="
    alert('click event occurred')">
</form>

<script>
function myFunction() {
  document.getElementById("myCheck").click();
}
</script>
```



Testuje emulacje zdarzenia "click" po najechaniu na "checkboxa" kursorem myszy:.





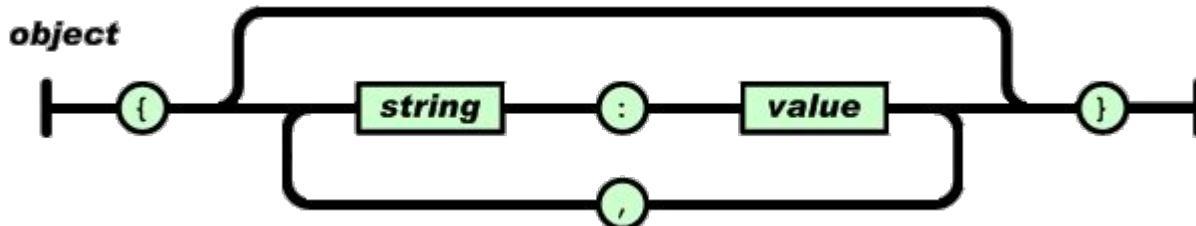
JSON

JSON (JavaScript Object Notation) [Notacja Obiektowa JavaScriptu] - jest to “lekki” format do przenoszenia danych oparty o literały obiektowe JavaScriptu. Jest podzbiorem JS, ale kompletnie niezależnym i może być używany do wymiany danych w zasadzie w każdym współczesnym języku programowania.

JSON

JSON (JavaScript Object Notation) [*Notacja Obiektowa JavaScriptu*] - jest to “lekki” format do przenoszenia danych oparty o literały obiektowe JavaScriptu. Jest podzbiorem JS, ale kompletnie niezależnym i może być używany do wymiany danych w zasadzie w każdym współczesnym języku programowania.

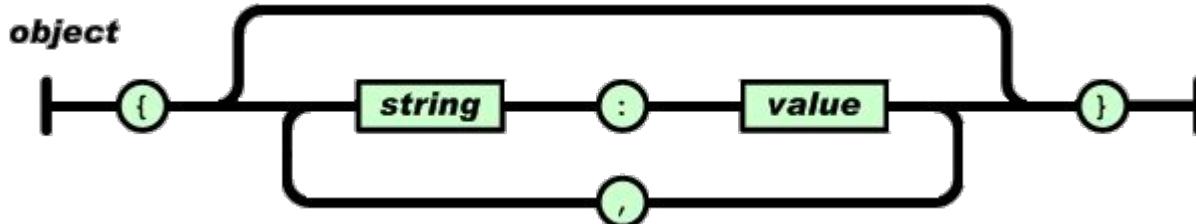
<http://www.json.org/>



JSON

JSON (JavaScript Object Notation) [*Notacja Obiektowa JavaScriptu*] - jest to “lekki” format do przenoszenia danych oparty o literały obiektowe JavaScriptu. Jest podzbiorem JS, ale kompletnie niezależnym i może być używany do wymiany danych w zasadzie w każdym współczesnym języku programowania.

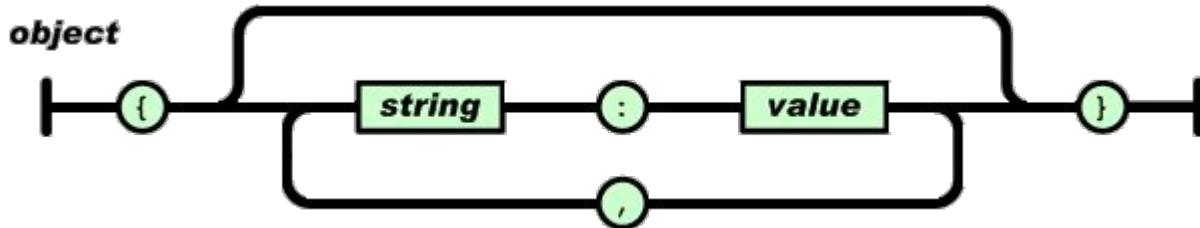
<http://www.json.org/>



```
{  
  "firstname": "Jan",  
  "lastname": "Kowalski",  
  "age": 20  
}
```

Obiekt w formacie JSON jest nieuporządkowanym zbiorem klucz-wartość, gdzie klucz może być dowolnym łańcuchem, natomiast wartość jednym z dowolnych typów (integer, string etc) włączając w to tablice i inne obiekty.

JSON



JSON posiada sześć rodzajów wartości (value):

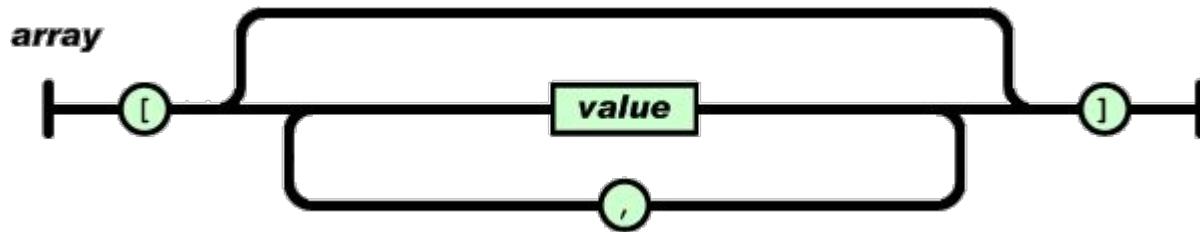
1. obiekty,
2. tablice,
3. łańcuchy,
4. liczby,
5. wartości logiczne (TRUE FALSE),
6. wartość NULL.

Obiekty JSON można zagnieźdzać w dowolnym stopniu jednak z reguły dla zachowania przejrzystości stosowana jest zasada “im bardziej płaska struktura tym lepiej”. Jednak nie w każdym przypadku i dla wszystkich zastosowań jest to możliwe do utrzymania.

JSON

W formacie JSON możemy także posługiwać się tablicami, które tworzą uporządkowane ciągi wartości o dowolnych typach dozwolonych przez JSONa (w tym tablice i obiekty).

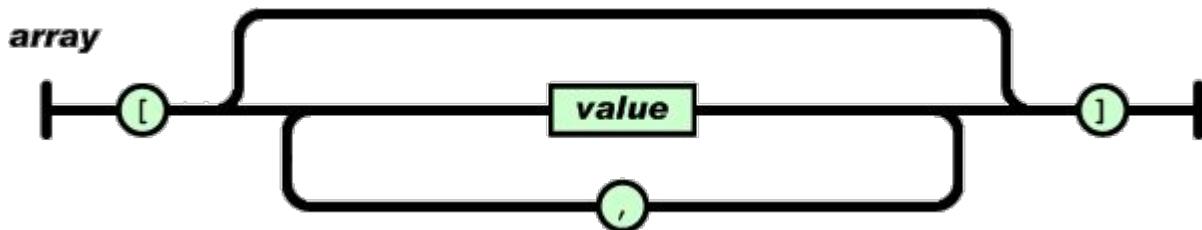
<http://www.json.org/>



JSON

W formacie JSON możemy także posługiwać się tablicami, które tworzą uporządkowane ciągi wartości o dowolnych typach dozwolonych przez JSONa (w tym tablice i obiekty).

<http://www.json.org/>



```
[ {  
    "firstname": "Jan",  
    "lastname": "Kowalski",  
    "age": 20  
,  
{  
    "firstname": "Anna",  
    "lastname": "Nowak",  
    "age": 25  
} ]
```

Uwaga:

Podobnie jak w JavaScript niedopuszczalne jest rozpoczęwanie liczb całkowitych od "zera" np.:

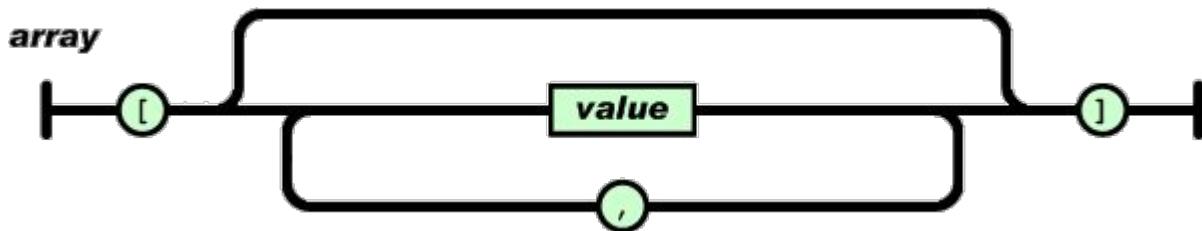
```
{ "liczba": 023 }
```

W niektórych przypadkach zapis taki może zostać zinterpretowany jako liczba w formacie ósemkowym.

JSON

W formacie JSON możemy także posługiwać się tablicami, które tworzą uporządkowane ciągi wartości o dowolnych typach dozwolonych przez JSONa (w tym tablice i obiekty).

<http://www.json.org/>



```
[ {  
    "firstname": "Jan",  
    "lastname": "Kowalski",  
    "age": 20  
,  
{  
    "firstname": "Anna",  
    "lastname": "Nowak",  
    "age": 25  
} ]
```

Uwaga:

Format tekstowy (podobnie zresztą jak XML-a) zapewnia czytelność nie tylko dla parserów, ale również dla ludzi.

JSON

Przykład tablicy obiektów:

```
{ "samochod": [ { "Marka": "VW", "Model": "Golf", "Rocznik": 1999 }, { "Marka": "BMW", "Model": "S6", "Rocznik": 2007 }, { "Marka": "Audi", "Model": "A4", "Rocznik": 2009 } ] }
```



XML

XML (eXtensible Markup Language) [Rozszerzalny Język Znaczników] - jest to format do przenoszenia danych oparty o znaczniki. W zasadzie jest przeznaczony do opisu struktury danych oraz powiązań między nimi. Jest rekommendowany przez W3C jako standard wymiany danych. Jego niezwykłą zaletą (zresztą podobną do JSONa) jest to że jest jest zapisywany w postaci zwykłych plików ASCII.

XML

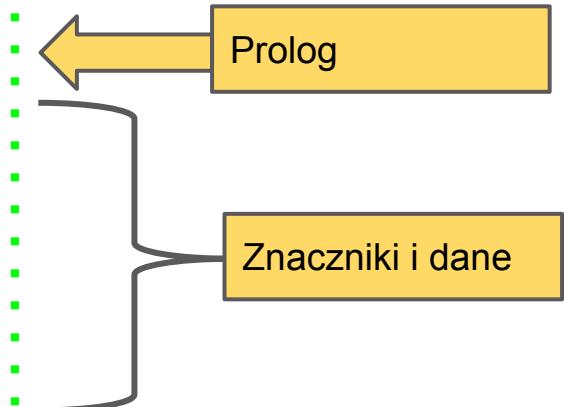
XML (eXtensible Markup Language) [Rozszerzalny Język Znaczników] - jest to format do przenoszenia danych oparty o znaczniki. W zasadzie jest przeznaczony do opisu struktury danych oraz powiązań między nimi. Jest rekommendowany przez W3C jako standard wymiany danych. Jego niezwykłą zaletą (zresztą podobną do JSONa) jest to że jest zapisywany w postaci zwykłych plików ASCII.

```
<?xml version="1.0" encoding="UTF-8"?>
<osoba>
    <imie>Jan</imie>
    <nazwisko>Kowalski</nazwisko>
    <pesel>83010198741</pesel>
</osoba>
```

XML

XML (eXtensible Markup Language) [Rozszerzalny Język Znaczników] - jest to format do przenoszenia danych oparty o znaczniki. W zasadzie jest przeznaczony do opisu struktury danych oraz powiązań między nimi. Jest rekomendowany przez W3C jako standard wymiany danych. Jego niezwykłą zaletą (zresztą podobną do JSONa) jest to że jest zapisywany w postaci zwykłych plików ASCII.

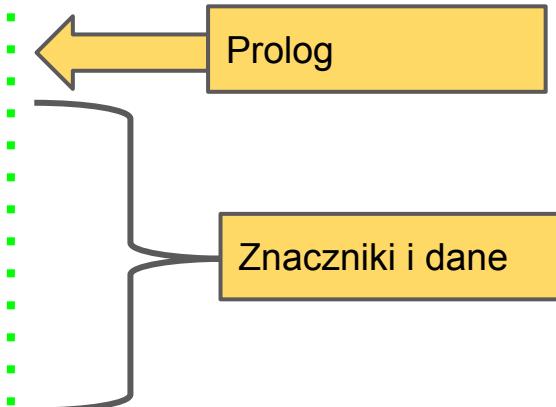
```
<?xml version="1.0" encoding="UTF-8"?>
<osoba>
    <imie>Jan</imie>
    <nazwisko>Kowalski</nazwisko>
    <pesel>83010198741</pesel>
</osoba>
```



XML

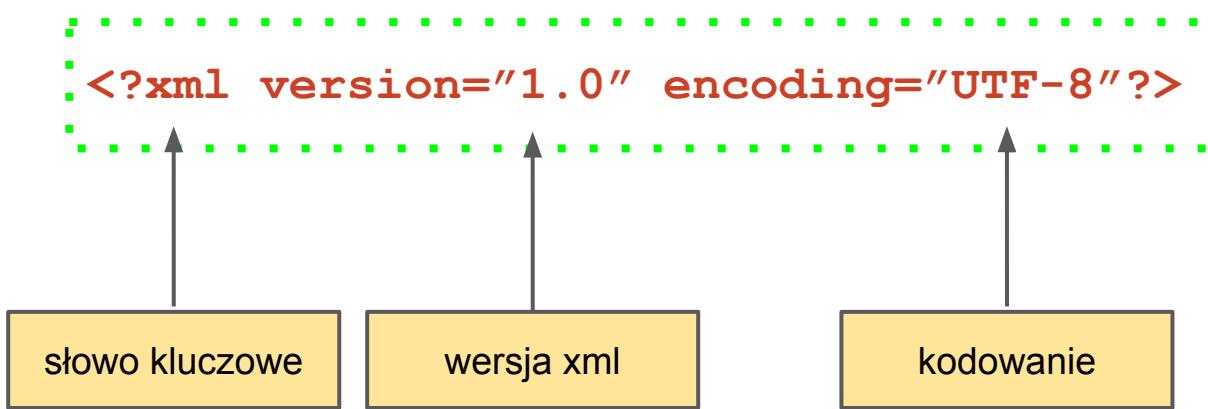
XML (eXtensible Markup Language) [Rozszerzalny Język Znaczników] - jest to format do przenoszenia danych oparty o znaczniki. W zasadzie jest przeznaczony do opisu struktury danych oraz powiązań między nimi. Jest rekommendowany przez W3C jako standard wymiany danych. Jego niezwykłą zaletą (zresztą podobną do JSONa) jest to że jest zapisywany w postaci zwykłych plików ASCII.

```
<?xml version="1.0" encoding="UTF-8"?>
<osoba>
    <imie>Jan</imie>
    <nazwisko>Kowalski</nazwisko>
    <pesel>83010198741</pesel>
</osoba>
```

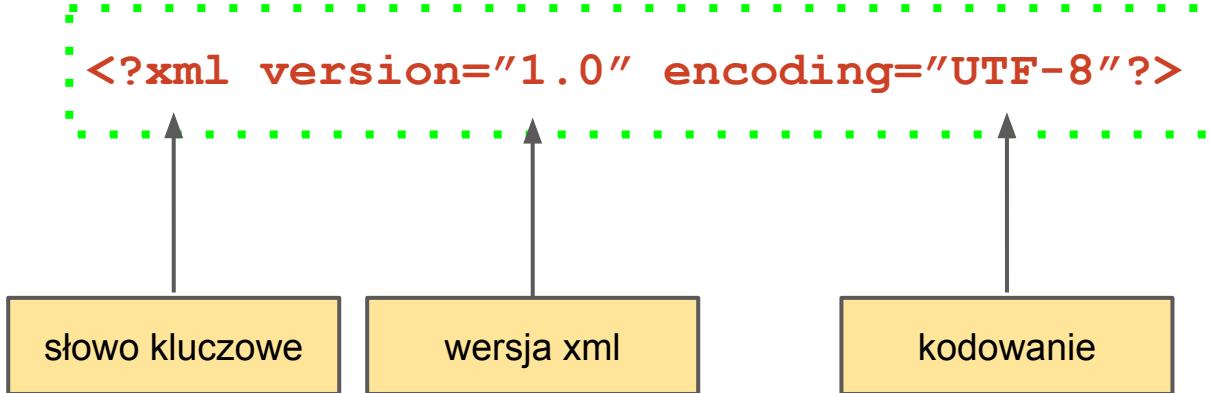


Prolog - powinien być ale nie zawsze jest konieczny!!!

XML

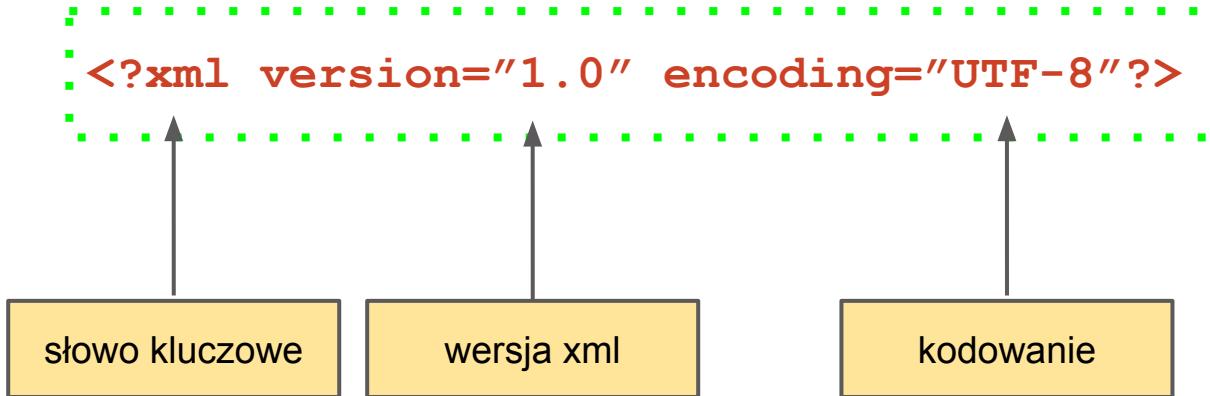


XML



W prologu mogą znaleźć się też inne elementy np. informacja o stylach lub DTD.

XML

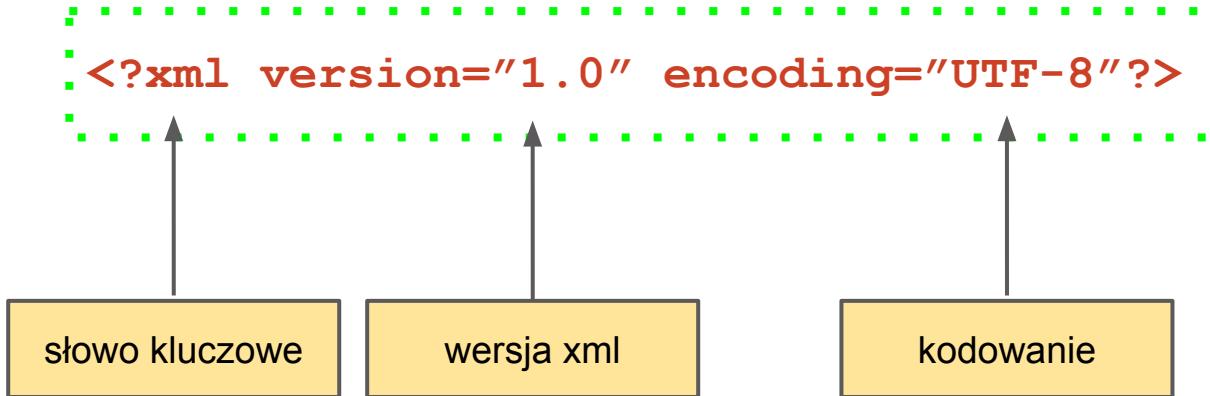


W prologu mogą znaleźć się też inne elementy np. informacja o stylach lub DTD.

```
<osoba>
  <imie>Jan</imie>
  <nazwisko>Kowalski</nazwisko>
  <pesel>83010198741</pesel>
</osoba>
```

Nazwy kolejnych elementów mogą być dowolnym siągiem znaków rozpoczynającym się do litery lub podkreślenia.

XML



W prologu mogą znaleźć się też inne elementy np. informacja o stylach lub DTD.

```
<osoba>
  <imie>Jan</imie>
  <nazwisko>Kowalski</nazwisko>
  <pesel>83010198741</pesel>
</osoba>
```

Nazwy kolejnych elementów mogą być dowolnym ciągiem znaków rozpoczynającym się od litery lub podkreślenia.

Znacznik z tekstem (lub innymi znacznikami wewnętrz) jest bardzo często nazywany węzłem (node). Natomiast pierwszy element “węzłem głównym” (root-node).

XML

W plikach XML możemy również podobnie jak w HTML stosować kaskadowe arkusze stylu, które pozwalają odpowiednio sformatować wygląd przenoszonych danych. Dołączenie CSS odbywa się przez dyrektywę “xmlstylesheet”.

```
KOMIS {  
    display: block;  
    background-colr: #AAAAAA;  
    color: #FFFFFF;  
}  
  
<?xml version="1.0" encoding="utf-8"?>  
<?xml-stylesheet type="text/css" href="styl.css"?>  
<KOMIS>  
    <Samochod>  
        <marka>BMW</marka>  
    </Samochod>  
</KOMIS>
```

XML

W plikach XML możemy również podobnie jak w HTML stosować kaskadowe arkusze stylu, które pozwalają odpowiednio sformatować wygląd przenoszonych danych. Dołączenie CSS odbywa się przez dyrektywę “xml-stylesheet”.

```
KOMIS {  
    display: block;  
    background-colr: #AAAAAA;  
    color: #FFFFFF;  
}  
  
<?xml version="1.0" encoding="utf-8"?>  
<?xml-stylesheet type="text/css" href="styl.css"?>  
<KOMIS>  
    <Samochod>  
        <marka>BMW</marka>  
    </Samochod>  
</KOMIS>
```

Sposób przygotowanie CSS jest dokładnie taki sam jak dla HTML.

Porównanie JSON vs. XML

```
{ "samochod": [ { "Marka": "VW", "Model": "Golf", "Rocznik": 1999 }, { "Marka": "BMW", "Model": "S6", "Rocznik": 2007 }, { "Marka": "Audi", "Model": "A4", "Rocznik": 2009 } ] }
```

```
<?xml version="1.0" encoding="utf-8"?>
<KOMIS>
  <Samochod>
    <Marka>VW</Marka>
    <Model>Golf</Model>
    <Rok>1999</Rok>
  </Samochod>
  <Samochod>
    <Marka>BMW</Marka>
    <Model>S6</Model>
    <Rok>2007</Rok>
  </Samochod>
  <Samochod>
    <Marka>Audi</Marka>
    <Model>A3</Model>
    <Rok>2009</Rok>
  </Samochod>
</KOMIS>
```

Porównanie JSON vs. XML

Obecnie wiele usług sieciowych wykorzystuje do przekazywania danych pliki XML oraz JSON. Standard XML jest tzw. “ścisły formatem”, który pozwala na łatwą, szybką i wydajną interpretację, niezależnie od platformy sprzętowej czy systemowej. Natomiast JSON cechuje się “lekkością” oraz bardzo przerzystą strukturą. Dla obu formatów w większości języków programowania (kompilowalnych oraz interpretowalnych) istnieją parsery pozwalające na łatwe odczytywanie, zapisywanie i modyfikowanie danych.

Który format wybrać ?

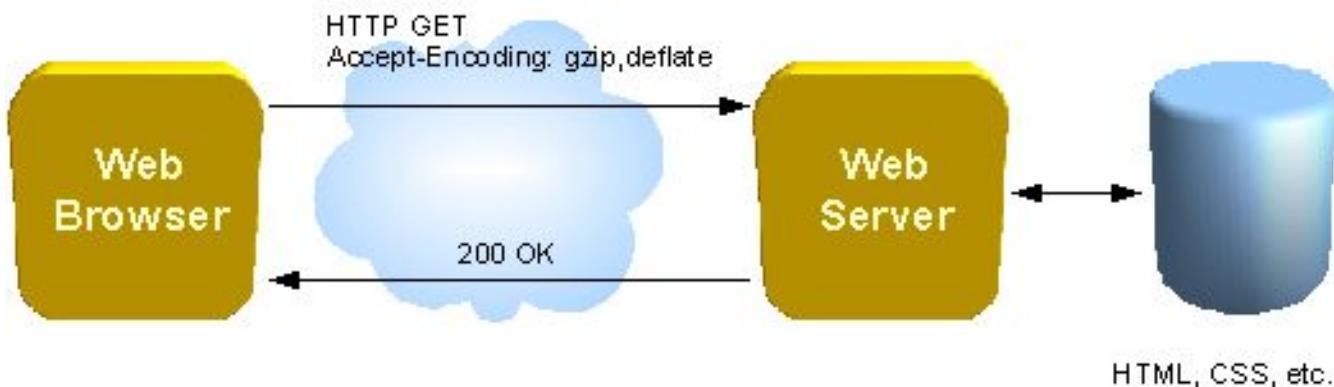
Porównanie JSON vs. XML

Obecnie wiele usług sieciowych wykorzystuje do przekazywania danych pliki XML oraz JSON. Standard XML jest tzw. “ścisły formatem”, który pozwala na łatwą, szybką i wydajną interpretację, niezależnie od platformy sprzętowej czy systemowej. Natomiast JSON cechuje się “lekkością” oraz bardzo przerzystą strukturą. Dla obu formatów w większości języków programowania (kompilowalnych oraz interpretowalnych) istnieją parsery pozwalające na łatwe odczytywanie, zapisywanie i modyfikowanie danych.

Który format wybrać ?

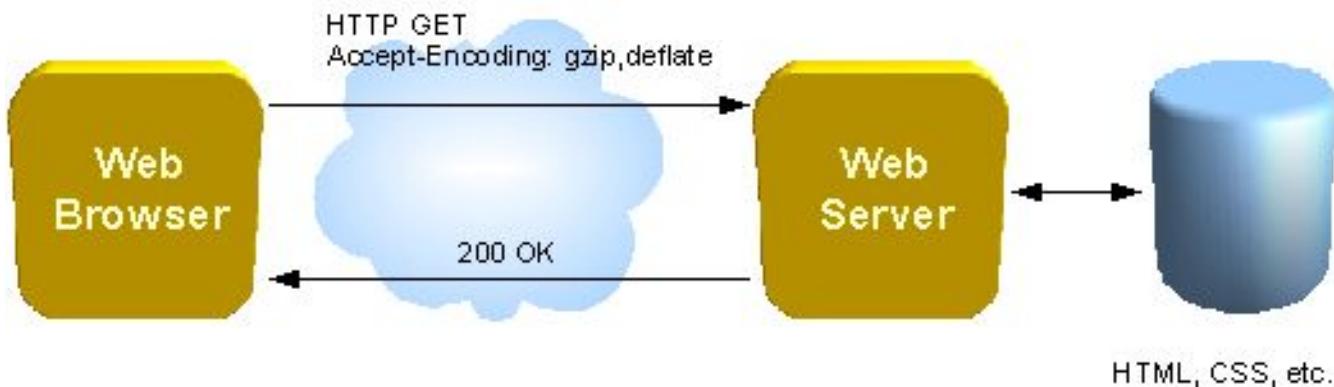
- każdy z nich ma wady i zalety, dlatego wybór zależy od tego do czego dany format ma służyć oraz od preferencji programisty.
- w przypadku JavaScriptu, lepiej wybrać JSONa, ponieważ łatwiej jest operować na obiektach JSONowych niż paroswać i XML.

Synchroniczne żądanie http



Jest to model synchronicznej komunikacji HTTP, gdzie klient wysyła żądanie, serwer je odbiera następnie przetwarza i na końcu generuje odpowiedź. W sytuacji takiej klient musi czekać z kolejnym żądaniem do momentu kiedy nie dostanie odpowiedzi od serwera.

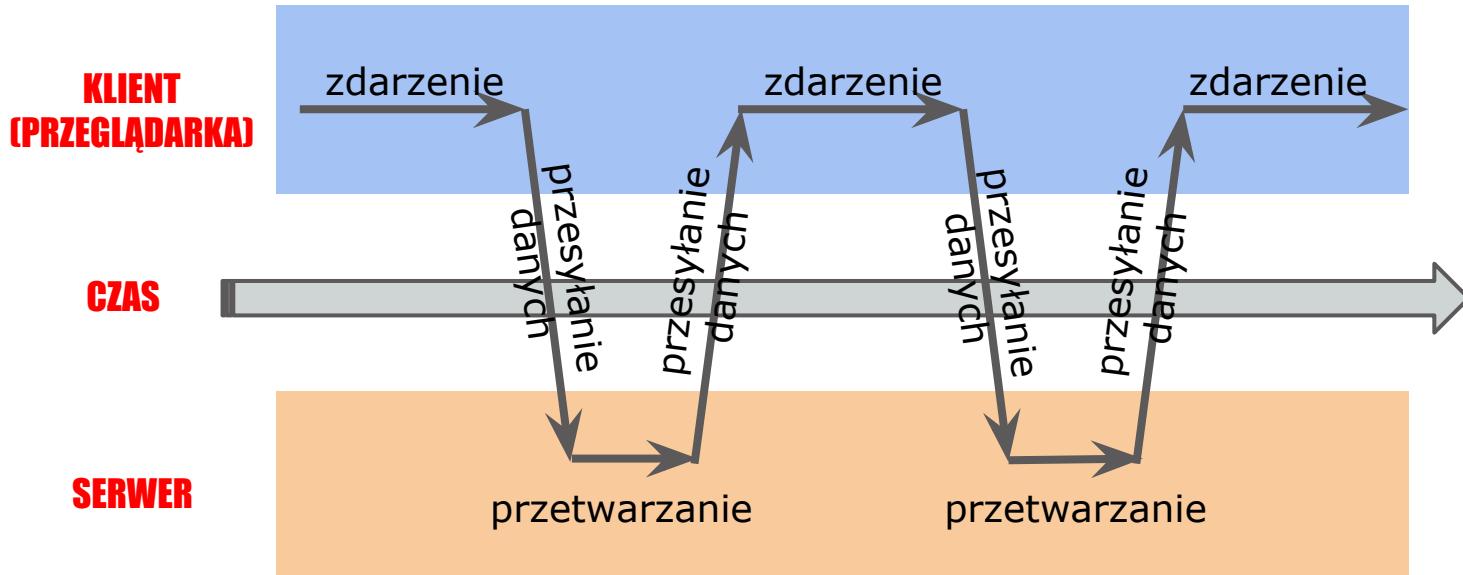
Synchroniczne żądanie http



Jest to model synchronicznej komunikacji HTTP, gdzie klient wysyła żądanie, serwer je odbiera następnie przetwarza i na końcu generuje odpowiedź. W sytuacji takiej klient musi czekać z kolejnym żądaniem do momentu kiedy nie dostanie odpowiedzi od serwera.

W modelu synchronicznym mamy bardzo mały poziom aktywności oraz interaktywności strony, strona musi być przeładowana (pobrać) po każdym żądaniu klienta, jeśli strony są złożone to proces ten jest długi.

Synchroniczne żądanie http



Jest to model synchronicznej komunikacji HTTP, gdzie klient wysyła żądanie, serwer je odbiera następnie przetwarza i na końcu generuje odpowiedź. W sytuacji takiej klient musi czekać z kolejnym żądaniem do momentu kiedy nie dostanie odpowiedzi od serwera.

W modelu synchronicznym mamy bardzo mały poziom aktywności oraz interaktywności strony, strona musi być przeładowana (pobrać) po każdym żądaniu klienta, jeśli strony są złożone to proces ten jest długi.

AJAX (“asynchroniczność”)

W klasycznych stronach (takich jak omawialiśmy do tej chwili) wszystkie żadania jakie były przekazywane przez protokół http (lub https) były synchroniczne. Oznaczało to, że strona po wykonaniu “działania” przez użytkownika musiał zostać przeładowana. Było to wygodne jednak nosło ze sobą konieczności “oczekiwania” na zakończenie poprzedniej operacji aby można było wysłać kolejne żądanie oraz niepotrzebnie generowała ruch sieciowy przez ciągłe przesyłanie tych samych plików.

AJAX (“asynchroniczność”)

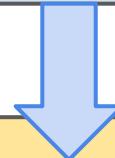
W klasycznych stronach (takich jak omawialiśmy do tej chwili) wszystkie żadania jakie były przekazywane przez protokół http (lub https) były synchroniczne. Oznaczało to, że strona po wykonaniu “działania” przez użytkownika musiał zostać przeładowana. Było to wygodne jednak nosło ze sobą konieczności “oczekiwania” na zakończenie poprzedniej operacji aby można było wysłać kolejne żądanie oraz niepotrzebnie generowała ruch sieciowy przez ciągłe przesyłanie tych samych plików.

Jak rozwiązać ten problem ??

AJAX (“asynchroniczność”)

W klasycznych stronach (takich jak omawialiśmy do tej chwili) wszystkie żadania jakie były przekazywane przez protokół http (lub https) były synchroniczne. Oznaczało to, że strona po wykonaniu “działania” przez użytkownika musiał zostać przeładowana. Było to wygodne jednak nosło ze sobą konieczności “oczekiwania” na zakończenie poprzedniej operacji aby można było wysłać kolejne żądanie oraz niepotrzebnie generowała ruch sieciowy przez ciągłe przesyłanie tych samych plików.

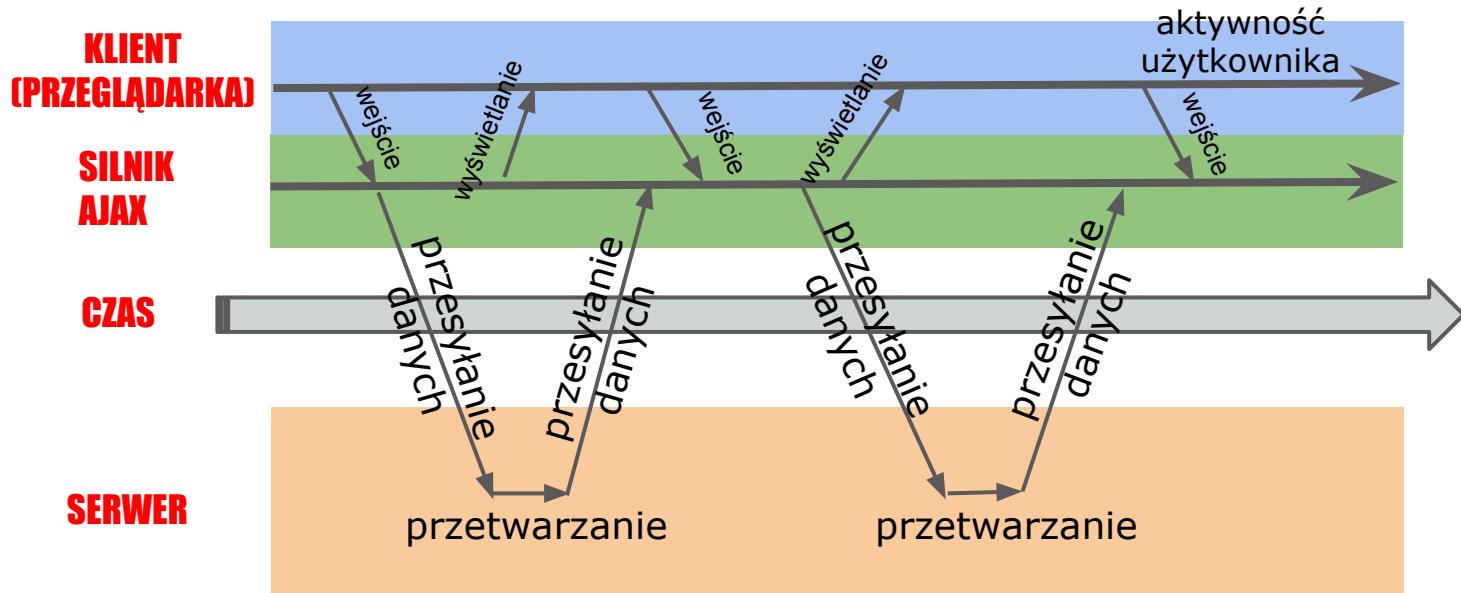
Jak rozwiązać ten problem ??



AJAX

(asynchroniczne przesyłanie danych przez protokół HTTP (HTTPS))

Asynchroniczne żądanie http



Jest to model asynchronicznej komunikacji HTTP, gdzie klient (przeglądarka) nie czeka na przyjście odpowiedzi na żądanie serwera, a wykonuje dalsze żądania.

W takim modelu nie ma konieczności przeładowania strony przy każdej operacji klineta, wystarczy, że zostaną doczytane brakujące dane, a dzięki odpowiednim narzędziom zmodyfikowana zostanie zawartość strony.

AJAX (asynchroniczność)

AJAX (Asynchronous JavaScript and XML) [Asynchroniczny JavaScript i XML] - model komunikacji sieciowej w której komunikacja pomiędzy klientem a serwerem odbywa się bez przeładowania dokumentu. Techniki służące do obsługi tej usługi są następujące:

1. **XMLHttpRequest** - klasa która umożliwiająca asynchroniczne przesyłanie danych pomiędzy klientem a serwerem.
2. **JavaScript** - język skryptowy pośredniczący w komunikacji.
3. **XML** - język znaczników który opisuje dane (w ogólności może to być dowolny format np. JSON).

Głównym zadaniem modelu AJAX jest otwarcie połączenia z pomiędzy klientem a serwerem.

AJAX (asynchroniczność)

Aby rozpocząć korzystanie z AJAX należy utworzyć obiekt klasy XMLHttpRequest():

```
var request = new XMLHttpRequest();
```

Od tego miejsca mamy dostęp do zestawu metod i właściwości przynależących do klasy XMLHttpRequest, które pozwalają na obsługę połączenia.

Zestaw metod:

- **open("metoda", "url", isAsynchronous, "user", "password")** - metoda otwiera połaczenie do serwera. Przyjmuje połaczenie parametry: "metoda" - definijemy GET czy POST, "url" - adres pliku na zdalnym serwerze, "isAsynchronous" - paramter true/false określa czy żądanie ma być asynchroniczne.
- **send("content")** - przesyłają żądanie do serwera.
- **abort()** - przerywa żądanie do serwera.
- **setRequestHeader()** - wysyła nagłówek do serwera.
- **getResponseHeader("nazwa_nagłówka")** - pobiera nagłówek z serwera
- **getAllResponseHeaders()** - pobiera wszystkie wysłane nagłówki http jako string.

AJAX (asynchroniczność)

Aby rozpocząć korzystanie z AJAX należy utworzyć obiekt klasy XMLHttpRequest():

```
var request = new XMLHttpRequest();
```

Od tego miejsca mamy dostęp do zestawu metod i właściwości przynależących do klasy XMLHttpRequest, które pozwalają na obsługę połączenia.

Zestaw właściwości:

- **onreadystatechange** - zdarzenie odpalane w chwili zmiany stanu danego połączenia
- **readyState** - zawiera aktualny status połączenia
- responseText** - zawiera zwrócone dane w formie tekstu
- **responseXML** - zawiera zwrócone dane w formie drzewa XML (jeżeli zwrócone dane są prawidłowym dokumentem XML)
- **status** - zwraca status połączenia np 404 - gdy strona nie istnieje, itp)
- **statusText** - zwraca status połączenia w formie tekstowej - np 404 zwróci Not Found

AJAX (asynchroniczność)

Przykład utworzenia żadania asynchronicznego metodą GET:

```
var request = new XMLHttpRequest();
request.open("GET", "/test/test.php", true);
request.send();
```

Parametr włączający
żądanie asynchroniczne

```
var request = new XMLHttpRequest();
request.open("POST", "/test/test.php", true);
request.setRequestHeader("Content-Type","text/xml");
request.send("<osoba>Jan Kowalski</osoba>");
```

AJAX (asynchroniczność)

Możemy również sprawdzać stan połączenia z serwerem za pomocą właściwości **readyState**:

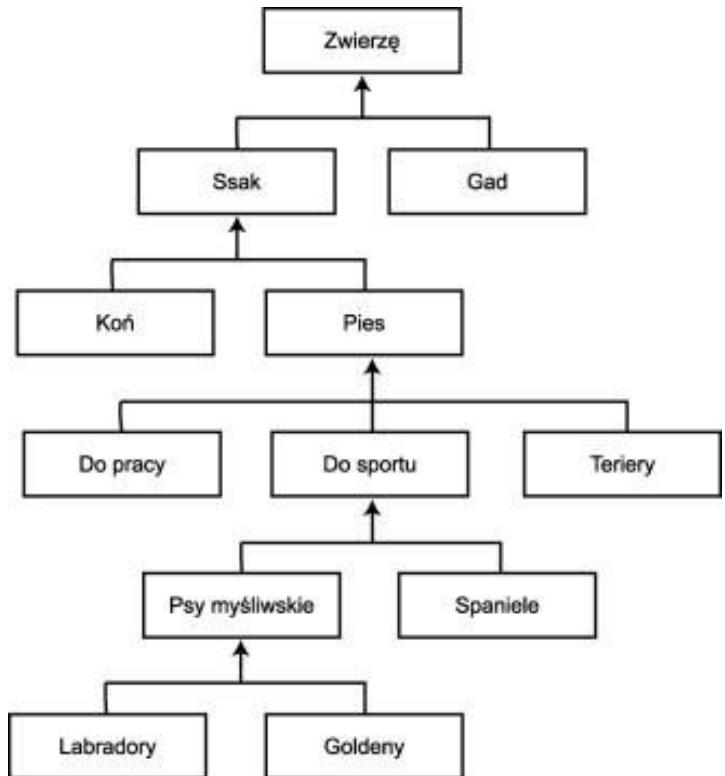
- 0: połączenie nie nawiązane,
- 1: połączenie nawiązane,
- 2: żądanie odebrane,
- 3: przetwarzanie,
- 4: dane zwrocone i gotowe do użycia

```
var request = new XMLHttpRequest();

request.open("GET", "/test/test.php", true);
request.onreadystatechange = function(){
    if ( request.readyState == 4 ) {
        request = null;
    }
};
request.send();
```

Dziedziczenie

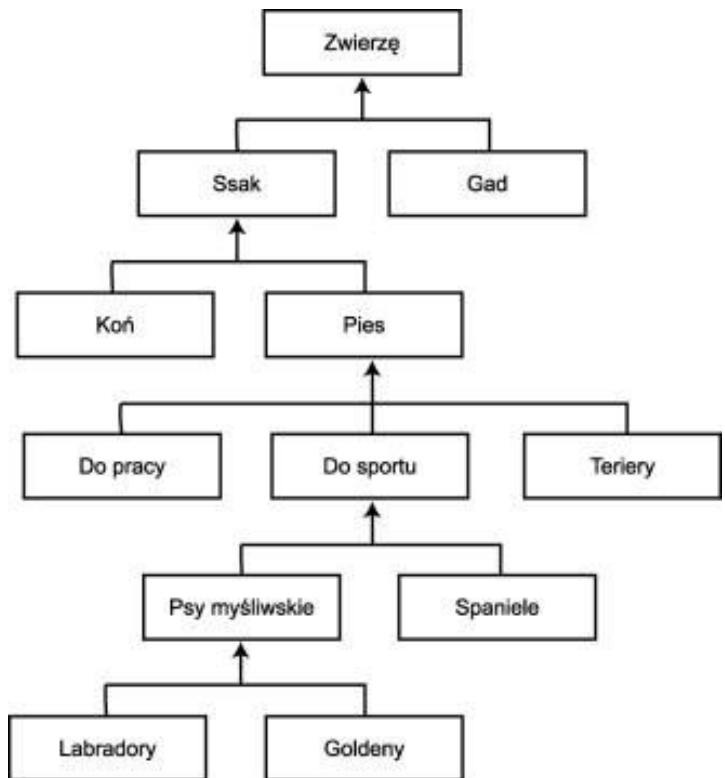
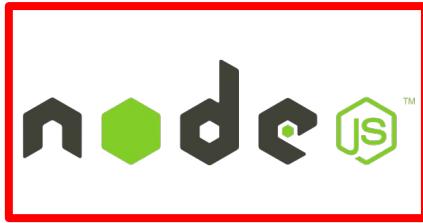
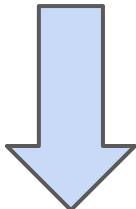
JavaScript jak już wielokrotnie wspominano jest językiem **bezklasowym** i nie przenosi typowego z języków kompilowalnych sposobu dziedziczenia.



Dziedziczenie

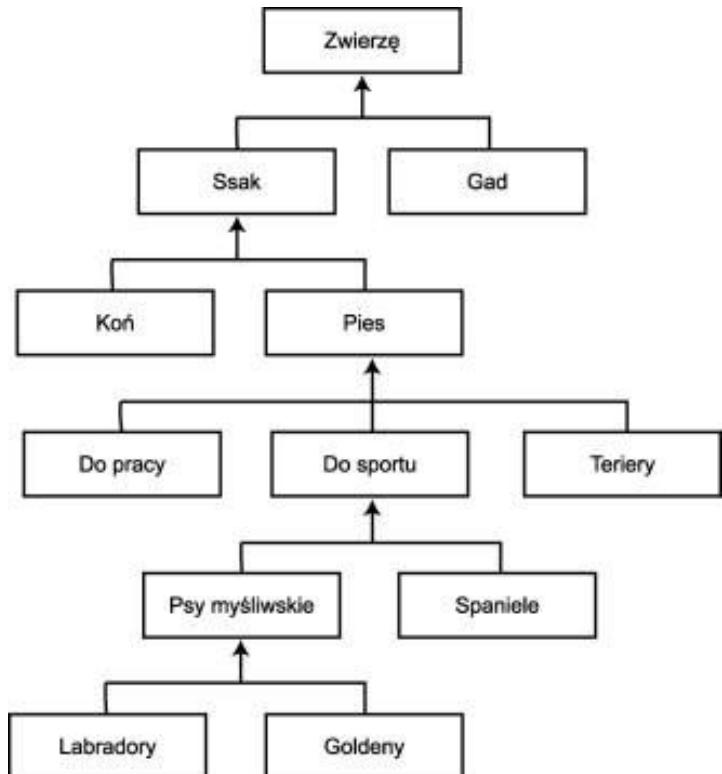
JavaScript jak już wielokrotnie wspominano jest językiem **bezklasowym** i nie przenosi typowego z języków kompilowalnych sposobu dziedziczenia.

Sposób dziedziczenia w JS
różni się od klasycznego
podejścia a tym samym i w ...



Dziedziczenie

Dziedziczenie w językach klasycznych ma przede wszystkim za zadanie umożliwiać wielokrotne wykorzystanie tego samego kodu: np. jeśli dwie klasy są podobne, należy określić tylko dzielące je różnice. Po drugie dziedziczenie ma w sobie specyfikację typów co pozwala na zmniejszenie konieczności rzutowania

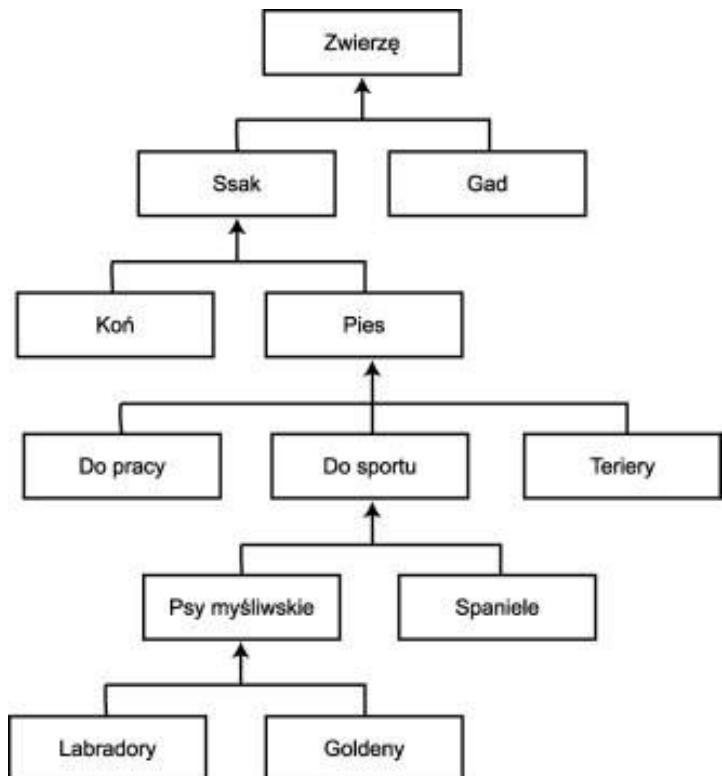


Dziedziczenie

Dziedziczenie w językach klasycznych ma przede wszystkim za zadanie umożliwiać wielokrotne wykorzystanie tego samego kodu: np. jeśli dwie klasy są podobne, należy określić tylko dzielące je różnice. Po drugie dziedziczenie ma w sobie specyfikację typów co pozwala na zmniejszenie konieczności rzutowania

JS nie posiada typów!

Co wtedy z dziedziczeniem ??





Dziedziczenie prototypowe w JS

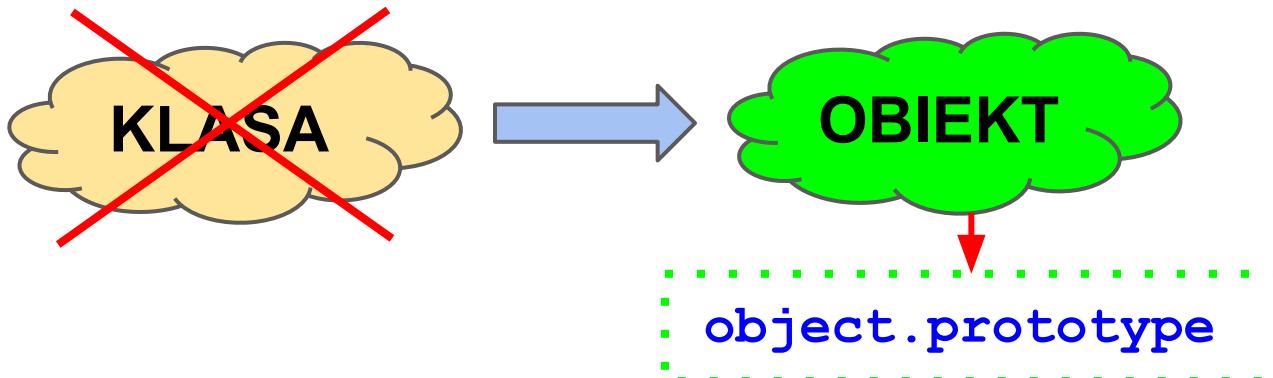


KLASA

Dziedziczenie prototypowe w JS

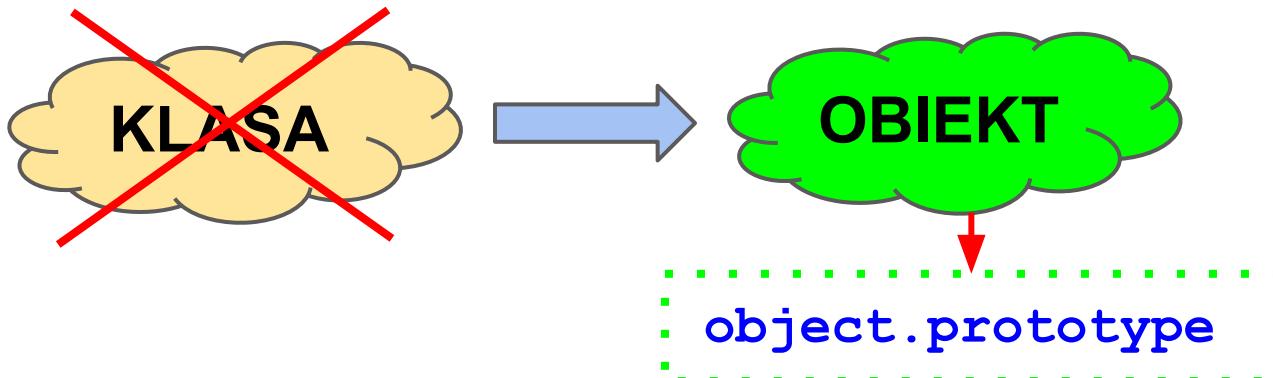


Dziedziczenie prototypowe w JS



W języku JavaScript (a tym samym we wszystkich środowiskach programistycznych opartych o ten język) dziedziczenie odbywa się w sposób prototypowy. Każdy obiekt w JavaScript można traktować jako prototyp który w dowolnym momencie można rozszerzać, a jego własności przekazywać (cedować) na inne dziedziczące obiekty.

Dziedziczenie prototypowe w JS



W języku JavaScript (a tym samym we wszystkich środowiskach programistycznych opartych o ten język) dziedziczenie odbywa się w sposób prototypowy. Każdy obiekt w JavaScript można traktować jako prototyp który w dowolnym momencie można rozszerzać, a jego własności przekazywać (cedować) na inne dziedziczące obiekty.

Obiekty dziedziczą po obiektach!

Dziedziczenie prototypowe w JS

Obiekty dziedziczą po obiektach!

`object.prototype`

Właściwość `prototype` posiadają obiekty które są tworzone przez funkcję konstruktora. Jednocześnie sam prototyp jest obiektem. Często dziedziczenie w JS określa się mianem dziedziczenia opartego o instancje.



Dziedziczenie prototypowe w JS

```
var Obiekt = function() {};  
var o = new Obiekt();  
console.log(o.czesc);
```



Dziedziczenie prototypowe w JS

```
var Obiekt = function() {};  
var o = new Obiekt();  
console.log(o.czesc);
```

> undefined



Dziedziczenie prototypowe w JS

Do zmiany lub rozszerzenia utworzonego obiektu możemy użyć właściwości prototype (który też jest obiektem):

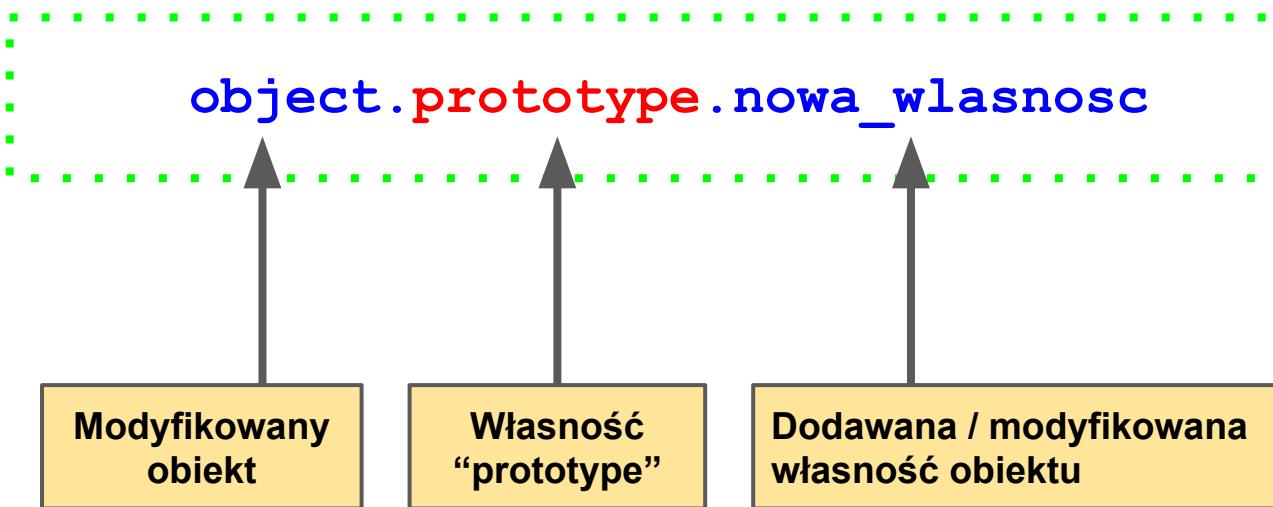
Dziedziczenie prototypowe w JS

Do zmiany lub rozszerzenia utworzonego obiektu możemy użyć właściwości prototype (który też jest obiektem):

```
object.prototype.nowa_wlasnosc
```

Dziedziczenie prototypowe w JS

Do zmiany lub rozszerzenia utworzonego obiektu możemy użyć właściwości prototype (który też jest obiektem):





Dziedziczenie prototypowe w JS

```
var Obiekt = function() {};  
var o = new Obiekt();  
console.log(o.czesc);  
  
Obiekt.prototype.czesc = 'czesc';  
console.log(o.czesc);
```

Dziedziczenie prototypowe w JS

```
var Obiekt = function() {};  
var o = new Obiekt();  
console.log(o.czesc);  
  
Obiekt.prototype.czesc = 'czesc';  
console.log(o.czesc);
```

```
> undefined  
czesc
```

Dziedziczenie prototypowe w JS

```
var Obiekt = function() {};  
var o = new Obiekt();  
console.log(o.czesc);  
  
Obiekt.prototype.czesc = 'czesc';  
console.log(o.czesc);  
  
var p = new Obiekt();  
console.log(p.czesc);
```

> undefined
czesc
czesc

Dziedziczenie prototypowe w JS

```
var Obiekt = function() {};  
var o = new Obiekt();  
console.log(o.czesc);  
  
Obiekt.prototype.czesc = 'czesc';  
console.log(o.czesc);  
  
var p = new Obiekt();  
console.log(p.czesc);  
  
Obiekt.prototype.PowiedzCzesc = function() {  
    console.log(this.czesc);  
}
```

Dziedziczenie prototypowe w JS

```
var Obiekt = function() {};  
var o = new Obiekt();  
console.log(o.czesc);  
  
Obiekt.prototype.czesc = 'czesc';  
console.log(o.czesc);  
  
var p = new Obiekt();  
console.log(p.czesc);  
  
Obiekt.prototype.PowedzCzesc = function() {  
    console.log(this.czesc);  
}  
o.PowedzCzesc();  
p.PowedzCzesc();
```

> undefined
czesc
czesc
czesc
czesc

Dziedziczenie prototypowe w JS

```
var Obiekt = function() {};  
var o = new Obiekt();  
console.log(o.czesc);
```



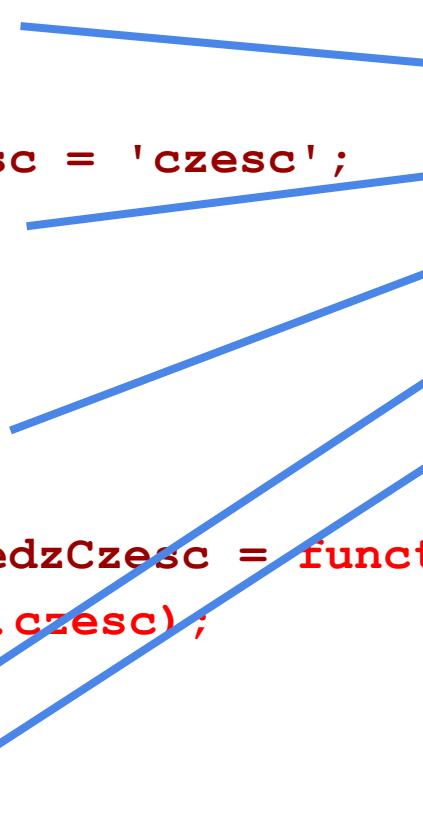
```
Obiekt.prototype.czesc = 'czesc';  
console.log(o.czesc);
```



```
var p = new Obiekt();  
console.log(p.czesc);
```



```
Obiekt.prototype.PowiedzCzesc = function() {  
    console.log(this.czesc);  
}  
o.PowiedzCzesc();  
p.PowiedzCzesc();
```



The diagram illustrates the prototype chain and object state. It shows four objects: o, p, Obiekt.prototype, and this. Arrows point from each object to its current value. The objects are represented by blue boxes:

- Object o: Value undefined
- Object p: Value czesc
- Object Obiekt.prototype: Value czesc
- Object this: Value czesc

Dziedziczenie prototypowe w JS

```
var Wspolrzedne = function(x,y){  
    this.x = x;  
    this.y = y;  
    this.dodaj = function(){  
        return this.x + this.y;  
    };  
};  
var f = new Wspolrzedne(4,7);
```

Dziedziczenie prototypowe w JS

```
var Wspolrzedne = function(x,y){  
    this.x = x;  
    this.y = y;  
    this.dodaj = function(){  
        return this.x + this.y;  
    };  
};  
var f = new Wspolrzedne(4,7);  
  
Wspolrzedne.prototype.pokazX = function(){  
    console.log(this.x);  
};  
Wspolrzedne.prototype.pokazY = function(){  
    console.log(this.y);  
};  
Wspolrzedne.prototype.pomnoz = function(){  
    return this.x * this.y;  
};
```

Dziedziczenie prototypowe w JS

```
var Wspolrzedne = function(x,y){  
    this.x = x;  
    this.y = y;  
    this.dodaj = function(){  
        return this.x + this.y;  
    };  
};  
var f = new Wspolrzedne(4,7);  
  
Wspolrzedne.prototype.pokazX = function(){  
    console.log(this.x);  
};  
Wspolrzedne.prototype.pokazY = function(){  
    console.log(this.y);  
};  
Wspolrzedne.prototype.pomnoz = function(){  
    return this.x * this.y;  
};  
Wspolrzedne.prototype.pokazWynikDodaj = function(){  
    console.log(this.dodaj());}  
Wspolrzedne.prototype.pokazWynikPomnoz = function(){  
    console.log(this.pomnoz());}
```

```
f.pokazX();  
f.pokazY();  
f.pokazWynikDodaj();  
f.pokazWynikPomnoz();
```

> 4
7
11
28

Dziedziczenie prototypowe w JS

```
function Mother() {  
    this.methodA = function() {  
        console.log('Mother::methodA()');  
    }  
    this.methodB = function() {  
        console.log('Mother::methodB()');  
    }  
}  
  
function Child() {  
    this.methodC = function () {  
        console.log('Child::methodC()');  
    }  
}
```

Dziedziczenie prototypowe w JS

```
function Mother() {  
    this.methodA = function() {  
        console.log('Mother::methodA()');  
    }  
    this.methodB = function() {  
        console.log('Mother::methodB()');  
    }  
}  
  
function Child() {  
    this.methodC = function () {  
        console.log('Child::methodC()');  
    }  
}  
  
Child.prototype = new Mother();  
  
m = new Mother();  
c = new Child();
```

Dziedziczenie prototypowe w JS

```
function Mother() {  
    this.methodA = function() {  
        console.log('Mother::methodA()');  
    }  
    this.methodB = function() {  
        console.log('Mother::methodB()');  
    }  
}  
  
function Child() {  
    this.methodC = function () {  
        console.log('Child::methodC()');  
    }  
}  
  
Child.prototype = new Mother();  
  
m = new Mother();  
c = new Child();
```

```
m.methodA();  
m.methodB();  
c.methodA();  
c.methodB();  
c.methodC();
```

> **Mother::methodA()**
Mother::methodB()
Mother::methodA()
Mother::methodB()
Child::methodC()



Dziedziczenie prototypowe w JS

Możemy też bez żadnych problemów “przeciążyć” metodę B zdefiniowaną w funkcji “Child”, oddziedziczoną po funkcji “Mother”:

```
Child.prototype.methodB = function () {
    console.log('Child::methodB()');

}

m.methodA();
m.methodB();
c.methodA();
c.methodB();
c.methodC();
```

Dziedziczenie prototypowe w JS

Możemy też bez żadnych problemów “przeciążyć” metodę B zdefiniowaną w funkcji “Child”, oddziedziczoną po funkcji “Mother”:

```
Child.prototype.methodB = function () {  
    console.log('Child::methodB()');  
}  
  
m.methodA();  
m.methodB();  
c.methodA();  
c.methodB();  
c.methodC();
```

> Mother::methodA()
Mother::methodB()
Mother::methodA()
Child::methodB()
Child::methodC()



KONIEC WYKŁADU 6



UNIWERSYTET
JAGIELŁOŃSKI
W KRAKOWIE

Zaawansowane Techniki WWW (HTML, CSS i JavaScript)

Dr inż. Marcin Zieliński

Środa 15:30 - 17:00 sala: A-1-04

WYKŁAD 7

Wykład dla kierunku: Informatyka Stosowana II rok

Rok akademicki: 2015/2016 - semestr zimowy

Przypomnienie z poprzedniego wykładu

JavaScript jako język zdarzeniowy

Format JSON vs. XML

Asynchroniczność w JS

Dziedziczenie prototypowe w JS

Obiekty i zdarzenia na stronie

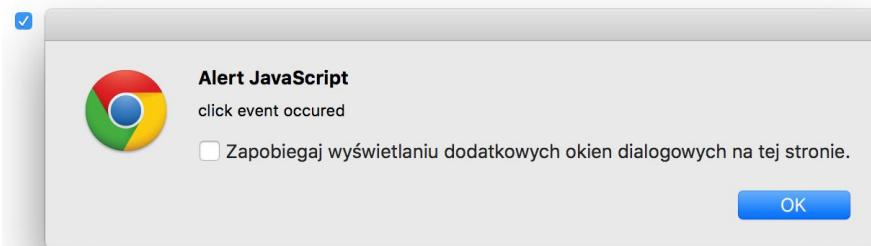
Przykład emulowania zdarzenia "click":

```
<form>
  <input type="checkbox" id="myCheck" onmouseover="myFunction()" onclick="
    alert('click event occurred')">
</form>

<script>
function myFunction() {
  document.getElementById("myCheck").click();
}
</script>
```



Testuje emulacje zdarzenia "click" po najechaniu na "checkboxa" kursorem myszy:.



Porównanie JSON vs. XML

```
{ "samochod": [ { "Marka": "VW", "Model": "Golf", "Rocznik": 1999 }, { "Marka": "BMW", "Model": "S6", "Rocznik": 2007 }, { "Marka": "Audi", "Model": "A4", "Rocznik": 2009 } ] }
```

```
<?xml version="1.0" encoding="utf-8"?>
<KOMIS>
  <Samochod>
    <Marka>VW</Marka>
    <Model>Golf</Model>
    <Rok>1999</Rok>
  </Samochod>
  <Samochod>
    <Marka>BMW</Marka>
    <Model>S6</Model>
    <Rok>2007</Rok>
  </Samochod>
  <Samochod>
    <Marka>Audi</Marka>
    <Model>A3</Model>
    <Rok>2009</Rok>
  </Samochod>
</KOMIS>
```

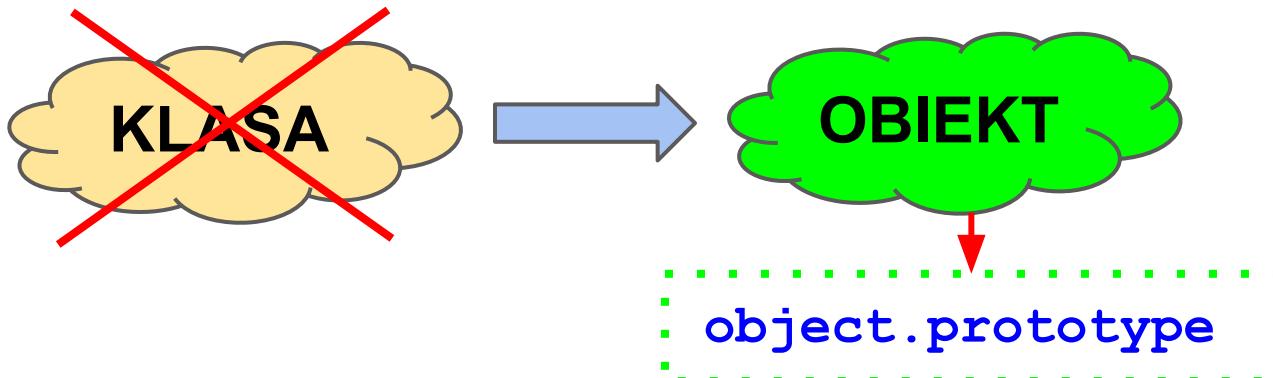
AJAX (asynchroniczność)

AJAX (Asynchronous JavaScript and XML) [Asynchroniczny JavaScript i XML] - model komunikacji sieciowej w której komunikacja pomiędzy klientem a serwerem odbywa się bez przeładowania dokumentu. Techniki służące do obsługi tej usługi są następujące:

1. **XMLHttpRequest** - klasa która umożliwiająca asynchroniczne przesyłanie danych pomiędzy klientem a serwerem.
2. **JavaScript** - język skryptowy pośredniczący w komunikacji.
3. **XML** - język znaczników który opisuje dane (w ogólności może to być dowolny format np. JSON).

Głównym zadaniem modelu AJAX jest otwarcie połączenia z pomiędzy klientem a serwerem.

Dziedziczenie prototypowe w JS

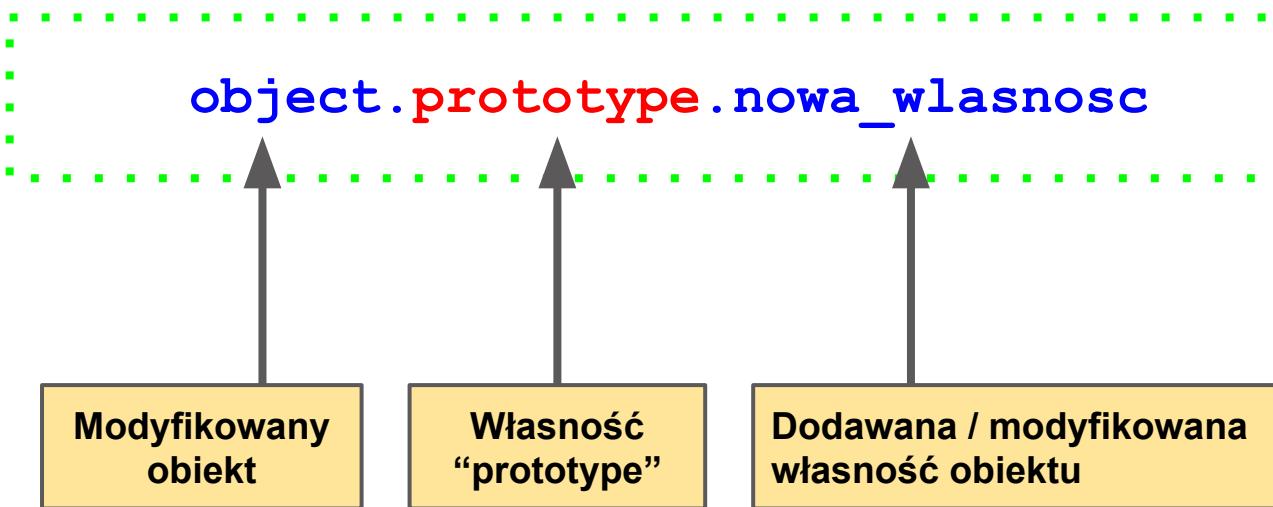


W języku JavaScript (a tym samym we wszystkich środowiskach programistycznych opartych o ten język) dziedziczenie odbywa się w sposób prototypowy. Każdy obiekt w JavaScript można traktować jako prototyp który w dowolnym momencie można rozszerzać, a jego własności przekazywać (cedować) na inne dziedziczące obiekty.

Obiekty dziedziczą po obiektach!

Dziedziczenie prototypowe w JS

Do zmiany lub rozszerzenia utworzonego obiektu możemy użyć właściwości `prototype` (który też jest obiektem):



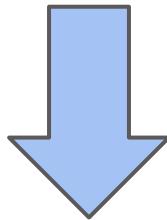


UNIWERSYTET
JAGIELLOŃSKI
W KRAKOWIE

Wprowadzenie do jQuery



Wprowadzenie do jQuery



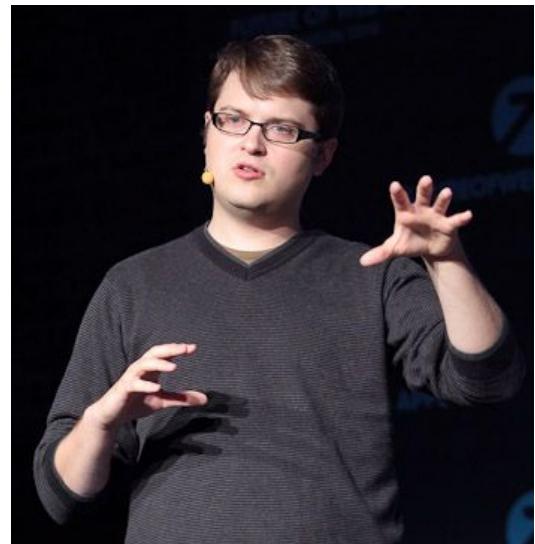
<http://www.jquery.com/>

Wprowadzenie do jQuery

Biblioteka napisana w języku JavaScript o “lekkim” charakterze, obsługująca przestrzeń nazw z mechanizmem łatwej rozszerzalności umożliwiającą manipulowanie elementami struktury DOM (Document Object Model).

Biblioteka jest dostępna na licencji: GPL i MIT.

Pierwsze wydanie biblioteki w wersji miało miejsce 26 sierpnia 2006 roku



John Resig

Wprowadzenie do jQuery

Biblioteka napisana w języku JavaScript o “lekkim” charakterze, obsługująca przestrzeń nazw z mechanizmem łatwej rozszerzalności umożliwiającą manipulowanie elementami struktury DOM (Document Object Model).

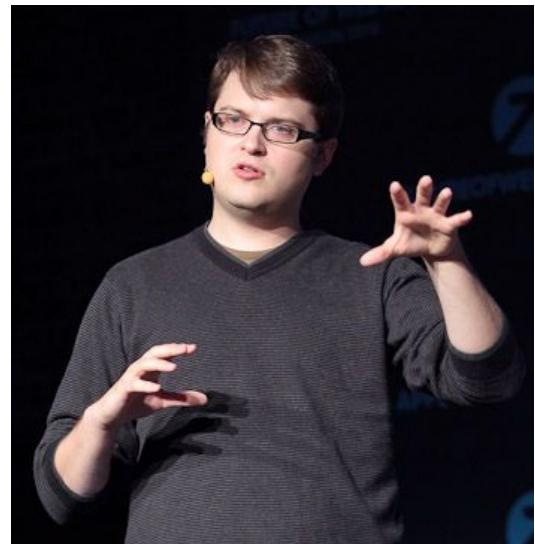
Biblioteka jest dostępna na licencji: GPL i MIT.

Pierwsze wydanie biblioteki w wersji miało miejsce 26 sierpnia 2006 roku

Najnowsze wydanie stabilne:

jQuery 2.1.4 (z 28 kwietnia 2015 r.)

jQuery 1.11.3 (z 28 kwietnia 2015 r.)



John Resig

Wprowadzenie do jQuery

Biblioteka napisana w języku JavaScript o “lekkim” charakterze, obsługująca przestrzeń nazw z mechanizmem łatwej rozszerzalności umożliwiającą manipulowanie elementami struktury DOM (Document Object Model).

Biblioteka jest dostępna na licencji: GPL i MIT.

Pierwsze wydanie biblioteki w wersji miało miejsce 26 sierpnia 2006 roku

Najnowsze wydanie stabilne:

jQuery 2.1.4 (z 28 kwietnia 2015 r.)

jQuery 1.11.3 (z 28 kwietnia 2015 r.)

Uwaga od wersji 2.X.X brak wsparcia dla IE 6 - 8 (redukcja rozmiaru pliku biblioteki)



John Resig



Wprowadzenie do jQuery

Najnowsze wydanie stabilne:

jQuery 2.1.4 (z 28 kwietnia 2015 r.)



Biblioteka jQuery występuje w dwóch wersjach:

Wprowadzenie do jQuery

Najnowsze wydanie stabilne:



jQuery 2.1.4 (z 28 kwietnia 2015 r.)

Biblioteka jQuery występuje w dwóch wersjach:



Zwykłej:
jquery-2.1.4.js

rozmiar: 82.4 KB

Czytelny kod, struktura
hierarchiczna, komentarze

Spakowanej:
jquery-2.1.4.min.js

rozmiar: 27.7 KB

Brak czytelności, wszystko w
jednej linii, brak komentarzy



Wprowadzenie do jQuery

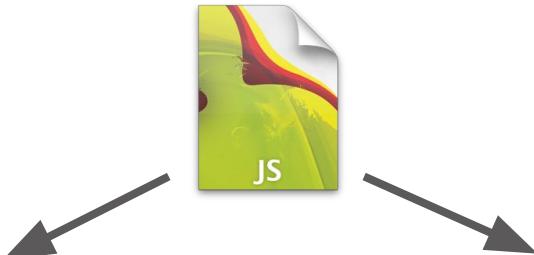
Najnowsze wydanie stabilne:



jQuery 2.1.4 (z 28 kwietnia 2015 r.)

Biblioteka jQuery występuje w dwóch wersjach:

jquery-2.1.4.js



jquery-2.1.4.min.js

```
(function( global, factory ) {  
  
    if ( typeof module === "object" && typeof module.exports === "object" ) {  
        // For CommonJS and CommonJS-like environments where a proper window  
        // is present, execute the factory and get jQuery.  
        // For environments that do not have a 'window' with a 'document'  
        // (such as Node.js), expose a factory as module.exports.  
        // This accentuates the need for the creation of a real 'window'.  
        // e.g. var jquery = require("jquery")(window);  
        // See ticket #14549 for more info.  
        module.exports = global.document ?  
            factory( global, true ) :  
            function( w ) {  
                if ( !w.document ) {  
                    throw new Error( "jQuery requires a window" );  
                }  
                return factory( w );  
            };  
    } else {  
        factory( global );  
    }  
  
    // Pass this if window is not defined yet  
    // (typeof window !== "undefined" ? window : this, function( window, noGlobal ) {  
  
        // Support: Firefox 18+  
        // Can't be in strict mode, several libs including ASP.NET trace  
        // the stack via arguments.caller.callee and Firefox dies if  
    })  
})
```

```
/*! jquery v2.1.4 | (c) 2005, 2015 jquery Foundation, Inc. | jquery!function(a,b){"object"==typeof module&&"object"==typeof module.exports&&"undefined"!=c.slice,e=c.concat,f=c.push,g=c.indexOf,h={},i=h.toString,j=h["\s|\uFFEF\xA0]+|[ \s|\uFFEF\xA0]+$/g,p=/^ms-/i,q=/([da-z])/gi,r=fu[jquery:m,constructor:n,selector:"",length:0,toArray:function(){(this).pushStack:(function(a){var b=n.merge(this.constructor(),a);(this,a,b)},map:function(a){return this.pushStack(n.map(this,function(this,arguments)),first:function(){return this.eq(0)},last:function(){return this.pushStack(c=0&&b>c?this[c]:[])}},end:function(){return this(null)},push:f,sort:c.sort,splice:c.splice},n.extend=n.fn.extend=f("boolean"==typeof g&&(j=g,g.arguments[h][i],h++),"object"==typeof a)c=g[b],d=a[b],g!==d&&(j&&d&&(n.isPlainObject(d))||(e=n.isArray(d)(j,f,d)):void 0!==d&&(g[b]=d));return g},n.extend({expando:"jQuery"},a),noop:function(){},isFunction:function(a){return"function"===(a),type:function(a){return null==a?a+"":"object"==typeof a?"object":typeof a},typeOf:function(a){return null==a?a+"":"object"==typeof a?"object":typeof a},eval:a.n.trim(a),a&&(i==a.indexof("use strict")?(b=l.createEl(a)),camelCase:function(a){return a.replace(p,"ms-").replace(q,r)
```



Wprowadzenie do jQuery

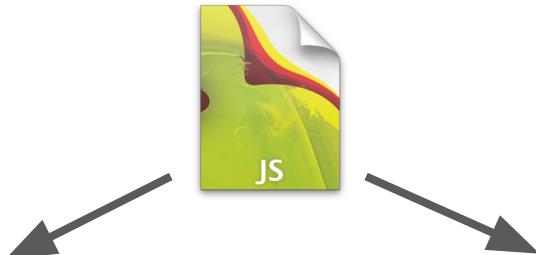
Najnowsze wydanie stabilne:



jQuery 2.1.4 (z 28 kwietnia 2015 r.)

Biblioteka jQuery występuje w dwóch wersjach:

jquery-2.1.4.js



jquery-2.1.4.min.js

```
(function( global, factory ) {  
  if ( typeof module === "object" && typeof module.exports === "object" ) {  
    // For CommonJS and CommonJS-like environments where a proper window  
    // is present, execute the factory and get jQuery.  
    // For environments that do not have a 'window' with a 'document'  
    // (such as Node.js), expose a factory as module.exports.  
    // This accentuates the need for the creation of a real 'window'.  
    // CommonJS users should not assign window to global in their environment.  
    module.exports = factory;  
  } else {  
    factory;  
  }  
}  
  
// Pass this if window is not defined yet  
}(typeof window !== "undefined" ? window : this, function( window, noGlobal ) {  
  
  // Support: Firefox 18+  
  // Can't be in strict mode, several libs including ASP.NET trace  
  // the stack via arguments.caller.callee and Firefox dies if  
  // you try to access the stack when strict mode is enabled.  
  // See https://github.com/mannat/jquery/tree/v1.5.1#test2  
  // See https://github.com/mannat/node/tree/v0.6.15#test
```

**Dla optymalizacji działania aplikacji
zaleca się używanie wersji “min”.**

```
/*! jquery v2.1.4 | (c) 2005, 2015 jquery Foundation, Inc. | jquery  
function(a,b){"object"==typeof module&&"object"==typeof module.ex  
requires a window with a document");return b(a);b(a).{"undefined"  
[],d=c.slice,e=c.concat,f=c.push,g=c.indexOf,h={},i=h.toString,j=h  
["\s\uFFEF\xA0]+|[ \s\uFFEF\xA0]+\$/g,p=/^ms-/q=-([\da-z])/gi,r=fu  
[jquery:m,constructor:n,selector:"",length:0,toArray:function(){re  
tructor(),a);  
ap(this,funct  
,last:functi  
{return this  
n.fn.extend=f  
object"==typeo  
n.isArray(d  
ando:"jQuery  
function"==n  
=a&&a==a.window},isNumeric:function(a){return!n.isArray(a)&&a.par  
n.isWindow(a)?1:a.constructor&&!j.call(a.constructor.prototype,"i  
0"},type:function(a){return null==a?a+"":"object"==typeof a|"funct  
b,c=eval;a=n.trim(a),a&&(1==a.indexOf("use strict")?(b=l.createEl  
()):c=eval(a)),camelCase:function(a){return a.replace(p,"ms-").replace(q,r)
```



Pobranie i instalacja jQuery

Pobieramy plik w najnowszej wersji ze strony:

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



Pobranie i instalacja jQuery

Pobieramy plik w najnowszej wersji ze strony:

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



[jquery-2.1.4.js](#)
[jquery-2.1.4.min.js](#)

```
<!DOCTYPE html>
<html>
<head>
<title>Pierwsza strona z jQuery</title>
<script type="text/javascript" src="jquery-2.1.4.min.js"></script>
</head>
<body>
<h1>Testujemy bibliotekę jQuery</h1>
</body>
</html>
```

Pobranie i instalacja jQuery

Pobieramy plik w najnowszej wersji ze strony:

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



jquery-2.1.4.js
jquery-2.1.4.min.js

```
>----->
[<!DOCTYPE html>
[<html>
[<head>
[<title>Pierwsza strona z jQuery</title>
[<script type="text/javascript" src="jquery-2.1.4.min.js"></script>
[</head>
[<body>
[<h1>Testujemy bibliotekę jQuery</h1>
[</body>
[</html>
[>----->
```

Od tego momentu mamy dostęp do wszystkich funkcjonalności biblioteki jQuery

Pobranie i instalacja jQuery

Pobieramy plik w najnowszej wersji ze strony:

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



jquery-2.1.4.js

jquery-2.1.4.min.js



```
<!DOCTYPE html>
<html>
<head>
<title>Pierwsza strona z jQuery</title>
<script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script>
</head>
<body>
<h1>Testujemy bibliotekę jQuery</h1>
</body>
</html>
```

W ten sposób odwołujemy się do zewnętrznego źródła biblioteki jQuery.

Jakie to ma zalety ??

Od tego momentu mamy dostęp do wszystkich funkcjonalności biblioteki jQuery

Jak używać jQuery



Kiedy mamy już dołączoną bibliotekę jQuery do naszej aplikacji możemy wykorzystywać wbudowane funkcjonalności.

```
<!DOCTYPE html>
<html>
<head>
<title>Pierwsza strona z jQuery</title>
<script type="text/javascript" src="jquery.2.4.1.min.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    alert('Pierwsza operacja jQuery');
});
</script>
</head>
<body>
<h1>Testujemy bibliotekę jQuery</h1>
</body>
</html>
```

Jak używać jQuery



Kiedy mamy już dołączoną bibliotekę jQuery do naszej aplikacji możemy wykorzystywać wbudowane funkcjonalności.

```
<!DOCTYPE html>
<html>
<head>
<title>Pierwsza strona z jQuery</title>
<script type="text/javascript" src="jquery.2.4.1.min.js"></script>
<script type="text/javascript">
    $(document).ready(function(){
        alert('Pierwsza operacja jQuery');
    });
</script>
</head>
<body>
<h1>Testujemy bibliotekę jQuery</h1>
</body>
</html>
```

```
jQuery(document).ready(function(){
    alert('Pierwsza operacja jQuery');
});
```

Jak używać jQuery



Kiedy mamy już dołączoną bibliotekę jQuery do naszej aplikacji możemy wykorzystywać wbudowane funkcjonalności.

```
<!DOCTYPE html>
<html>
<head>
<title>Pierwsza strona z jQuery</title>
<script type="text/javascript" src="jquery.2.4.1.min.js"></script>
<script type="text/javascript">
    $(document).ready(function(){
        alert('Pierwsza operacja jQuery');
    });
</script>
</head>
<body>
<h1>Testujemy bibliotekę jQuery</h1>
</body>
</html>
```

```
jQuery(document).ready(function(){
    alert('Pierwsza operacja jQuery');
});
```

Konstrukcja ta oznacza dostęp do zdefiniowanej przestrzeni nazw biblioteki:

\$()

(wersja skrócona)

Jak używać jQuery



Kiedy mamy już dołączoną bibliotekę jQuery do naszej aplikacji możemy wykorzystywać wbudowane funkcjonalności.

```
<!DOCTYPE html>
<html>
<head>
<title>Pierwsza strona z jQuery</title>
<script type="text/javascript" src="jquery.2.4.1.min.js"></script>
<script type="text/javascript">
    $(document).ready(function(){
        alert('Pierwsza operacja jQuery');
    });
</script>
</head>
<body>
<h1>Testujemy bibliotekę jQuery</h1>
</body>
</html>
```

```
jQuery(document).ready(function(){
    alert('Pierwsza operacja jQuery');
});
```

Konstrukcja ta oznacza dostęp do zdefiniowanej przestrzeni nazw biblioteki:

\$()

(wersja skrócona)

=

jQuery()

(wersja pełna)

Jak używać jQuery



`$()`

`=`

`jQuery()`

```
<script type="text/javascript">
    $(document).ready(function(){
        alert('Pierwsza operacja jQuery');
    });
</script>
```

Jak używać jQuery



`\$()`

`=`

`jQuery()`

```
<script type="text/javascript">
    $(document).ready(function(){
        alert('Pierwsza operacja jQuery');
    });
</script>
```

Co robi ta funkcja ?

Jak używać jQuery



`$()`

`=`

`jQuery()`

```
<script type="text/javascript">
    $(document).ready(function(){
        alert('Pierwsza operacja jQuery');
    });
</script>
```

Co robi ta funkcja ?

W wyniku zadziałania funkcji `$()` tworzony jest obiekt, który posiada metodę `ready()`, a argumentem tej metody jest wywołanie zwrotne (funkcja nienazwana), która wykonuje określone działanie.

Jak używać jQuery



`$()`

`=`

`jQuery()`

```
<script type="text/javascript">
    $(document).ready(function(){
        alert('Pierwsza operacja jQuery');
    });
</script>
```

Co robi ta funkcja ?

W wyniku zadziałania funkcji `$()` tworzony jest obiekt, który posiada metodę `ready()`, a argumentem tej metody jest wywołanie zwrotne (funkcja nienazwana), która wykonuje określone działanie.

Co robi metoda `ready()` oraz czym jest argument `document` ?

Jak używać jQuery



Co robi metoda **ready()** oraz czym jest argument **document** ?

Metoda **ready()**, zachodzi wtedy kiedy zajdzie zdarzenie “ready”, czyli cały (kompletny) dokument hipertekstowy zostanie załadowany, natomiast argument “**document**” symbolizuje ten dokument.

```
<script type="text/javascript">
    $(document).ready(function() {
        alert('Pierwsza operacja jQuery');
    });
</script>
```

Jak używać jQuery



Co robi metoda **ready()** oraz czym jest argument **document** ?

Metoda **ready()**, zachodzi wtedy kiedy zajdzie zdarzenie “ready”, czyli cały (kompletny) dokument hipertekstowy zostanie załadowany, natomiast argument “**document**” symbolizuje ten dokument.

```
<script type="text/javascript">
    $(document).ready(function() {
        alert('Pierwsza operacja jQuery');
    });
</script>
```

Czy da się w związku z tym jeszcze bardziej uprościć ten zapis ?

Jak używać jQuery



Co robi metoda **ready()** oraz czym jest argument **document** ?

Metoda **ready()**, zachodzi wtedy kiedy zajdzie zdarzenie “ready”, czyli cały (kompletny) dokument hipertekstowy zostanie załadowany, natomiast argument “**document**” symbolizuje ten dokument.

```
<script type="text/javascript">
    $(document).ready(function() {
        alert('Pierwsza operacja jQuery');
    });
</script>
```

Czy da się w związku z tym jeszcze bardziej uprościć ten zapis ?

TAK

```
<script type="text/javascript">
    $(function() {
        alert('Pierwsza operacja jQuery');
    });
</script>
```

Jak używać jQuery



Co robi metoda **ready()** oraz czym jest argument **document** ?

Metoda **ready()**, zachodzi wtedy kiedy zajdzie zdarzenie “ready”, czyli cały (kompletny) dokument hipertekstowy zostanie załadowany, natomiast argument “**document**” symbolizuje ten dokument.

```
<script type="text/javascript">
    $(document).ready(function() {
        alert('Pierwsza operacja jQuery');
    });
</script>
```

Czy da się w związku z tym jeszcze bardziej uprościć ten zapis ?

TAK

Dlaczego ?

```
<script type="text/javascript">
    $(function() {
        alert('Pierwsza operacja jQuery');
    });
</script>
```



jQuery i selektory



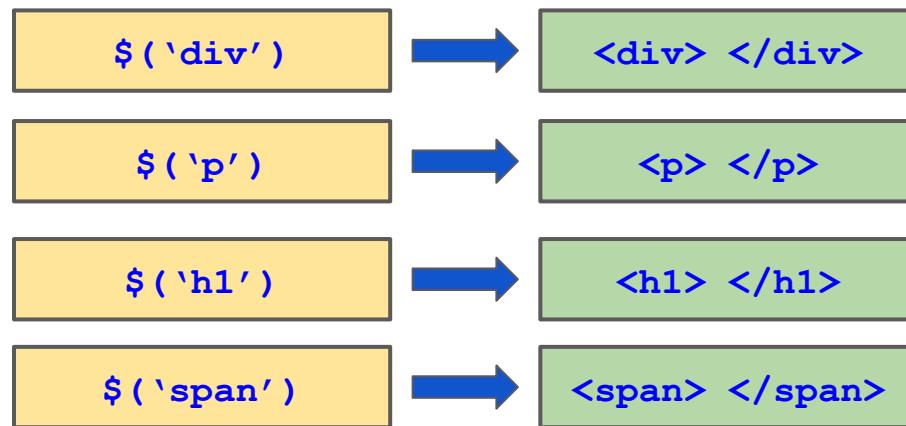
Jedną z podstawowych zalet biblioteki jQuery jest to, że umożliwia manipulowanie wybranymi elementami DOM, wykorzystując do tego celu selektory - tak jak jest robione w przypadku arkuszy CSS.

jQuery i selektory



Jedną z podstawowych zalet biblioteki jQuery jest to, że umożliwia manipulowanie wybranymi elementami DOM, wykorzystując do tego celu selektory - tak jak jest robione w przypadku arkuszy CSS.

Przykład selektora:

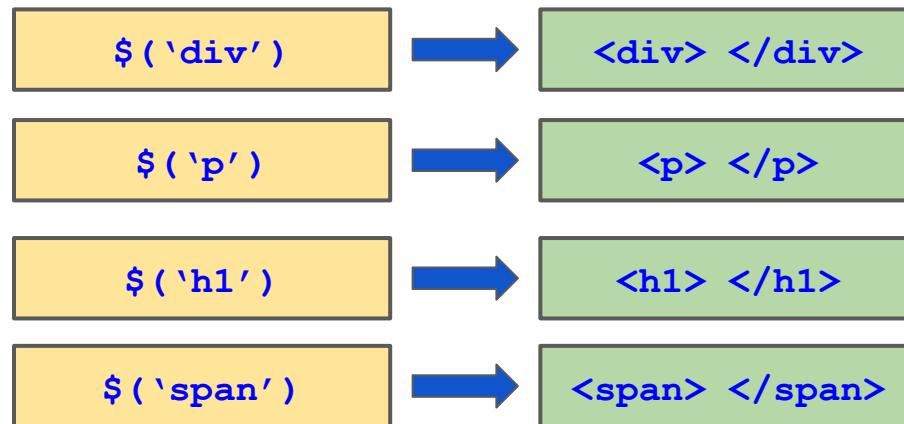


jQuery i selektory



Jedną z podstawowych zalet biblioteki jQuery jest to, że umożliwia manipulowanie wybranymi elementami DOM, wykorzystując do tego celu selektory - tak jak jest robione w przypadku arkuszy CSS.

Przykład selektora:



Selektorami tak jak w CSS mogą być klasy i identyfikatory:



jQuery i selektory



W jQuery obowiązują również, wszystkie reguły dziedziczenia, łączenia selektorów oraz używania selektorów atrybutów:

```
$('tresc h2')
```



```
<div id="tresc">  
  <h2> Tytuł treści </h2>  
</div>
```

jQuery i selektory



W jQuery obowiązują również, wszystkie reguły dziedziczenia, łączenia selektorów oraz używania selektorów atrybutów:

```
$('tresc h2')
```



```
<div id="tresc">  
  <h2> Tytuł treści </h2>  
</div>
```

```
<script type="text/javascript">  
  $(function(){  
    $('tresc h2').click(function(){  
      alert("Klikam w h2");  
    });  
  });  
</script>
```

Zdarzenie zostanie obsłużone tylko wtedy gdy znacznik `<h2>` jest potomkiem znacznika `<div>` o identyfikatorze "tresc".

jQuery i selektory



W jQuery obowiązują również, wszystkie reguły dziedziczenia, łączenia selektorów oraz używania selektorów atrybutów:

```
$('tresc h2')
```



```
<div id="tresc">  
  <h2> Tytuł treści </h2>  
</div>
```

```
<script type="text/javascript">  
  $(function(){  
    $('tresc h2').click(function(){  
      alert("Klikam w h2");  
    });  
  });  
</script>  
  
<script type="text/javascript">  
  $(function(){  
    $('tresc').find('h2').click(function(){  
      alert("Klikam w h2");  
    });  
  });  
</script>
```

Zdarzenie zostanie obsłużone tylko wtedy gdy znacznik `<h2>` jest potomkiem znacznika `<div>` o identyfikatorze "tresc".

Elementy można wybierać również, za pomocą metod wbudowanych takich jak "find()"

jQuery i selektory



W jQuery obowiązują również, wszystkie reguły dziedziczenia, łączenia selektorów oraz używania selektorów atrybutów:

```
$(`a[href="www.uj.edu.pl"]'')
```



```
<div id="tresc">
  <a href="www.uj.edu.pl"> Link do
  strony UJ</h2>
</div>
```

jQuery i selektory



W jQuery obowiązują również, wszystkie reguły dziedziczenia, łączenia selektorów oraz używania selektorów atrybutów:

```
$(`a[href="www.uj.edu.pl"]'')
```



```
<div id="tresc">
  <a href="www.uj.edu.pl"> Link do
  strony UJ</h2>
</div>
```

```
.....
<script type="text/javascript">
  $(function(){
    $('a[href="www.uj.edu.pl"]').click(function(){
      alert("Klikam w link do strony UJ");
      return false;
    });
  });
</script>
.

.

<div id="tresc">
  <a href="www.uj.edu.pl"> Link do strony UJ</h2>
</div>
.....
```



jQuery i zdarzenia



Biblioteka jQuery potrafi również operować na zdarzeniach i funkcjach obsługi zdarzeń. Przypomnijmy typy najpopularniejszych zdarzeń:

jQuery i zdarzenia



Biblioteka jQuery potrafi również operować na zdarzeniach i funkcjach obsługi zdarzeń. Przypomnijmy typy najpopularniejszych zdarzeń:

Fun. obsługi zdarzenia	Metoda jQuery	Opis
onclick	click()	Kliknięcie elementu
dblclick	dblclick()	Podwójne kliknięcie elem.
onmousedown	mousedown()	Naciśnięcie przycisku myszy
onmouseover	mouseover()	Najechanie kursem myszy na elem.

jQuery i zdarzenia



Biblioteka jQuery potrafi również operować na zdarzeniach i funkcjach obsługi zdarzeń. Przypomnijmy typy najpopularniejszych zdarzeń:

Fun. obsługi zdarzenia	Metoda jQuery	Opis
onclick	click()	Kliknięcie elementu
dblclick	dblclick()	Podwójne kliknięcie elem.
onmousedown	mousedown()	Naciśnięcie przycisku myszy
onmouseover	mouseover()	Najechanie kursem myszy na elem.

Podobnie jak w samym JavaScript można używać tych zdarzeń do sterowanie aplikacją:

```
<script type="text/javascript">
  $(function(){
    $('#button').mouseover(function(){
      alert("Klikam w link do strony UJ");
    });
  });
</script>
```

```
$(` selektor`).zdarzenie();
```

Modyfikacja elementów



jQuery umożliwia w łatwy sposób manipulacje elementami struktury DOM każdego dokumentu hipertekstowego. Jedną z podstawowych operacji jaką można wykonać je możliwość zmodyfikowania wyglądu danego elementu przez nadanie mu odpowiedniego stylu CSS. Służy do tego metoda:

.css()

Modyfikacja elementów



jQuery umożliwia w łatwy sposób manipulacje elementami struktury DOM każdego dokumentu hipertekstowego. Jedną z podstawowych operacji jaką można wykonać je możliwość zmodyfikowania wyglądu danego elementu przez nadanie mu odpowiedniego stylu CSS. Służy do tego metoda:

.css()

Przykład wykorzystania:

```
<script type="text/javascript">
    $(function() {
        $('span').css('background', '#CCC000');
    });
</script>
```

Ta funkcja ustali wartość cechy “background” elementu “span” na odpowiedni kolor.

Modyfikacja elementów



jQuery umożliwia w łatwy sposób manipulacje elementami struktury DOM każdego dokumentu hipertekstowego. Jedną z podstawowych operacji jaką można wykonać je możliwość zmodyfikowania wyglądu danego elementu przez nadanie mu odpowiedniego stylu CSS. Służy do tego metoda:

.css()

Przykład wykorzystania:

```
<script type="text/javascript">
  $(function() {
    $('span').css('background', '#CCC000');
  });
</script>
```

natomiast:

```
<script type="text/javascript">
  $(function() {
    $('span').css('background');
  });
</script>
```

Ta funkcja ustali wartość cechy “background” elementu “span” na odpowiedni kolor.

Ta funkcja odczyta tylko wartość ustawioną dla cechy “background” dla elementu “span”.

Modyfikacja elementów



jQuery umożliwia w łatwy sposób manipulacje elementami struktury DOM każdego dokumentu hipertekstowego. Jedną z podstawowych operacji jaką można wykonać je możliwość zmodyfikowania wyglądu danego elementu przez nadanie mu odpowiedniego stylu CSS. Służy do tego metoda:

.css()

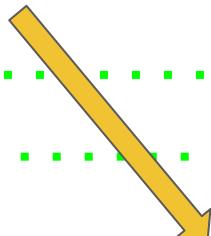
Przykład wykorzystania:

```
<script type="text/javascript">
  $(function() {
    $('span').css('background', '#CCC000');
  });
</script>
```

natomiast:

```
<script type="text/javascript">
  $(function() {
    $('p').css('background' : $('span').css('background'));
  });
</script>
```

Ta funkcja ustali wartość cechy "background" elementu "span" na odpowiedni kolor.



Modyfikacja elementów



Poza operowaniem na pojedynczych cechach CSS elementów mamy możliwość ustawiania “niejako w locie”, że dany element należy do danej klasy lub nie za pomocą metod:

`.addClass()`

`.removeClass()`

Modyfikacja elementów



Poza operowaniem na pojedynczych cechach CSS elementów mamy możliwość ustawiania “niejako w locie”, że dany element należy do danej klasy lub nie za pomocą metod:

.addClass()

.removeClass()

Przykład zastosowania:

```
<style type="text/css">
    .wazny { color: red; }
</style>

<script type="text/javascript">
    $(function() {
        $('li').mouseover(function() {
            $(this).addClass('wazny');
        }).mouseout(function() {
            $(this).removeClass('wazny');
        });
    });
</script>
```

Modyfikacja elementów



Poza operowaniem na pojedynczych cechach CSS elementów mamy możliwość ustawiania “niejako w locie”, że dany element należy do danej klasy lub nie za pomocą metod:

.addClass()

.removeClass()

Przykład zastosowania:

```
<style type="text/css">
    .wazny { color: red; }
</style>

<script type="text/javascript">
    $(function() {
        $('li').mouseover(function() {
            $(this).addClass('wazny');
        }).mouseout(function() {
            $(this).removeClass('wazny');
        });
    });
</script>
```

Co w tym
kontekście znaczy
słowo “**this**” ??

Modyfikacja elementów



Poza operowaniem na pojedynczych cechach CSS elementów mamy możliwość ustawiania “niejako w locie”, że dany element należy do danej klasy lub nie za pomocą metod:

.addClass()

.removeClass()

Przykład zastosowania:

```
<style type="text/css">
    .wazny { color: red; }
</style>

<script type="text/javascript">
    $(function() {
        $('li').mouseover(function() {
            $(this).addClass('wazny');
        }).mouseout(function() {
            $(this).removeClass('wazny');
        });
    });
</script>
```

Co w tym kontekście znaczy słowo “**this**” ??

Słowo “**this**” wskazuje na kontekst zdarzenia, czyli wskazuje na element DOM, który to zdarzenie wygenerował.

W tym konkretnym przykładzie:

`$(this) = $('li')`

Modyfikacja elementów



Poza operowaniem na pojedynczych cechach CSS elementów mamy możliwość ustawiania “niejako w locie”, że dany element należy do danej klasy lub nie za pomocą metod:

.addClass()

.removeClass()

Przykład zastosowania:

```
<script type="text/javascript">
    $(function() {
        $('div#tresc').mouseover(function() {
            $('p').css('background', $(this).css('background-color'));
        }).mouseout(function() {
            $('p').css('background', 'white');
        });
    });
</script>

<div id="tresc" style="background-color: red;">
    <p> Testowa tresc </p>
</div>
```

Modyfikacja elementów



Poza operowaniem na pojedynczych cechach CSS elementów mamy możliwość ustawiania “niejako w locie”, że dany element należy do danej klasy lub nie za pomocą metod:

.addClass()

.removeClass()

Przykład zastosowania:

```
<script type="text/javascript">
    $(function(){
        $('div#tresc').mouseover(function(){
            $('p').css('background', $(this).css('background-color'));
        }).mouseout(function(){
            $('p').css('background', 'white');
        });
    });
</script>

<div id="tresc" style="background-color: red;">
    <p> Testowa tresc </p>
</div>
```

Inny przykład użycia “this” :

Uwaga:

kolor tła można ustawić za pomocą zbiorczej cechy “background”, natomiast odczytu możemy dokonać tylko z konkretnej pojedynczej cechy takiej jak: “background-color”.

Wywołania łańcuchowe



W poprzednim przykładzie posłużyliśmy się dość dziwaczna, ale bardzo zgrabną konwencją zapisu:

```
$('li').mouseover(function() {
    $(this).addClass('wazny');
}).mouseout(function() {
    $(this).removeClass('wazny');
});
```

Jedna metoda była wykonywana na tym samym obiekcie co poprzednia

Wywołania łańcuchowe



W poprzednim przykładzie posłużyliśmy się dość dziwaczna, ale bardzo zgrabną konwencją zapisu:

```
$('li').mouseover(function() {
    $(this).addClass('wazny');
}).mouseout(function() {
    $(this).removeClass('wazny');
});
```

Jedna metoda była wykonywana na tym samym obiekcie co poprzednia

jest to tzw. “łączenie wywołań” lub “łańcuch wywołań”. W ogólności przyjmuje on postać:

```
$('selektor').metoda1().metoda2().metoda3() . . . . metodaN();
```

co odpowiada postaci:

```
$('selektor').metoda1();
$('selektor').metoda2();
$('selektor').metoda3();
.....
$('selektor').metodaN();
```

Wywołania łańcuchowe



W poprzednim przykładzie posłużyliśmy się dość dziwaczna, ale bardzo zgrabną konwencją zapisu:

```
$('li').mouseover(function() {
    $(this).addClass('wazny');
}).mouseout(function() {
    $(this).removeClass('wazny');
});
```

Jedna metoda była wykonywana na tym samym obiekcie co poprzednia

jest to tzw. “łączenie wywołań” lub “łańcuch wywołań”. W ogólności przyjmuje on postać:

```
$(selektor).metoda1().metoda2().metoda3() . . . . metodaN();
```

co odpowiada postaci:

```
$(selektor).metoda1();
$(selektor).metoda2();
$(selektor).metoda3();
.....
$(selektor).metodaN();
```

Która z postaci zapisów jest lepszy ??

Wywołania łańcuchowe



W poprzednim przykładzie posłużyliśmy się dość dziwaczna, ale bardzo zgrabną konwencją zapisu:

```
$($('li').mouseover(function() {
    $(this).addClass('wazny');
}).mouseout(function() {
    $(this).removeClass('wazny');
});
```

Jedna metoda była wykonywana na tym samym obiekcie co poprzednia

jest to tzw. “łączenie wywołań” lub “łańcuch wywołań”. W ogólności przyjmuje on postać:

```
$(selektor).metoda1().metoda2().metoda3() . ... .metodaN();
```

co odpowiada postaci:

```
$(selektor).metoda1();
$(selektor).metoda2();
```

Która z postaci zapisów jest lepszy ??

Zapis łańcuchowy jest “lepszy” ponieważ w wyniku jego działania powstaje tylko jeden obiekt, który jest wynikiem przeszukania elementu pasującego do danego selektora. W drugim przypadku dokument byłby przeszukiwany wielokrotnie.

Ukrywanie elementów



Za pomocą CSS możemy ukrywać elementy strony dokumentu hipertekstowego używając cechy “display”:

```
<style type="text/css">  
    div { display: none; }  
</style>
```

Aby przywrócić widoczność elementu ustawimy wartość cechy na: “block” lub “inline”.

Ukrywanie elementów



Za pomocą CSS możemy ukrywać elementy strony dokumentu hipertekstowego używając cechy “display”:

```
<style type="text/css">
    div { display: none; }
</style>
```

Aby przywrócić widoczność elementu ustawimy wartość cechy na: “block” lub “inline”.

W jQuery również możemy wykonywać takie operacje za pomocą metod:

.hide()

(ukrywanie)

.show()

(pokazywanie)

Przykład zastosowania:

```
$(function() {
    $('button#hide').click(function() {
        $('p').hide();
    });
    $('button#show').click(function() {
        $('p').show();
    });
});
```

Ukrywanie elementów



Za pomocą CSS możemy ukrywać elementy strony dokumentu hipertekstowego używając cechy “display”:

```
<style type="text/css">
    div { display: none; }
</style>
```

Aby przywrócić widoczność elementu ustawimy wartość cechy na: “block” lub “inline”.

W jQuery również możemy wykonywać takie operacje za pomocą metod:

.hide()

(ukrywanie)

.show()

(pokazywanie)

Przykład zastosowania:

```
$(function() {
    $('button#hide').click(function() {
        $('p').hide();
    });
    $('button#show').click(function() {
        $('p').show();
    });
});
```

Dodatkowo argumentem obu funkcji może być opcjonalny paramter “slow” lub “fast” lub określenie w milisekundach szybkości wykonania danej czynności.

Ukrywanie elementów



Za pomocą CSS możemy ukrywać elementy strony dokumentu hipertekstowego używając cechy “display”:

```
<style type="text/css">
    div { display: none; }
</style>
```

Aby przywrócić widoczność elementu ustawimy wartość cechy na: “block” lub “inline”.

W jQuery również możemy wykonywać takie operacje za pomocą metod:

.hide()

(ukrywanie)

.show()

(pokazywanie)

Przykład zastosowania:

```
$(function() {
    $('button#hide').click(function() {
        $('p').hide("slow");
    });
    $('button#show').click(function() {
        $('p').show(2000);
    });
});
```

Dodatkowo argumentem obu funkcji może być opcjonalny paramter “slow” lub “fast” lub określenie w milisekundach szybkości wykonania danej czynności.

Ukrywanie elementów



W jQuery istnieje również metoda obsługująca obie operacje (ukrywania i pokazywania elementu):

.toggle()

(ukrywanie / pokazywanie)

Przykład użycia:

```
$function() {
    $('button').click(function() {
        $('span').toggle();
    });
}
```

Ukrywanie elementów



W jQuery istnieje również metoda obsługująca obie operacje (ukrywania i pokazywania elementu):

.toggle()

(ukrywanie / pokazywanie)

Przykład użycia:

```
$function() {
  $('button').click(function() {
    $('span').toggle();
  });
}
```

Również w tym przypadku możemy określić szybkość zachodzenia operacji tymi samymi parametrami co poprzednio.

Ukrywanie elementów



W jQuery istnieje również metoda obsługująca obie operacje (ukrywania i pokazywania elementu):

.toggle()

(ukrywanie / pokazywanie)

Przykład użycia:

```
$function() {
  $('button').click(function() {
    $('span').toggle();
  });
}
```

Również w tym przypadku możemy określić szybkość zachodzenia operacji tymi samymi parametrami co poprzednio.

Ćwiczenie:

Napisać menu UL/LI z dwoma poziomami zagnieżdżenia, w którym drugi poziom menu będzie widoczny dopiero po najechaniu na niego kursem myszy. Wykorzystać do tego metodę .toggle()

Odczyt i modyfikacja zawartości elementów



Modyfikacja zawartości elementów oraz i ich treści jest podobnie prosta jak w przypadku czystego JavaScriptu. Służą do tego celu dwie metody:





Odczyt i modyfikacja zawartości elementów



Modyfikacja zawartości elementów oraz i ich treści jest podobnie prosta jak w przypadku czystego JavaScriptu. Służą do tego celu dwie metody:

.text()



modyfikuje zawartość **tekstową** elementu

.html()



modyfikuje zawartość **hipertekstową** elementu

```
$ (function() {  
    $('div#tresc1').text('Nowa treść wpisana dynamicznie w jQuery');  
    $('div#tresc2').html('Nowa treść wpisana dynamicznie w jQuery');  
});
```



Odczyt i modyfikacja zawartości elementów



Modyfikacja zawartości elementów oraz i ich treści jest podobnie prosta jak w przypadku czystego JavaScriptu. Służą do tego celu dwie metody:

.text()



modyfikuje zawartość **tekstową** elementu

.html()



modyfikuje zawartość **hipertekstową** elementu

```
$ (function() {  
    $('div#tresc1').text('Nowa treść wpisana dynamicznie w jQuery');  
    $('div#tresc2').html('Nowa treść wpisana dynamicznie w jQuery');  
});
```

Czym będzie różnił się wynik działania obu funkcji ??



Odczyt i modyfikacja zawartości elementów



Modyfikacja zawartości elementów oraz i ich treści jest podobnie prosta jak w przypadku czystego JavaScriptu. Służą do tego celu dwie metody:



```
$ (function() {  
    $('div#tresc1').text('Nowa treść wpisana dynamicznie w jQuery');  
    $('div#tresc2').html('Nowa treść wpisana dynamicznie w jQuery');  
});
```

Czym będzie różnił się wynik działania obu funkcji ??

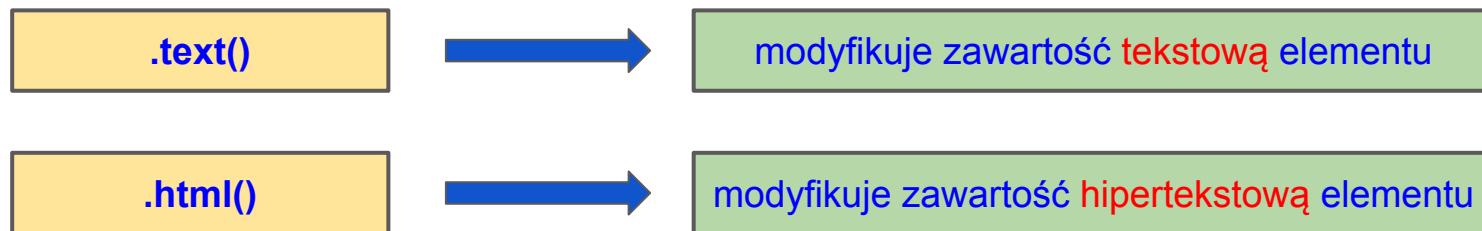
NICZYM !



Odczyt i modyfikacja zawartości elementów



Modyfikacja zawartości elementów oraz i ich treści jest podobnie prosta jak w przypadku czystego JavaScriptu. Służą do tego celu dwie metody:



```
$ (function() {  
    $('div#tresc1').text('Nowa treść wpisana dynamicznie w jQuery');  
    $('div#tresc2').html('Nowa treść wpisana dynamicznie w jQuery');  
});
```

Czym będzie różnił się wynik działania obu funkcji ??

NICZYM !

ALE

Odczyt i modyfikacja zawartości elementów



.text()



modyfikuje zawartość **tekstową** elementu

.html()



modyfikuje zawartość **hipertekstową** elementu

```
$ (function() {  
    $('div#tresc1').text('<span> To jest treść</span>');  
    $('div#tresc2').html('<span> To jest treść</span>');  
});
```

Odczyt i modyfikacja zawartości elementów



.text()



modyfikuje zawartość **tekstową** elementu

.html()



modyfikuje zawartość **hipertekstową** elementu

```
$ (function() {
    $('div#tresc1').text('<span> To jest treść</span>');
    $('div#tresc2').html('<span> To jest treść</span>');
});
```

```
<div id="tresc1">
    &lt;span&gt; To jest treść &lt;/span&gt;
</div>

<div id="tresc2">
    <span> To jest treść</span>
</div>
```



Treść w metodzie ".text()" zostanie sparsowana i wszystkie znaki specjalne zostaną zmienione na ich odpowiedniki zapisane za pomocą endji html.

Odczyt i modyfikacja zawartości elementów



.text()



modyfikuje zawartość **tekstową** elementu

.html()



modyfikuje zawartość **hipertekstową** elementu

```
$ (function() {
    $('div#tresc1').text('<span> To jest treść</span>');
    $('div#tresc2').html('<span> To jest treść</span>');
});
```

```
<div id="tresc1">
    &lt;span&gt; To jest treść &lt;/span&gt;
</div>

<div id="tresc2">
    <span> To jest treść</span>
</div>
```



Treść w metodzie ".text()" zostanie sparsowana i wszystkie znaki specjalne zostaną zmienione na ich odpowiedniki zapisane za pomocą endji html.

Zatem za pomocą metody .text() dodajemy treści, a za pomocą metody .html() możemy dodawać nowe zagnieżdzone struktury znacznikowe.



Odczyt i modyfikacja atrybutów elementów



Bardzo często w dynamicznym dokumencie hipertekstowym zachodzi potrzeba manipulacji wartościami atrybutów konkretnego znacznika. Dostęp do atrybutów znaczników uzyskujemy za pomocą metody:

```
.attr( nazwa atrybutu , wartosc atrybutu )
```



Odczyt i modyfikacja atrybutów elementów



Bardzo często w dynamicznym dokumencie hipertekstowym zachodzi potrzeba manipulacji wartościami atrybutów konkretnego znacznika. Dostęp do atrybutów znaczników uzyskujemy za pomocą metody:

```
.attr( nazwa atrybutu , wartosc atrybutu )
```

Jako argument tej metody podajemy nazwę atrybutu który chcemy zamodyfikować:

```
$function() {
    $('img').attr('src', 'obrazek.png');
};

<img src="" alt="obrazek"/>
```



Odczyt i modyfikacja atrybutów elementów



Bardzo często w dynamicznym dokumencie hipertekstowym zachodzi potrzeba manipulacji wartościami atrybutów konkretnego znacznika. Dostęp do atrybutów znaczników uzyskujemy za pomocą metody:

```
.attr( nazwa atrybutu , wartosc atrybutu )
```

Jako argument tej metody podajemy nazwę atrybutu który chcemy zmodyfikować:

```
$function() {
    $('img').attr('src', 'obrazek.png');
};

<img src="" alt="obrazek"/>
```

Jak taką funkcjonalność wykorzystać w przypadku elementów img ???



Odczyt i modyfikacja atrybutów elementów



Bardzo często w dynamicznym dokumencie hipertekstowym zachodzi potrzeba manipulacji wartościami atrybutów konkretnego znacznika. Dostęp do atrybutów znaczników uzyskujemy za pomocą metody:

```
.attr( nazwa atrybutu , wartosc atrybutu )
```

Jako argument tej metody podajemy nazwę atrybutu który chcemy zmodyfikować:

```
$function() {
    $('img').attr('src', 'obrazek.png');
};

<img src="" alt="obrazek"/>
```

Jak taką funkcjonalność wykorzystać w przypadku elementów img ???



Można stworzyć prostą galerię

Odczyt i modyfikacja atrybutów elementów



```
.attr(  nazwa atrybutu ,  wartosc atrybutu )
```

```
$function(){
    $('td img').click(function(){
        var adres = $(this).attr('src').replace('maly', 'duzy');
        $('div img').attr('src', adres);
    });
}

<tr>
    <td></td>
    <td></td>
    <td></td>
    <td></td>
</tr>

<div>
    
</div>
```



Odczyt i modyfikacja atrybutów elementów



`.attr(nazwa atrybutu , wartosc atrybutu)`

```
$function(){
    $('td img').click(function(){
        var adres = $(this).attr('src').replace('maly', 'duzy');
        $('div img').attr('src', adres);
    });
}

<tr>
    <td></td>
    <td></td>
    <td></td>
    <td></td>
</tr>

<div>
    
</div>
```

Metoda `.replace()` zastępuje w stringu wybrany wzorzec (podany jako argument 1) i zastępuje go podanym jako argument 2 tekstem.



KONIEC WYKŁADU 7



UNIWERSYTET
JAGIELŁOŃSKI
W KRAKOWIE

Zaawansowane Techniki WWW (HTML, CSS i JavaScript)

Dr inż. Marcin Zieliński

Środa 15:30 - 17:00 sala: A-1-04

WYKŁAD 8

Wykład dla kierunku: Informatyka Stosowana II rok

Rok akademicki: 2015/2016 - semestr zimowy

Przypomnienie z poprzedniego wykładu

Wprowadzenie do biblioteki jQuery

Pobranie i instalacja jQuery

Pobieramy plik w najnowszej wersji ze strony:

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



[jquery-2.1.4.js](#)
[jquery-2.1.4.min.js](#)

```
<!DOCTYPE html>
<html>
<head>
<title>Pierwsza strona z jQuery</title>
<script type="text/javascript" src="jquery-2.1.4.min.js"></script>
</head>
<body>
<h1>Testujemy bibliotekę jQuery</h1>
</body>
</html>
```

Modyfikacja drzewa DOM



Do bardziej złożonych operacji które możemy wykonywać za pomocą biblioteki jQuery jest dowolna modyfikacja całej struktury drzewa DOM. Mówiliśmy już o jednej metodzie która na to pozwala `.html()` . Natomiast metod do wykonywania operacji na drzewie DOM jest dużo więcej:

Modyfikacja drzewa DOM



Do bardziej złożonych operacji które możemy wykonywać za pomocą biblioteki jQuery jest dowolna modyfikacja całej struktury drzewa DOM. Mówiliśmy już o jednej metodzie która na to pozwala [.html\(\)](#) . Natomiast metod do wykonywania operacji na drzewie DOM jest dużo więcej:

`.append()`

`.appendTo()`

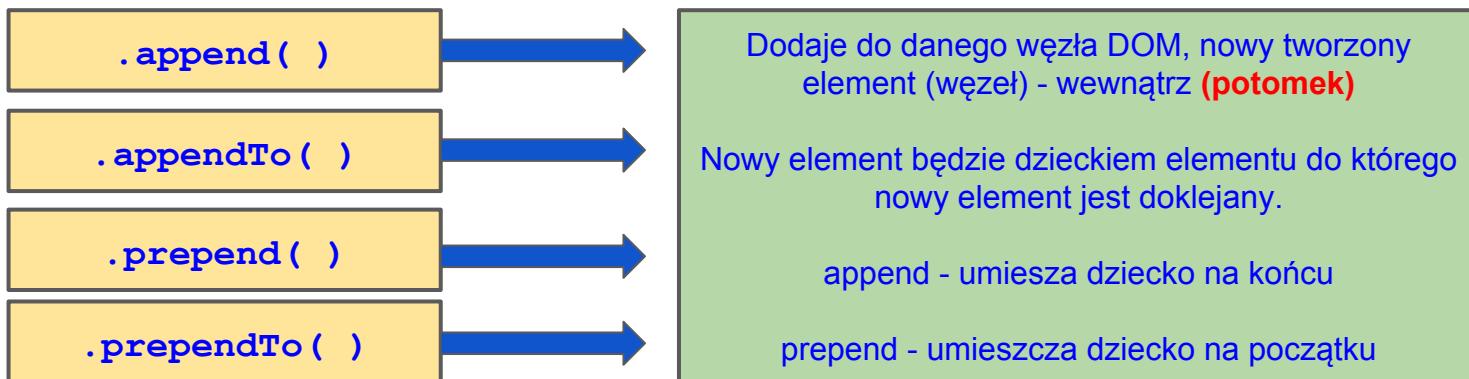
`.prepend()`

`.prependTo()`

Modyfikacja drzewa DOM



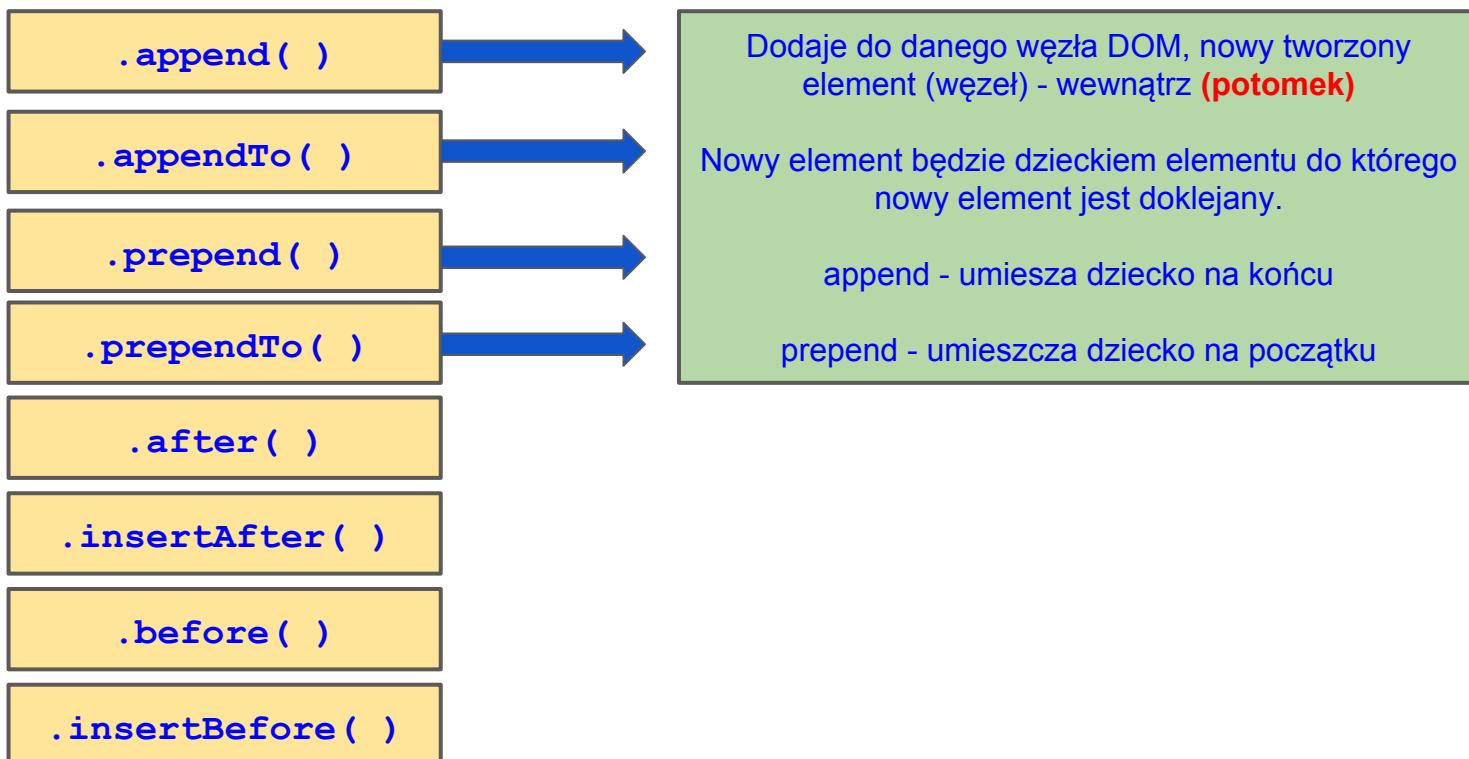
Do bardziej złożonych operacji które możemy wykonywać za pomocą biblioteki jQuery jest dowolna modyfikacja całej struktury drzewa DOM. Mówiliśmy już o jednej metodzie która na to pozwala `.html()` . Natomiast metod do wykonywania operacji na drzewie DOM jest dużo więcej:



Modyfikacja drzewa DOM



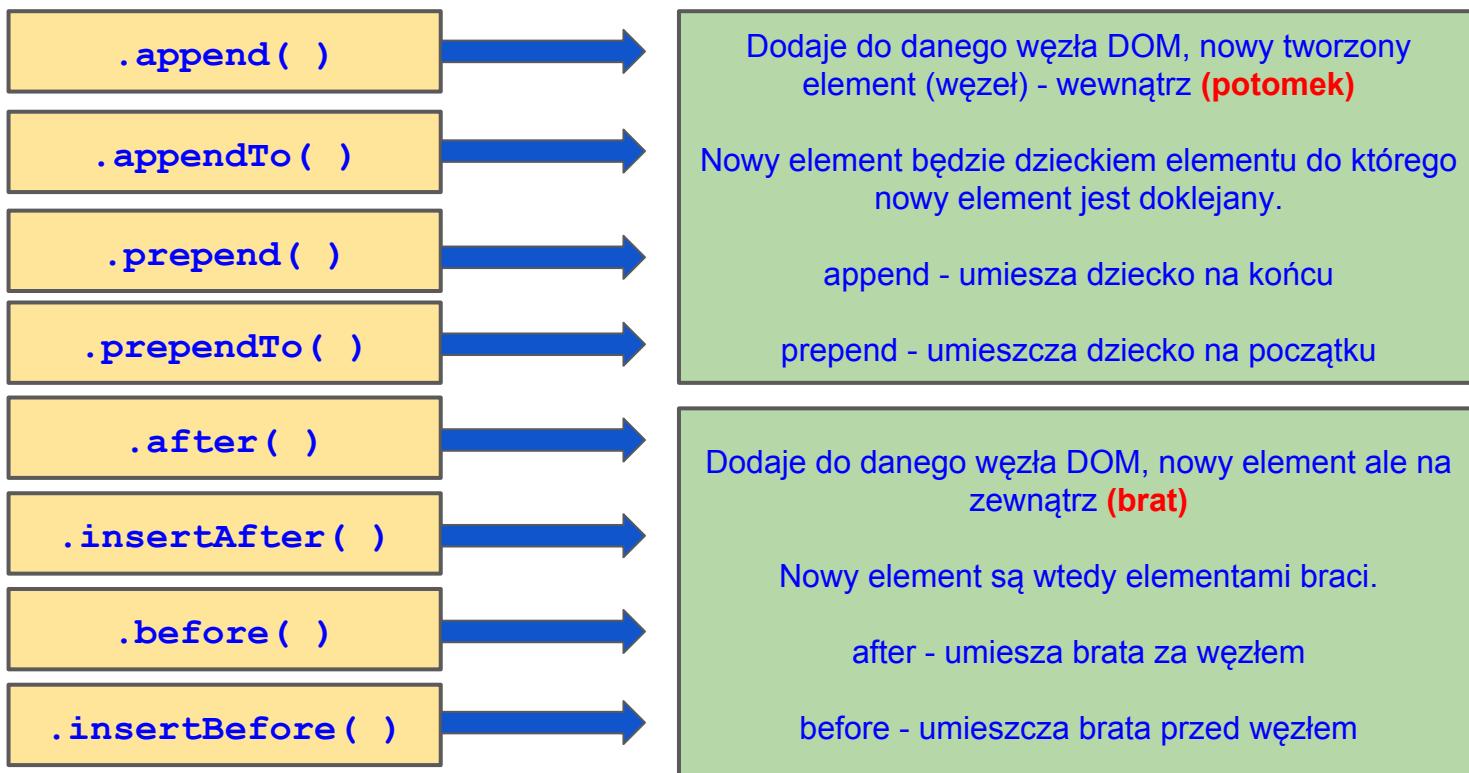
Do bardziej złożonych operacji które możemy wykonywać za pomocą biblioteki jQuery jest dowolna modyfikacja całej struktury drzewa DOM. Mówiliśmy już o jednej metodzie która na to pozwala `.html()` . Natomiast metod do wykonywania operacji na drzewie DOM jest dużo więcej:



Modyfikacja drzewa DOM



Do bardziej złożonych operacji które możemy wykonywać za pomocą biblioteki jQuery jest dowolna modyfikacja całej struktury drzewa DOM. Mówiliśmy już o jednej metodzie która na to pozwala `.html()` . Natomiast metod do wykonywania operacji na drzewie DOM jest dużo więcej:



Modyfikacja drzewa DOM



Różnica pomiędzy metodami z postfixem “To”, a bez niego:

.append()

.appendTo()

Modyfikacja drzewa DOM



Różnica pomiędzy metodami z postfixem “To”, a bez niego:

.append()



.appendTo()

W tych metodach najpierw podajemy nazwę węzła który chcemy zmodyfikować, a dopiero później węzeł dodawany.

Modyfikacja drzewa DOM



Różnica pomiędzy metodami z postfixem “To”, a bez niego:

.append()



W tych metodach najpierw podajemy nazwę węzła który chcemy zmodyfikować, a dopiero później węzeł dodawany.

.appendTo()



W tych metodach najpierw podajemy węzeł dodawany, a dopiero jako drugi argument nazwę węzła który chcemy zmodyfikować.

Modyfikacja drzewa DOM



Różnica pomiędzy metodami z postfixem “To”, a bez niego:

.append()



W tych metodach najpierw podajemy nazwę węzła który chcemy zmodyfikować, a dopiero później węzeł dodawany.

.appendTo()



W tych metodach najpierw podajemy węzeł dodawany, a dopiero jako drugi argument nazwę węzła który chcemy zmodyfikować.

```
$(function() {
  $('body').append('<div></div>');
});
```

```
$(function() {
  $('
```

Modyfikacja drzewa DOM



Różnica pomiędzy metodami z postfixem “To”, a bez niego:

.append()



W tych metodach najpierw podajemy nazwę węzła który chcemy zmodyfikować, a dopiero później węzeł dodawany.

.appendTo()



W tych metodach najpierw podajemy węzeł dodawany, a dopiero jako drugi argument nazwę węzła który chcemy zmodyfikować.

```
$function() {
  $('body').append('<div></div>');
}
```

1. Wybranie elementu <body> z drzewa DOM.
2. Wywołanie metody append().
3. Utworzenie węzła <div>.
4. Dowiązanie elementu <div> jako dziecka.

```
$function() {
  '<div>
    </div>'.appendTo('body');
}
```

1. Utworzenie węzła <div>.
2. Wywołanie metody appendTo().
3. Wybranie elementu <body> z drzewa DOM.
4. Dowiązanie elementu <div> jako dziecka.

Modyfikacja drzewa DOM



Różnica pomiędzy metodami z postfixem “To”, a bez niego:

.append()



W tych metodach najpierw podajemy nazwę węzła który chcemy zmodyfikować, a dopiero później węzeł dodawany.

.appendTo()



W tych metodach najpierw podajemy węzeł dodawany, a dopiero jako drugi argument nazwę węzła który chcemy zmodyfikować.

```
$function() {
  $('body').append('<div></div>');
}
```

1. Wybranie elementu <body> z drzewa DOM.
2. Wywołanie metody append().
3. Utworzenie węzła <div>.
4. Dowiązanie elementu <div> jako dziecka.

```
<body>
  <div></div>
</body>
```

```
$function() {
  '<div>
    </div>'.appendTo('body');
}
```

1. Utworzenie węzła <div>.
2. Wywołanie metody appendTo().
3. Wybranie elementu <body> z drzewa DOM.
4. Dowiązanie elementu <div> jako dziecka.

Modyfikacja drzewa DOM



Różnica pomiędzy metodami z postfixem “To”, a bez niego:

.append()

.appendTo()

W tych metodach najpierw podajemy nazwę węzła który chcemy zmodyfikować, a dopiero później węzeł dodawany.

W tych metodach najpierw podajemy węzeł dodawany, a dopiero jako drugi argument nazwę węzła który chcemy zmodyfikować.

```
$function() {
  $('body').append('<div></div>');
}
```

1. Wybranie elementu <body> z drzewa DOM.
2. Wywołanie metody append().
3. Utworzenie węzła <div>.
4. Dowiązanie elementu <div> jako dziecka.

```
<body>
  <div></div>
</body>
```

```
$function() {
  '<div>
    </div>'.appendTo('body');
}
```

1. Utworzenie węzła <div>.
2. Wywołanie metody appendTo().
3. Wybranie elementu <body> z drzewa DOM.
4. Dowiązanie elementu <div> jako dziecka.

Wynik działania
jest identyczny!



Modyfikacja drzewa DOM



Przykłady zastosowania metod “before()” i “after()” :

```
$ (function () {
    $('p').before('<div id="przed"></div>');
});

$(function () {
    $('p').after('<div id="po"></div>');
});

<body>
    <p>Tu jest akapit</p>
</body>
```



Modyfikacja drzewa DOM



Przykłady zastosowania metod “before()” i “after()” :

```
$ (function () {
    $('p').before('<div id="przed"></div>');
});

$(function () {
    $('p').after('<div id="po"></div>');
});

<body>
    <p>Tu jest akapit</p>
</body>
```

Modyfikacja drzewa DOM



Przykłady zastosowania metod “before()” i “after()” :

```
$ (function () {
    $('p').before('<div id="przed"></div>');
});

$(function () {
    $('p').after('<div id="po"></div>');
});

<body>
    <p>Tu jest akapit</p>
</body>
```



```
<body>
    <div id="przed"></div>
    <p>Tu jest akapit</p>
    <div id="po"></div>
</body>
```

Modyfikacja drzewa DOM



Przykłady zastosowania metod “insertBefore()” i “insertAfter()” :

```
$ (function () {
    $('<div id="przed"></div>').insertBefore('p');
});

$(function () {
    $('<div id="po"></div>').insertAfter('p');
});

<body>
    <p>Tu jest akapit</p>
</body>
```

Modyfikacja drzewa DOM



Przykłady zastosowania metod “insertBefore()” i “insertAfter()” :

```
$ (function () {
    $('<div id="przed"></div>').insertBefore('p');
});

$(function () {
    $('<div id="po"></div>').insertAfter('p');
});

<body>
    <p>Tu jest akapit</p>
</body>
```



```
<body>
    <div id="przed"></div>
    <p>Tu jest akapit</p>
    <div id="po"></div>
</body>
```

Modyfikacja drzewa DOM



Przykłady dodawania nowego elementy z “appendTo()”:

```
$ (function () {
  $('<div>',{
    'class': 'nowy',
    text: 'Nowy element blokowy',
    click: function(){
      $(this).css('color': 'red');
    }
  }).appendTo('body');
});

<body>
</body>
```

Dodawanie nowego elementu i określanie jego atrybutów i własności.

Modyfikacja drzewa DOM



Przykłady dodawania nowego elementy z “appendTo()”:

```
$ (function () {
    $('<div>', {
        'class': 'nowy',
        text: 'Nowy element blokowy',
        click: function () {
            $(this).css('color': 'red');
        }
    }).appendTo('body');
});
```

Dodawanie nowego elementu i określanie jego atrybutów i własności.



```
<body>
    <div class="nowy">Nowy element blokowy</div>
</body>
```

Modyfikacja drzewa DOM



Poprzednich metod można używać do dodawania nowych węzłów na początku i na końcu dowolnego istniejącego węzła. Istnieją natomiast funkcje które pozwalają na “owijanie” węzłów innymi węzłami:

`.wrap()`

`.wrapInner()`

Modyfikacja drzewa DOM



Poprzednich metod można używać do dodawania nowych węzłów na początku i na końcu dowolnego istniejącego węzła. Istnieją natomiast funkcje które pozwalają na “owijanie” węzłów innymi węzłami:

`.wrap()`



`.wrapInner()`

Metoda ta “owija” istniejący element w nowy element podany jako argument omawianej metody.

Modyfikacja drzewa DOM



Poprzednich metod można używać do dodawania nowych węzłów na początku i na końcu dowolnego istniejącego węzła. Istnieją natomiast funkcje które pozwalają na “owijanie” węzłów innymi węzłami:

`.wrap()`



Metoda ta “owija” istniejący element w nowy element podany jako argument omawianej metody.

`.wrapInner()`



Metoda ta owija zawartość wybranego elementu elementem podanym jako argument omawianej metody.

Modyfikacja drzewa DOM



Poprzednich metod można używać do dodawania nowych węzłów na początku i na końcu dowolnego istniejącego węzła. Istnieją natomiast funkcje które pozwalają na “owijanie” węzłów innymi węzłami:

.wrap()



Metoda ta “owija” istniejący element w nowy element podany jako argument omawianej metody.

.wrapInner()



Metoda ta owija zawartość wybranego elementu elementem podanym jako argument omawianej metody.

<p>Tekst akapitu</p>

```
$function() {
  $('p').wrap('<div></div>');
});
```

```
$function() {
  $('p').wrap('<strong></strong>');
});
```

Modyfikacja drzewa DOM



Poprzednich metod można używać do dodawania nowych węzłów na początku i na końcu dowolnego istniejącego węzła. Istnieją natomiast funkcje które pozwalają na “owijanie” węzłów innymi węzłami:

.wrap()



Metoda ta “owija” istniejący element w nowy element podany jako argument omawianej metody.

.wrapInner()



Metoda ta owija zawartość wybranego elementu elementem podanym jako argument omawianej metody.

<p>Tekst akapitu</p>

```
$function() {
  $('p').wrap('<div></div>');
});
```



```
<div>
  <p>Tekst akapitu</p>
</div>
```

```
$function() {
  $('p').wrap('<strong></strong>');
});
```



```
<p><strong>Tekst akapitu</strong></p>
```

Modyfikacja drzewa DOM



Poprzednich metod można używać do dodawania nowych węzłów na początku i na końcu dowolnego istniejącego węzła. Istnieją natomiast funkcje które pozwalają na “owijanie” węzłów innymi węzłami:

.wrap()



Metoda ta “owija” istniejący element w nowy element podany jako argument omawianej metody.

.wrapInner()



Metoda ta owija zawartość wybranego elementu elementem podanym jako argument omawianej metody.

<p>Tekst akapitu</p>

```
$function() {  
  $('p').wrap('<div></div>');  
}  
$function() {  
  $('p').wrap('<strong></strong>');  
}
```

Możliwy jest jeszcze bardziej zwięzły zapis...

```
</div>
```

Modyfikacja drzewa DOM



Poprzednich metod można używać do dodawania nowych węzłów na początku i na końcu dowolnego istniejącego węzła. Istnieją natomiast funkcje które pozwalają na “owijanie” węzłów innymi węzłami:

.wrap()

Metoda ta “owija” istniejący element w nowy element podany jako argument omawianej metody.

.wrapInner()

Metoda ta owija zawartość wybranego elementu elementem podanym jako argument omawianej metody.

Można stosować zapis skrócony znaczników...

<p>Tekst akapitu</p>

```
$function() {
  $('p').wrap('<div>');
});
```

```
<div>
  <p>Tekst akapitu</p>
</div>
```

```
$function() {
  $('p').wrap('<strong>');
});
```

<p>Tekst akapitu</p>

Modyfikacja drzewa DOM



Do usuwania elementów drzewa DOM służy tylko jedna metoda:



Modyfikacja drzewa DOM



Do usuwania elementów drzewa DOM służy tylko jedna metoda:



```
$function() {
  $('p').remove();
};

<body>
  <div>
    <p>Tekst akapitu </p>
  </div>
</body>
```

Modyfikacja drzewa DOM



Do usuwania elementów drzewa DOM służy tylko jedna metoda:

`.remove()`

Metoda usuwa elementy drzewa DOM dla którego została wywołana.

```
$function() {
  $('p').remove();
}

<body>
  <div>
    <p>Tekst akapitu </p>
  </div>
</body>
```



```
<body>
  <div>
  </div>
</body>
```

Modyfikacja drzewa DOM



Do usuwania elementów drzewa DOM służy tylko jedna metoda:

`.remove()`

Metoda usuwa elementy drzewa DOM dla którego została wywołana.

```
$ (function() {
  $('#tytul').click(function() {
    $(this).remove();
  });
}

<body>
  <div>
    <h1>Tekst akapitu </h1>
    <h1 id="tytul"> Tytuł akapitu </h1>
  </div>
</body>
```

Modyfikacja drzewa DOM



Do usuwania elementów drzewa DOM służy tylko jedna metoda:

`.remove()`

Metoda usuwa elementy drzewa DOM dla którego została wywołana.

```
$ (function() {
  $('#tytul').click(function() {
    $(this).remove();
  });
})
```



```
<body>
  <div>
    <h1>Tekst akapitu </h1>
    <h1 id="tytul"> Tytuł akapitu </h1>
  </div>
</body>
```



```
<body>
  <div>
    <h1>Tekst akapitu </h1>
  </div>
</body>
```



Poruszanie się po strukturze DOM



Do poruszania się i przechodzenia po elementach struktury drzewa DOM służą cztery metody (jedną z nich już poznaliśmy):

Poruszanie się po strukturze DOM



Do poruszania się i przechodzenia po elementach struktury drzewa DOM służą cztery metody (jedną z nich już poznaliśmy):



Poruszanie się po strukturze DOM



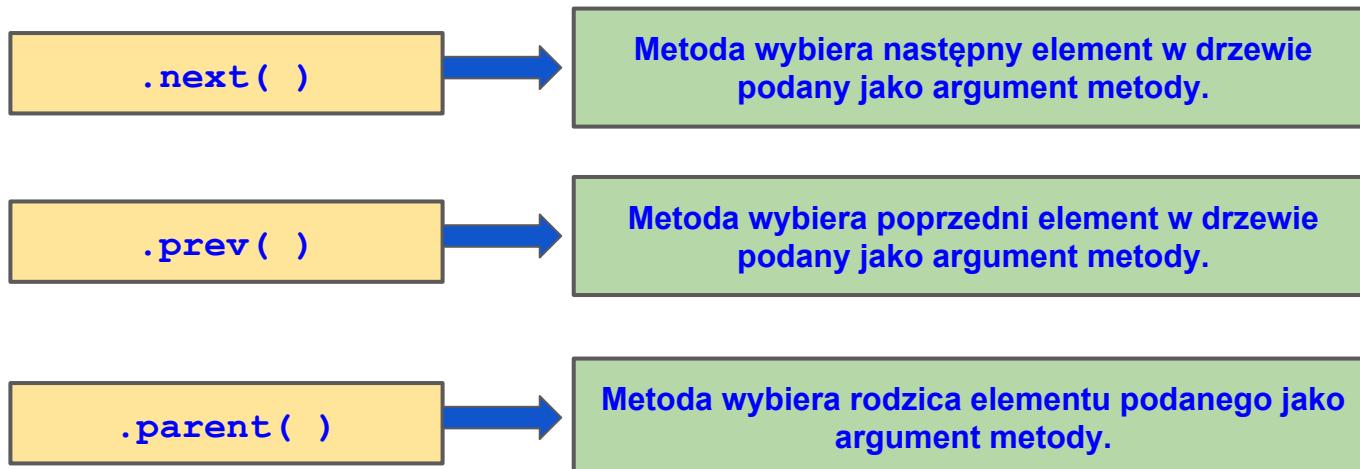
Do poruszania się i przechodzenia po elementach struktury drzewa DOM służą cztery metody (jedną z nich już poznaliśmy):



Poruszanie się po strukturze DOM



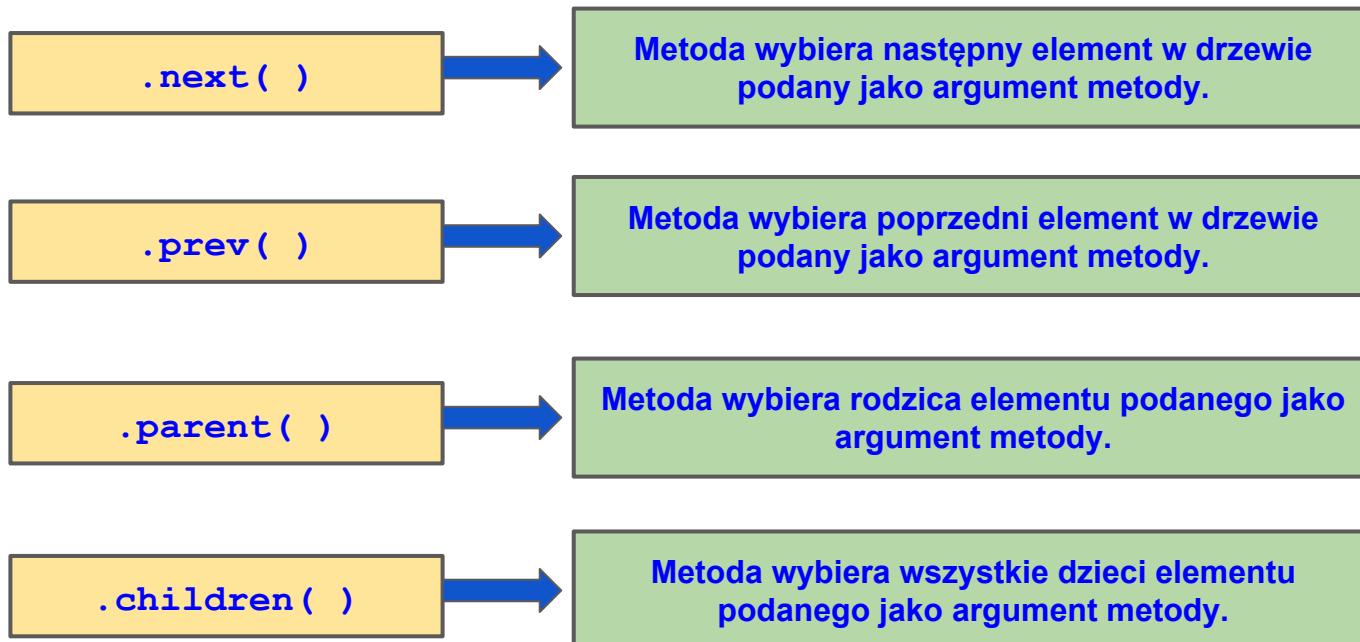
Do poruszania się i przechodzenia po elementach struktury drzewa DOM służą cztery metody (jedną z nich już poznaliśmy):



Poruszanie się po strukturze DOM



Do poruszania się i przechodzenia po elementach struktury drzewa DOM służą cztery metody (jedną z nich już poznaliśmy):





Poruszanie się po strukturze DOM



Przykład:

```
<body>
  <div id="kontener">
    <h1 id="naglowek">Naglowek akapitu</h1>
    <div id="akapit">
      <p> Tresc akapitu 1 </p>
      <p> Tresc akapitu 2 </p>
      <p> Tresc akapitu 3 </p>
    </div>
    <div id="stopka"></div>
  </div>
</body>
```

Poruszanie się po strukturze DOM



Przykład:

```
<body>
  <div id="kontener">
    <h1 id="naglowek">Naglowek akapitu</h1>
    <div id="akapit">
      <p> Tresc akapitu 1 </p>
      <p> Tresc akapitu 2 </p>
      <p> Tresc akapitu 3 </p>
    </div>
    <div id="stopka"></div>
  </div>
</body>
```

`$('#akapit').next()`



`<div id="stopka"></div>`

Poruszanie się po strukturze DOM



Przykład:

```
<body>
  <div id="kontener">
    <h1 id="naglowek">Naglowek akapitu</h1>
    <div id="akapit">
      <p> Tresc akapitu 1 </p>
      <p> Tresc akapitu 2 </p>
      <p> Tresc akapitu 3 </p>
    </div>
    <div id="stopka"></div>
  </div>
</body>
```

`$('#akapit').next()`



`<div id="stopka"></div>`

`$('#akapit').prev()`



`<h1 id="naglowek"></h1>`

Poruszanie się po strukturze DOM



Przykład:

```
<body>
  <div id="kontener">
    <h1 id="naglowek">Naglowek akapitu</h1>
    <div id="akapit">
      <p> Tresc akapitu 1 </p>
      <p> Tresc akapitu 2 </p>
      <p> Tresc akapitu 3 </p>
    </div>
    <div id="stopka"></div>
  </div>
</body>
```

`$('#akapit').next()`



`<div id="stopka"></div>`

`$('#akapit').prev()`



`<h1 id="naglowek"></h1>`

`$('#akapit').parent()`



`<div id="kontener"></div>`

Poruszanie się po strukturze DOM



Przykład:

```
<body>
  <div id="kontener">
    <h1 id="naglowek">Naglowek akapitu</h1>
    <div id="akapit">
      <p> Tresc akapitu 1 </p>
      <p> Tresc akapitu 2 </p>
      <p class="opis"> Opis opisowy akapitu </p>
    </div>
    <div id="stopka"></div>
  </div>
</body>
```

```
$( '#akapit' ).children()
```



```
<p> Tresc akapitu 1 </p>
<p> Tresc akapitu 2 </p>
<p> Tresc akapitu 3 </p>
```

Poruszanie się po strukturze DOM



Przykład:

```
<body>
  <div id="kontener">
    <h1 id="naglowek">Naglowek akapitu</h1>
    <div id="akapit">
      <p> Tresc akapitu 1 </p>
      <p> Tresc akapitu 2 </p>
      <p class="opis"> Opis opisowy akapitu </p>
    </div>
    <div id="stopka"></div>
  </div>
</body>
```

`$('#akapit').children()`



```
<p> Tresc akapitu 1 </p>
<p> Tresc akapitu 2 </p>
<p> Tresc akapitu 3 </p>
```

`$('#akapit').children().eq(0)`



```
<p> Tresc akapitu 1 </p>
```

`$('#akapit').children().eq(1)`



```
<p> Tresc akapitu 2 </p>
```

`$('#akapit').children().eq(2)`



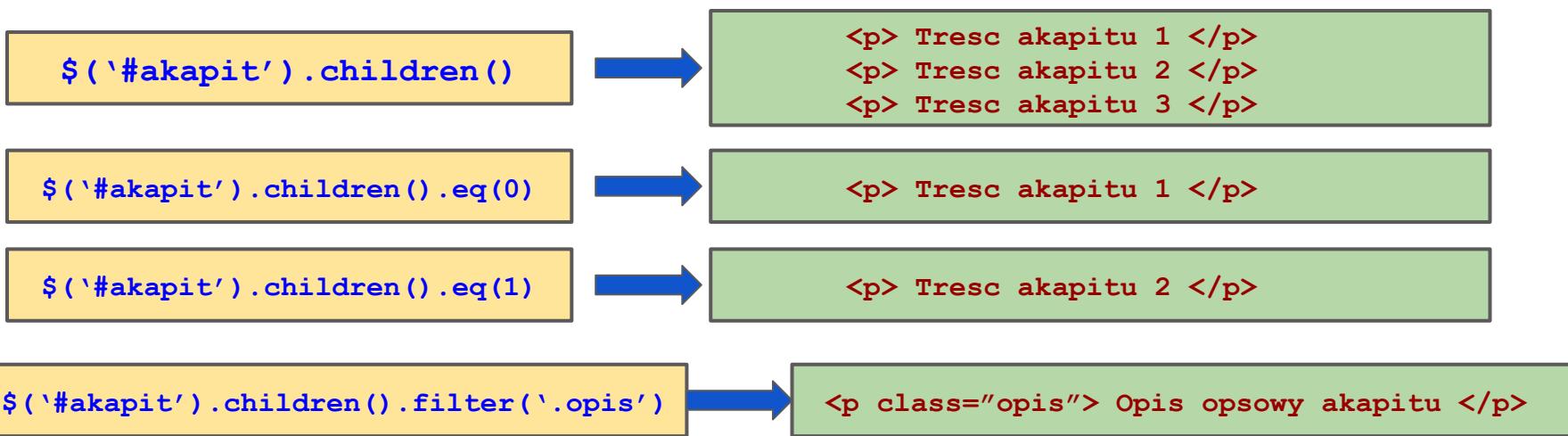
```
<p class="opis"> Opis opisowy akapitu </p>
```

Poruszanie się po strukturze DOM



Przykład:

```
<body>
  <div id="kontener">
    <h1 id="naglowek">Naglowek akapitu</h1>
    <div id="akapit">
      <p> Tresc akapitu 1 </p>
      <p> Tresc akapitu 2 </p>
      <p class="opis"> Opis opisowy akapitu </p>
    </div>
    <div id="stopka"></div>
  </div>
</body>
```





Wykorzystanie poznanych metod do zrobienia okna “pop-up””



```
<body>
  <h1 id="tytul">Profesor Adam Strzałkowski</h1>
  <div id="akapit">
    <p>Prof. dr hab. Adam Strzałkowski urodził się 26 listopada 1923 roku w
      Tenczynku. Studia fizyki ukończył na Uniwersytecie Jagiellońskim w 1948
      roku. Karierę zaczynał od astronomii w Obserwatorium Astronomicznym UJ u
      profesora Tadeusza Banachiewicza, którego uważa za swojego pierwszego
      mistrza.
    </p>
  </div>
</body>
```



Wykorzystanie poznanych metod do zrobienia okna “pop-up”



Chcemy aby po najechaniu na ten napis pojawiał się opis z dodatkowymi wyjaśnieniami np. jako okno pop-up.

```
<body>
  <h1 id="tytul">Profesor Adam Strzałkowski</h1>
  <div id="akapit">
    <p>Prof. dr hab. Adam Strzałkowski urodził się 26 listopada 1923 roku w Tenczynku. Studia fizyki ukończył na Uniwersytecie Jagiellońskim w 1948 roku. Karierę zaczynał od astronomii w Obserwatorium Astronomicznym UJ u profesora Tadeusza Banachiewicza, którego uważa za swojego pierwszego mistrza.
    </p>
  </div>
</body>
```



Wykorzystanie poznanych metod do zrobienia okna “pop-up”



Chcemy aby po najechaniu na ten napis pojawiał się opis z dodatkowymi wyjaśnieniami np. jako okno pop-up.

```
<body>
  <h1 id="tytul">Profesor Adam Strzałkowski</h1>
  <div id="akapit">
    <p>Prof. dr hab. Adam Strzałkowski urodził się 26 listopada 1923 roku w Tenczynku. Studia fizyki ukończył na Uniwersytecie Jagiellońskim w 1948 roku. Karierę zaczynał od astronomii w Obserwatorium Astronomicznym UJ u profesora Tadeusza Banachiewicza, którego uważa za swojego pierwszego mistrza.
    </p>
  </div>
</body>
```



Wykorzystanie poznanych metod do zrobienia okna “pop-up”



Chcemy aby po najechaniu na ten napis pojawiał się opis z dodatkowymi wyjaśnieniami np. jako okno pop-up.

```
<body>
  <h1 id="tytul">Profesor Adam Strzałkowski</h1>
  <div id="akapit">
    <p>Prof. dr hab. Adam Strzałkowski urodził się 26 listopada 1923 roku w Tenczynku. Studia fizyki ukończył na Uniwersytecie Jagiellońskim w 1948 roku. Karierę zaczynał od astronomii w Obserwatorium Astronomicznym UJ u profesora Tadeusza Banachiewicza, którego uważa za swojego pierwszego mistrza.
    </p>
  </div>
</body>
```

Na początek musimy zmodyfikować kod HTML



Wykorzystanie poznanych metod do zrobienia “pop-up”



```
<body>
    <h1 id="tytul">Profesor Adam Strzałkowski</h1>
    <div id="akapit">
        <p>Prof. dr hab. Adam Strzałkowski urodził się 26 listopada 1923 roku w
            Tenczynku. Studia fizyki ukończył na
            <span class="przypis">
                <span class="slowo">Uniwersytecie Jagiellońskim</span>
                <span class="wyraz">Uniwersytet Jagielloński w Krakowie</span>
                <span class="opis">Najstarszy uniwersytet w Polsce</span>
            </span>
            w 1948 roku. Karierę zaczynał od astronomii w Obserwatorium Astronomicznym
            UJ u profesora Tadeusza Banachiewicza, którego uważa za swojego pierwszego
            mistrza.
        </p>
    </div>
</body>
```



Wykorzystanie poznanych metod do zrobienia okna “pop-up”



```
<body>
  <h1 id="tytul">Profesor Adam Strzałka</h1>
  <div id="akapit">
    <p>Prof. dr hab. Adam Strzałkowski, urodzony w 1928 roku w Tenczynku. Studia fizyki ukončil w 1951 roku na Uniwersytecie Jagiellońskim</p>
    <span class="przypis">
      <span class="slowo">Uniwersytecie Jagiellońskim</span>
      <span class="wyraz">Uniwersytet Jagielloński w Krakowie</span>
      <span class="opis">Najstarszy uniwersytet w Polsce</span>
    </span>
    <p>w 1948 roku. Karierę zaczynał od astronomii w Obserwatorium Astronomicznym UJ u profesora Tadeusza Banachiewicza, którego uważa za swojego pierwszego mistrza.</p>
  </div>
</body>
```

To co zaznaczone zostało na kolor czerwony będzie treścią okna “pop-up” po najechaniu na przypis kursorem myszy.

Wykorzystanie poznanych metod do zrobienia “pop-up”



```
$function() {
  $('div').attr('id','popup').prependTo('.przypis').hide();
  $('.przypis').mouseover(function(){
    $('#popup').text('').
      append('<h3>' + $(this).children().eq(1).text() + '</h3>').
      append($(this).children().eq(2).text()).
      show();
  }).mouseout(function(){
    $('#popup').hide();
  }).mousemove(function(e){
    $('#popup').css('left', e.pageX + 10).css('top', e.pageY + 10);
  });
});
```

Wykorzystanie poznanych metod do zrobienia “pop-up”



```
$function() {
  $('div').attr('id','popup').prependTo('.przypis').hide();
  $('.przypis').mouseover(function(){
    $('#popup').text('').
      append('<h3>' + $(this).children().eq(1).text() + '</h3>').
      append($(this).children().eq(2).text()).
      show();
  }).mouseout(function(){
    $('#popup').hide();
  }).mousemove(function(e){
    $('#popup').css('left', e.pageX + 10).css('top', e.pageY + 10);
  });
});
```

Sam kod jQuery nie wystarczy trzeba jeszcze napisać reguły CSS stylizujące wygląd okna “pop-up”.

AJAX i jQuery



Już wcześniej poznaliśmy różnice pomiędzy synchronicznym i asynchronicznym modelem przetwarzania danych poprzez protokół HTTP.

Wykonywanie asynchronicznych zapytań AJAX było możliwe dzięki wykorzystaniu własności obiektu “XMLHttpRequest()”. Dzięki tej technologii możliwe jest np. uzupełnienie danego dokumentu HTML od dane pobrane z serwera bez konieczności przeładowania danego dokumentu HTML. Podobną funkcjonalność oferuje biblioteka jQuery.

AJAX i jQuery



Już wcześniej poznaliśmy różnice pomiędzy synchronicznym i asynchronicznym modelem przetwarzania danych poprzez protokół HTTP.

Wykonywanie asynchronicznych zapytań AJAX było możliwe dzięki wykorzystaniu własności obiektu “XMLHttpRequest()”. Dzięki tej technologii możliwe jest np. uzupełnienie danego dokumentu HTML od dane pobrane z serwera bez konieczności przeładowania danego dokumentu HTML. Podobną funkcjonalność oferuje biblioteka jQuery.

Możemy asynchronicznie pobrać dodatkowe dane z serwera a następnie korzystając z metod którymi możliwe jest modyfikowanie struktury DOM wyświetlić te dane użytkownikowi.

AJAX i jQuery



Już wcześniej poznaliśmy różnice pomiędzy synchronicznym i asynchronicznym modelem przetwarzania danych poprzez protokół HTTP.

Wykonywanie asynchronicznych zapytań AJAX było możliwe dzięki wykorzystaniu własności obiektu “XMLHttpRequest()”. Dzięki tej technologii możliwe jest np. uzupełnienie danego dokumentu HTML od dane pobrane z serwera bez konieczności przeładowania danego dokumentu HTML. Podobną funkcjonalność oferuje biblioteka jQuery.

Możemy asynchronicznie pobrać dodatkowe dane z serwera a następnie korzystając z metod którym możliwe jest modyfikowanie struktury DOM wyświetlić te dane użytkownikowi.

W jQuery istnieje wiele metod pozwalających na asynchroniczne działanie kodu. Jedną z nich jest metoda `.load()`.

Pozwala na ustalenie zawartości elementu treścią pobraną z adresu URL podanego jako argument metody.

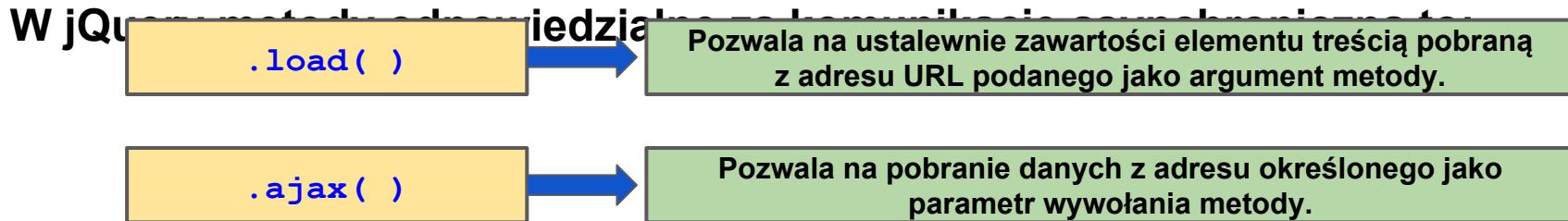
AJAX i jQuery



Już wcześniej poznaliśmy różnice pomiędzy synchronicznym i asynchronicznym modelem przetwarzania danych poprzez protokół HTTP.

Wykonywanie asynchronicznych zapytań AJAX było możliwe dzięki wykorzystaniu własności obiektu “XMLHttpRequest()”. Dzięki tej technologii możliwe jest np. uzupełnienie danego dokumentu HTML od dane pobrane z serwera bez konieczności przeładowania danego dokumentu HTML. Podobną funkcjonalność oferuje biblioteka jQuery.

Możemy asynchronicznie pobrać dodatkowe dane z serwera a następnie korzystając z metod którym możliwe jest modyfikowanie struktury DOM wyświetlić te dane użytkownikowi.





AJAX i jQuery



Spróbujmy najpierw określić możliwości działania metody “load()”:

```
$( selektor ).load( URL, parametry-URL, function() );
```



AJAX i jQuery



Spróbujmy najpierw określić możliwości działania metody “load()”:

```
$( selektor ).load( URL, parametry-URL, function() );
```

Przykład wywołania - zakładamy że ładujemy treść z strony “tresc.html”

```
$(function() {
    $('#tresc').load('tresc.html', 'bcs=34&sort=asc', function(html) {
        $('span#info').text('Tresc zostala zaladowana!');
    });
});

<body>
    <div id="tresc"></div>
    <span id="info"></span>
</body>
```



AJAX i jQuery



Spróbujmy najpierw określić możliwości działania metody “load()”:

```
$( selektor ).load( URL, parametry-URL, function() );
```

Przykład wywołania - zakl.

Postać wywoływanego adresu:

“**tresc.html?bcs=34&sort=asc**”

“**tresc.html**”

```
$(function() {
    $('#tresc').load('tresc.html', 'bcs=34&sort=asc', function(html) {
        $('span#info').text('Tresc zostala zaladowana!');
    });
});

<body>
    <div id="tresc"></div>
    <span id="info"></span>
</body>
```



AJAX i jQuery



Spróbujmy najpierw określić możliwości działania metody “load()”:

```
$( selektor ).load( URL, parametry-URL, function() );
```

Przykład wywołania - zakładamy że ładowamy :

W argumencie przechowywane są
pobrane dane z adresu URL.

```
$function() {
    $('#tresc').load('tresc.html', 'bcs=34&sort=asc', function(html) {
        $('span#info').text('Tresc zostala zaladowana!');
    });
}

<body>
    <div id="tresc"></div>
    <span id="info"></span>
</body>
```

AJAX i jQuery



Spróbujmy najpierw określić możliwości działania metody “load()”:

```
$( selektor ).load( URL, parametry-URL, function() );
```

Przykład wywołania - zakładamy że ładowamy :

W argumencie przechowywane są
pobrane dane z adresu URL.

```
$function() {
    $('#tresc').load('tresc.html', 'bcs=34&sort=asc', function(html) {
        $('span#info').text('Tresc zostala zaladowana!');
    });
}

<body>
    <div id="tresc"></div>
    <span id="info"></span>
</body>
```

Działanie metody “load()” przypomina funkcjonalnością metodę
“include()” z języka PHP.



AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$.ajax( {  
    type: "POST",  
    url: "http://www.jag.tu.pl/~kamil/test.html",  
    data: "id=1&name=kamil",  
    success: function(response) {  
        alert(response);  
    }  
});
```



AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$ .ajax( {  
    url: URL,  
    timeout: ms,  
    cache: true/false,  
    success: function(html){},  
    beforeSend: function(){},  
    error: function(){  
});
```



AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$ .ajax( {  
    url: URL,  
    timeout: ms,  
    cache: true/false,  
    success: function(html){},  
    beforeSend: function(){},  
    error: function(){  
});
```

Metoda “ajax()” nie działa na żadnym obiekcie drzewa DOM !!!!

Parametry do wywołania tej metody przekazywane są w postaci obiektu (tablicy asocjacyjnej) określającej parametry połączenia oraz zdania jakie mają zostać wykonane po zakończeniu żądania.

AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$ .ajax( {  
    url: URL,  
    timeout: ms,  
    cache: true/false,  
    success: function(html){},  
    beforeSend: function(){},  
    error: function()  
});
```

adres URL

AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$ .ajax( {  
    url: URL,  
    timeout: ms,  
    cache: true/false,  
    success: function(html){},  
    beforeSend: function(){},  
    error: function()  
});
```

adres URL

AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$ajax( {  
    url: URL,  
    timeout: ms,  
    cache: true/false,  
    success: function(html){},  
    beforeSend: function(){},  
    error: function(){  
});
```

adres URL

maksymalny czas oczekiwania
na odpowiedz serwera

AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$ajax( {  
    url: URL,  
    timeout: ms,  
    cache: true/false,  
    success: function(html){},  
    beforeSend: function(){},  
    error: function()  
});
```

adres URL

maksymalny czas oczekiwania
na odpowiedz serwera

wyłącza stosowanie pamięci
podręcznej przeglądarki

AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$ajax( {  
    url: URL,  
    timeout: ms,  
    cache: true/false,  
    success: function(html){},  
    beforeSend: function(){},  
    error: function(){  
});
```

adres URL

maksymalny czas oczekiwania
na odpowiedz serwera

wyłącza stosowanie pamięci
podręcznej przeglądarki

wywołanie zwrotne
realizowane w przypadku
powodzenia żądania

AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$ajax( {  
    url: URL,  
    timeout: ms,  
    cache: true/false,  
    success: function(html){},  
    beforeSend: function(){},  
    error: function(){  
});
```

adres URL

maksymalny czas oczekiwania
na odpowiedz serwera

wyłącza stosowanie pamięci
podręcznej przeglądarki

wywołanie zwrotne
realizowane w przypadku
powodzenia żądania

wywołanie zwrotne realizowane w
przed wysłaniem żądania

AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$ajax( {  
    url: URL,  
    timeout: ms,  
    cache: true/false,  
    success: function(html){},  
    beforeSend: function(){},  
    error: function(){}  
});
```

adres URL

maksymalny czas oczekiwania
na odpowiedz serwera

wyłącza stosowanie pamięci
podręcznej przeglądarki

wywołanie zwrotne
realizowane w przypadku
powodzenia żądania

wywołanie zwrotne
realizowane w przypadku
niepowodzenia żądania

AJAX i jQuery



Przykład 1:

```
$(document).ready(function() {
    $.ajax({
        url: "localhost/api/getDataHtml",
        success: function(html) {
            $("#tresp").html(html);
        }
    });
});<body>
<div id="tresp"></div>
</body>
```

Pobieramy asynchronicznie dane ze adresu URL i w razie sukcesu zakończenia żądania wyświetlamy treść (html) w elemencie #tresp.

AJAX i jQuery



Przykład 2:

```
$().ready(function() {
    $.ajax({
        url: "localhost/api/getDataJson",
        dataType: "json",
        success: function(json) {
            $("#tresc").html(JSON.stringify(json));
        }
    });
});
```

określamy typ danych
przychodzących

Pobieramy asynchronicznie dane ze adresu URL i w razie sukcesu zakończenia żądania wyświetlamy treść (json) w elemencie #tresc.

AJAX i jQuery



Przykład 3:

```
$
$
$
$(function() {
  $.ajax({
    url: 'localhost/api/getData',
    type: 'GET',
    data: 'Username=jquery4u',
    success: function(data) {
      $('#results').html(data);
    },
    error: function(e) {
      console.log(e.message);
    }
  });
})
$
```

określamy metody wysyłania danych

przesyłane dane w metodzie GET

AJAX i jQuery



Przykład 4:

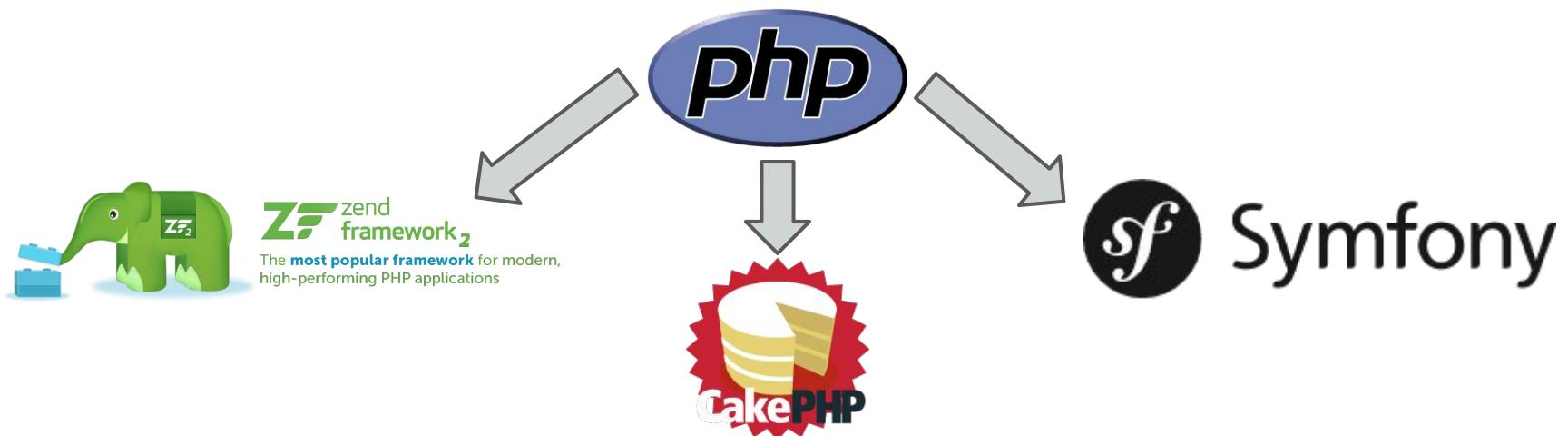
```
$
$
$
$
$
$ $(function() {
$   $.ajax({
$     method: 'POST',
$     url: 'localhost/api/sendData',
$     data: { name: "Jan", fname: "Kowalski" },
$     success: function() {
$       alert( 'Dane zostały przesłane ' );
$     });
$   });
$ }) ;
```

określamy metody wysyłania danych



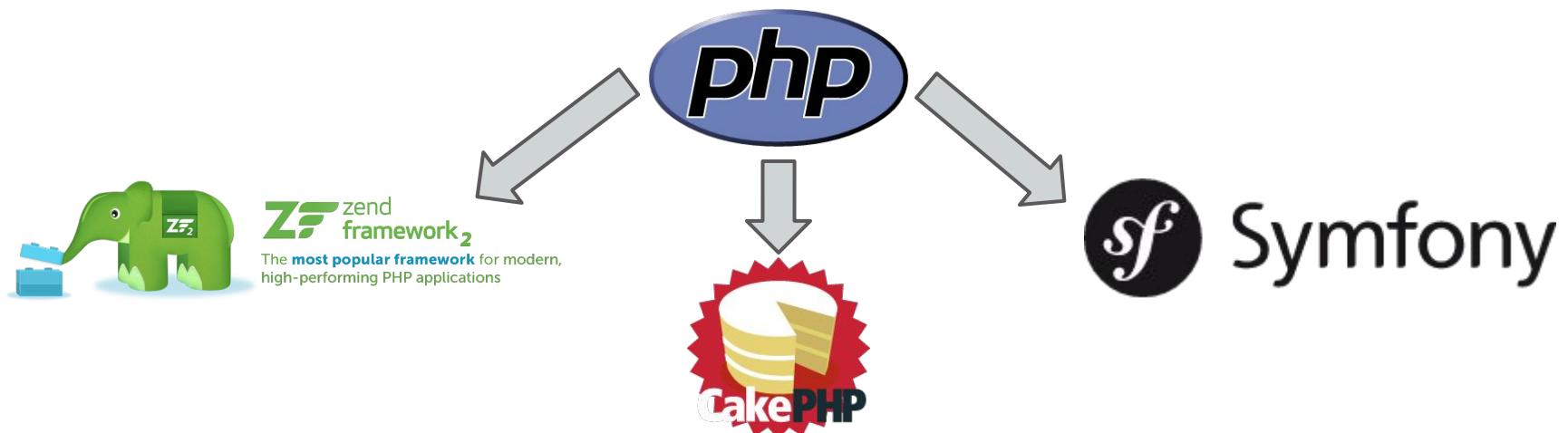
Popularne dostępne rozwiązania

Najpopularniejsze środowiska programistyczne:



Popularne dostępne rozwiązania

Najpopularniejsze środowiska programistyczne:



oraz systemy CMS (Content Manager System):

Drupal™

PHP FUSION

Joomla!™

WORDPRESS

PHP nuke

MediaWiki
Because ideas want to be free.

Dlaczego NODE.JS a nie PHP



vs.



Obecnie PHP jest wykorzystywane na około 75% wszystkich stron internetowych.

Dlaczego NODE.JS a nie PHP



vs.



Obecnie PHP jest wykorzystywane na około 75% wszystkich stron internetowych.

Wiele nowych projektów odchodzi od PHP na rzecz NODE.JS jako rozwiązania bardziej wydajnego, elastycznego i niezależnego.

Dlaczego NODE.JS a nie PHP



vs.

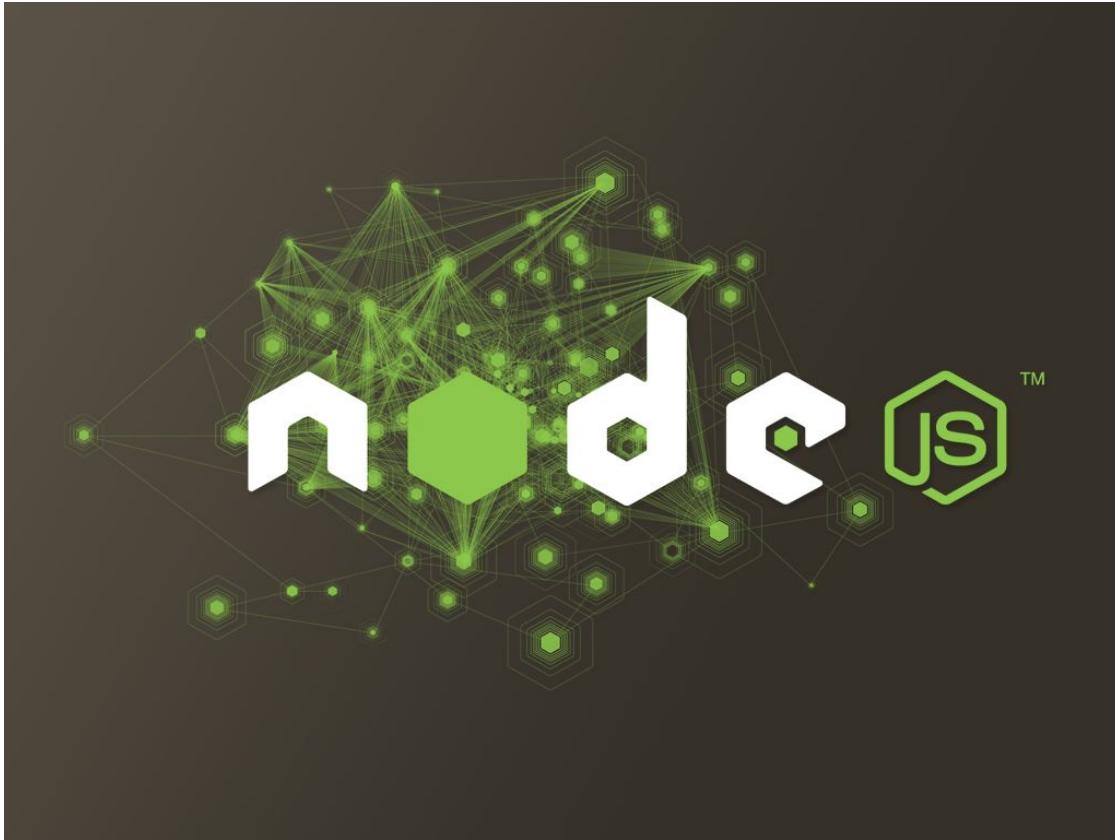


Obecnie PHP jest wykorzystywane na około 75% wszystkich stron internetowych.

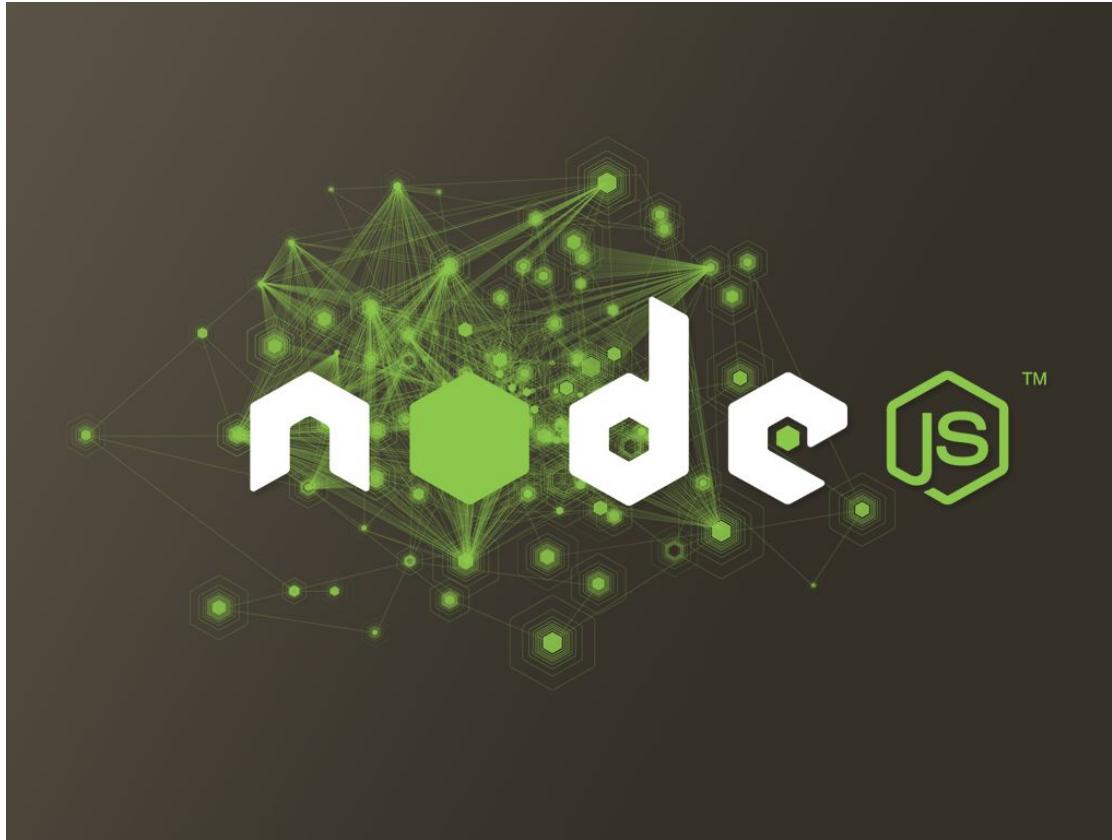
Wiele nowych projektów odchodzi od PHP na rzecz NODE.JS jako rozwiązania bardziej wydajnego, elastycznego i niezależnego.

Daje możliwości tworzenia łatwo skalowalnych aplikacji internetowych, których działanie jest sterowane zdarzeniowo wykorzystując żądania asynchroniczne czyli bez konieczności przeładowania danego zasobu.

Node.js



Node.js



<http://nodejs.org/>

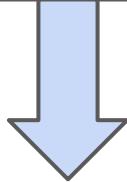


Node.js

Co to jest NODE.JS ?

Node.js

Co to jest NODE.JS ?



Nowoczesne środowisko
programistyczne

Środowisko programistyczne w sensie zestawu gotowych klas i metod których można używać do przygotowania własnych skalowalnych i wydajnych aplikacji internetowych.

Krótka historia Node.js

Środowisko Node.js zostało stworzone w 2009 roku, przez Ryana Dahlę, który w tamtym czasie pracował w firmie Joyent.



Ryan Dahl



Krótka historia Node.js

Środowisko Node.js zostało stworzone w 2009 roku, przez Ryana Dahlę, który w tamtym czasie pracował w firmie Joyent.



Oficjalna premiera i prezentacja pakietu odbyła się podczas wykładu na konferencji JSConf EU 2009

Ryan Dahl



Krótka historia Node.js

Node.js jest środowiskiem programistycznym napisanym w języku JavaScript. Aplikacje stworzone w środowisku Node.js są sterowane zdarzeniami wykorzystując system wejścia/wyjścia (I/O) który jest asynchroniczny.



Ryan Dahl

Krótka historia Node.js

Node.js jest środowiskiem programistycznym napisanym w języku JavaScript. Aplikacje stworzone w środowisku Node.js są sterowane zdarzeniami wykorzystując system wejścia/wyjścia (I/O) który jest asynchroniczny.



Ryan Dahl



NODE.JS ON THE ROAD

PRODUCTION NODE HITS 1
GET INSPIRED & INVOLVED

<http://nodejs.org/>

Krótka historia Node.js

Node.js jest środowiskiem programistycznym napisanym w języku JavaScript. Aplikacje stworzone w środowisku Node.js są sterowane zdarzeniami wykorzystując system wejścia/wyjścia (I/O) który jest asynchroniczny.



Ryan Dahl



NODE.JS ON THE ROAD

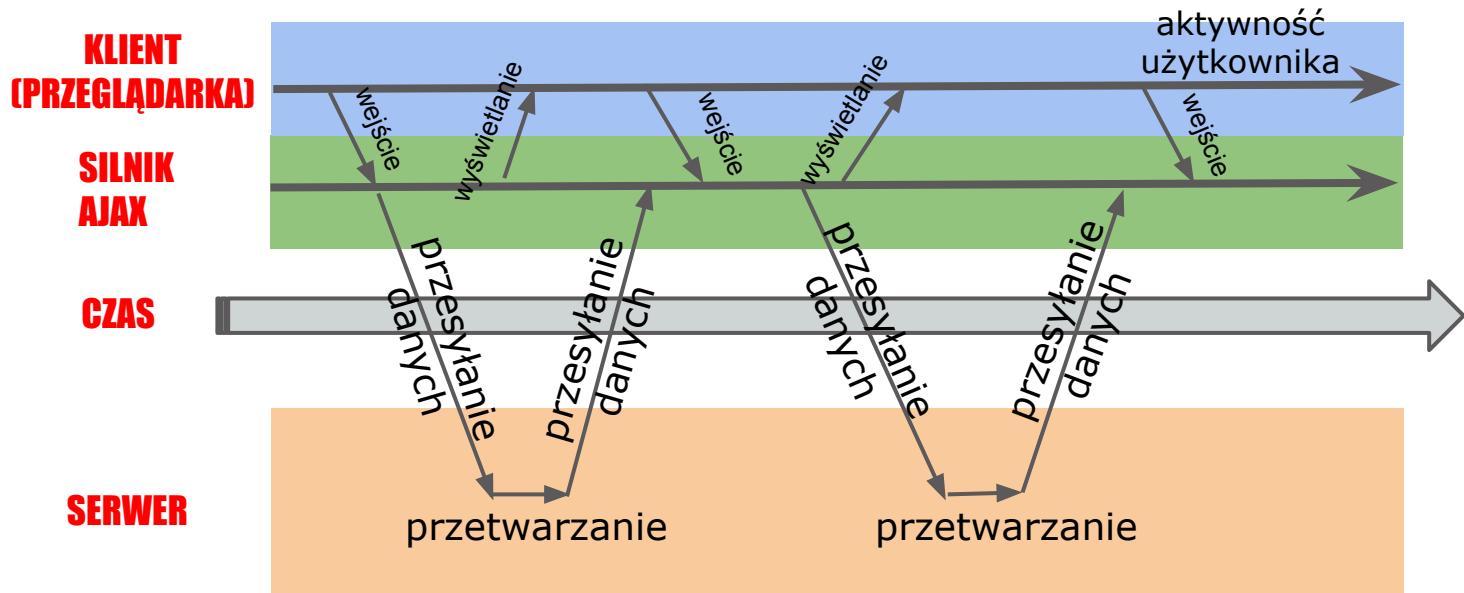
PRODUCTION NODE HITS 1
GET INSPIRED & INVOLVED

<http://nodejs.org/>

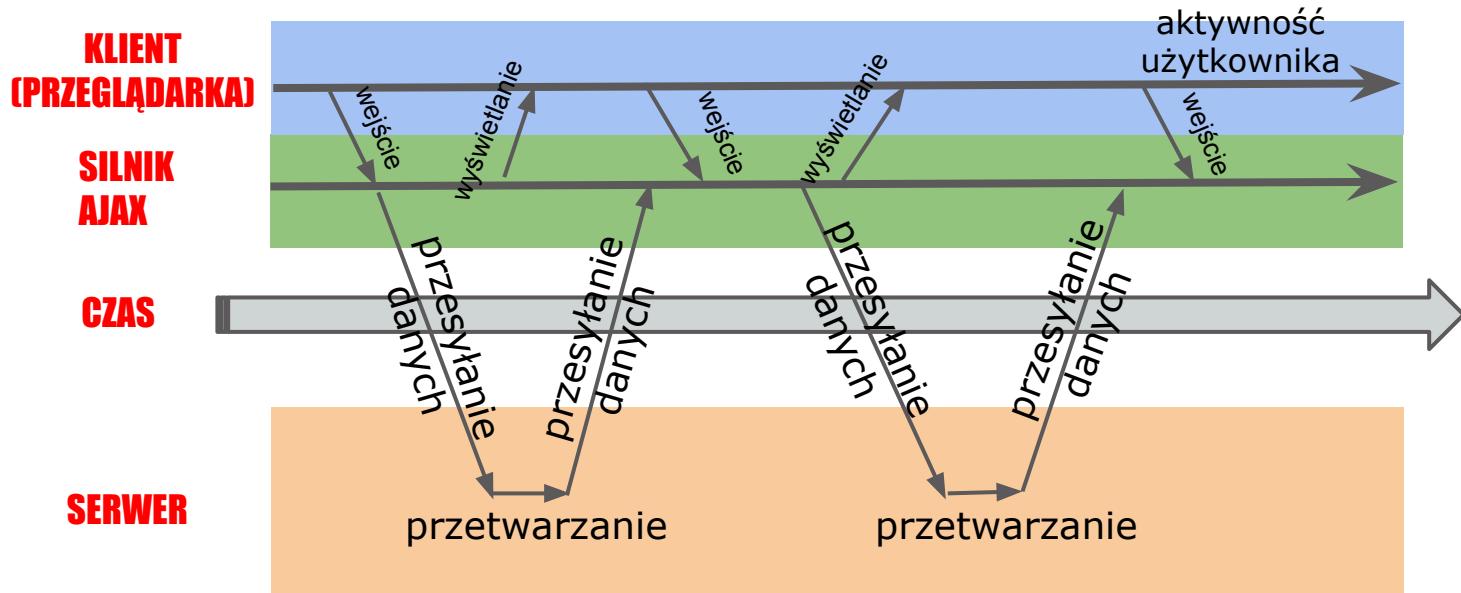
Najnowsza wersja: v.5.1.0

(17 października 2015 r.)

Przypomnienie: asynchroniczność



Przypomnienie: asynchroniczność



Jest to model asynchronicznej komunikacji HTTP, gdzie klient (przeglądarka) nie czeka na przyjście odpowiedzi na żądanie serwera, a wykonuje dalsze żądania.

W takim modelu nie ma konieczności przeładowania strony przy każdej operacji klineta, wystarczy, że zostaną doczytane brakujące dane, a dzięki odpowiednim narzędziom zmodyfikowana zostanie zawartość strony.



Krótka historia Node.js

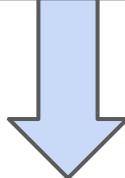
Powstanie Node.js było zainspirowane przez funkcjonalności “push” jakie oferuje np. poczta GMAIL...

Co to jest “push” ?

Krótka historia Node.js

Powstanie Node.js było zainspirowane przez funkcjonalności “push” jakie oferuje np. poczta GMAIL...

Co to jest “push” ?

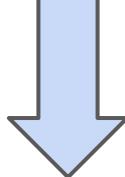


Usługi push działają w oparciu o przekazane przez klienta wcześniej informacje, na podstawie których serwer dostarcza klientowi nowych danych w zależności od tego czy są one dostępne.

Krótka historia Node.js

Powstanie Node.js było zainspirowane przez funkcjonalności “push” jakie oferuje np. poczta GMAIL...

Co to jest “push” ?



Usługi push działają w oparciu o przekazane przez klienta wcześniej informacje, na podstawie których serwer dostarcza klientowi nowych danych w zależności od tego czy są one dostępne.

Przykład:

Przykładem usługi “push” jest synchroniczny chat.



Krótka historia Node.js

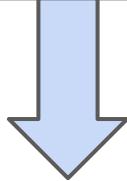
Node. jest czysto JavaScriptowym środowiskiem tworzenia stron internetowych działającym po stronie serwera.

Do tej pory uczyliśmy się że...

Krótka historia Node.js

Node. jest czysto JavaScriptowym środowiskiem tworzenia stron internetowych działającym po stronie serwera.

Do tej pory uczyliśmy się że...

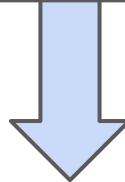


JavaScript jest wykonywany po stronie klineta (przeglądarki)!

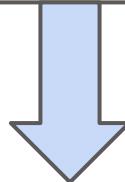
Krótka historia Node.js

Node.js jest czysto JavaScriptowym środowiskiem tworzenia stron internetowych działającym po stronie serwera.

Do tej pory uczyliśmy się że...



JavaScript jest wykonywany po stronie klineta (przeglądarki)!



Jak zatem jest możliwe że Node.js działa na serwerze ?

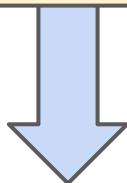


Google V8

Jak zatem jest możliwe że Node.js działają na serwerze ?

Google V8

Jak zatem jest możliwe że Node.js działają na serwerze ?



Silnik V8 został stworzony na potrzeby przeglądarki Chrome, do obsługi skryptów napisanych w języku JavaScript.

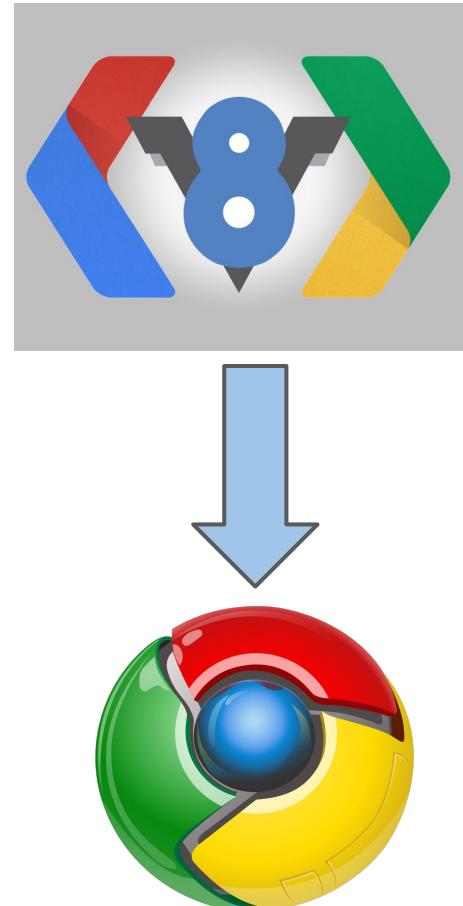
Google V8

Silnik V8 został stworzony przez firmę Google na potrzeby przeglądarki Chrome, do obsługi skryptów napisanych w języku JavaScript.



Google V8

Silnik V8 został stworzony przez firmę Google na potrzeby przeglądarki Chrome, do obsługi skryptów napisanych w języku JavaScript.



Jest to program napisany w języku C++, który “kompiluje” skrypty do kodu maszynowego, a dopiero następnie wykonuje. Dzięki takiej procedurze uzyskano wzrost wydajności i szybkości wykonywania skryptów JS.

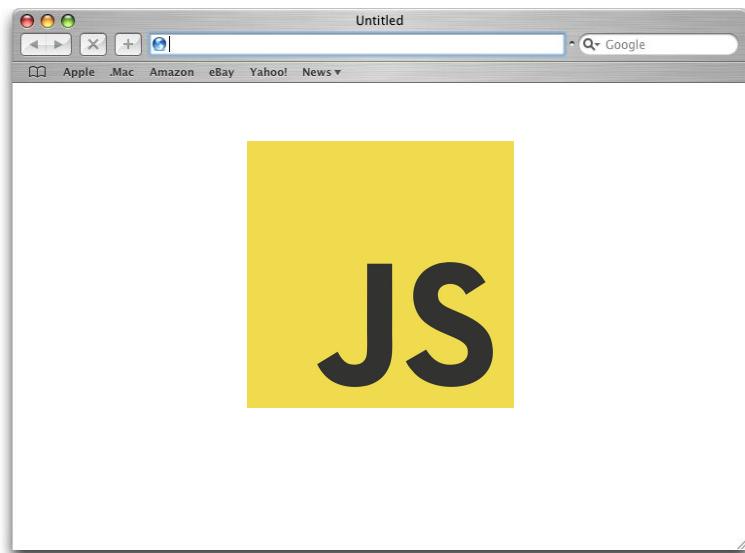


JavaScript po stronie serwera

Dzięki zastosowaniu silnika V8 wprowadzono zupełnie nowe podejście w myśleniu o wykonywaniu kodu JavaScript

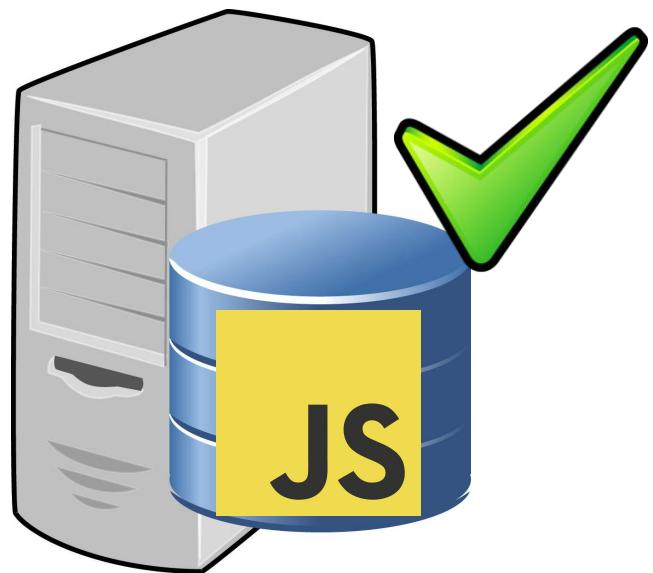
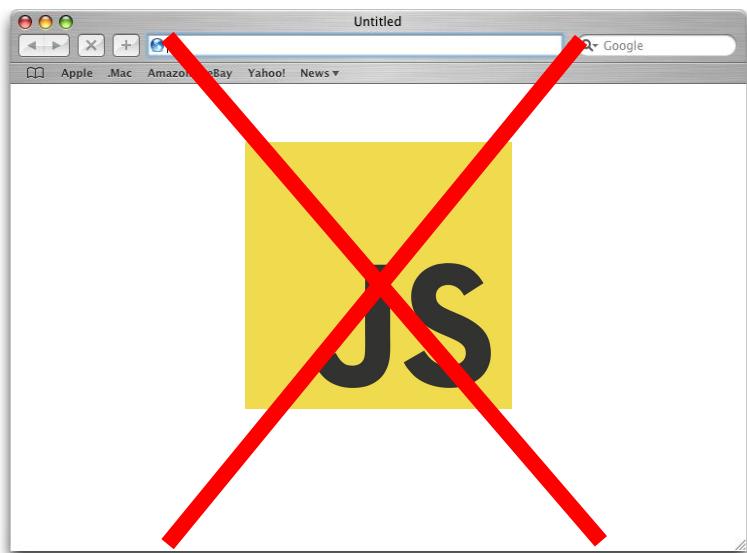
JavaScript po stronie serwera

Dzięki zastosowaniu silnika V8 wprowadzono zupełnie nowe podejście w myśleniu o wykonywaniu kodu JavaScript



JavaScript po stronie serwera

Dzięki zastosowaniu silnika V8 wprowadzono zupełnie nowe podejście w myśleniu o wykonywaniu kodu JavaScript



JavaScript po stronie serwera

Musimy odwrócić sposób myślenia jeśli chodzi wykonywanie JavaScriptu.





JavaScript po stronie serwera

SERVER SIDE

CLIENT SIDE

JavaScript po stronie serwera

SERVER SIDE

```
1 var edge = require('edge');
2
3 var hello = edge.func(function () {/*
4     async (input) =>
5     {
6         return ".NET welcomes " + input.ToString();
7     }
8 */});
9
10 hello('Node.js', function (error, result) {
11     if (error) throw error;
12     console.log(result);
13});
```

CLIENT SIDE

JavaScript po stronie serwera

SERVER SIDE

```
1 var edge = require('edge');
2
3 var hello = edge.func(function () {/*
4     async (input) =>
5     {
6         return ".NET welcomes " + input.ToString();
7     }
8 */});
9
10 hello('Node.js', function (error, result) {
11     if (error) throw error;
12     console.log(result);
13});
```

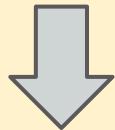


CLIENT SIDE

JavaScript po stronie serwera

SERVER SIDE

```
1 var edge = require('edge');
2
3 var hello = edge.func(function () {/*
4     async (input) =>
5     {
6         return ".NET welcomes " + input.ToString();
7     }
8 */});
9
10 hello('Node.js', function (error, result) {
11     if (error) throw error;
12     console.log(result);
13});
```



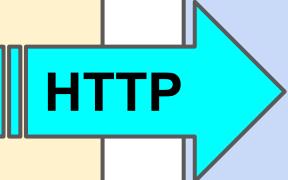
CLIENT SIDE

JavaScript po stronie serwera

SERVER SIDE

```
1 var edge = require('edge');
2
3 var hello = edge.func(function () {/*
4     async (input) =>
5     {
6         return ".NET welcomes " + input.ToString();
7     }
8 */});
9
10 hello('Node.js', function (error, result) {
11     if (error) throw error;
12     console.log(result);
13});
```

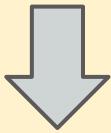
CLIENT SIDE



JavaScript po stronie serwera

SERVER SIDE

```
1 var edge = require('edge');
2
3 var hello = edge.func(function () {/*
4     async (input) =>
5     {
6         return ".NET welcomes " + input.ToString();
7     }
8 */});
9
10 hello('Node.js', function (error, result) {
11     if (error) throw error;
12     console.log(result);
13});
```



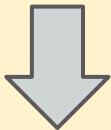
CLIENT SIDE

```
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>CSS3</title>
<link href="style.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="container"> <!-- Main Container -->
<div class="menu"> <!-- Menu here --></div>
<div class="article"><h2>CSS3 Sample</h2>
<p>CSS3 & HTML5 are so good!</p>
</div>
</div>
```

JavaScript po stronie serwera

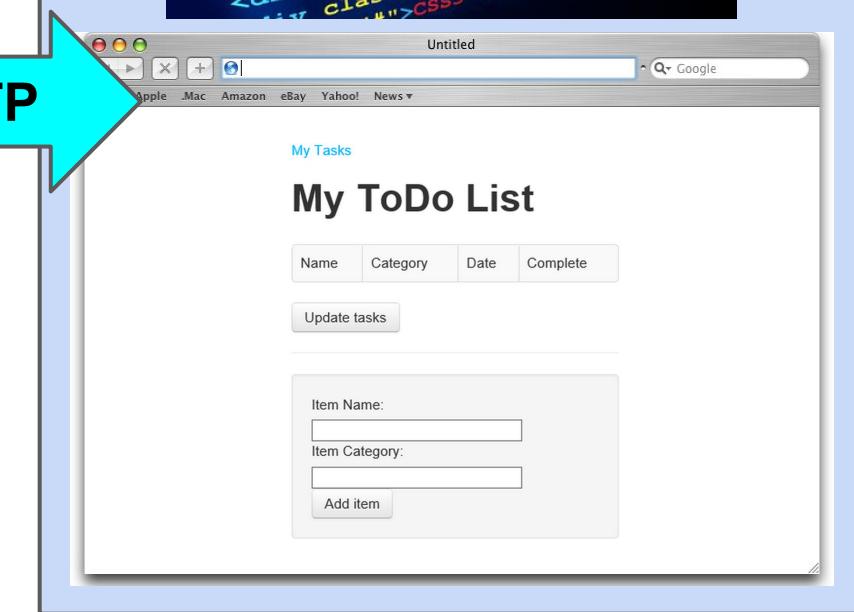
SERVER SIDE

```
1 var edge = require('edge');
2
3 var hello = edge.func(function () {/*
4     async (input) =>
5     {
6         return ".NET welcomes " + input.ToString();
7     }
8 */});
9
10 hello('Node.js', function (error, result) {
11     if (error) throw error;
12     console.log(result);
13});
```



CLIENT SIDE

```
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>CSS3</title>
<link href="style.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="container"> <!-- Main Container -->
<div class="menu"> <!-- Menu here --></div>
<div class="article"><h2>CSS3 Sample</h2>
<p>CSS3 & HTML5 are so good!</p>
</div>
</div>
```



HTTP



Obsługa żądań

MODEL TRADYCYJNY

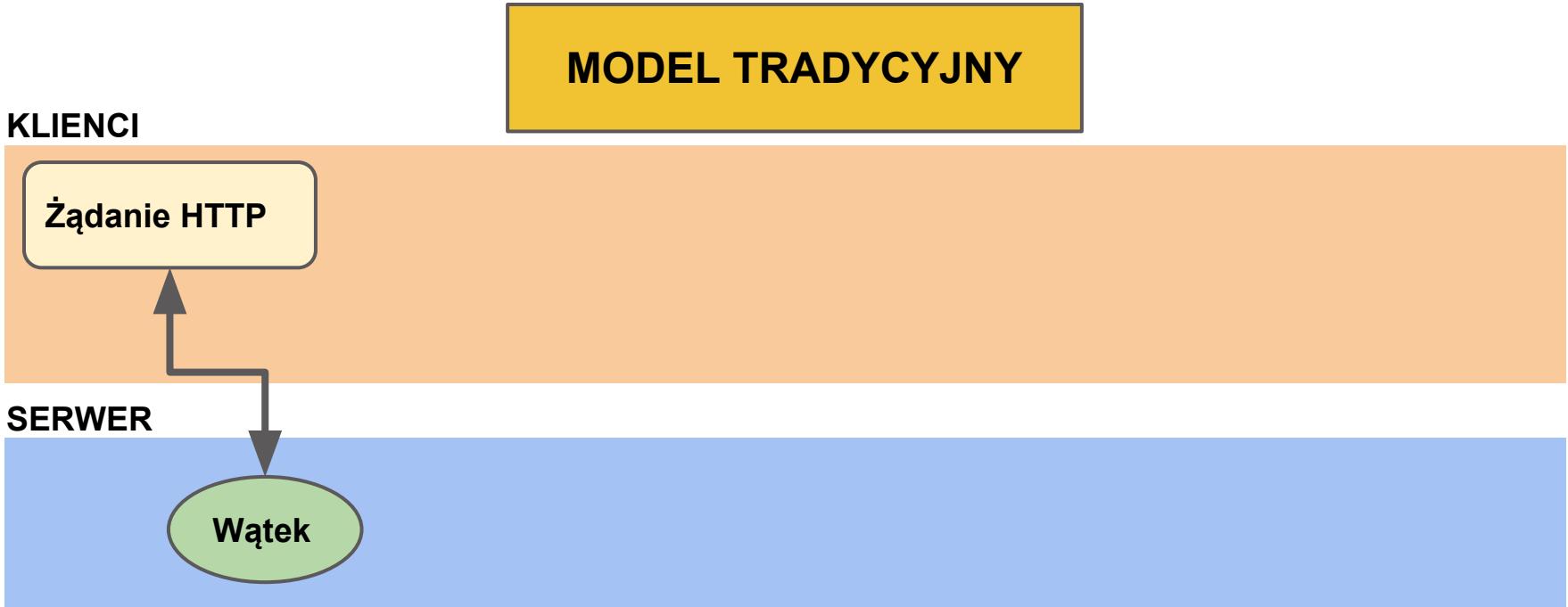
KLIENCI



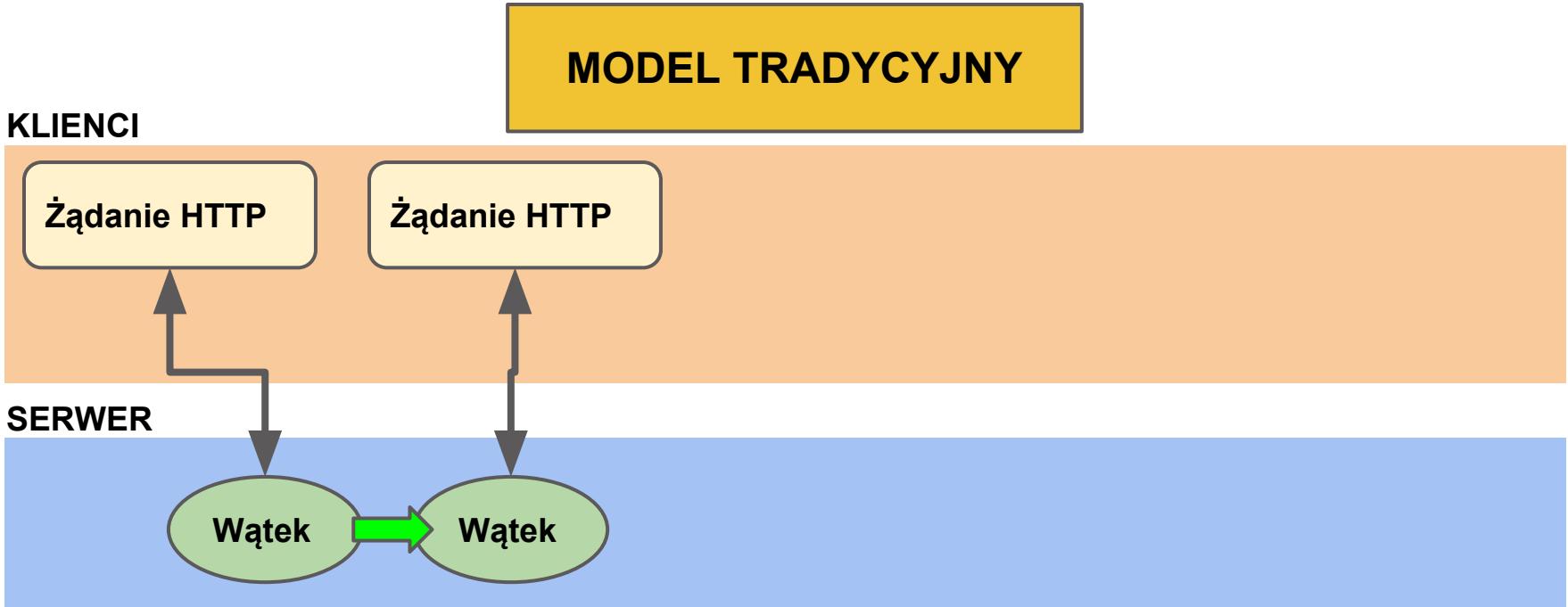
SERWER



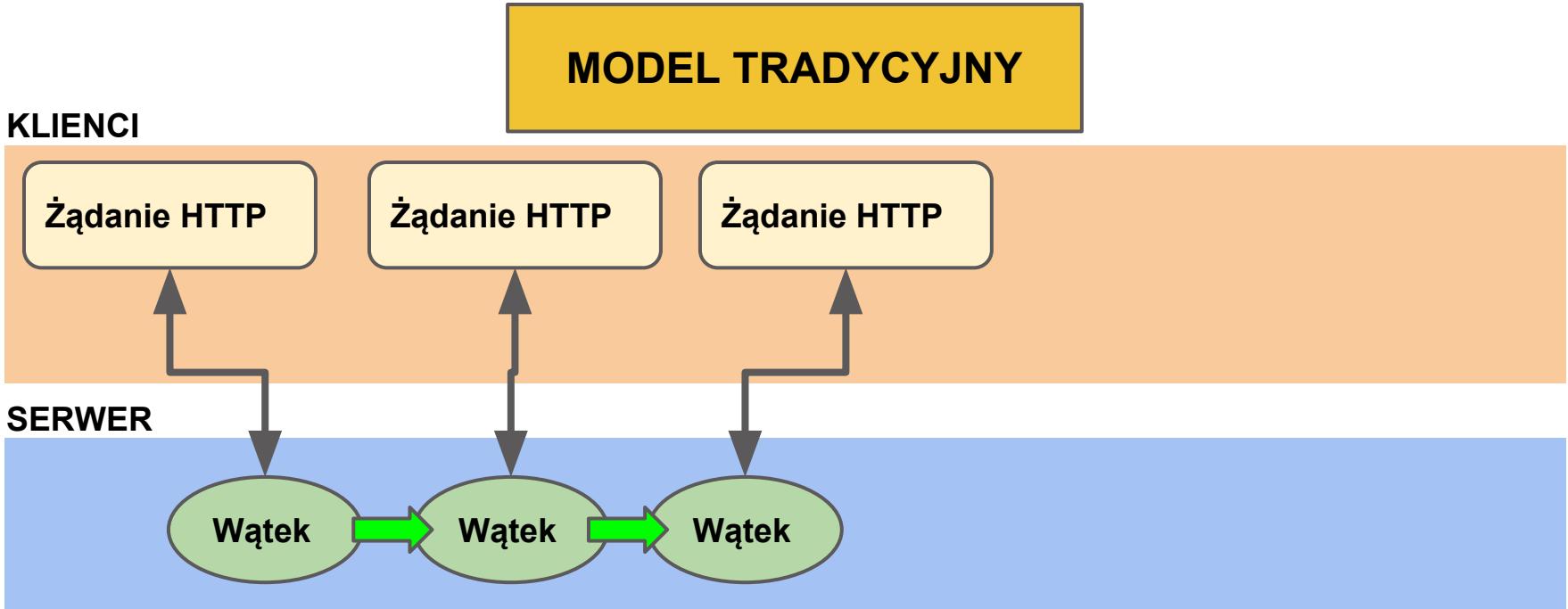
Obsługa żądań



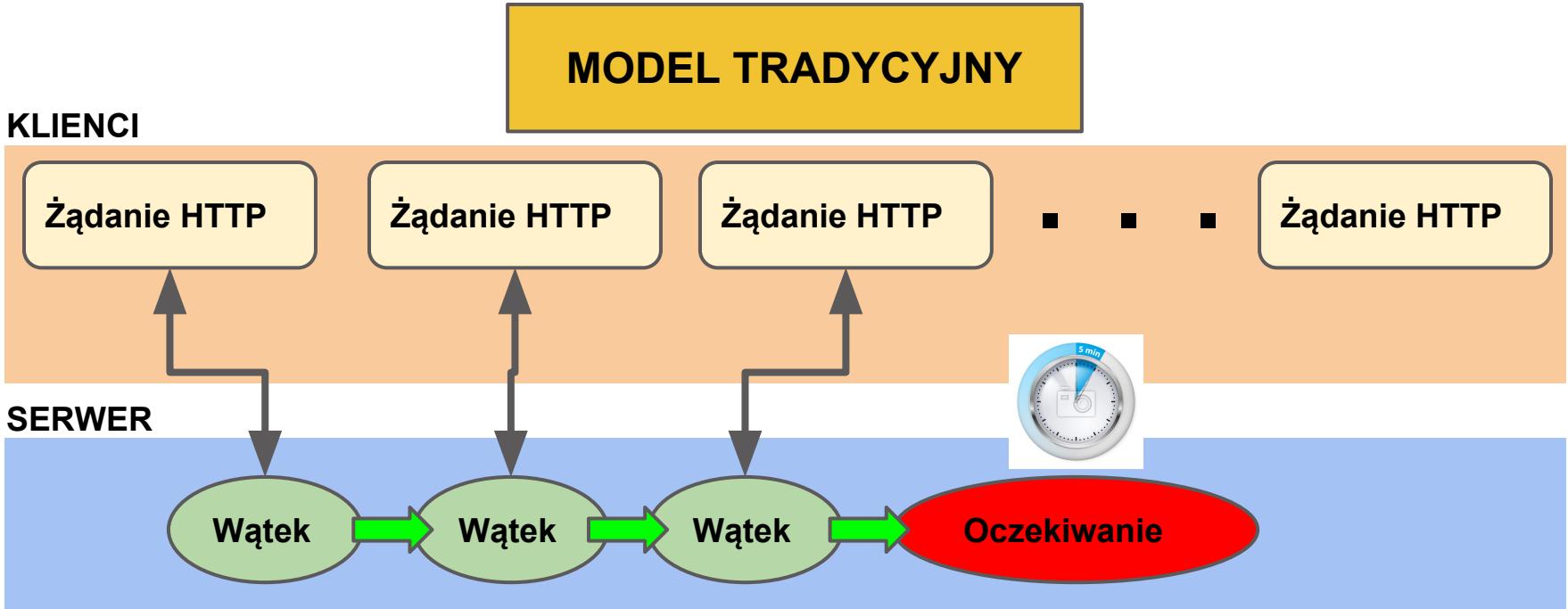
Obsługa żądań



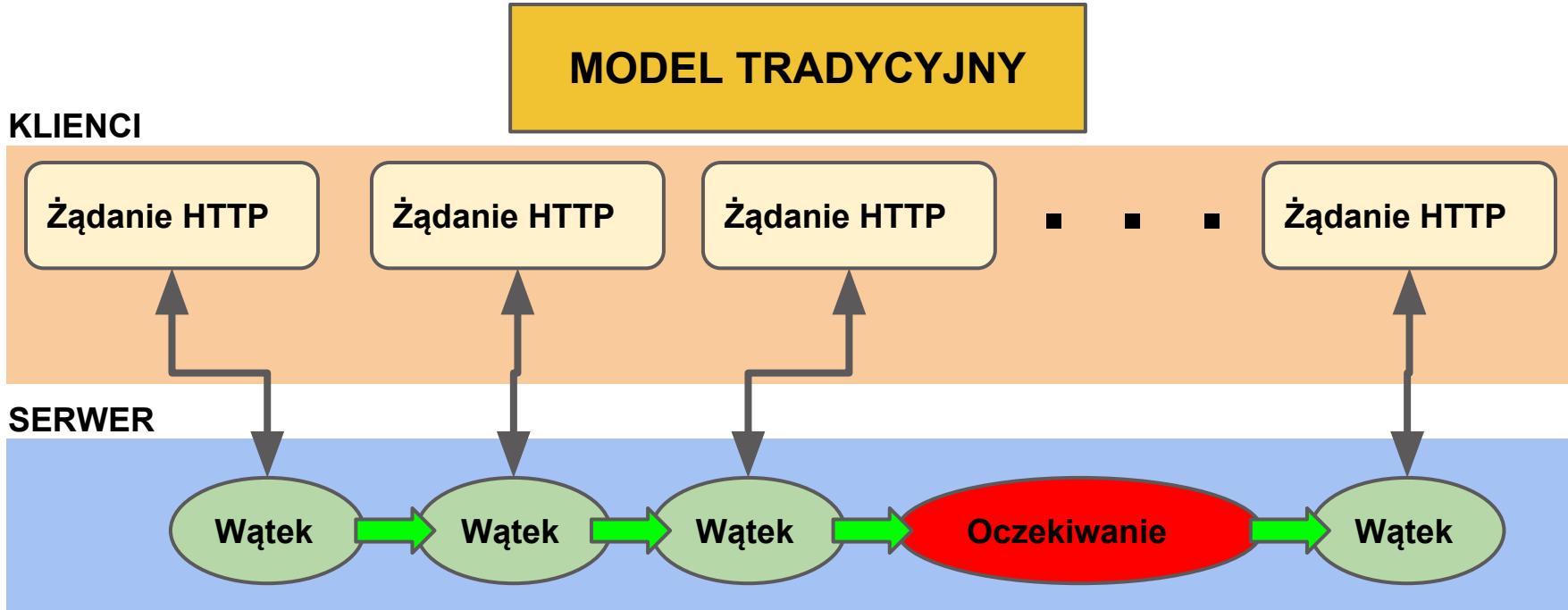
Obsługa żądań



Obsługa żądań

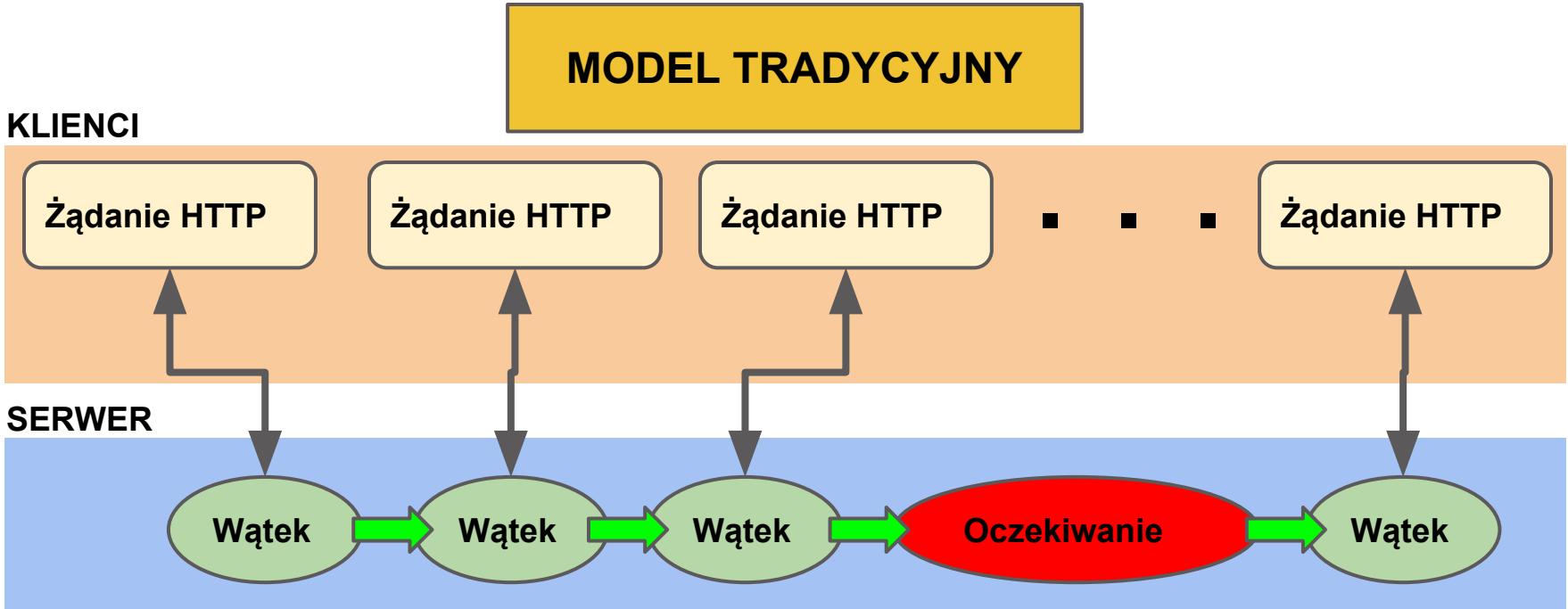


Obsługa żądań



W tym modelu serwer do obsługi każdego żadania musi stworzyć osobny wątek z ograniczonej puli jaką może obsłużyć CPU. Powoduje to że przy dużej ilości żądań niektóre z nich muszą czekać w “kolejce” na zrealizowanie. W wyniku czego serwer zaczyna zaczyna wolniej działać co wydłuża czas oczekiwania na odpowiedź.

Obsługa żądań



Przykład:

Dla systemu z 8GB pamięci RAM przydzielającego 2MB pamięci na wątek możemy obsłużyć maksymalnie w tym samym czasie **4000 żądań** (w rzeczywistości jest to mniej ponieważ zużywamy jeszcze pamięć na inne operacje).



Obsługa żądań

MODEL ZDARZENIOWY

KLIENCI



SERWER



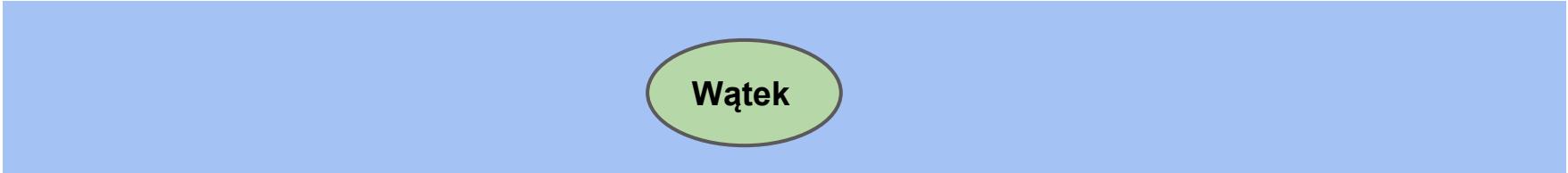
Obsługa żądań

MODEL ZDARZENIOWY

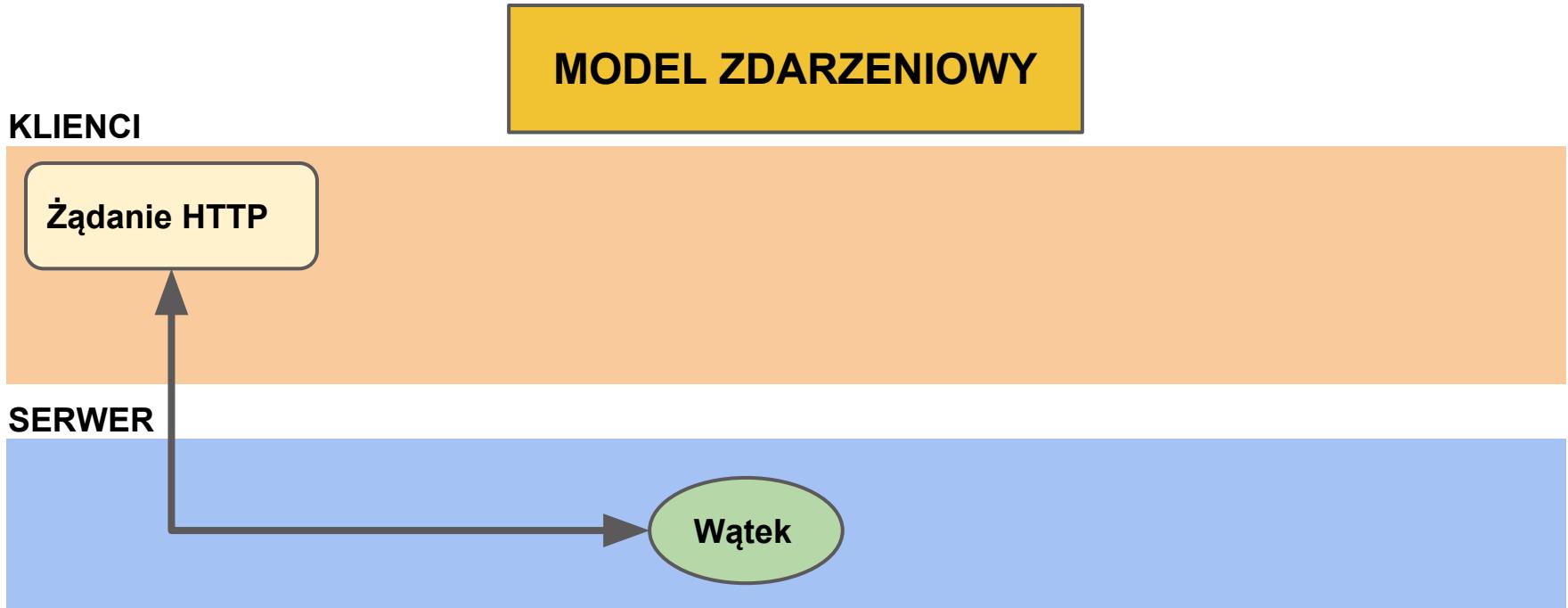
KLIENCI



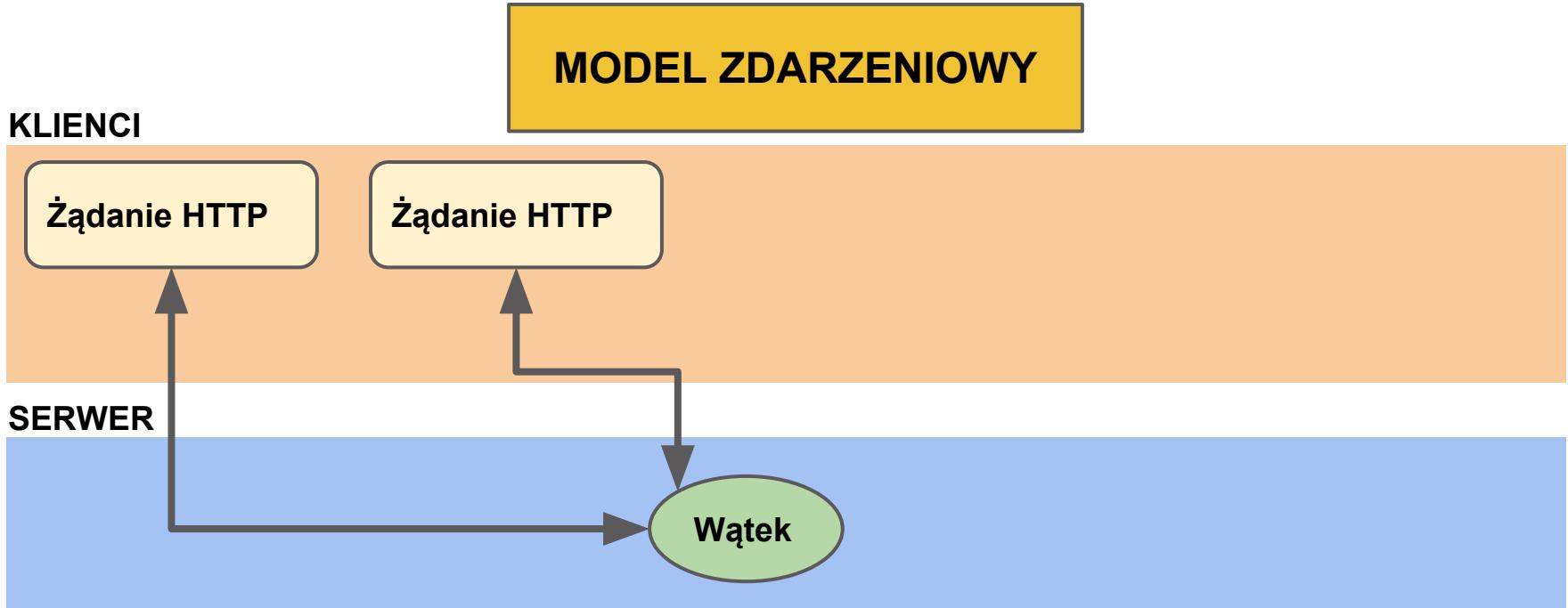
SERWER



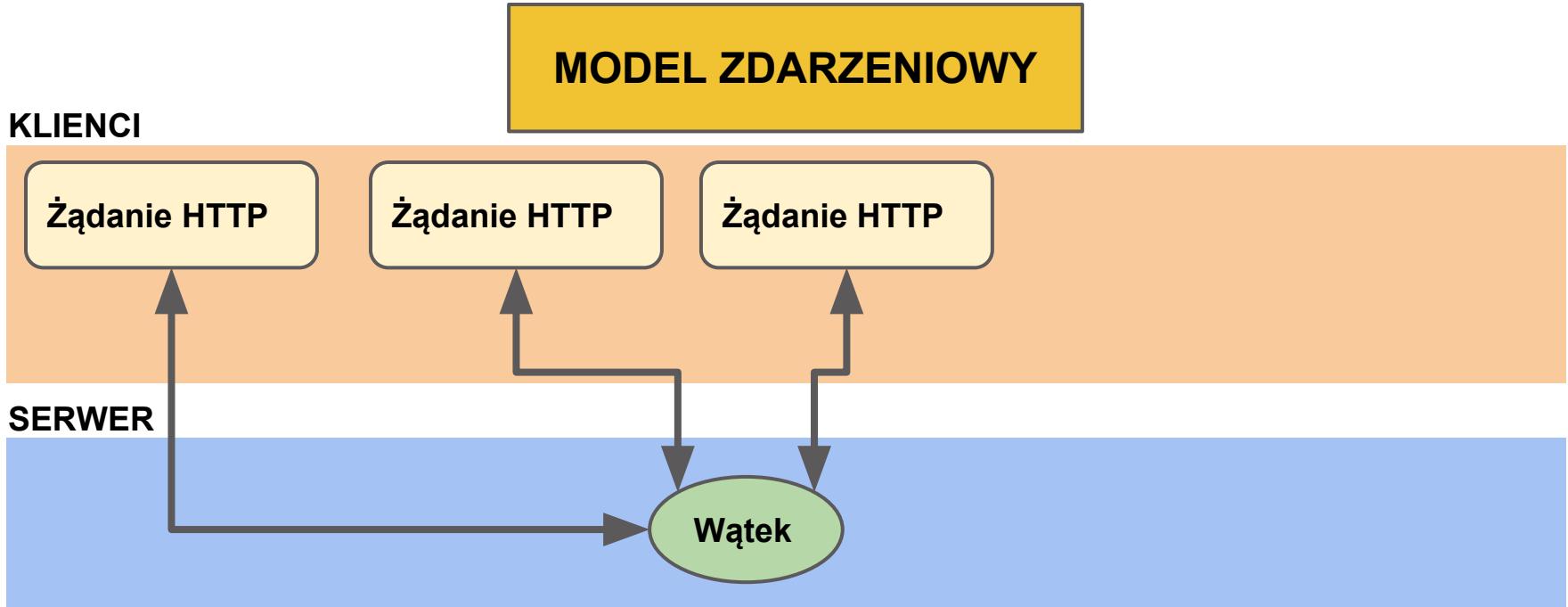
Obsługa żądań



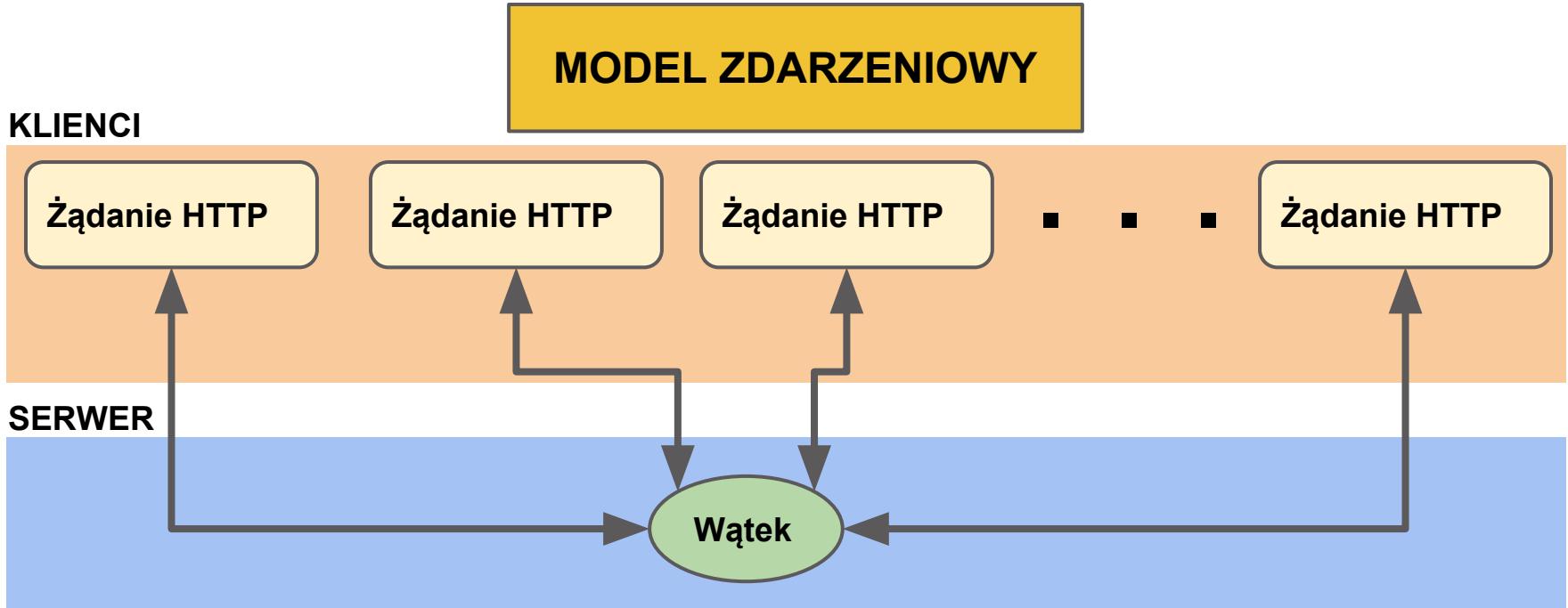
Obsługa żądań



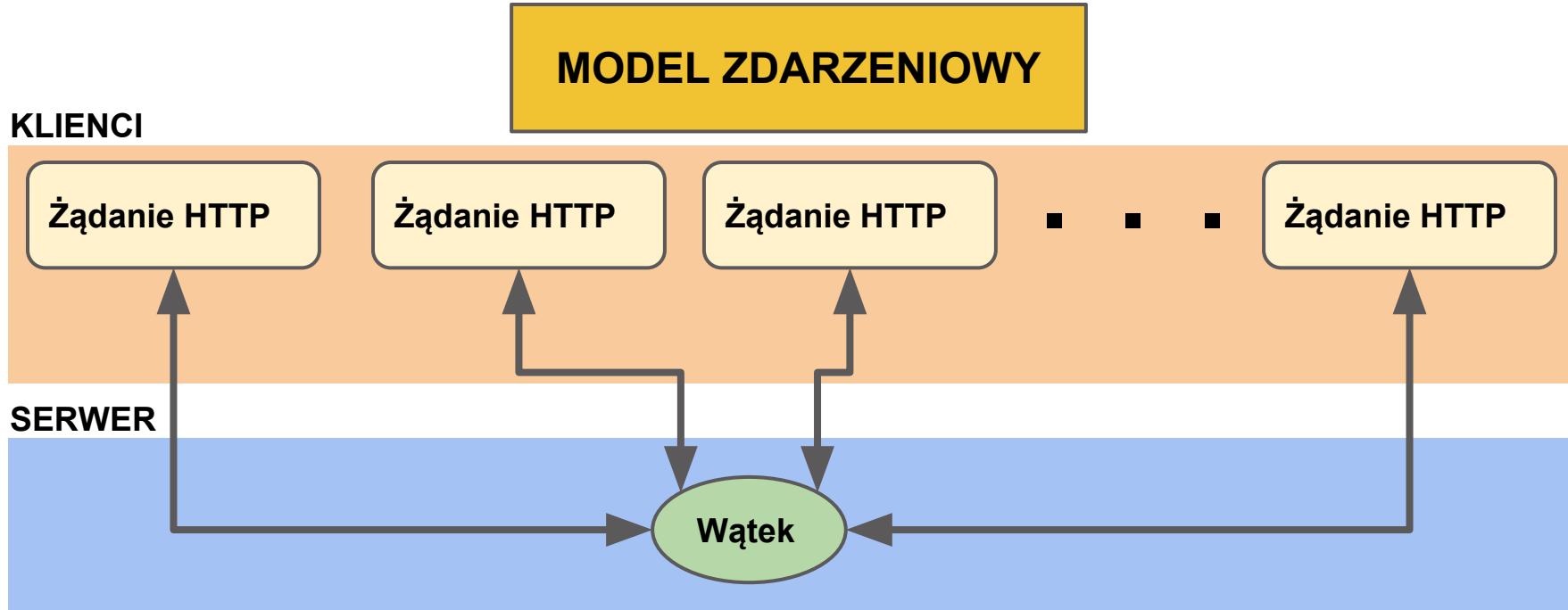
Obsługa żądań



Obsługa żądań



Obsługa żądań



W modelu zdarzeniowym Node.js wykorzystuje tylko jeden wątek do obsługi wielu żądań, oraz “pętlę zdarzeń” co powoduje że aplikacja taka jest bardzo wydajna i skalowalna. W praktyce przy żadaniach które nie wymagają złożonych operacji obliczeniowych można obsłużyć nawet do **1 miliona żądań jednocześnie**.



Pętla zdarzeń (Event loop)

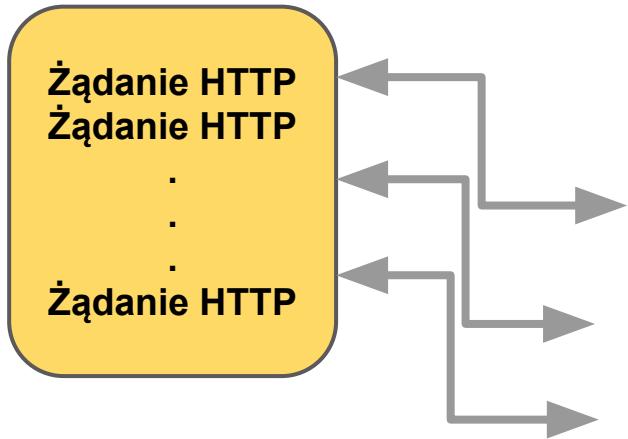
Żądanie HTTP
Żądanie HTTP

.

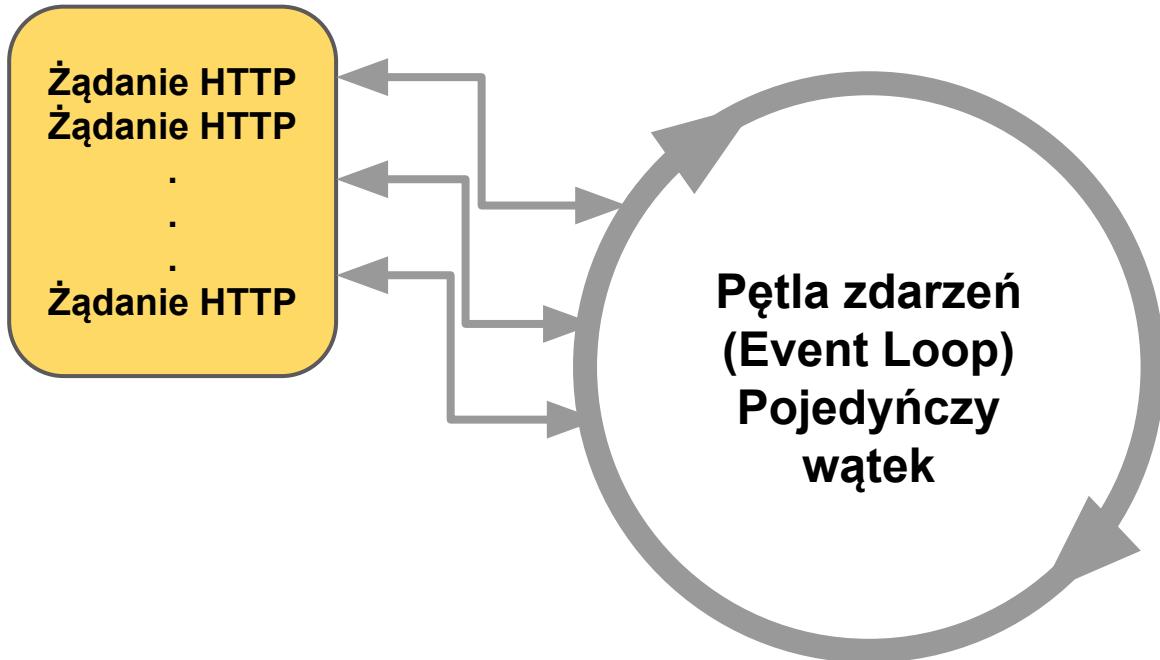
.

Żądanie HTTP

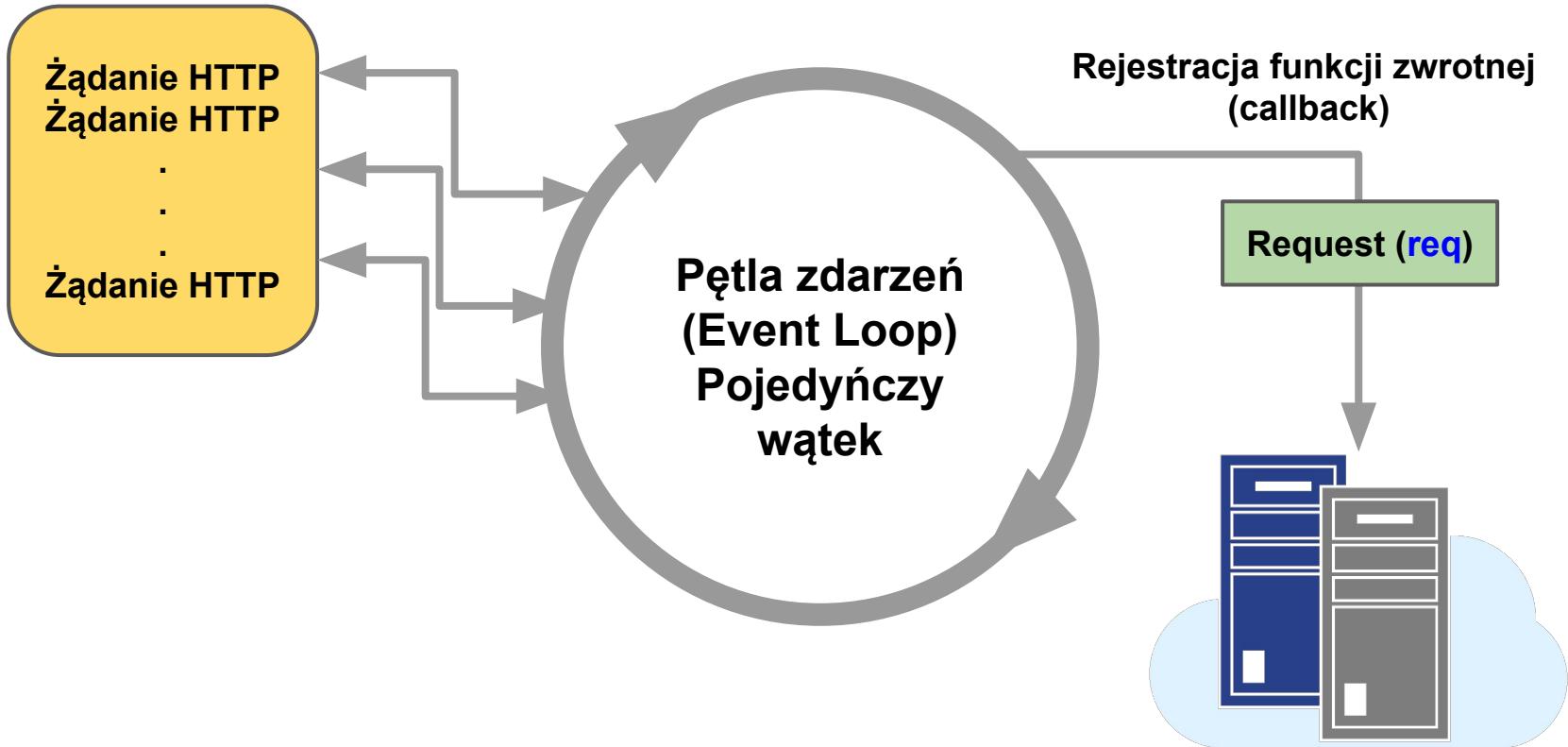
Pętla zdarzeń (Event loop)



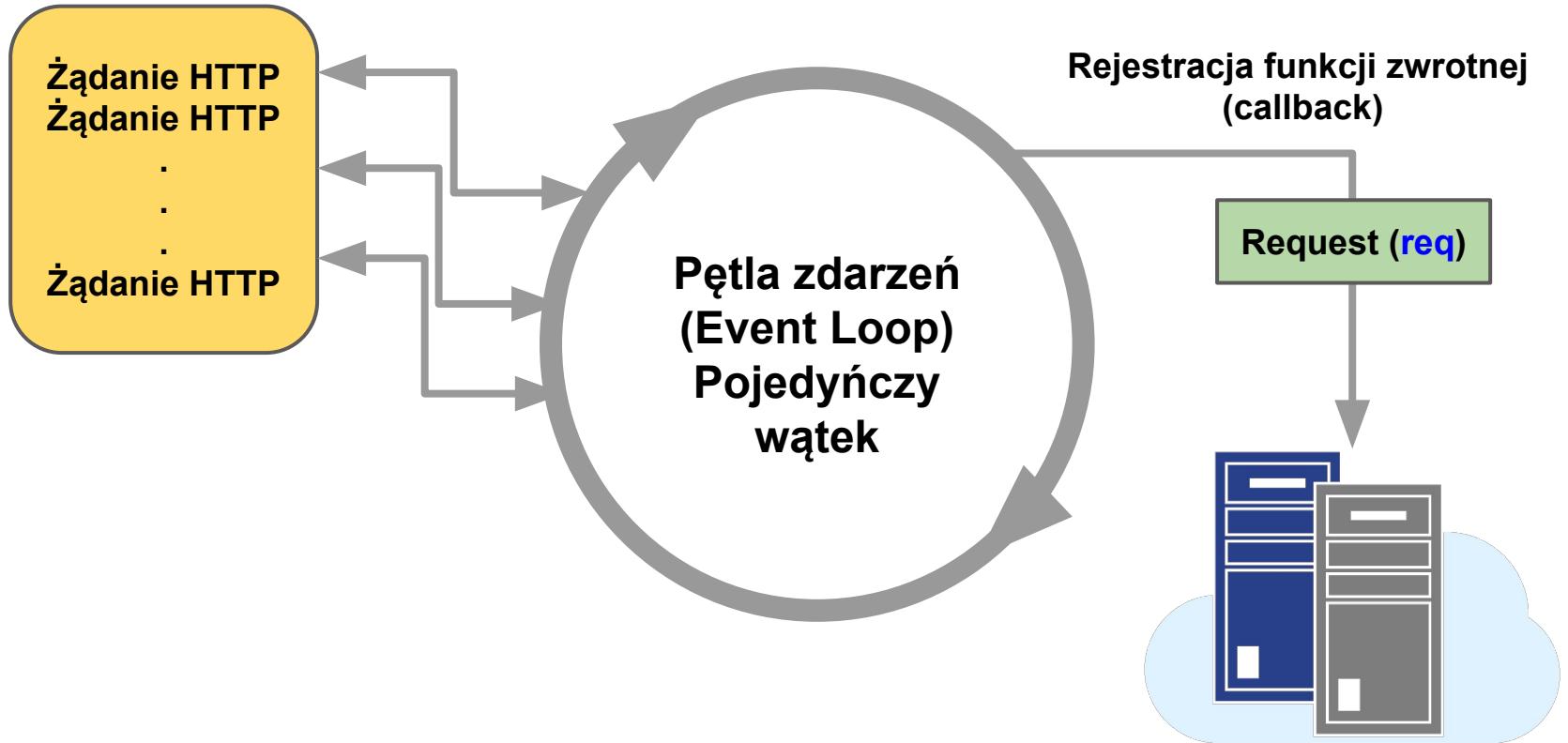
Pętla zdarzeń (Event loop)



Pętla zdarzeń (Event loop)



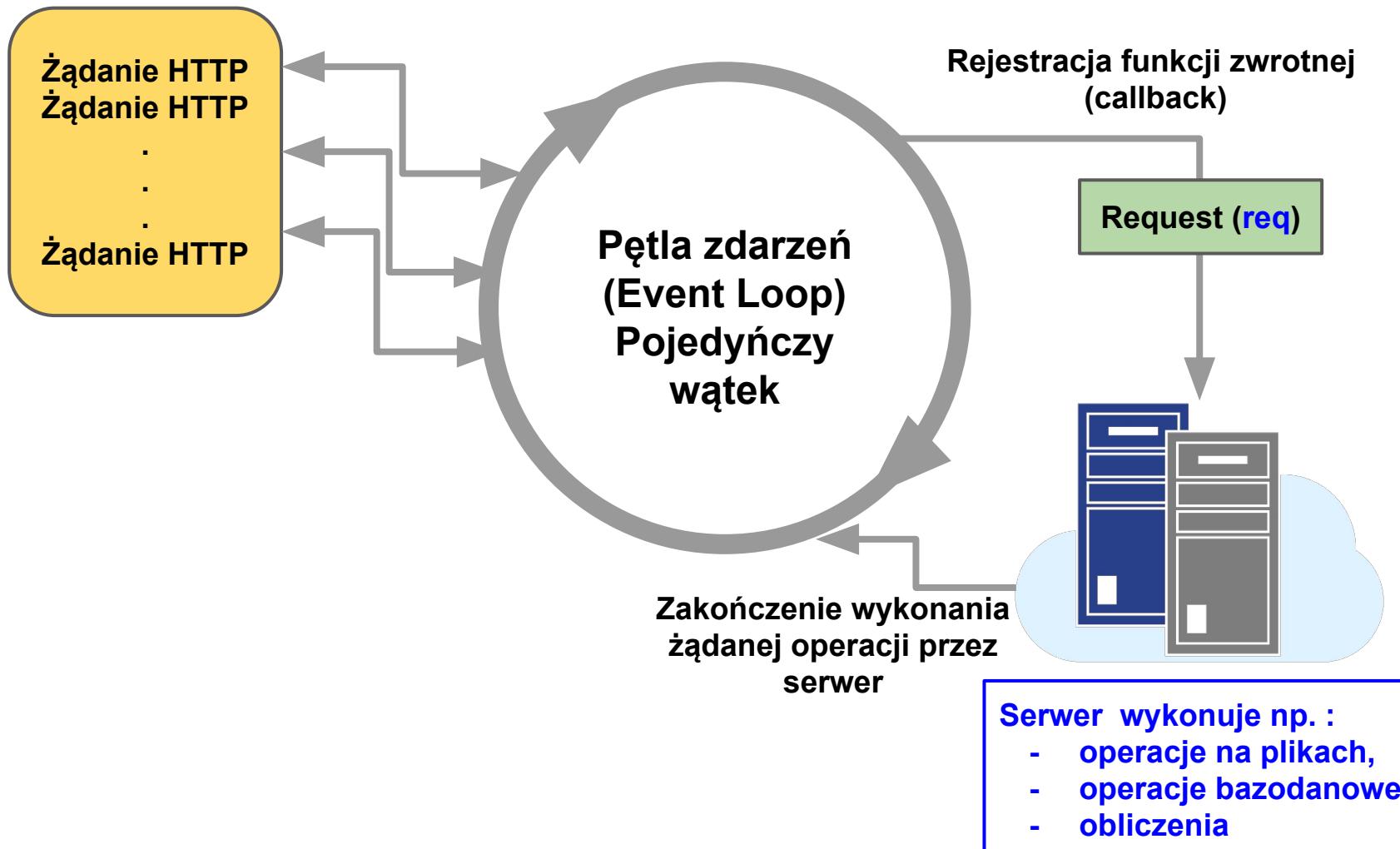
Pętla zdarzeń (Event loop)



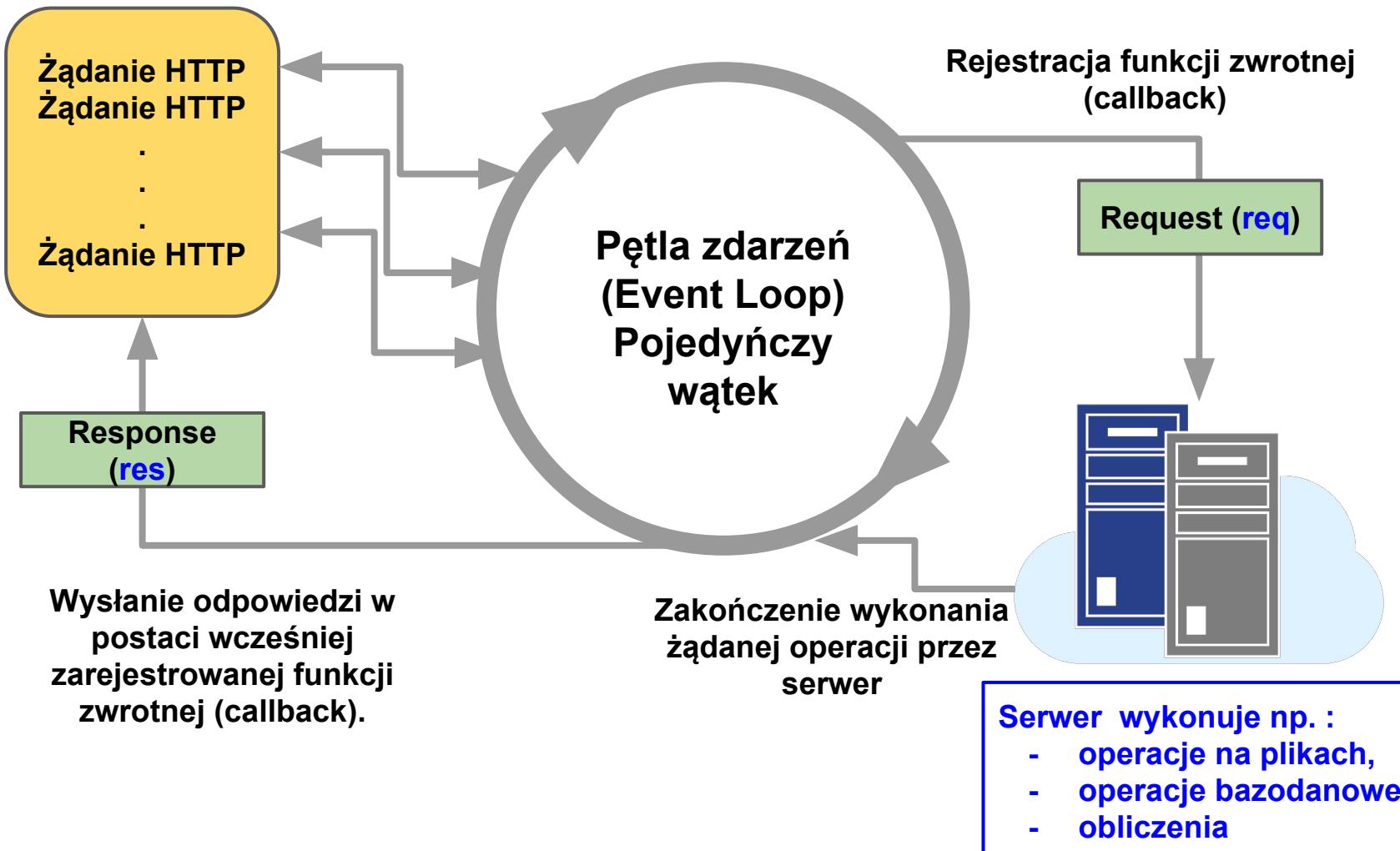
Serwer wykonuje np. :

- operacje na plikach,
- operacje bazodanowe,
- obliczenia

Pętla zdarzeń (Event loop)



Pętla zdarzeń (Event loop)





KONIEC WYKŁADU 8



UNIWERSYTET
JAGIELŁOŃSKI
W KRAKOWIE

Zaawansowane Techniki WWW (HTML, CSS i JavaScript)

Dr inż. Marcin Zieliński

Środa 15:30 - 17:00 sala: A-1-04

WYKŁAD 9

Wykład dla kierunku: Informatyka Stosowana II rok

Rok akademicki: 2015/2016 - semestr zimowy

Przypomnienie z poprzedniego wykładu

**Wykorzystanie biblioteki jQuery do realizacji
asynchronicznej kumunikacji klient-serwer**

Wprowadzenie do środowiska Node.js

AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$ajax( {  
    url: URL,  
    timeout: ms,  
    cache: true/false,  
    success: function(html){},  
    beforeSend: function(){},  
    error: function(){}  
});
```

adres URL

maksymalny czas oczekiwania
na odpowiedz serwera

wyłącza stosowanie pamięci
podręcznej przeglądarki

wywołanie zwrotne
realizowane w przypadku
powodzenia żądania

wywołanie zwrotne
realizowane w przypadku
niepowodzenia żądania

AJAX i jQuery



Przykład 2:

```
$().ready(function() {
    $.ajax({
        url: "localhost/api/getDataJson",
        dataType: "json",
        success: function(json) {
            $("#tresc").html(JSON.stringify(json));
        }
    });
});
```

określamy typ danych
przychodzących

Pobieramy asynchronicznie dane ze adresu URL i w razie sukcesu zakończenia żądania wyświetlamy treść (json) w elemencie #tresc.

AJAX i jQuery



Przykład 3:

```
$
$
$
$ $(function() {
$   $.ajax({
$     url: 'localhost/api/getData',
$     type: 'GET',
$     data: 'Username=jquery4u',
$     success: function(data) {
$       $('#results').html(data);
$     },
$     error: function(e) {
$       console.log(e.message);
$     }
$   });
$ });
$ $
```

określamy metody wysyłania danych

przesyłane dane w metodzie GET

AJAX i jQuery



Przykład 4:

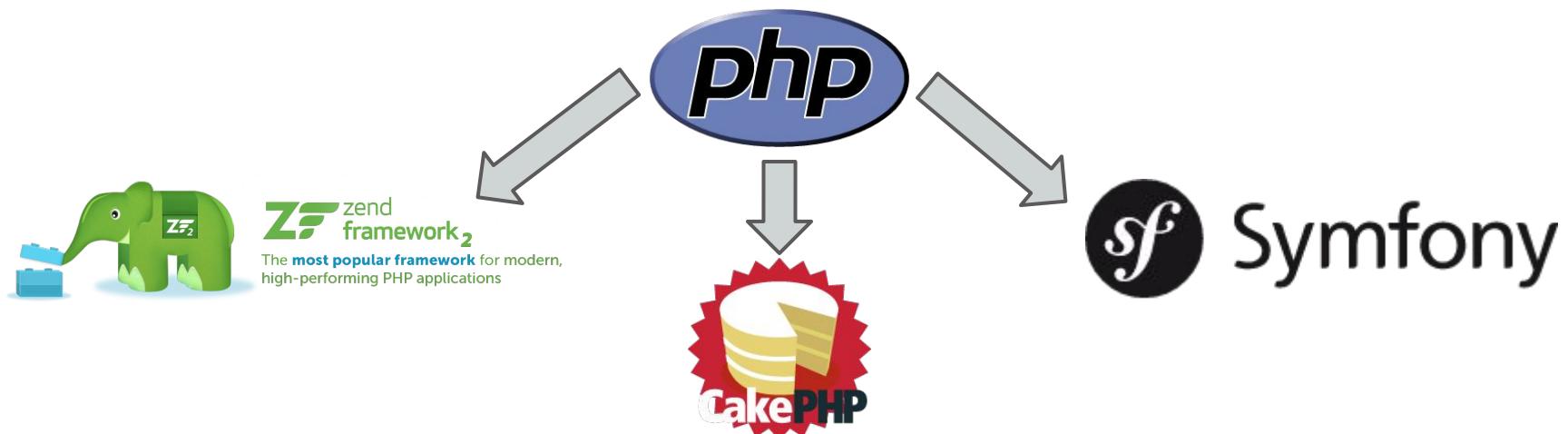
```
$
$
$
$
$
$ $(function() {
$   $.ajax({
$     method: 'POST',
$     url: 'localhost/api/sendData',
$     data: { name: "Jan", fname: "Kowalski" },
$     success: function() {
$       alert( 'Dane zostały przesłane ' );
$     });
$   });
$ }) ;
```

określamy metody wysyłania danych



Popularne dostępne rozwiązania

Najpopularniejsze środowiska programistyczne:



oraz systemy CMS (Content Manager System):

Drupal™

PHP FUSION

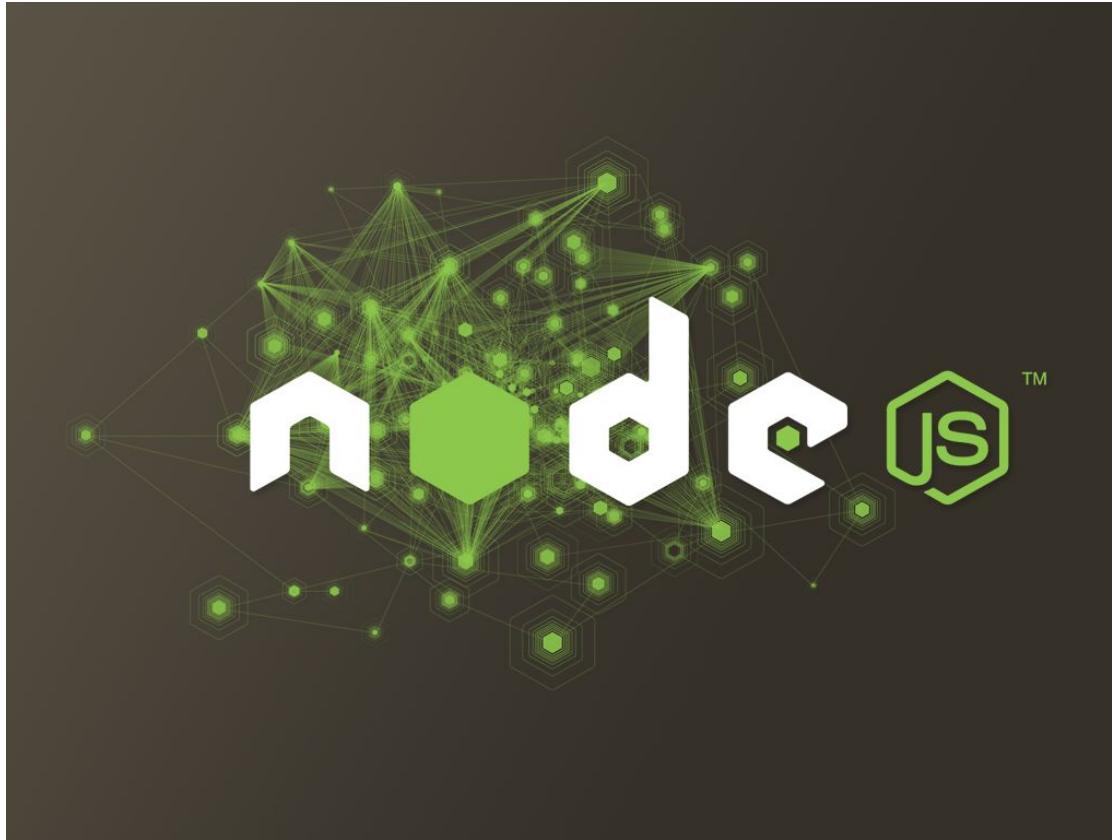
Joomla!™

WORDPRESS

PHP nuke

MediaWiki
Because ideas want to be free.

Node.js

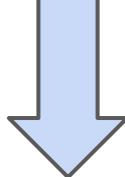


<http://nodejs.org/>

Krótka historia Node.js

Powstanie Node.js było zainspirowane przez funkcjonalności “push” jakie oferuje np. poczta GMAIL...

Co to jest “push” ?



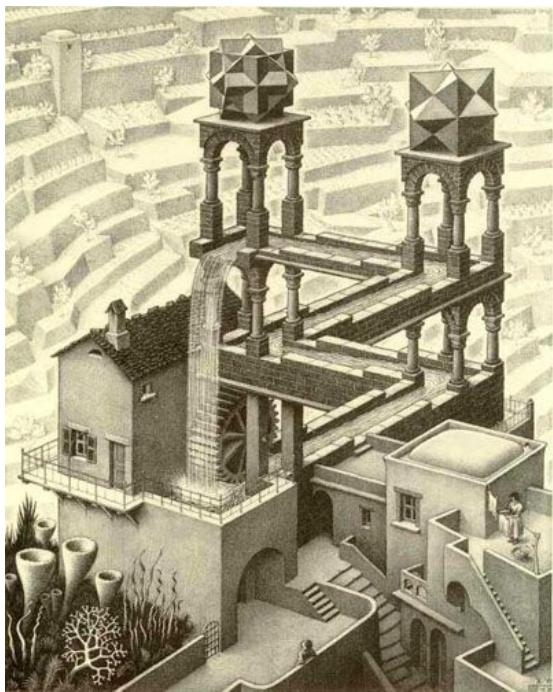
Usługi push działają w oparciu o przekazane przez klienta wcześniej informacje, na podstawie których serwer dostarcza klientowi nowych danych w zależności od tego czy są one dostępne.

Przykład:

Przykładem usługi “push” jest synchroniczny chat.

JavaScript po stronie serwera

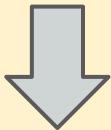
Musimy odwrócić sposób myślenia jeśli chodzi wykonywanie JavaScriptu.



JavaScript po stronie serwera

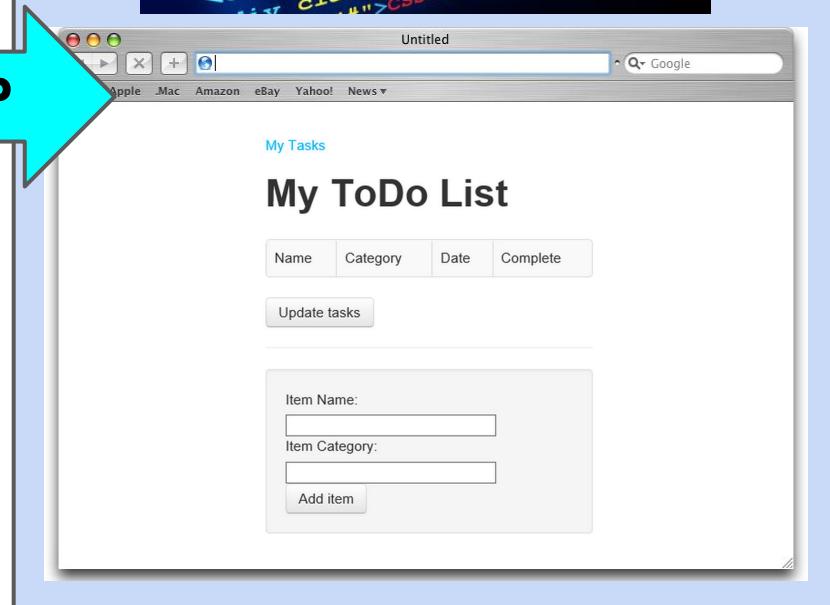
SERVER SIDE

```
1 var edge = require('edge');
2
3 var hello = edge.func(function () {/*
4     async (input) =>
5     {
6         return ".NET welcomes " + input.ToString();
7     }
8 */});
9
10 hello('Node.js', function (error, result) {
11     if (error) throw error;
12     console.log(result);
13});
```



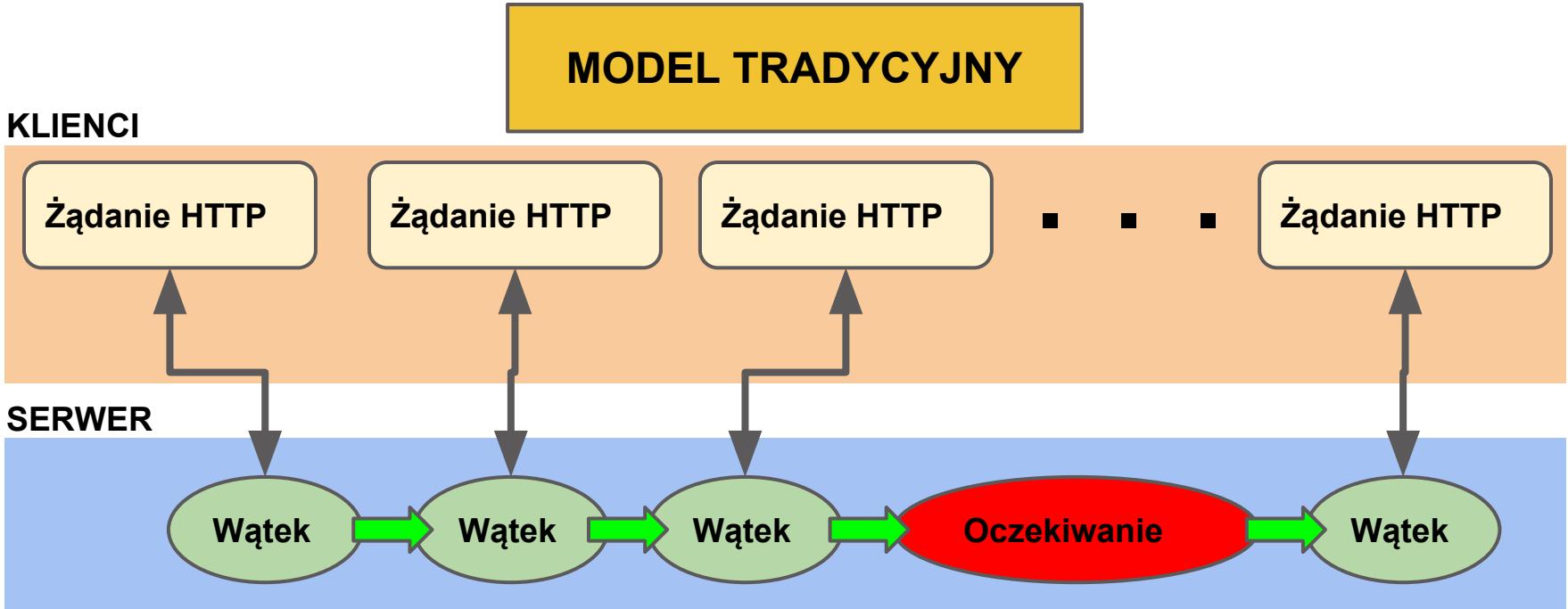
CLIENT SIDE

```
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>CSS3</title>
<link href="style.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="container"> <!-- Main Container -->
<div class="menu"> <!-- Menu here --></div>
<div class="article"><h2>CSS3 Sample</h2>
<p>CSS3 & HTML5 are so good!</p>
</div>
</div>
```



HTTP

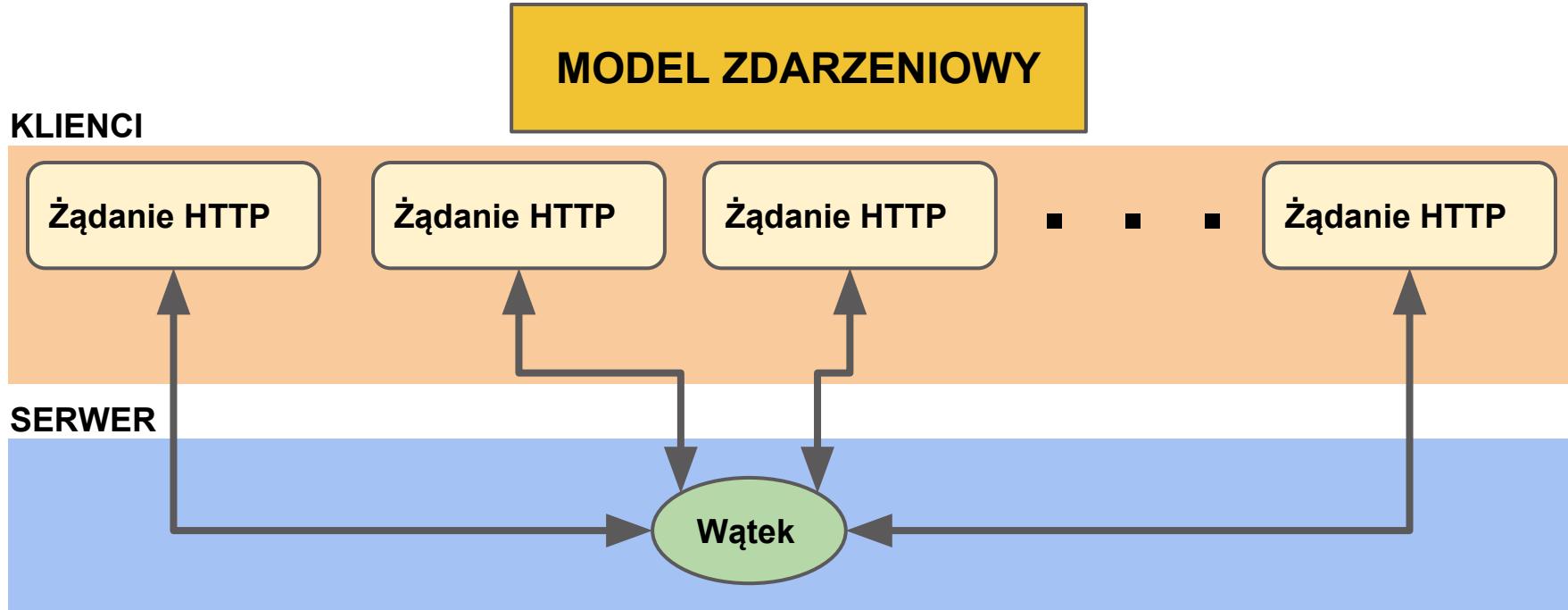
Obsługa żądań



Przykład:

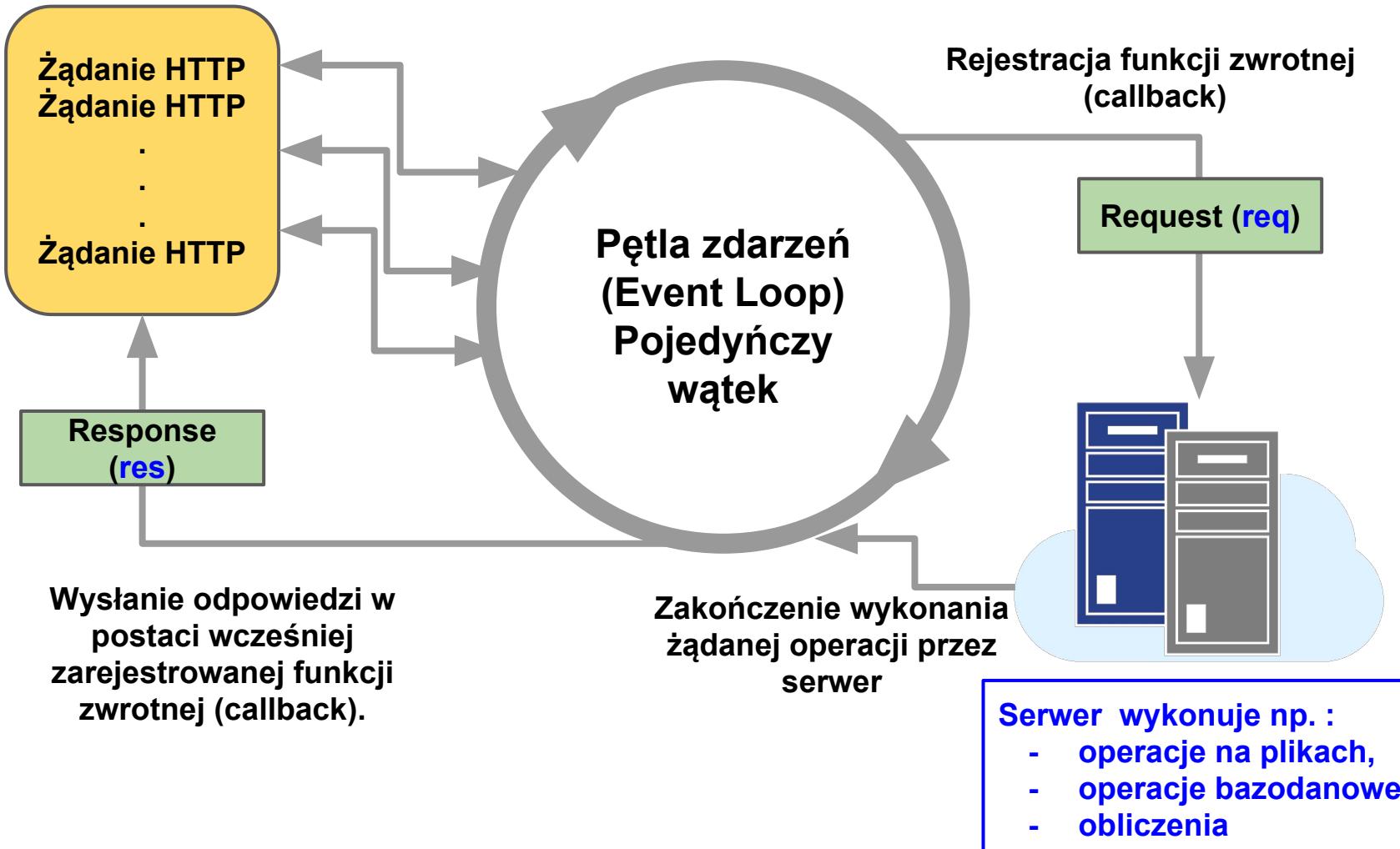
Dla systemu z 8GB pamięci RAM przydzielającego 2MB pamięci na wątek możemy obsłużyć maksymalnie w tym samym czasie **4000 żądań** (w rzeczywistości jest to mniej ponieważ zużywamy jeszcze pamięć na inne operacje).

Obsługa żądań



W modelu zdarzeniowym Node.js wykorzystuje tylko jeden wątek do obsługi wielu żądań, oraz “pętlę zdarzeń” co powoduje że aplikacja taka jest bardzo wydajna i skalowalna. W praktyce przy żadaniach które nie wymagają złożonych operacji obliczeniowych można obsłużyć nawet do **1 miliona żądań jednocześnie**.

Pętla zdarzeń (Event loop)





Zalety wykorzystania Node.js

1. Wbudowana asynchroniczność.
2. Sterowany zdarzeniami.
3. “non-blocking” I/O (wykorzystanie jednego wątku).
4. Praca w czasie rzeczywistym.
5. Skalowalność.
6. Duża biblioteka gotowych modułów (NPM).
7. Język JavaScript.

Liderzy wykorzystujący Node.js





Przykład prostego kodu (z strony nodejs.org)

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');

console.log('Server running at http://127.0.0.1:1337/');
```

Prosty serwer przyjmujący żądania http



Przykład prostego kodu (z strony nodejs.org)

```
var http = require('http');

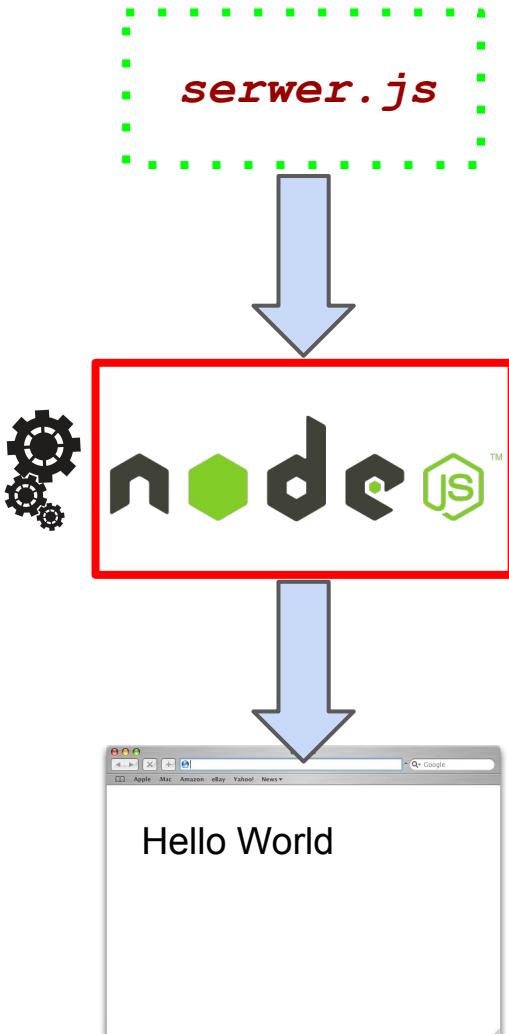
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');

console.log('Server running at http://127.0.0.1:1337/');
```

Prosty serwer przyjmujący żądania http

Założymy że powyższy kod jest zapisany w pliku “serwer.js”

Przykład prostego kodu (z strony nodejs.org)



Jeśli node.js został zainstalowany globalnie na komputerze, jego uruchomienie polega na wywołaniu z konsoli polecenia:

```
> node serwer.js
```

```
MacBook-Pro-Marcin-2:~ marcin$ node serwer.js
Server running at http://127.0.0.1:1337/
```



Przykład prostego kodu (z strony nodejs.org)

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');

console.log('Server running at http://127.0.0.1:1337/');
```

Po uruchomianiu tej “aplikacji” powstał serwer http oczekujący na żądania na porcie 1337.

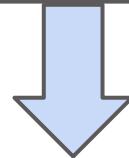
Przykład prostego kodu (z strony nodejs.org)

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');

console.log('Server running at http://127.0.0.1:1337/');
```

Po uruchomianiu tej “aplikacji” powstał serwer http oczekujący na żądania na porcie 1337.



Po otrzymaniu żądania http serwer zwraca odpowiedź OK przesyłając do wyświetlenia zwykły tekst: “Hello World”, który jest widoczny w przeglądarce.



Przykład prostego kodu (z strony nodejs.org)

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('<html><head></head><body>Test</body></html>');
}).listen(1337, '127.0.0.1');

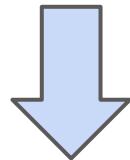
console.log('Server running at http://127.0.0.1:1337/');
```

We wcześniejszym przykładzie serwer zwracał w odpowiedzi zwykły tekst, natomiast teraz zwraca dokument hipertekstowy ze statyczną stroną internetową.

Przykład prostego kodu (z strony nodejs.org)

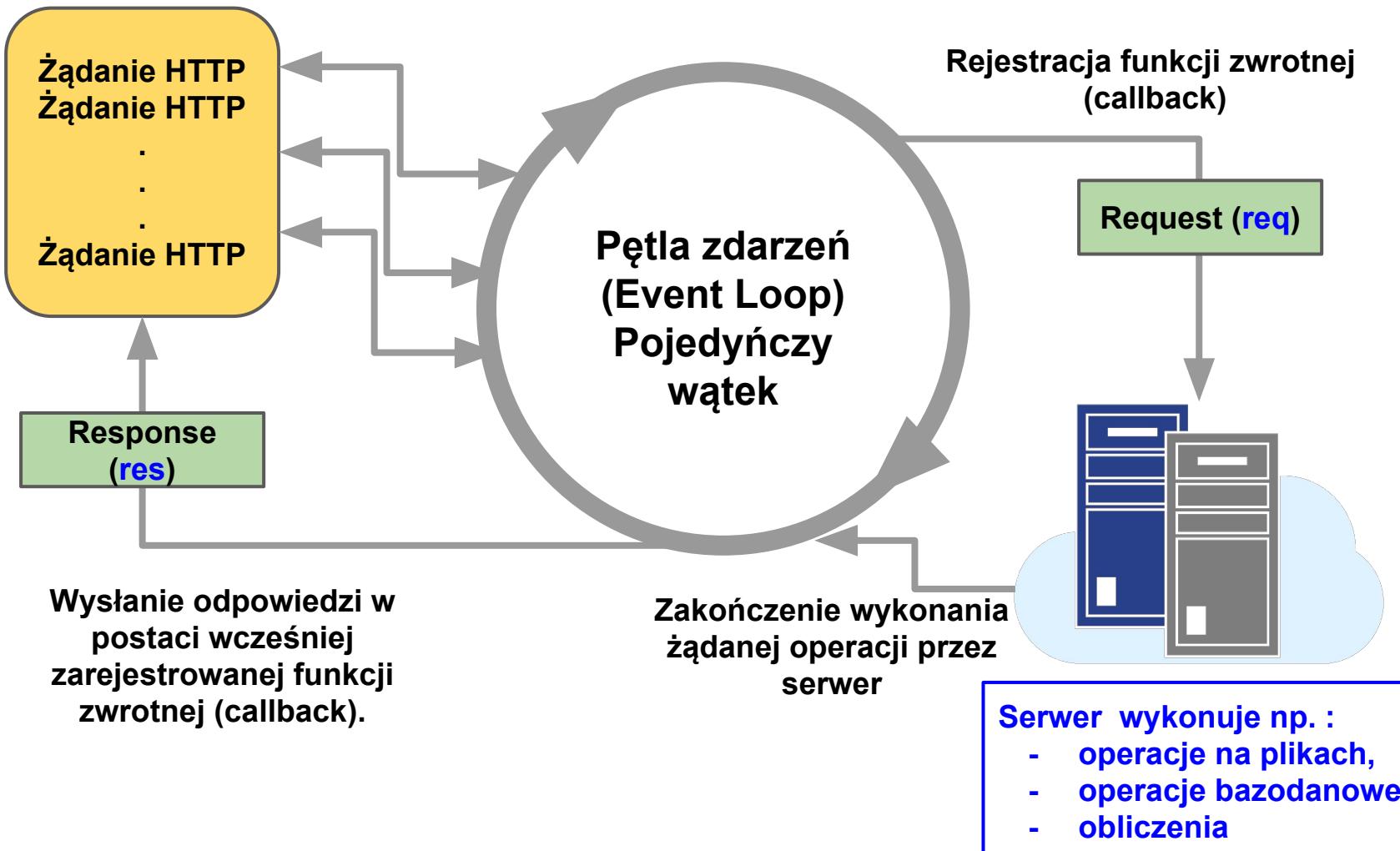
```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('<html><head></head><body>Test</body></html>');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

We wcześniejszym przykładzie serwer zwracał w odpowiedzi zwykły tekst, natomiast teraz zwraca dokument hipertekstowy ze statyczną stroną internetową.



Stworzyliśmy serwer obsługujący żądania HTTP !!!

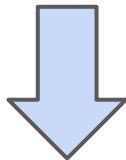
Pętla zdarzeń (Event loop)



Moduły i obiekty

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('hello world');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

We wcześniejszym przykładzie serwer zwracał w odpowiedzi zwykły tekst, natomiast teraz zwraca dokument hipertekstowy ze statyczną stroną internetową.



Stworzyliśmy serwer obsługujący żądania HTTP !!!



Moduły i obiekty

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('hello world');
}).listen(1337, '127.0.0.1');

console.log('Server running at http://127.0.0.1:1337/');
```

Wróćmy do naszego “prostego” przykładu który spełniał rolę serwera http...



Moduły i obiekty

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('hello world');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

Wróćmy do naszego “prostego” przykładu który spełniał rolę serwera http...

W języku JavaScript a tym samym w node.js wszystko jest “obiektem” i dlatego możemy zawsze obejrzeć “podejrzeć” każdy obiekt. Można to zrobić korzystając ze specjalnej biblioteki “utils”, którą można pobrać z repozytorium NPM:

```
| > npm -g install utils
```

Moduły i obiekty

```
var http = require('http');
var utils = require('utils');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end(utils.inspect(req.headers));
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

Dodajemy moduł za pomocą metody "require".

Moduły i obiekty

```
var http = require('http');
var utils = require('utils');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end(utils.inspect(req.headers));
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

Dodajemy moduł za pomocą metody "require".

Korzystamy z metody "inspect()" z modułu "utils" która zwraca postać obiektu w formie tekstu, który możemy wypisać.

Moduły i obiekty

```
var http = require('http');
var utils = require('utils');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end(utils.inspect(req.headers));
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

Dodajemy moduł za pomocą metody "require".

Korzystamy z metody "inspect()" z modułu "utils" która zwraca postać obiektu w formie tekstu, który możemy wypisać.

```
{ host: 'localhost:1337',
connection: 'keep-alive',
'cache-control': 'max-age=0',
accept: 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8',
'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36',
'accept-encoding': 'gzip, deflate, sdch',
'accept-language': 'pl-PL,pl;q=0.8,en-US;q=0.6,en;q=0.4' }
```

Obiekt nagłówka żądania http

Moduły i obiekty

```
: var module = require('module_name');
```

Nazwa obiektu odnoszącego się do modułu

Nazwa modułu

Dzięki modułom można tworzyć zestawy metod (biblioteki), mogą być wykorzystane wielokrotnie w różnych aplikacjach.

Moduły i obiekty

```
: var module = require('module_name');
```

Nazwa obiektu odnoszącego się do modułu

Nazwa modułu

Dzięki modułom można tworzyć zestawy metod (biblioteki), mogą być wykorzystane wielokrotnie w różnych aplikacjach.

Powstały w ten sposób obiekt jest niejako wskaźnikiem na dany moduł przez który możemy wywoływać z modułu dostępne metody i własności:

```
: module.zrobCos();
```

Instalacja

<http://nodejs.org/>



Node.js® is a JavaScript runtime built on [Chrome's V8 JavaScript engine](#). Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, [npm](#), is the largest ecosystem of open source libraries in the world.

Download for OS X (x64)

v4.2.2 LTS

Mature and Dependable

v5.1.0 Stable

Latest Features

Other Downloads: LTS / Stable | Changelog: LTS / Stable | API Docs: LTS / Stable

Pobieramy pakiet w postaci archiwum tar który instalujemy:

node-v5.1.0.tar.gz



Instalacja

<http://nodejs.org/>



Node.js® is a JavaScript runtime built on [Chrome's V8 JavaScript engine](#). Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, [npm](#), is the largest ecosystem of open source libraries in the world.

Download for OS X (x64)

v4.2.2 LTS

Mature and Dependable

v5.1.0 Stable

Latest Features

Other Downloads: LTS / Stable | Changelog: LTS / Stable | API Docs: LTS / Stable

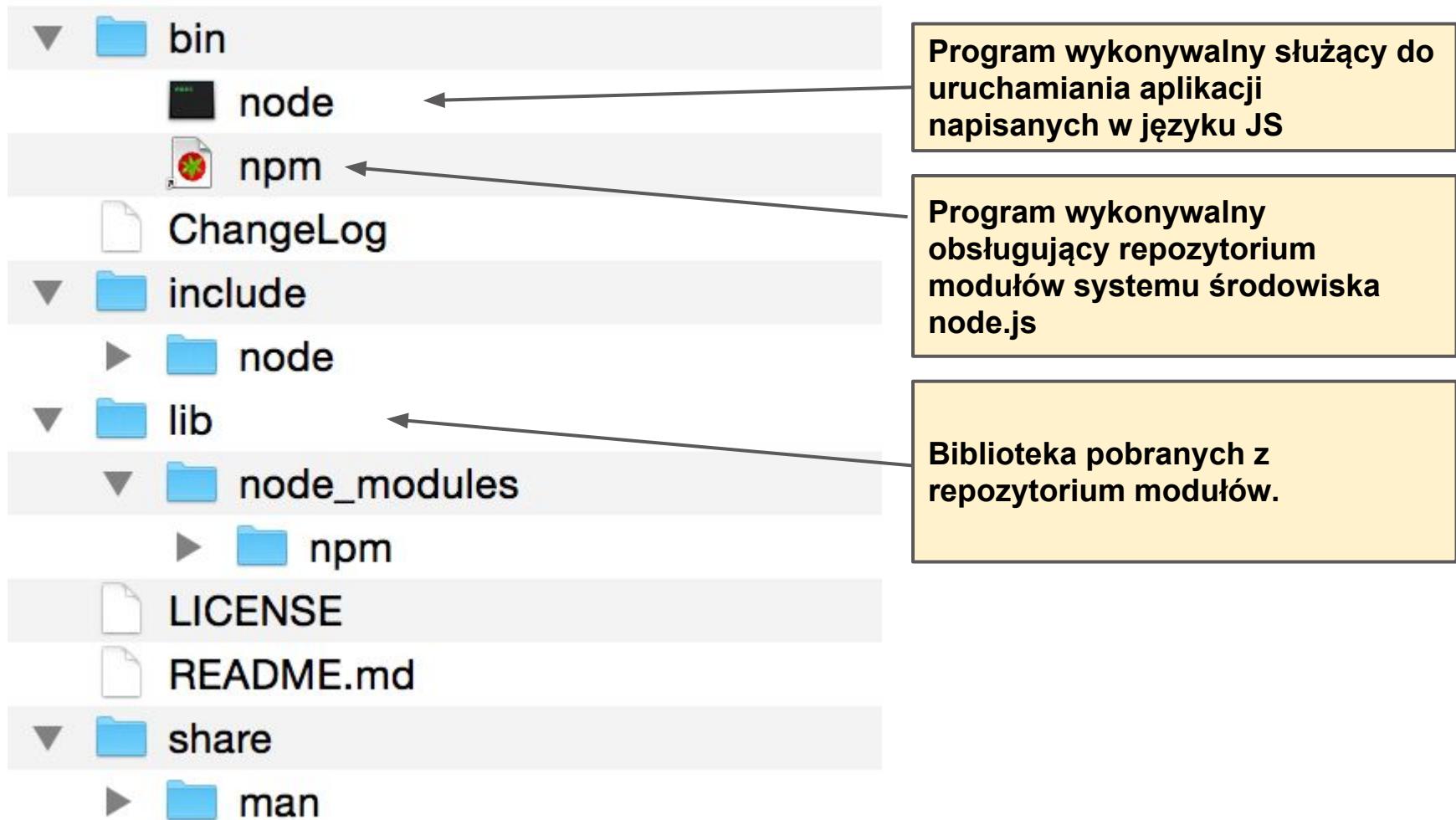


Pobieramy pakiet w postaci archiwum tar który instalujemy:

`node-v5.1.0.tar.gz`

Alternatywnie mamy dostęp do wersji na różne systemy operacyjne oraz wersje skompilowane w postaci plików wykonywalnych.

Instalacja dla Linux





Instalacja dla Linux

Przechodzimy do kartoteki bin gdzie wykonujemy polecenie:

```
MacBook-Pro-Marcin-2:bin marcin$ ./node -v  
v0.10.33
```

Program odpowiada numerem wersji

Zaleca się zainstalowanie globalnej wersji node.js dostępnej dla wszystkich użytkowników oraz możliwej do uruchomianie a każdego miejsca w strukturze kartotek.

Uruchomienie skryptu polega na wywołaniu programu “node” z parametrem określającym nazwę skryptu (gdy node.js jest zainstalowany lokalnie):

```
| > ./node serwer.js |
```

Repozytorium NPM

Integralną częścią środowiska NODE.JS, jest bogate repozytorium modułów (bibliotek), dzięki któremu mamy dostęp do wielu gotowych funkcji.



<http://npmjs.org/>



Repozytorium NPM

Integralną częścią środowiska NODE.JS,
jest bogate repozytorium modułów
(bibliotek), dzięki któremu mamy dostęp do
wielu gotowych funkcji.



npm is the package manager for javascript.

112 811
total packages

32 164 109
downloads in the last day

174 569 610
downloads in the last week

675 621 989
downloads in the last month

packages people 'npm install' a lot



browserify
browser-side require() the node way
6.2.0 published 2 months ago by substack

express

express
Fast, unopinionated, minimalist web framework
4.10.1 published 2 months ago by dougwilson



pm2
Modern CLI process manager for Node apps with a builtin...
0.11.1 published 2 months ago by tknew



grunt-cli
The grunt command line interface.
0.1.13 published a year ago by tkellen



npm
A package manager for node
2.16 published 2 months ago by othym23



karma
Spectacular Test Runner for JavaScript.
0.12.24 published 3 months ago by vojtajina



<http://npmjs.org/>

Repozytorium NPM

Integralną częścią środowiska NODE.JS, jest bogate repozytorium modułów (bibliotek), dzięki któremu mamy dostęp do wielu gotowych funkcji.



<http://npmjs.org/>

Na stronie internetowej projektu można przeglądać i wyszukiwać pakiety, jednak ich instalacja odbywa się w poziomie wiersza poleceń:

```
| > ./npm install nazwa-pakietu
```

Pakiety instalują się w kartotece lib/node_modules.



Repozytorium NPM

Integralną częścią środowiska NODE.JS, jest bogate repozytorium modułów (bibliotek), dzięki któremu mamy dostęp do wielu gotowych funkcji.



<http://npmjs.org/>

Na stronie internetowej projektu można przeglądać i wyszukiwać pakiety, jednak ich instalacja odbywa się w poziomie wiersza poleceń:

```
| > ./npm install nazwa-pakietu
```

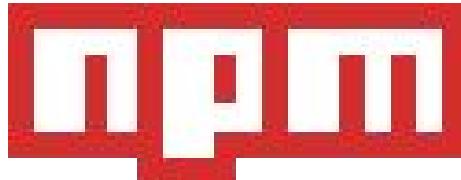
Pakiety instalują się w kartotece **lib/node_modules**.

Do globalnej instalacji pakietów NPM należy posiadać prawa administratora i wydać polecenie:

```
| > ./npm -g install nazwa-pakietu
```

Repozytorium NPM

```
|> ./npm install -g nazwa-pakietu
```



```
MacBook-Pro-Marcin-2:bin marcin$ sudo npm -g install xml2json
npm http GET https://registry.npmjs.org/xml2json
npm http 304 https://registry.npmjs.org/xml2json
npm http GET https://registry.npmjs.org/node-expat
npm http 200 https://registry.npmjs.org/node-expat
npm http GET https://registry.npmjs.org/node-expat/-/node-expat-2.3.3.tgz
npm http 200 https://registry.npmjs.org/node-expat/-/node-expat-2.3.3.tgz
npm http GET https://registry.npmjs.org/bindings
```



Model-View-Controller

Model-View-Controller (MVC) [Model-Widok-Kontroler] - jest to wzorzec projektowy (podejście które jest bazą w oparciu o którą tworzymy aplikację), dzielący projektowaną aplikację na trzy warstwy:



Model-View-Controller

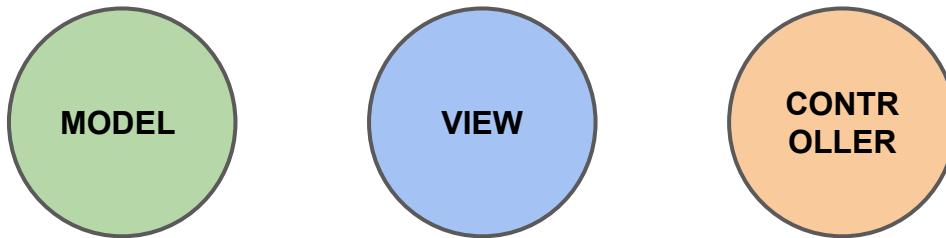
Model-View-Controller (MVC) [Model-Widok-Kontroler] - jest to wzorzec projektowy (podejście które jest bazą w oparciu o którą tworzymy aplikację), dzielący projektowaną aplikację na trzy warstwy:

- **Model (dane / logika)**
- **Widok (prezentacja danych)**
- **Kontroler (interakcja z użytkownikiem + sterowanie aplikacją)**

Model-View-Controller

Model-View-Controller (MVC) [Model-Widok-Kontroler] - jest to wzorzec projektowy (podejście które jest bazą w oparciu o którą tworzymy aplikację), dzielący projektowaną aplikację na trzy warstwy:

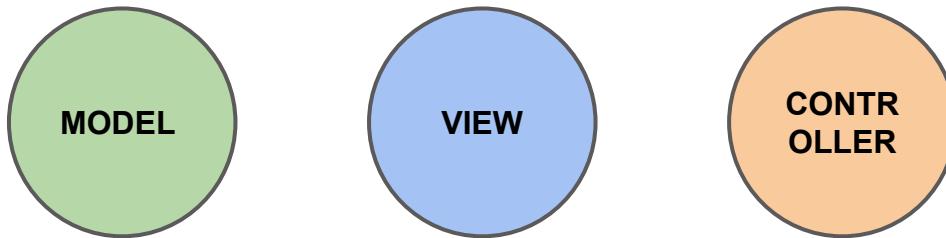
- **Model (dane / logika)**
- **Widok (prezentacja danych)**
- **Kontroler (interakcja z użytkownikiem + sterowanie aplikacją)**



Model-View-Controller

Model-View-Controller (MVC) [Model-Widok-Kontroler] - jest to wzorzec projektowy (podejście które jest bazą w oparciu o którą tworzymy aplikację), dzielący projektowaną aplikację na trzy warstwy:

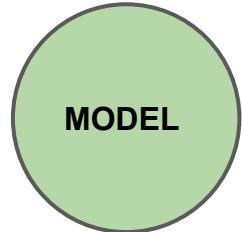
- **Model (dane / logika)**
- **Widok (prezentacja danych)**
- **Kontroler (interakcja z użytkownikiem + sterowanie aplikacją)**



Można go zaimplementować bez użycia bibliotek czy specjalistycznych platform programistycznych, stosując jasne reguły podziału na konkretne komponenty w kodzie źródłowym. W ten sposób każdy komponent aplikacji można niezależnie od siebie rozwijać, implementować i testować.

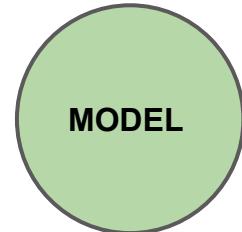
Model-View-Controller

Model - jest odpowiedzialny za przechowywanie wszystkich obiektów danych wykorzystywanych w aplikacji. Model odwozuje logikę danych i fizyczność przetwarzanych danych. Model “nie wie” nic o widokach i kontrolerach. W modelach nie może być zapisany żaden szablon widoku warstwy prezentacji danych ponieważ jest to sprzeczne z wzorcem MVC.



Model-View-Controller

Model - jest odpowiedzialny za przechowywanie wszystkich obiektów danych wykorzystywanych w aplikacji. Model odwozorowuje logikę danych i fizyczność przetwarzanych danych. Model “nie wie” nic o widokach i kontrolerach. W modelach nie może być zapisany żaden szablon widoku warstwy prezentacji danych ponieważ jest to sprzeczne z wzorcem MVC.



Przykład:

Model użytkownika (**USER**) może przechowywać listę użytkowników, ich atrybuty oraz wyznacza logikę powiązania tych danych z innymi modelami danych.

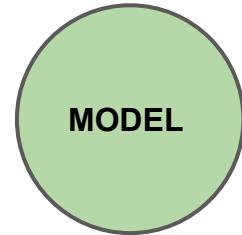
Kiedy kontroler pobiera dane z modelu tworzy na jego podstawie nową “instancję” tego modelu.

Tworzone modele powinny być zorientowane obiektowo !!!

```
var user = Users["Jan"];  
UsunUzytkownika(user);
```

Model-View-Controller

Model - jest odpowiedzialny za przechowywanie wszystkich obiektów danych wykorzystywanych w aplikacji. Model odwozuje logikę danych i fizyczność przetwarzanych danych. Model “nie wie” nic o widokach i kontrolerach. W modelach nie może być zapisany żaden szablon widoku warstwy prezentacji danych ponieważ jest to sprzeczne z wzorcem MVC.

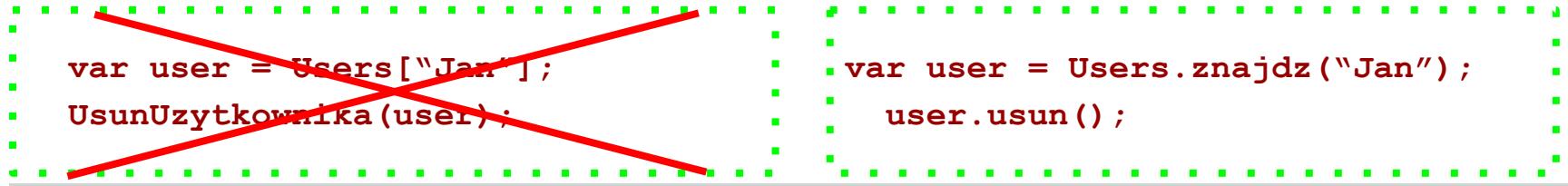


Przykład:

Model użytkownika (**USER**) może przechowywać listę użytkowników, ich atrybuty oraz wyznacza logikę powiązania tych danych z innymi modelami danych.

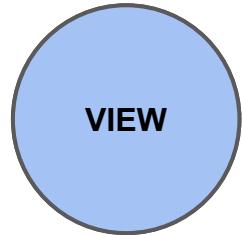
Kiedy kontroler pobiera dane z modelu tworzy na jego podstawie nową “instancję” tego modelu.

Tworzone modele powinny być zorientowane obiektowo !!!



Model-View-Controller

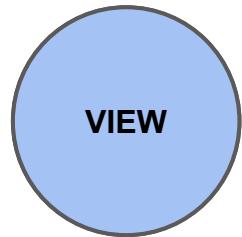
Widok - odpowiada za prezentację danych pobranych z modelu dla użytkownika. Widoki z reguły nie mają żadnej logiki poza kilkoma prostymi instrukcjami warunkowymi. Zgodnie z regułami widoki powinny być całkowicie oddzielone od innych części aplikacji. Logika prezentacji powinna być ujęta w funkcjach “helperach” czyli specjalnych narzędzi do obsługi widoków.



Model-View-Controller

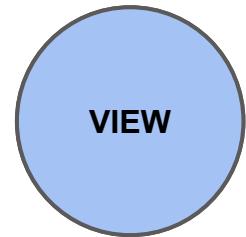
Widok - odpowiada za prezentację danych pobranych z modelu dla użytkownika. Widoki z reguły nie mają żadnej logiki poza kilkoma prostymi instrukcjami warunkowymi. Zgodnie z regułami widoki powinny być całkowicie oddzielone od innych części aplikacji. Logika prezentacji powinna być ujęta w funkcjach “helperach” czyli specjalnych narzędzi do obsługi widoków.

```
.....
<div>
  <script>
    function ZmienDate() {};
  </script>
</div>
.....
```



Model-View-Controller

Widok - odpowiada za prezentację danych pobranych z modelu dla użytkownika. Widoki z reguły nie mają żadnej logiki poza kilkoma prostymi instrukcjami warunkowymi. Zgodnie z regułami widoki powinny być całkowicie oddzielone od innych części aplikacji. Logika prezentacji powinna być ujęta w funkcjach “helperach” czyli specjalnych narzędzi do obsługi widoków.



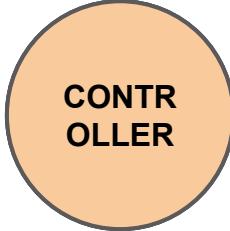
```
<div>
  <script>
    function ZmienDate() {};
  </script>
</div>
```

```
// w osobnym małym skrypcie
var helper = {};
helper.ZmienDate = function() {};

// w widoku
<div>
  ${helper.ZmienDate()}
</div>
```

Model-View-Controller

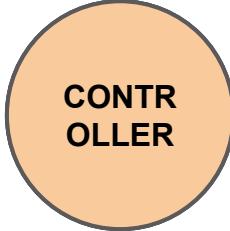
Kontroler - spełnia rolę pośrednika pomiędzy warstwą modelu danych, a widokami na których te dane mają zostać zaprezentowane. Przyjmują zdarzenia i dane przychodzące od użytkowników, przetwarzają je z zastosowaniem modeli i kierują do odpowiednich widoków. W trakcie ładowania strony kontroler dodaje do widoków procesy nasłuchiwanie zdarzeń.



CONTR
OLLER

Model-View-Controller

Kontroler - spełnia rolę pośrednika pomiędzy warstwą modelu danych, a widokami na których te dane mają zostać zaprezentowane. Przyjmują zdarzenia i dane przychodzące od użytkowników, przetwarzają je z zastosowaniem modeli i kierują do odpowiednich widoków. W trakcie ładowania strony kontroler dodaje do widoków procesy nasłuchiwanie zdarzeń.

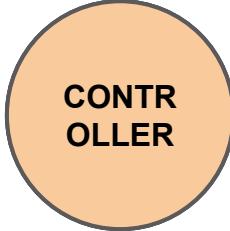


CONTROLLER

```
app.get('/login', function(req, res) {  
    res.render('users/login', {data:data});  
});
```

Model-View-Controller

Kontroler - spełnia rolę pośrednika pomiędzy warstwą modelu danych, a widokami na których te dane mają zostać zaprezentowane. Przyjmują zdarzenia i dane przychodzące od użytkowników, przetwarzają je z zastosowaniem modeli i kierują do odpowiednich widoków. W trakcie ładowania strony kontroler dodaje do widoków procesy nasłuchiwanie zdarzeń.

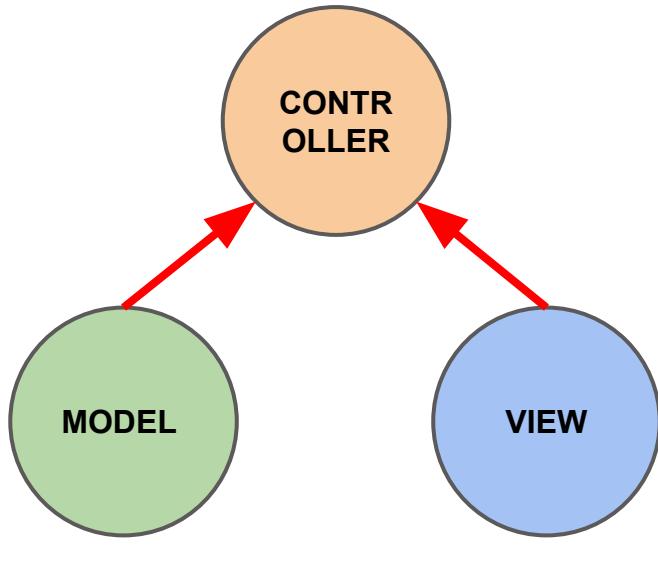


CONTROLLER

```
app.get('/login', function(req, res) {  
    res.render('users/login', {data:data});  
});
```

Do implementacji kontrolerów nie są potrzebne żadne specjalne biblioteki ani platformy programistyczne, jednak wygodniej jest z nich korzystać.

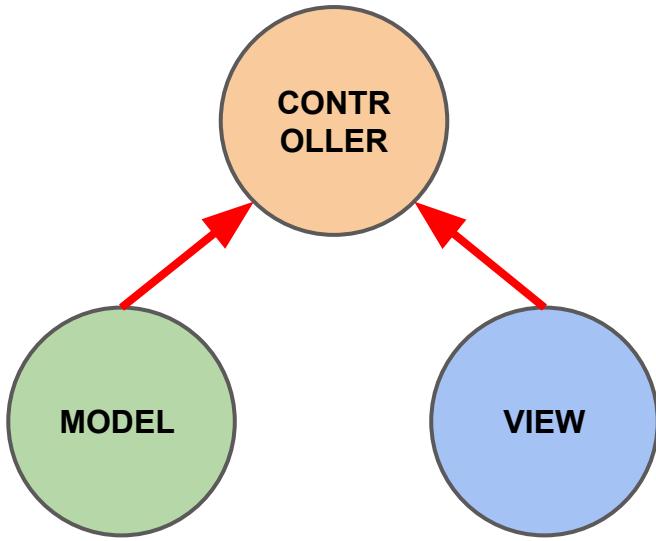
Model-View-Controller



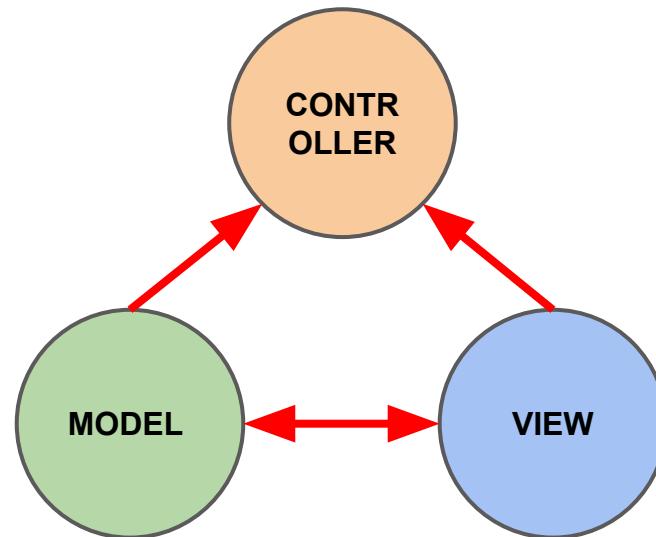
Model Pasywny

W modelu “pasywny” nie ma wymiany danych pomiędzy modelem a widokiem z pominięciem kontrolera, wszystkie akcje są wywoływane i sterowane przez kontroler.

Model-View-Controller



Model Pasywny



Model Aktywny

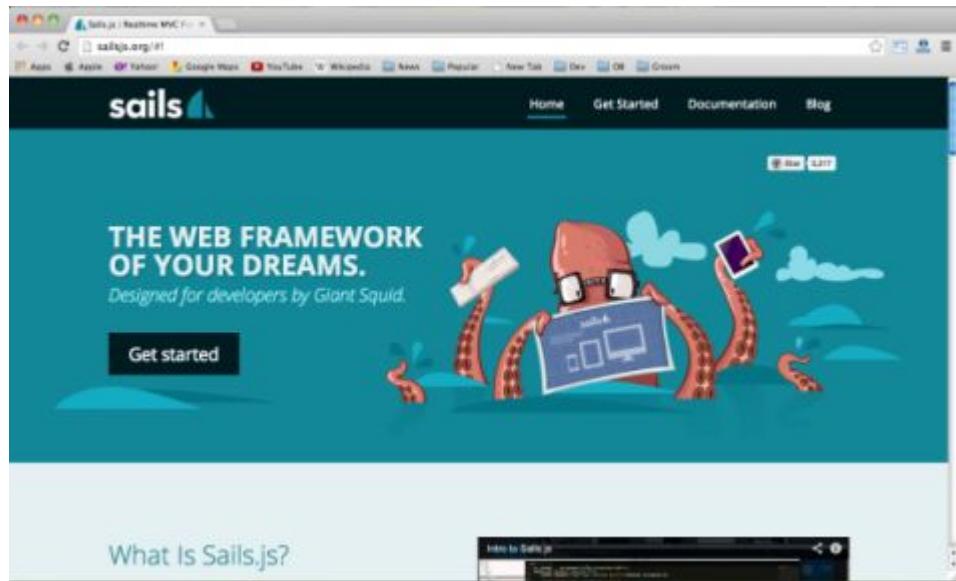
W modelu “pasywny” nie ma wymiany danych pomiędzy modelem a widokiem z pominięciem kontrolera, wszystkie akcje są wywoływane i sterowane przez kontroler.

W modelu aktywnym model może bezpośrednio przekazywać dane do widoku z pominięciem kontrolera.

Node.js + MVC

Istnieje kilka środowisk ułatwiających tworzenie aplikacji według wzorca projektowego MVC w systemie NODE.JS:

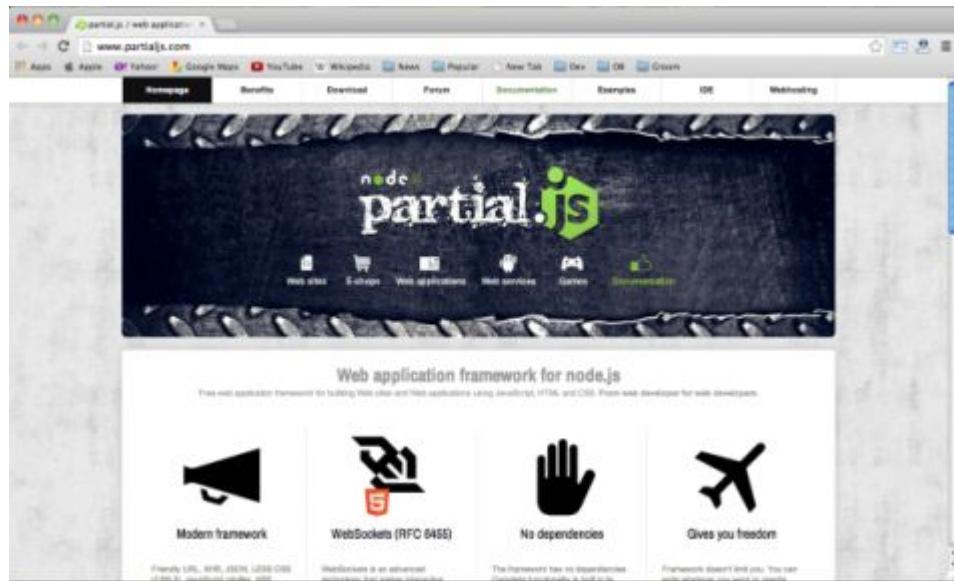
<http://sailsjs.org/>



Node.js + MVC

Istnieje kilka środowisk ułatwiających tworzenie aplikacji według wzorca projektowego MVC w systemie NODE.JS:

<http://www.partialjs.com>



Node.js + MVC

Istnieje kilka środowisk ułatwiających tworzenie aplikacji według wzorca projektowego MVC w systemie NODE.JS:

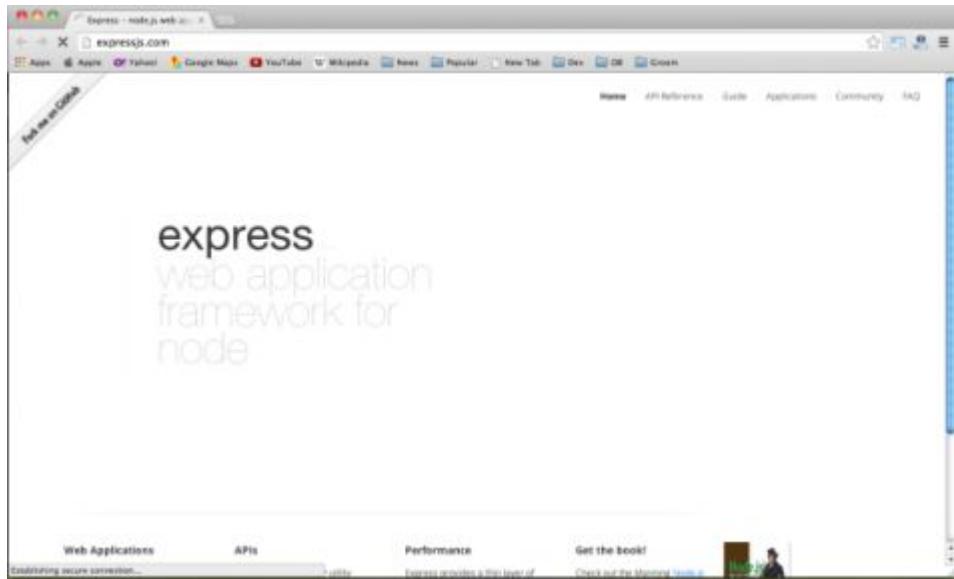
<https://www.totaljs.com/>



Node.js + MVC

Istnieje kilka środowisk ułatwiających tworzenie aplikacji według wzorca projektowego MVC w systemie NODE.JS:

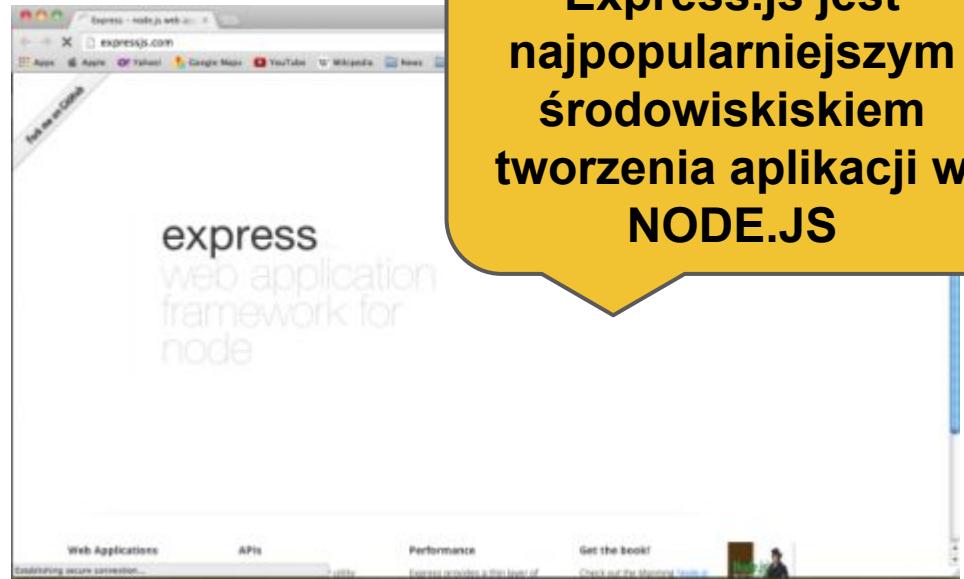
<http://expressjs.com/>



Node.js + MVC

Istnieje kilka środowisk ułatwiających tworzenie aplikacji według wzorca projektowego MVC w systemie NODE.JS:

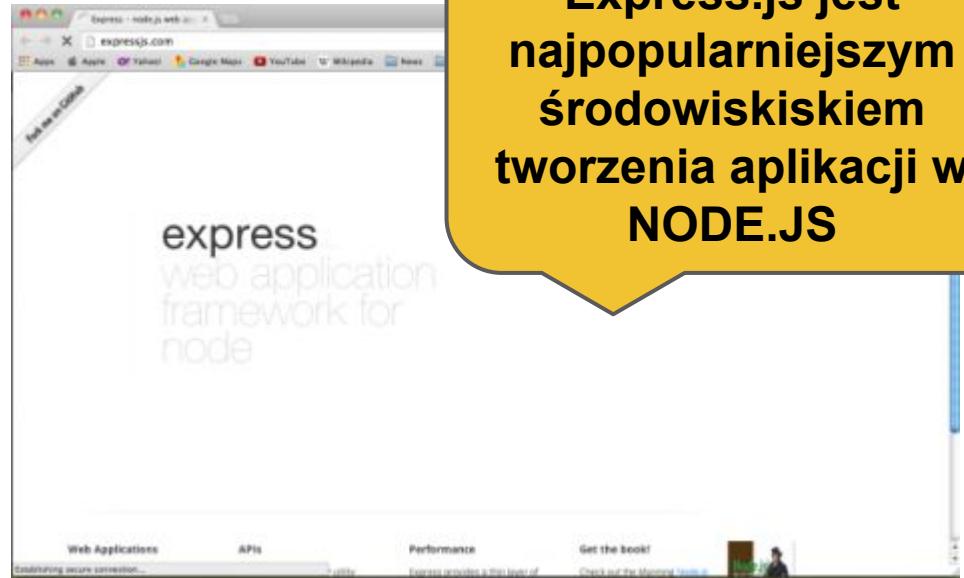
<http://expressjs.com/>



Node.js + MVC

Istnieje kilka środowisk ułatwiających tworzenie aplikacji według wzorca projektowego MVC w systemie NODE.JS:

<http://expressjs.com/>



To środowisko będziemy poznawać na zajęciach ...

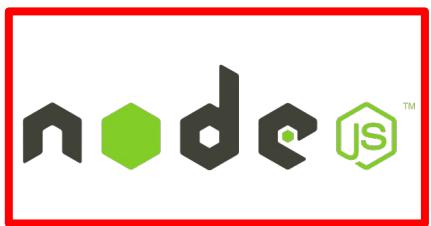


Express JS

Express.js jest środowiskiem które pozwala na tworzenie aplikacji internetowych w formie jednostronnicowych oraz wielostronowych witryn dostosowanych do wyświetlania na urządzeniach mobilnych oraz normalnych komputerach.

Express JS

Express.js jest środowiskiem które pozwala na tworzenie aplikacji internetowych w formie jednostronnicowych oraz wielostronowych witryn dostosowanych do wyświetlania na urządzeniach mobilnych oraz normalnych komputerach.



Express JS - instalacja

Instalacja Express.js odbywa się za pomocą menadżera pakietów NPM.

nutty penguin music

private npm npm Enterprise documentation blog support

npm find packages sign up or log in 

npm is the package manager for **g**

 116 339 total packages  31 204 894 downloads in the last day  132 594 356 downloads in the last week  637 550 458 downloads in the last month

packages people 'npm install' a lot



browserify
browser-side require() the node way
6.2.0 published 2 months ago by substack



express
Fast, unopinionated, minimalist web framework
4.10.1 published 2 months ago by dougwilson



pm2
Modern CLI process manager for Node apps with a builtin...
0.11.1 published 3 months ago by tknew



grunt-cli
The grunt command line interface.
0.1.13 published a year ago by tkellen



npm
A package manager for node
2.1.6 published 2 months ago by othym23



karma
Spectacular Test Runner for JavaScript.
0.12.24 published 3 months ago by vojtajina

Express JS - instalacja

Instalacja Express.js odbywa się za pomocą menadżera pakietów NPM.



The screenshot shows the top navigation bar of the npm website. It includes a search bar with the placeholder "find packages", a magnifying glass icon, and links for "private npm", "npm Enterprise", "documentation", "blog", and "support". There is also a "sign up or log in" button and a user profile icon.

npm is the package manager for 

 116 339
total packages

 31 204 894
downloads in the last day

 132 594 356
downloads in the last week

 637 550 458
downloads in the last month

packages people 'npm install' a lot



browserify

browser-side require() the node way
6.2.0 published 2 months ago by substack

express

express
Fast, unopinionated, minimalist web framework
10.1 published 2 months ago by dougwil...



pm2

Modern CLI process manager for Node apps with a builtin...
0.11.1 published 3 months ago by tknew



grunt-cli

The grunt command line interface.
0.1.13 published a year ago by tkellen



npm

A package manager for node
2.1.6 published 2 months ago by othym23



karma

Spectacular Test Runner for JavaScript.
0.12.24 published 3 months ago by vojtaojina



Express JS - instalacja

Instalacja Express.js odbywa się za pomocą menadżera pakietów NPM.

The screenshot shows the npm homepage. At the top left is the red 'npm' logo. To its right is a search bar with the placeholder 'find packages'. Next to the search bar is a magnifying glass icon. Further to the right are links for 'sign up or log in' and a small user icon. The background is light grey.

★ express

Fast, unopinionated, minimalist web framework

npm v4.10.7 downloads 2M/month build passing coverage 100%

```
var express = require('express')
var app = express()

app.get('/', function (req, res) {
  res.send('Hello World')
})

app.listen(3000)
```

npm install express

Package Info

4.10.7 published 3 days ago by **dougwilson**

github.com/strongloop/express

expressjs.com

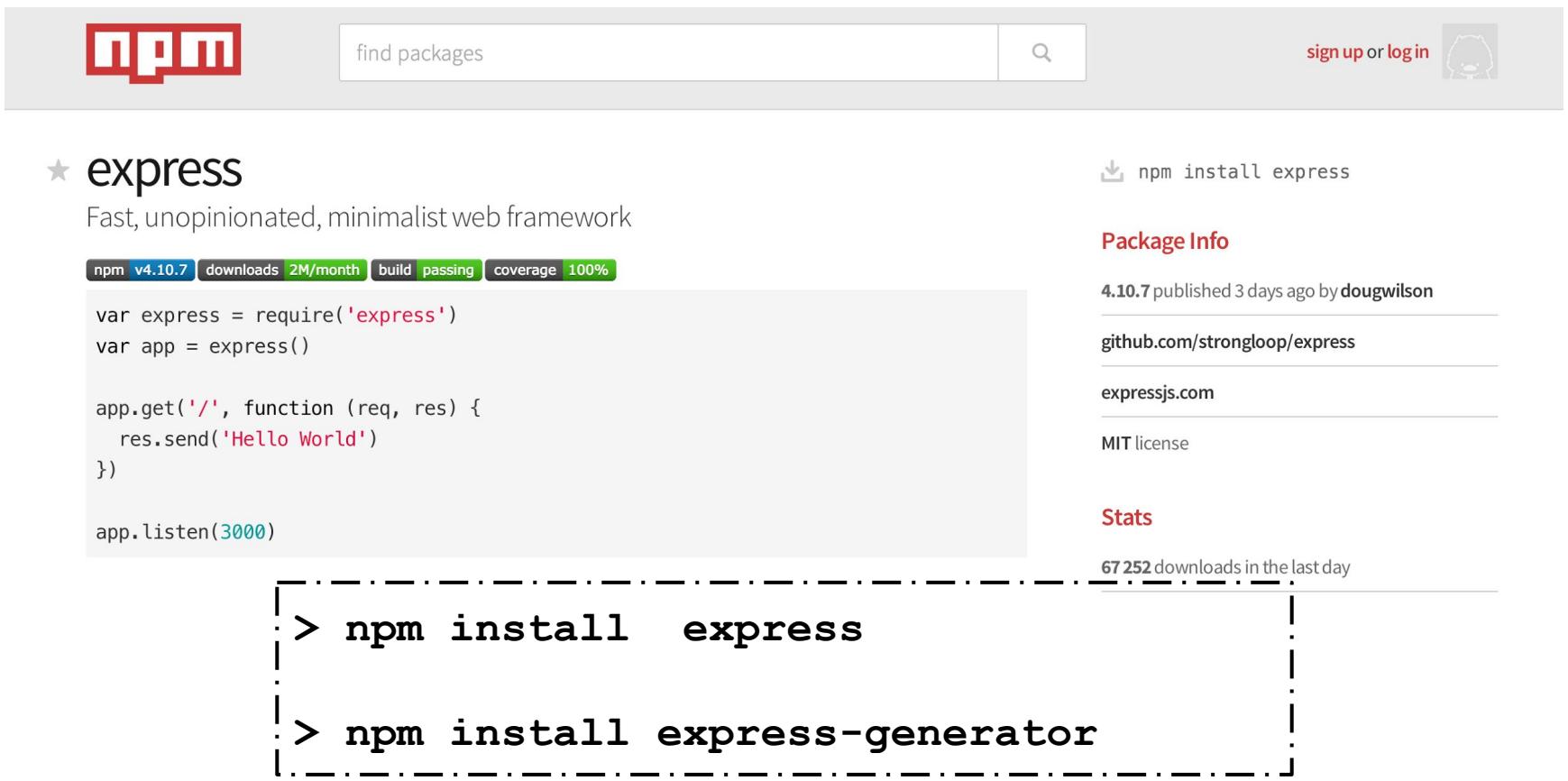
MIT license

Stats

67 252 downloads in the last day

Express JS - instalacja

Instalacja Express.js odbywa się za pomocą menadżera pakietów NPM.



The screenshot shows the npmjs.com website interface. At the top, there's a search bar with 'find packages' placeholder text and a magnifying glass icon. To the right of the search bar are 'sign up or log in' links and a user profile icon. Below the header, the 'express' package page is displayed. The package name 'express' is shown with a star rating. A brief description follows: 'Fast, unopinionated, minimalist web framework'. Below the description are several status indicators: 'npm v4.10.7', 'downloads 2M/month', 'build passing', and 'coverage 100%'. A code snippet shows the basic setup for the 'express' module:

```
var express = require('express')
var app = express()

app.get('/', function (req, res) {
  res.send('Hello World')
})

app.listen(3000)
```

To the right of the code snippet, there are links for 'npm install express', 'Package Info', '4.10.7 published 3 days ago by dougwilson', 'github.com/strongloop/express', 'expressjs.com', 'MIT license', 'Stats', and '67 252 downloads in the last day'. At the bottom, two command-line entries are shown in a dashed box:

```
> npm install express
> npm install express-generator
```

Express JS - instalacja

```
> npm install express
```

```
> npm install express-generator
```

Pierwsze polecenie instaluje środowisko Express.js w ramach posiadanej dystrybucji NODE.JS, natomiast drugim poleceniem instalujemy aplikację pomocniczą służącą do generowania standardowych i działających “wzorców” aplikacji (projektów “0”).



UNIWERSYTET
JAGIELŁOŃSKI
W KRAKOWIE

KONIEC WYKŁADU 9



UNIWERSYTET
JAGIELŁOŃSKI
W KRAKOWIE

Zaawansowane Techniki WWW (HTML, CSS i JavaScript)

Dr inż. Marcin Zieliński

Środa 15:30 - 17:00 sala: A-1-04

WYKŁAD 10

Wykład dla kierunku: **Informatyka Stosowana II rok**

Rok akademicki: **2015/2016 - semestr zimowy**

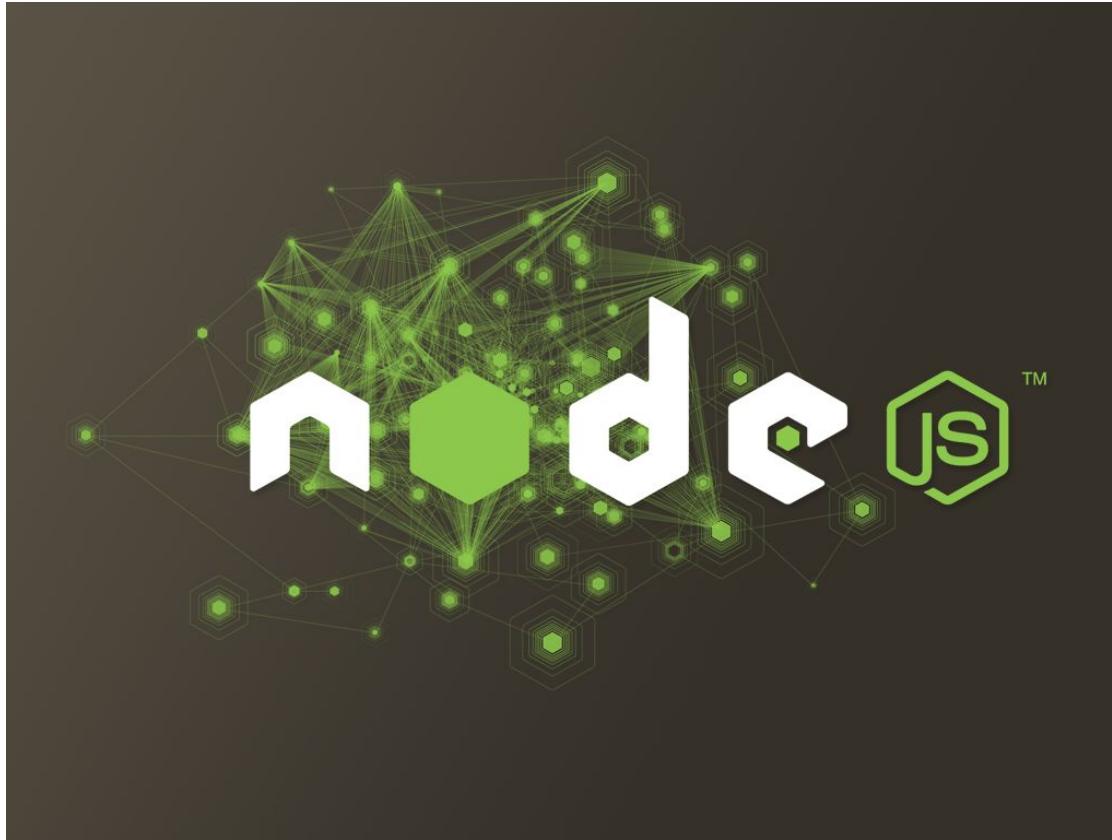
Przypomnienie z poprzedniego wykładu

Wprowadzenie do środowiska Node.js

Repozytorium NPM

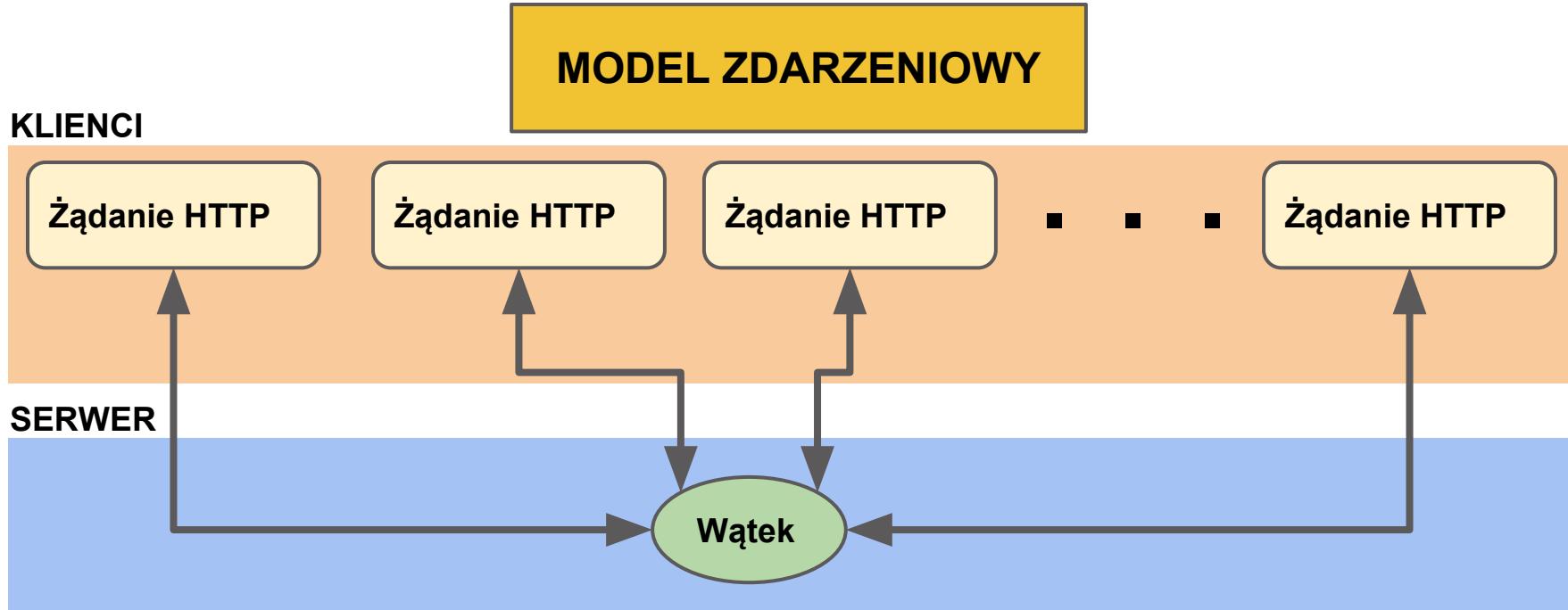
Wzorzec projektowy MVC (Model - Widok - Kontroler)

Node.js



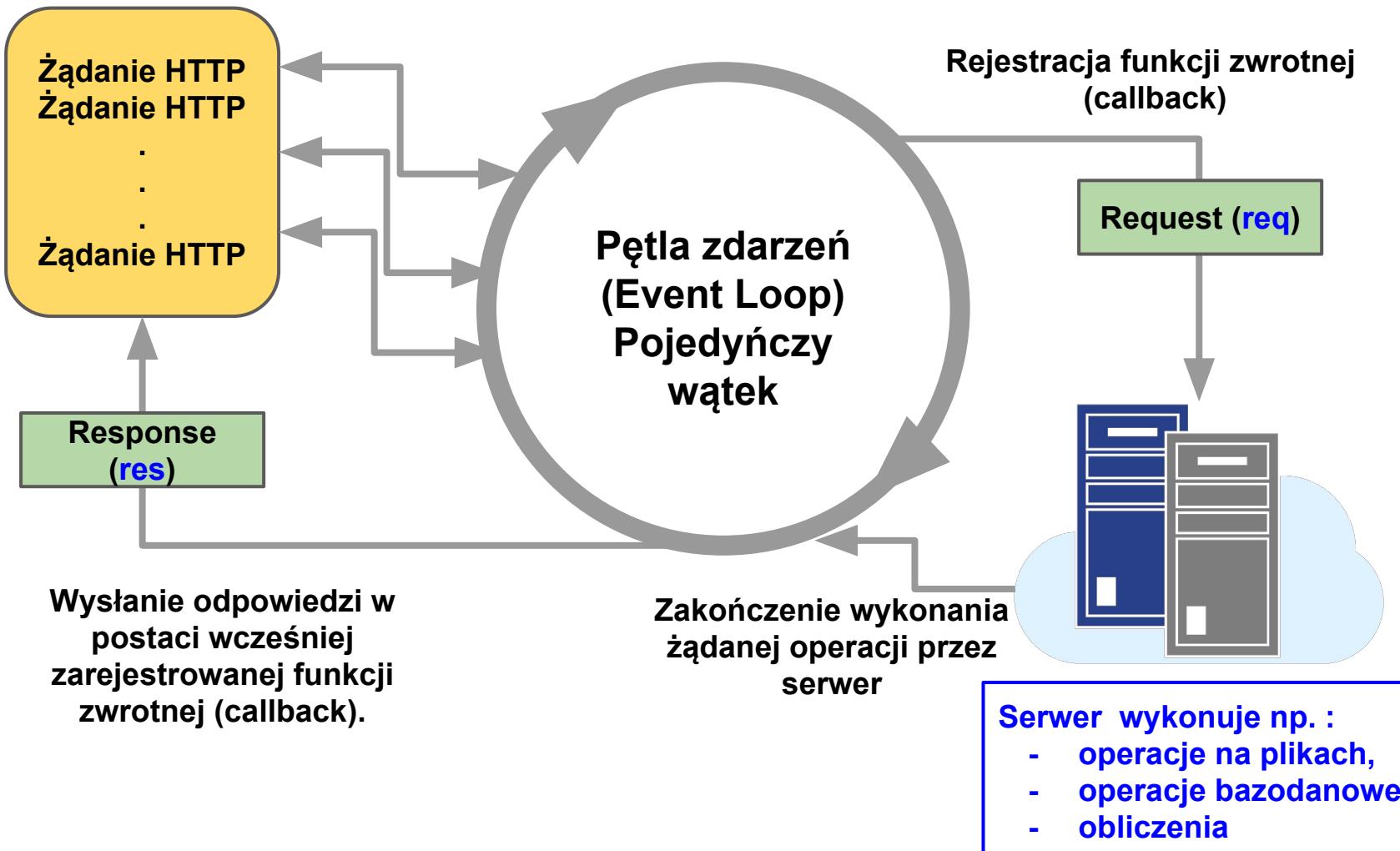
<http://nodejs.org/>

Obsługa żądań



W modelu zdarzeniowym Node.js wykorzystuje tylko jeden wątek do obsługi wielu żądań, oraz “pętlę zdarzeń” co powoduje że aplikacja taka jest bardzo wydajna i skalowalna. W praktyce przy żadaniach które nie wymagają złożonych operacji obliczeniowych można obsłużyć nawet do **1 miliona żądań jednocześnie**.

Pętla zdarzeń (Event loop)



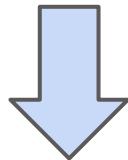
Przykład prostego kodu (z strony nodejs.org)

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('<html><head></head><body>Test</body></html>');
}).listen(1337, '127.0.0.1');

console.log('Server running at http://127.0.0.1:1337/');
```

We wcześniejszym przykładzie serwer zwracał w odpowiedzi zwykły tekst, natomiast teraz zwraca dokument hipertekstowy ze statyczną stroną internetową.



Stworzyliśmy serwer obsługujący żądania HTTP !!!



Instalacja dla Linux

Przechodzimy do kartoteki bin gdzie wykonujemy polecenie:

```
MacBook-Pro-Marcin-2:bin marcin$ ./node -v  
v0.10.33
```

Program odpowiada numerem wersji

Zaleca się zainstalowanie globalnej wersji node.js dostępnej dla wszystkich użytkowników oraz możliwej do uruchomianie a każdego miejsca w strukturze kartotek.

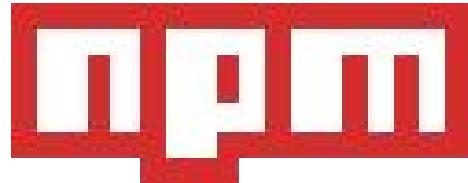
Uruchomienie skryptu polega na wywołaniu programu “node” z parametrem określającym nazwę skryptu (gdy node.js jest zainstalowany lokalnie):

```
| > ./node serwer.js |
```



Repozytorium NPM

Integralną częścią środowiska NODE.JS, jest bogate repozytorium modułów (bibliotek), dzięki któremu mamy dostęp do wielu gotowych funkcji.



<http://npmjs.org/>

Na stronie internetowej projektu można przeglądać i wyszukiwać pakiety, jednak ich instalacja odbywa się w poziomu wiersza poleceń:

```
| > ./npm install nazwa-pakietu
```

Pakiety instalują się w kartotece **lib/node_modules**.

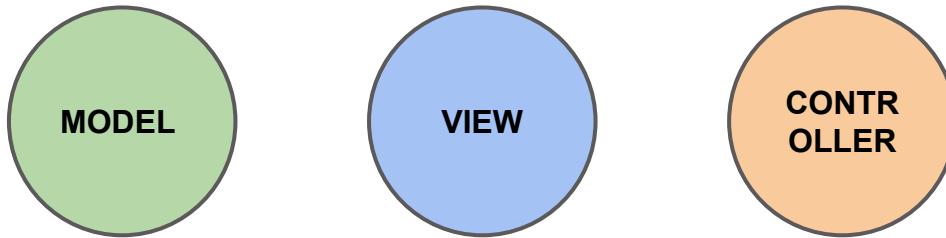
Do globalnej instalacji pakietów NPM należy posiadać prawa administratora i wydać polecenie:

```
| > ./npm -g install nazwa-pakietu
```

Model-View-Controller

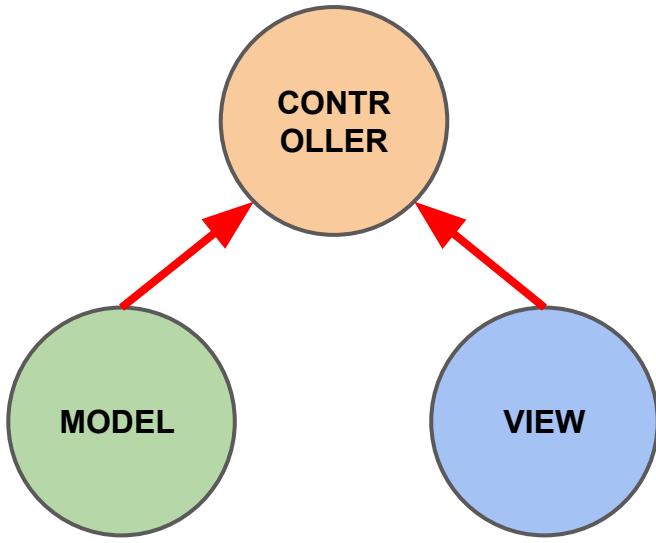
Model-View-Controller (MVC) [Model-Widok-Kontroler] - jest to wzorzec projektowy (podejście które jest bazą w oparciu o którą tworzymy aplikację), dzielący projektowaną aplikację na trzy warstwy:

- **Model (dane / logika)**
- **Widok (prezentacja danych)**
- **Kontroler (interakcja z użytkownikiem + sterowanie aplikacją)**

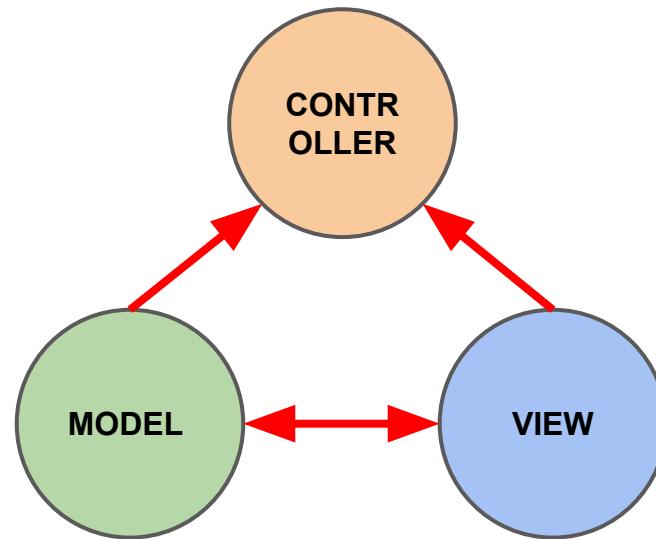


Można go zaimplementować bez użycia bibliotek czy specjalistycznych platform programistycznych, stosując jasne reguły podziału na konkretne komponenty w kodzie źródłowym. W ten sposób każdy komponent aplikacji można niezależnie od siebie rozwijać, implementować i testować.

Model-View-Controller



Model Pasywny



Model Aktywny

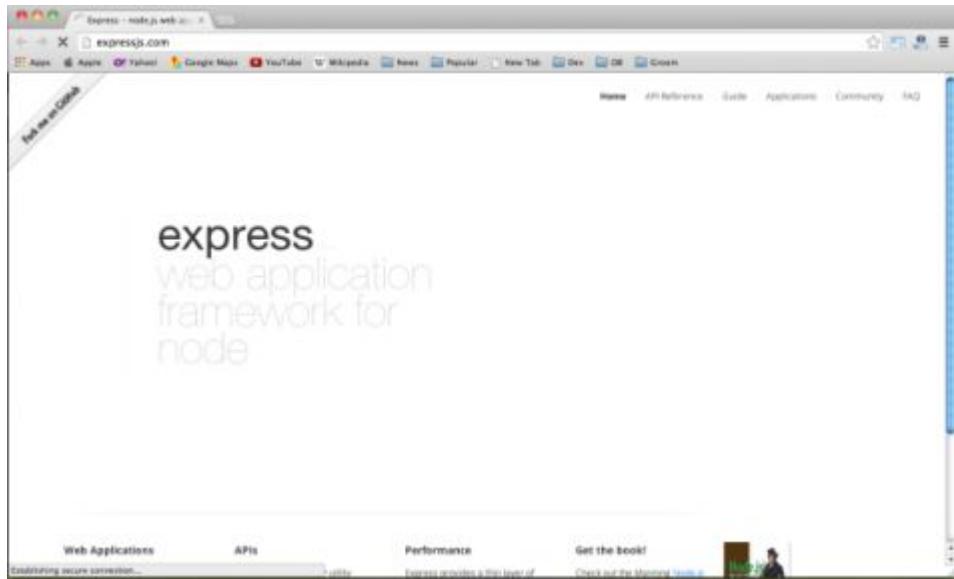
W modelu “pasywny” nie ma wymiany danych pomiędzy modelem a widokiem z pominięciem kontrolera, wszystkie akcje są wywoływanie i sterowane przez kontroler.

W model aktywnym model może bezpośrednio przekazywać dane do widoku z pominięciem kontrolera.

Node.js + MVC

Istnieje kilka środowisk ułatwiających tworzenie aplikacji według wzorca projektowego MVC w systemie NODE.JS:

<http://expressjs.com/>





Express JS - instalacja

Instalacja Express.js odbywa się za pomocą menadżera pakietów NPM.

The screenshot shows the npm homepage. At the top left is the red 'npm' logo. To its right is a search bar with the placeholder 'find packages'. Next to the search bar is a magnifying glass icon. Further to the right are links for 'sign up or log in' and a small user profile icon.

★ express

Fast, unopinionated, minimalist web framework

npm v4.10.7 downloads 2M/month build passing coverage 100%

```
var express = require('express')
var app = express()

app.get('/', function (req, res) {
  res.send('Hello World')
})

app.listen(3000)
```

⬇ npm install express

Package Info

4.10.7 published 3 days ago by **dougwilson**

github.com/strongloop/express

expressjs.com

MIT license

Stats

67 252 downloads in the last day

```
> npm install express
```

```
> npm install express-generator
```



Express JS - projekt “0”

Po zainstalowaniu środowiska express.js wraz z generatorem dostajemy nowe narzędzie służące do tworzenie aplikacji według wzorca projektowego MVC. Fizycznie jest to program wykonywalny który służy do generowania projektów “0” oraz obsługi całej aplikacji:



Express JS - projekt “0”

Po zainstalowaniu środowiska express.js wraz z generatorem dostajemy nowe narzędzie służące do tworzenie aplikacji według wzorca projektowego MVC. Fizycznie jest to program wykonywalny który służy do generowania projektów “0” oraz obsługi całej aplikacji:

- 1) W pierwszym kroku tworzymy nowy folder w którym będziemy przechowywać pliki projektu:

```
> mkdir test
```

Express JS - projekt “0”

Po zainstalowaniu środowiska express.js wraz z generatorem dostajemy nowe narzędzie służące do tworzenie aplikacji według wzorca projektowego MVC. Fizycznie jest to program program wykonywalny który służy do generowania projektów “0” oraz obsługi całej aplikacji:

- 1) W pierwszym kroku tworzymy nowy folder w którym będziemy przechowywać pliki projektu:

```
> mkdir test
```

- 2) W drugim kroku generujemy projekt “0”

```
> express test
```

Express JS - projekt “0”

Po wykonaniu powyższej operacji pusty do tej pory katalog został wypełniony plikami stanowiący w pełni dzielającą aplikację według wzorca MVC:

```
MacBook-Pro-Marcin-2:~ marcin$ express test

  create : test
  create : test/package.json
  create : test/app.js
  create : test/public
  create : test/public/images
  create : test/public/stylesheets
  create : test/public/stylesheets/style.css
  create : test/routes
  create : test/routes/index.js
  create : test/routes/users.js
  create : test/views
  create : test/views/index.jade
  create : test/views/layout.jade
  create : test/views/error.jade
  create : test/bin
  create : test/bin/www
  create : test/public/javascripts

install dependencies:
$ cd test && npm install

run the app:
$ DEBUG=test ./bin/www
```

Express JS - projekt “0”

Po wykonaniu powyższej operacji pusty do tej pory katalog został wypełniony plikami stanowiący w pełni dzielącą aplikację według wzorca MVC:

```
MacBook-Pro-Marcin-2:~ marcin$ express test

  create : test
  create : test/package.json
  create : test/app.js
  create : test/public
  create : test/public/images
  create : test/public/stylesheets
  create : test/public/stylesheets/style.css
  create : test/routes
  create : test/routes/index.js
  create : test/routes/users.js
  create : test/views
  create : test/views/index.jade
  create : test/views/layout.jade
  create : test/views/error.jade
  create : test/bin
  create : test/bin/www
  create : test/public/javascripts

install dependencies:
$ cd test && npm install

run the app:
$ DEBUG=test ./bin/www
```

Lista wygenerowanych plików
w kartotece “test”

Express JS - projekt “0”

Po wykonaniu powyższej operacji pusty do tej pory katalog został wypełniony plikami stanowiący w pełni dzielącą aplikację według wzorca MVC:

```
MacBook-Pro-Marcin-2:~ marcin$ express test

  create : test
  create : test/package.json
  create : test/app.js
  create : test/public
  create : test/public/images
  create : test/public/stylesheets
  create : test/public/stylesheets/style.css
  create : test/routes
  create : test/routes/index.js
  create : test/routes/users.js
  create : test/views
  create : test/views/index.jade
  create : test/views/layout.jade
  create : test/views/error.jade
  create : test/bin
  create : test/bin/www
  create : test/public/javascripts

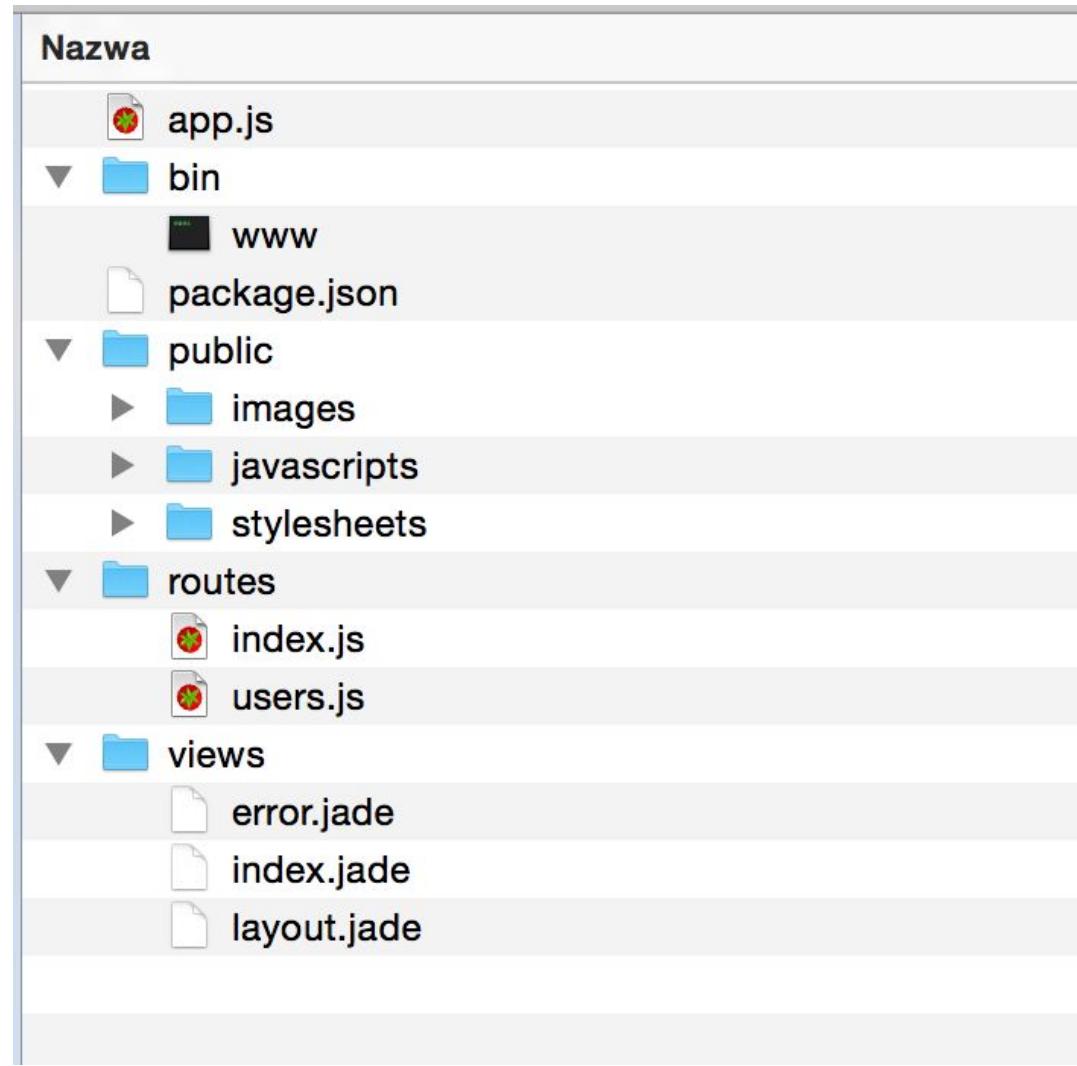
install dependencies:
$ cd test && npm install

run the app:
$ DEBUG=test ./bin/www
```

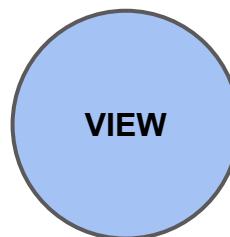
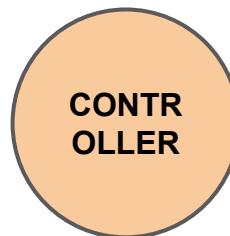
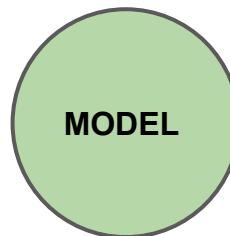
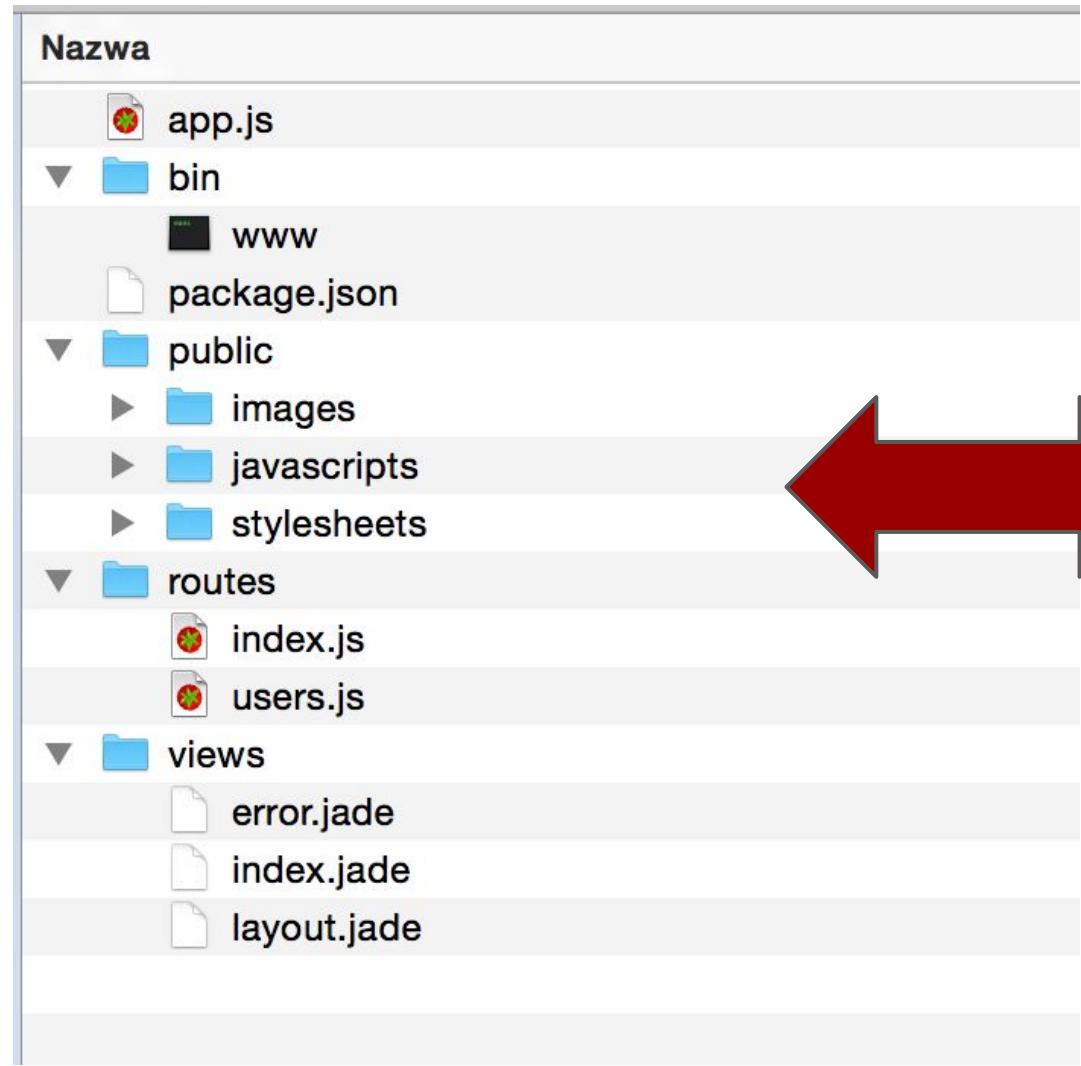
Lista wygenerowanych plików w kartotece “test”

Instalacja standardowych pakietów

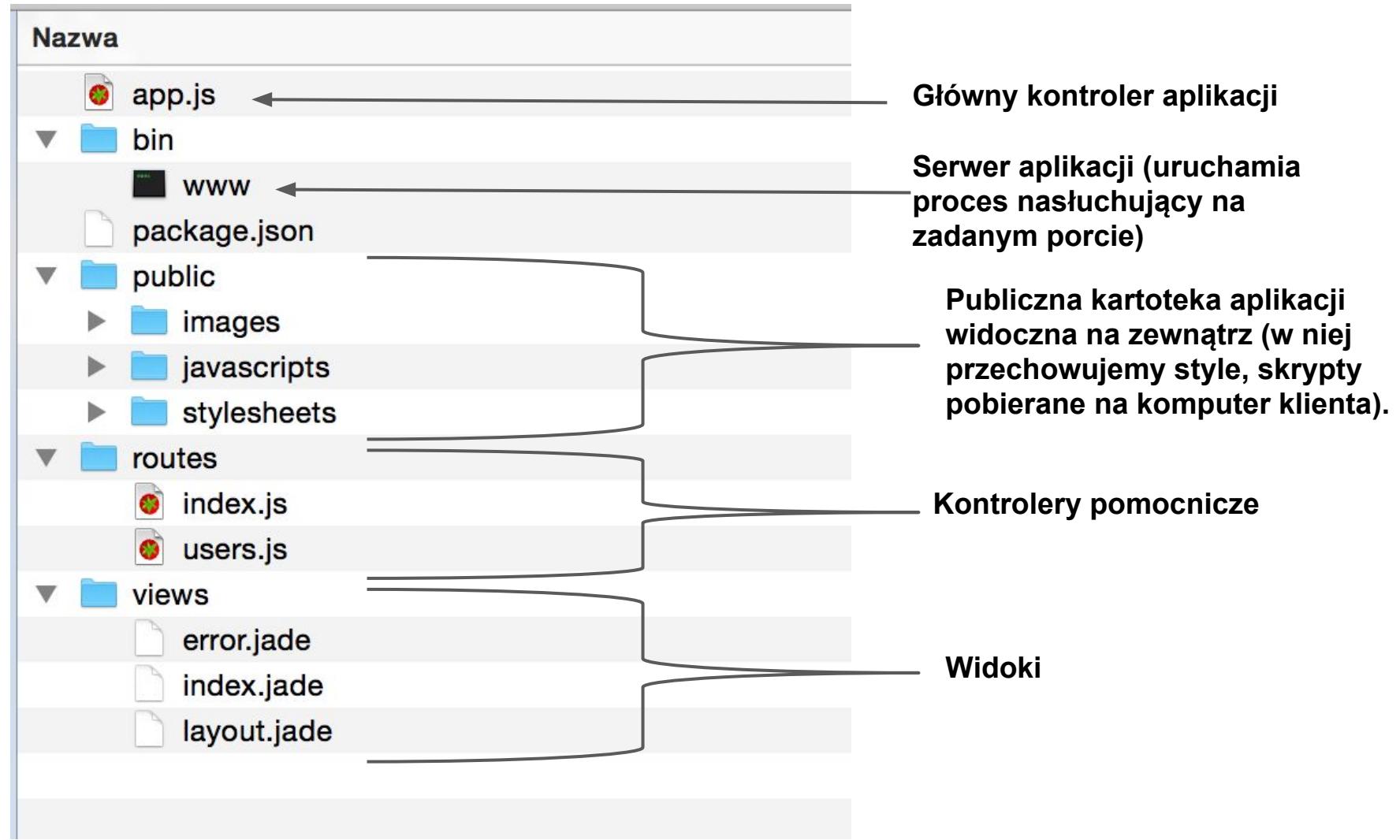
Express JS - struktura aplikacji



Express JS - struktura aplikacji



Express JS - struktura aplikacji



Repozytorium NPM

Integralną częścią środowiska NODE.JS, jest bogate repozytorium modułów (bibliotek), dzięki któremu mamy dostęp do wielu gotowych funkcji.



<http://npmjs.org/>

Standardowo w kartotece projektu powstaje plik `package.json`, który zawiera informację o zainstalowanych pakietach w danym projekcie:

```
{
  "name": "hello",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "body-parser": "~1.0.0",
    "express": "~4.2.0",
    "jade": "~1.3.0",
    "static-favicon": "~1.0.0"
  }
}
```

W projektach generowanych np. w Express plik ten powstaje automatycznie.

W innym przypadku należy taki plik wytworzyć samemu poleceniem:
> `npm init`

Aby przy instalacji dodać informację o nowym pakiecie do tego pliku należy wykonać polecenie:
> `npm install NAZWA_PAKIETU --save`



Express JS - Uruchamianie

Po wygenerowaniu projektu “0” dostajemy w pełni działający wzorzec aplikacji który można uruchomić. Każdą aplikację przygotowaną za pomocą środowiska Express.js uruchamiamy w następujący sposób:



Express JS - Uruchamianie

Po wygenerowaniu projektu “0” dostajemy w pełni działający wzorzec aplikacji który można uruchomić. Każdą aplikację przygotowaną za pomocą środowiska Express.js uruchamiamy w następujący sposób:

```
~/test> npm start
```

Express JS - Uruchamianie

Po wygenerowaniu projektu “0” dostajemy w pełni działający wzorzec aplikacji który można uruchomić. Każdą aplikację przygotowaną za pomocą środowiska Express.js uruchamiamy w następujący sposób:

```
~/test> npm start
```

MacBook-Pro-Marcin-2:test marcin\$ npm start

```
> test@0.0.0 start /Users/marcin/test
> node ./bin/www
```



Express JS - Uruchamianie

Po wygenerowaniu projektu “0” dostajemy w pełni działający wzorzec aplikacji który można uruchomić. Każdą aplikację przygotowaną za pomocą środowiska Express.js uruchamiamy w następujący sposób:

```
~/test> npm start
```

MacBook-Pro-Marcin-2:test marcin\$ npm start

```
> test@0.0.0 start /Users/marcin/test
> node ./bin/www
```

Po uruchomieniu aplikacja uruchamia serwer HTTP nasłuchujący na adresie IP: 127.0.0.1 oraz standardowym porcie 3000



Express JS - Uruchamianie



Strona widoczna po uruchomieniu aplikacji “0”

Express JS - Uruchamianie



```
MacBook-Pro-Marcin-2:test marcin$ npm start

> test@0.0.0 start /Users/marcin/test
> node ./bin/www

GET / 304 194.179 ms --
GET /stylesheets/style.css 304 2.135 ms --
GET /favicon.ico 404 31.420 ms - 1042
```



Struktura aplikacji

Plik tworzący serwer i obsługujący nasłuchiwanie żądań za pomocą protokołu http jest realizowane w następujący sposób:

Plik: ./bin/www

Struktura aplikacji

Plik tworzący serwer i obsługujący nasłuchiwanie żądań za pomocą protokołu http jest realizowane w następujący sposób:

Plik: ./bin/www

```
-----  
|  
|  
|  /**  
|  * Get port from environment and store in Express.  
|  */  
  
|  var port = parseInt(process.env.PORT, 10) || 3000;  
|  app.set('port', port);  
  
|  /**  
|  * Create HTTP server.  
|  */  
  
|  var server = http.createServer(app);  
|  
-----
```



Struktura aplikacji

Główny kontroler aplikacji jest zlokalizowany w pliku “app.js” znajdujący się w głównej kartotece:

Plik: app.js

Struktura aplikacji

Główny kontroler aplikacji jest zlokalizowany w pliku “app.js” znajdujący się w głównej kartotece:

```
var routes = require('./routes/index');
var users = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');

// uncomment after placing your favicon in /public
//app.use(favicon(__dirname + '/public/favicon.ico'));
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', routes);
app.use('/users', users);
```

Plik: app.js



Struktura aplikacji

Funkcje kontrolera odpowiedzialne za realizację konkretnych funkcjonalności znajdują się w kartotece routes. W wersji “0” plik zawiera tylko funkcję obsługującą wyświetlanie strony głównej aplikacji:

Plik: routes/index.js

Struktura aplikacji

Funkcje kontrolera odpowiedzialne za realizację konkretnych funkcjonalności znajdują się w kartotece routes. W wersji “0” plik zawiera tylko funkcję obsługującą wyświetlanie strony głównej aplikacji:

Plik: routes/index.js

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

Struktura aplikacji

Funkcje kontrolera odpowiedzialne za realizację konkretnych funkcjonalności znajdują się w kartotece routes. W wersji “0” plik zawiera tylko funkcję obsługującą wyświetlanie strony głównej aplikacji:

```
var express = require('express')
var router = express.Router()

/* GET home page. */
router.get('/', function(req, res) {
  res.render('index', { title: 'Express' });
})

module.exports = router;
```

Plik: routes/index.js

Metoda która realizuje wyświetlanie odpowiedniego widoku



Funkcje routera

```
.....  
:router.get('/', function(req, res) {  
  res.render('index', { title: 'Express' });  
});  
.....
```

Metoda GET, obsługująca żądania wyświetlania
zadanego adresu URL wpisanego w przeglądarce
przez użytkownika

Funkcje routera

```
.....  
:router.get('/', function(req, res) {  
  res.render('index', { title: 'Express' });  
});  
.....
```

Metoda GET, obsługująca żądania wyświetlenia
zadanego adresu URL wpisanego w przeglądarce
przez użytkownika

Możemy również obsługiwać żądania typu POST:

```
.....  
.....  
:router.post();  
.....  
.....
```

Funkcje routera

```
.....  
:router.get('/', function(req, res) {  
  res.render('index', { title: 'Express' });  
});  
.....
```

Ścieżka (lub wyrażenie regularne), które mapuje adres wpisany w przeglądarce.

<http://127.0.0.1:3000/>

<http://127.0.0.1:3000/?name=test>



Funkcje routera

```
.....  
:router.get('/', function(req, res) {  
  res.render('index', { title: 'Express' });  
});  
.....
```

Wywołanie zwrotne (callback) mające zostać wykonanw w wyniku odpowiedzi na żdanie GET

Funkcje routera

```
.....  
:router.get('/', function(req, res) {  
  res.render('index', { title: 'Express' });  
});  
.....
```

Wywołanie zwrotne (callback) mające zostać wykonanw w wyniku odpowiedzi na żdanie GET

req - obiekt przenoszący zapytanie klienta (żądanie)

res - Obiekt przenoszący odpowiedź aplikacji do klienta

Funkcje routera

```
.....  
:router.get('/', function(req, res) {  
:  res.render('index', { title: 'Express' });  
});  
.....
```

Metoda “renderująca” widok (ostateczny HTML)
który jest następnie przesyłany do klienta.

Metoda ta jako pierwszy argument przyjmuje nazwę widoku który ma zostać wypisany (nazwa pliku z kartoteki views bez rozszerzenia) oraz opcjonalnie obiekt (JSON) z danymi które mają zostać przekazane do widoku.

W przykładzie powyżej renderujemy widok o nazwie “index” i przekazujemy obiekt zawierający jedną zmienną o nazwie title.

Funkcje routera

```
router.get('/', function(req, res) {  
  var file_name = __dirname + '/../public/' + 'file.pdf';  
  res.download(file);  
});
```

Metoda pobierająca pliki, a dokładniej mówiąc wskazująca na pliku które mają zostać przesłane klientowi po wywołaniu żądanego adresu URL.

Funkcje routera

```
:router.get('/', function(req, res) {  
  var file_name = __dirname + '/../public/' + 'file.pdf';  
  res.download(file);  
});
```

Metoda pobierająca pliki, a dokładniej mówiąc wskazująca na pliku które mają zostać przesłane klientowi po wywołaniu żądanego adresu URL.

Jako argument funkcji download musimy podać bezwzględną (lub względną w stosunku do kartoteki w której znajduje się plik kontrolera) ścieżkę do pliku. Jako drugi argument funkcji download można przekazać string mówiący pod jaką nazwą plik zostanie zapisany na dysk użytkownika:

```
res.download(file, 'moja_nazwa_pliku.pdf');
```

Funkcje routera

```
router.get('/', function(req, res) {  
  var options = {  
    root: __dirname + '/../public/',  
    dotfiles: 'deny',  
    headers: {  
      'x-timestamp': Date.now(),  
      'x-sent': true  
    }  
  };  
  
  var fileName = 'ph.pdf'  
  res.sendFile(fileName, options, function (err) {  
    if (err) {  
      console.log(err);  
      res.status(err.status).end();  
    }  
    else {  
      console.log('Sent:', fileName);  
    }  
  });  
});
```

Funkcje routera

```
router.get('/', function(req, res) {  
  var options = {  
    root: __dirname + '/../public/',  
    dotfiles: 'deny',  
    headers: {  
      'x-timestamp': Date.now(),  
      'x-sent': true  
    }  
  };  
  
  var fileName = 'ph.pdf'  
  res.sendFile(fileName, options, function (err) {  
    if (err) {  
      console.log(err);  
      res.status(err.status).end();  
    }  
    else {  
      console.log('Sent:', fileName);  
    }  
  });  
});
```

Metoda “sendFile” spełnia tą samą funkcjonalność do download. Jest dostępna od wersji 4.8

Funkcje routera

```
router.get('/', function(req, res) {  
  res.send('<h1>Witaj</h1>');  
});
```

Metoda `send`, służy do przesyłania prostych wartości tekstowych do użytkownika. Domyślny format przesyłany za pomocą tej metody jest ustawiony na "text/html".

Funkcje routera

```
: router.get('/', function(req, res) {  
  res.send('<h1>Witaj</h1>');  
});
```

Metoda send, służy do przesyłania prostych wartości tekstowych do użytkownika. Domyślny format przesyłany za pomocą tej metody jest ustawiony na "text/html".

Można również wysłać tablicę lub obiekt JSON, które zostaną następnie wyświetcone w formie preformatowanego html za pomocą znacznika `<pre>`:

```
:   res.send( [100, 200, 300] );  
:   res.send( {title: 'express'} );
```

Funkcje routera

```
router.get('/', function(req, res) {  
  res.sendStatus(200); // OK  
});
```

Metoda `sendStatus`, przesyła do przeglądarki tekstową postać odpowiedzi dla podanego jako argument kodu statusu serwera

Funkcje routera

```
: router.get('/', function(req, res) {  
  res.sendStatus(200); // OK  
});
```

Metoda `sendStatus`, przesyła do przeglądarki tekstową postać odpowiedzi dla podanego jako argument kodu statusu serwera

```
: res.sendStatus( 404 ); // Not Found  
: res.sendStatus( 500 ); // Internal Server Error
```

```
> test@0.0.0 start /Users/marcin/test  
> node ./bin/www  
  
GET /send 404 7.669 ms - 9  
GET /send 404 1.279 ms - 9  
GET /send 404 0.627 ms - 9  
GET /send 404 0.515 ms - 9
```



Funkcje routera

```
.....  
:router.post('/formularz', function(req, res) {  
  res.render('index', { title: req.body.fname });  
});  
.....
```

W podobny sposób możemy korzystać z metody “POST” do przekazywania danych wpisanych w formularzu. Adres URI stanowiący pierwszy argument tej metody jest tzw. akcją która ma zostać wykonane po przesłaniu formularza.

Wszystkie dane zebrane z pól przesyłanego formularza są przekazane przez obiekt “req. body.NAZWA_POLA”. Aby pobrać dane posługujemy się notacją obiektową.



Funkcje routera

```
.....  
:router.post('/formularz', function(req, res) {  
  res.render('index', { title: req.body.fname });  
});  
.....
```

W podobny sposób możemy korzystać z metody “POST” do przekazywania danych wpisanych w formularzu. Adres URI stanowiący pierwszy argument tej metody jest tzw. akcją która ma zostać wykonane po przesłaniu formularza.

Wszystkie dane zebrane z pól przesyłanego formularza są przekazane przez obiekt “req.body.NAZWA_POLA”. Aby pobrać dane posługujemy się notacją obiektową.

Formularz którego funkcja sterująca podana jest wyżej mógłby wyglądać następująco:

```
.....  
<form method='POST' action='/formularz'>  
  <input type='text' name='fname'>  
  <input type=submit name='Wyslij'>  
</form>  
.....
```

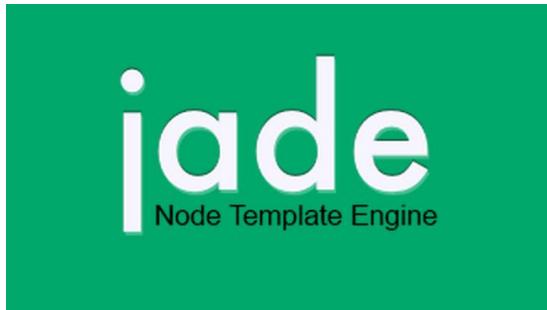


JADE

W środowisku Express.js w celu uproszczenia formy oraz usprawnienia tworzenia widoków wprowadzono nowy język pisania szablonów tzw. JADE (node template engine). Jest bardzo blisko językowi html, jednak w JADE nie występują znaczniki, a jedynie nazwy określające dane znacznik.

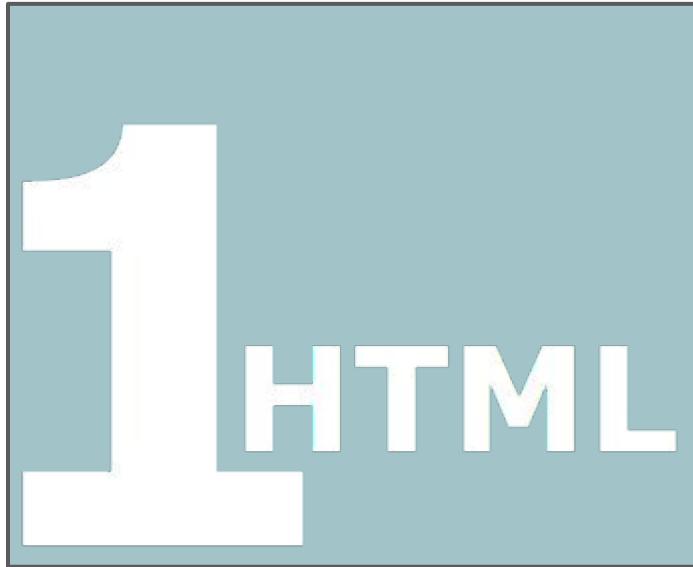
JADE

W środowisku Express.js w celu uproszczenia formy oraz usprawnienia tworzenia widoków wprowadzono nowy język pisania szablonów tzw. JADE (node template engine). Jest bardzo blisko językowi html, jednak w JADE nie występują znaczniki, a jedynie nazwy określające dane znacznik.



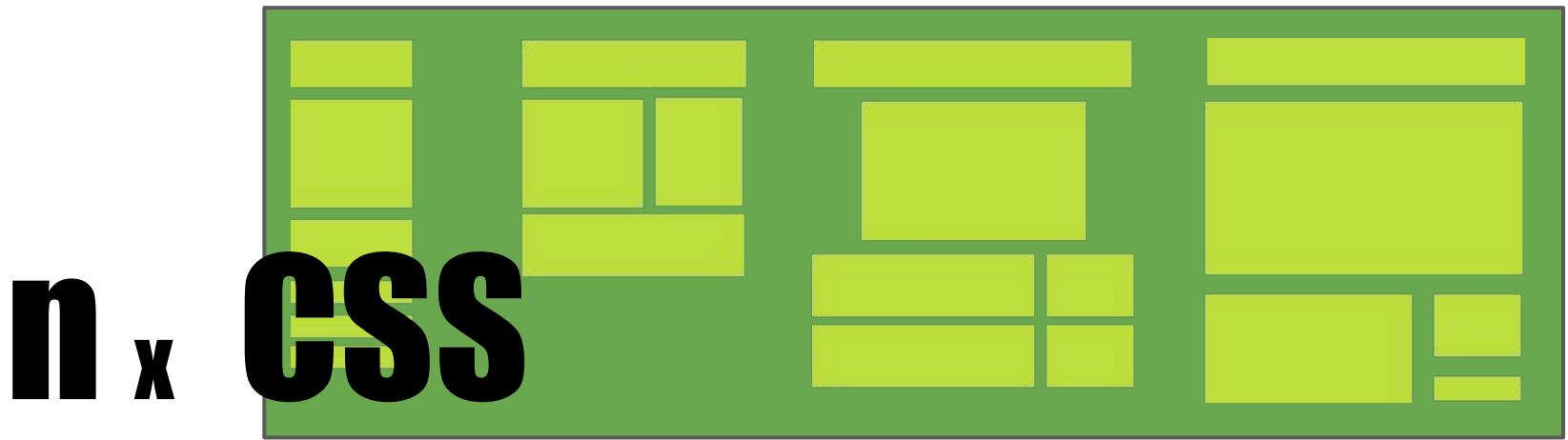
<http://jade-lang.com/>

Struktura modułowa HTML



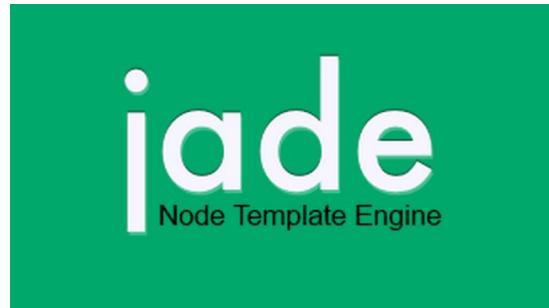
ZASADA:

W ogólności zasada jaka powinna przyświecać tworzeniu stron zgodnie z metodologią RWD jest tworzenie jednego pliku HTML, a dla formatowania jego wygądu wiele “layoutów” w CSS które będą zawierać odpowiednie reguły wyświetlania strony w zależności od rozdzielczości ekranu.



JADE

W środowisku Express.js w celu uproszczenia formy oraz usprawnienia tworzenia widoków wprowadzono nowy język pisania szablonów tzw. JADE (node template engine). Jest bardzo blisko językowi html, jednak w JADE nie występują znaczniki, a jedynie nazwy określające dane znacznik.



<http://jade-lang.com/>

```
<div>
  Element blokowy
</div>
```



```
div
  | Element blokowy
```

JADE

W języku JADE nie występują znaczniki otwierająca oraz następujące po nich znaczniki zamykające. Przykład:

```
<body>
  <h1>Jade</h1>
  <div id="container">
    <p>You are amazing</p>
    <p>
      JADE
    </p>
  </div>
</body>
```

JADE

W języku JADE nie występują znaczniki otwierająca oraz następujące po nich znaczniki zamykające. Przykład:

```
<body>
  <h1>Jade</h1>
  <div id="container">
    <p>You are amazing</p>
    <p>
      JADE
    </p>
  </div>
</body>
```



```
body
h1 Jade
#container
p You are amazinng
p.
  Jade
```

JADE

W języku JADE nie występują znaczniki otwierająca oraz następujące po nich znaczniki zamykające. Przykład:

```
<body>
  <h1>Jade</h1>
  <div id="container">
    <p>You are amazing</p>
    <p>
      JADE
    </p>
  </div>
</body>
```



```
body
h1 Jade
#container
p You are amazinng
p.
  Jade
```

Zatem jak zachować hierarchię występowania i zagnieżdżania znaczników??

JADE

W języku JADE nie występują znaczniki otwierająca oraz następujące po nich znaczniki zamykające. Przykład:

```
<body>
  <h1>Jade</h1>
  <div id="container">
    <p>You are amazing</p>
    <p>
      JADE
    </p>
  </div>
</body>
```



```
body
h1 Jade
#container
p You are amazinng
p.
  Jade
```

Hierarchia znaczników w plikach JADE jest zaznaczana za pomocą
“wcięć” w tekście.

JADE

Hierarchia znaczników w plikach JADE jest zaznaczana za pomocą “wcięć” w tekście.

```
<html>
  <head>
    <title> Jade </title>
  </head>
  <body>
    <p>
      JADE
    </p>
  </body>
</html>
```



```
.....html
.....
...head
.....
....title Jade
.....
...body
.....
....p
.....
....| JADE
```

W przykładzie powyżej kropki oznaczają wcięcia zrobione za pomocą spacji. Alternatywnie można używać “tabulatora” (stałego odstępu). Należy jednak w pojedyńczym pliku .jade używać spacji lub tabulacji (nigdy obu).

JADE

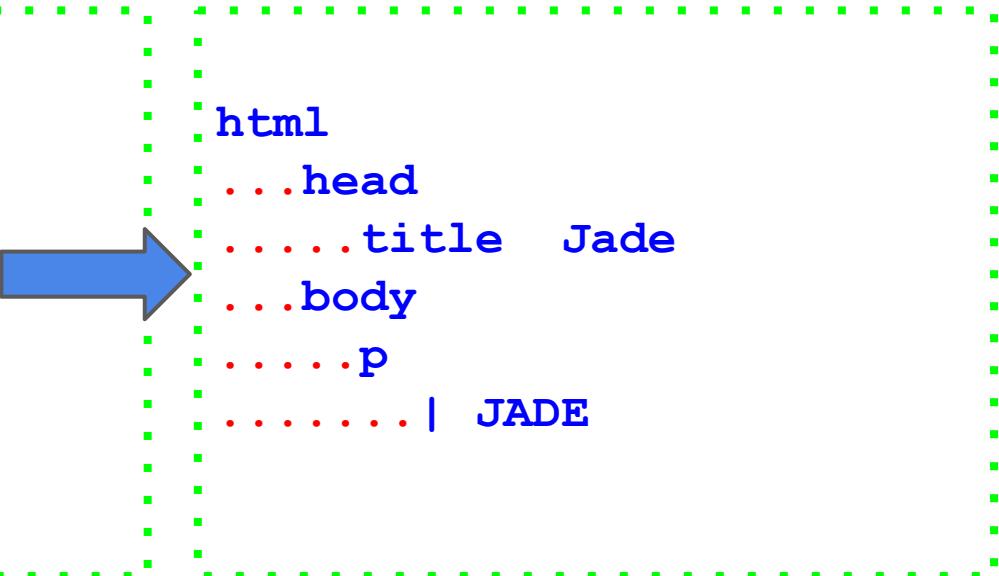
Hierarchia znaczników w plikach JADE jest zaznaczana za pomocą
“wcięć” w tekście.

```
<html>
  <head>
    <title> Jade </title>
  </head>
  <body>
    <p>
      JADE
    </p>
  </body>
</html>
```

JADE

Hierarchia znaczników w plikach JADE jest zaznaczana za pomocą “wcięć” w tekście.

```
<html>
  <head>
    <title> Jade </title>
  </head>
  <body>
    <p>
      JADE
    </p>
  </body>
</html>
```



```
          html
          ...head
          ....title Jade
          ...body
          ....p
          .....| JADE
```

W przykładzie powyżej kropki oznaczają wcięcia zrobione za pomocą spacji. Alternatywnie można używać “tabulatora” (stałego odstępu). Należy jednak w pojedyńczym pliku .jade używać spacji lub tabulacji (nigdy obu).



JADE

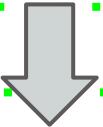
Zwykły tekst:

```
| Plain text can include <strong>html</strong>
| <p>It must always be on its own line</p>
```

JADE

Zwykły tekst:

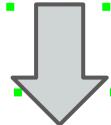
```
| Plain text can include <strong>html</strong>
| <p>It must always be on its own line</p>
|
| Plain text can include <strong>html</strong>
| p
|   | It must always be on its own line
```



JADE

Zwykły tekst:

```
| Plain text can include <strong>html</strong>
| <p>It must always be on its own line</p>
| |
| | Plain text can include <strong>html</strong>
| p
| | It must always be on its own line
```



Lista:

```
<ul>
  <li>Item A</li>
  <li>Item B</li>
  <li>Item C</li>
</ul>
```

<ul style="list-style-type: none"> Item A Item B Item C 	<ul style="list-style-type: none"> Item A Item B Item C
--	--

JADE

JADE umożliwia wykonywanie bardzo prostych operacji wyliczeniowych:

```
<ul>
  <li>1</li>
  <li>2</li>
  <li>3</li>
  <li>4</li>
  <li>5</li>
</ul>
```

ul
each val in [1, 2, 3, 4, 5]
li= val

JADE

JADE umożliwia wykonywanie bardzo prostych operacji wyliczeniowych:

```
<ul>
  <li>1</li>
  <li>2</li>
  <li>3</li>
  <li>4</li>
  <li>5</li>
</ul>
```

ul
each val in [1, 2, 3, 4, 5]
li= val

oraz zadań które mają być wykonane w pętli:

```
<li>item</li>
<li>item</li>
<li>item</li>
```

- for (var x = 0; x < 3; x++)
 li item

JADE

Instrukcja wyboru “case”:

```
.....
| - var friends = 10
| case friends
|   when 0
|     p you have no friends
|   when 1
|     p you have a friend
|   default
|     p you have #{friends} friends
| .....
| <p>you have 10 friends</p>
| .....
```



JADE

Zapis atrybutów w znacznikach:

```
<a href="google.com">Google</a>
<a href="google.com" class="button">Google</a>
```

```
a(href='google.com') Google
a(class='button', href='google.com') Google
```



JADE

Zapis atrybutów w znacznikach:

```
<a href="google.com">Google</a>
<a href="google.com" class="button">Google</a>
```

```
a(href='google.com') Google
a(class='button', href='google.com') Google
```

```
<a style="color:red;background:green"></a>
```

```
a(style={color: 'red', background: 'green'})
```

JADE

Zapis klas i identyfikatorów:

```
<a class="button"></a>
<div class="content"></div>
```

```
a.button
.content
```



JADE

Zapis klas i identyfikatorów:

```
<a class="button"></a>
<div class="content"></div>
```

```
a.button
.content
```

```
<a id="main-link"></a>
<div id="content"></div>
```

```
a#main-link
#content
```



JADE

Dziedziczenie widoków:

```
<!doctype html> . . . . .
<html>
  <head>
    <title>Article
  Title</title>
  </head>
  <body>
    <h1>My Article</h1>
  </body>
</html>
```

JADE

Dziedziczenie widoków:

```
<!doctype html>
<html>
  <head>
    <title>Article
Title</title>
  </head>
  <body>
    <h1>My Article</h1>
  </body>
</html>
```

```
//- layout.jade
doctype html
html
  head
    block title
      title Default title
  body
    block content
//- index.jade
extends ./layout.jade
block title
  title Article Title
block content
  h1 My Article
```

JADE

Dziedziczenie widoków:

```
<!doctype html>
<html>
  <head>
    <title>Article
Title</title>
  </head>
  <body>
    <h1>My Article</h1>
  </body>
</html>
```

```
//- layout.jade
doctype html
html
  head
    block title
      title Default title
  body
    block content
//- index.jade
extends ./layout.jade
block title
  title Article Title
block content
  h1 My Article
```

Taka organizacja widoków pozwala na stworzenie modularnej i hierarchicznej struktury strony.

JADE

Przykład formularza na stronie które są przesyłane do serwera za pomocą funkcji “post”:

```
div
  form(method=' POST'  action=' /formularz' )
    input(type='text'  name='imie')
    input(type='text'  name='nazwisko')
    input(type=submit name='Wyslij')
```

JADE

Przykład formularza na stronie które są przesyłane do serwera za pomocą funkcji “post”:

```
div
  form(method=' POST' action='/formularz')
    input(type='text' name='imie')
    input(type='text' name='nazwisko')
    input(type=submit name='Wyslij')
```

Metoda kontrolera odbierająca dane z formularza:

```
router.post('/formularz', function(req, res) {
  var _imie = req.body.imie;
  var _nazwisko = req.body.nazwisko;
  var osoba = _imie + _nazwisko;
  res.render('index', { title: osoba });
})
```



KONIEC WYKŁADU 10



UNIWERSYTET
JAGIELŁOŃSKI
W KRAKOWIE

Zaawansowane Techniki WWW (HTML, CSS i JavaScript)

Dr inż. Marcin Zieliński

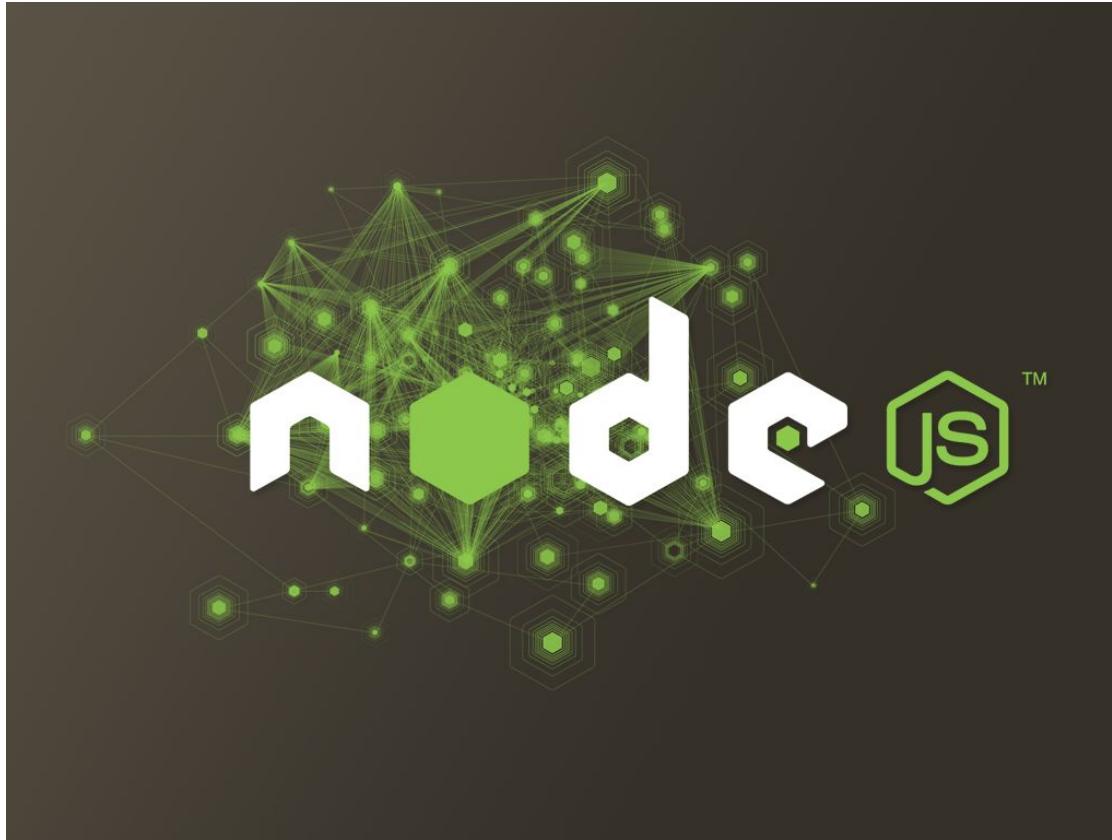
Środa 15:30 - 17:00 sala: A-1-04

WYKŁAD 11

Wykład dla kierunku: Informatyka Stosowana II rok

Rok akademicki: 2015/2016 - semestr zimowy

Node.js



<http://nodejs.org/>

JavaScript po stronie serwera

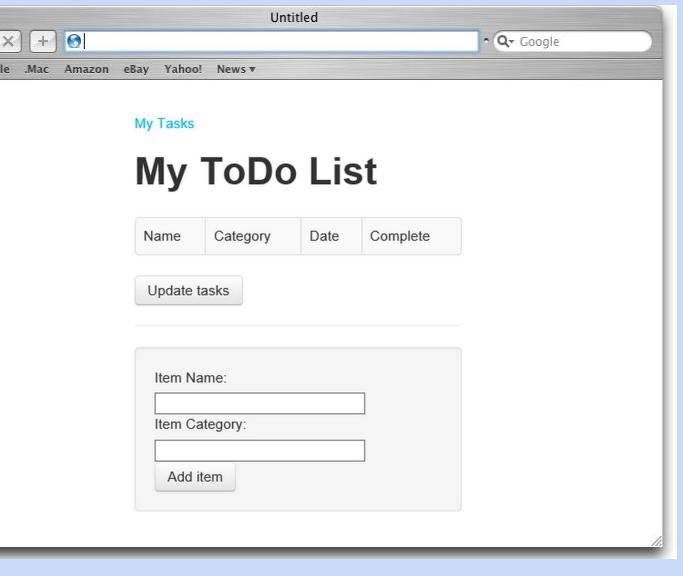
SERVER SIDE

```
1 var edge = require('edge');
2
3 var hello = edge.func(function () {/*
4     async (input) =>
5     {
6         return ".NET welcomes " + input.ToString();
7     }
8 */});
9
10 hello('Node.js', function (error, result) {
11     if (error) throw error;
12     console.log(result);
13});
```



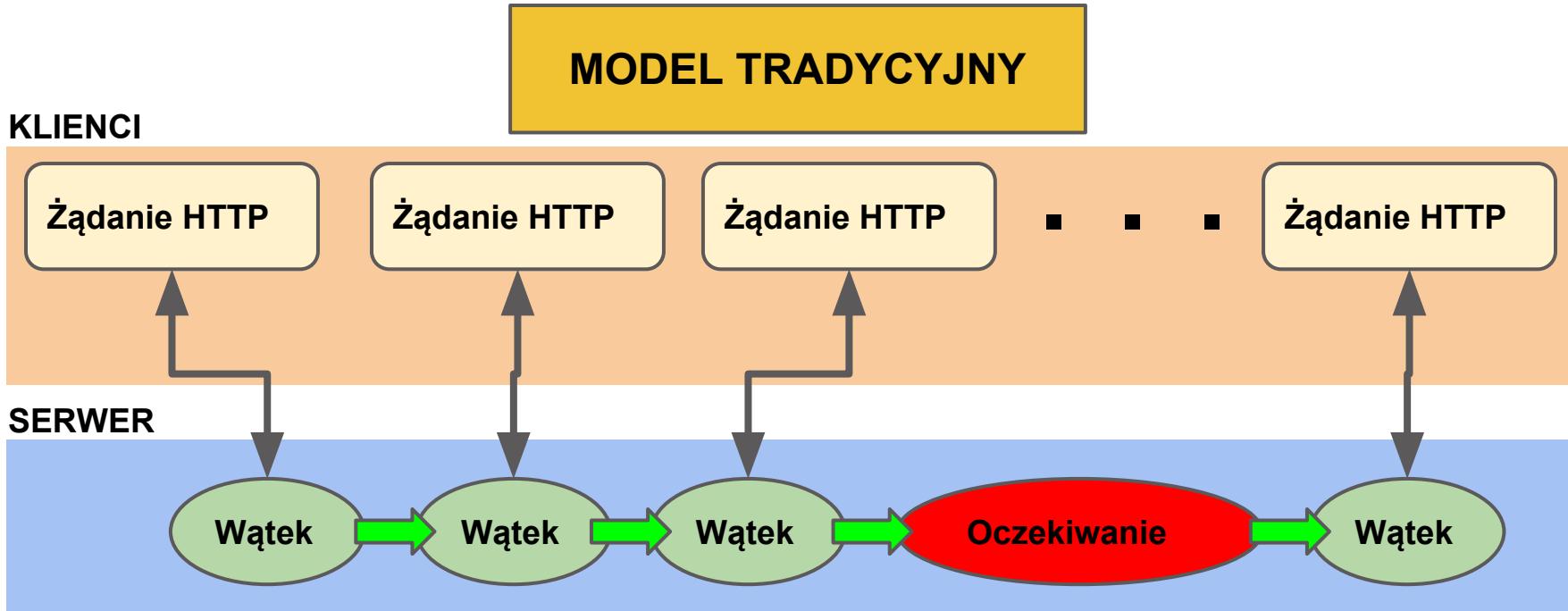
CLIENT SIDE

```
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>CSS3</title>
<link href="style.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="container"> <!-- Main Container -->
<div class="menu"> <!-- Menu here --></div>
<div class="article"><h2>CSS3 Sample</h2>
<p>CSS3 & HTML5 are so good!</p>
</div>
</div>
```



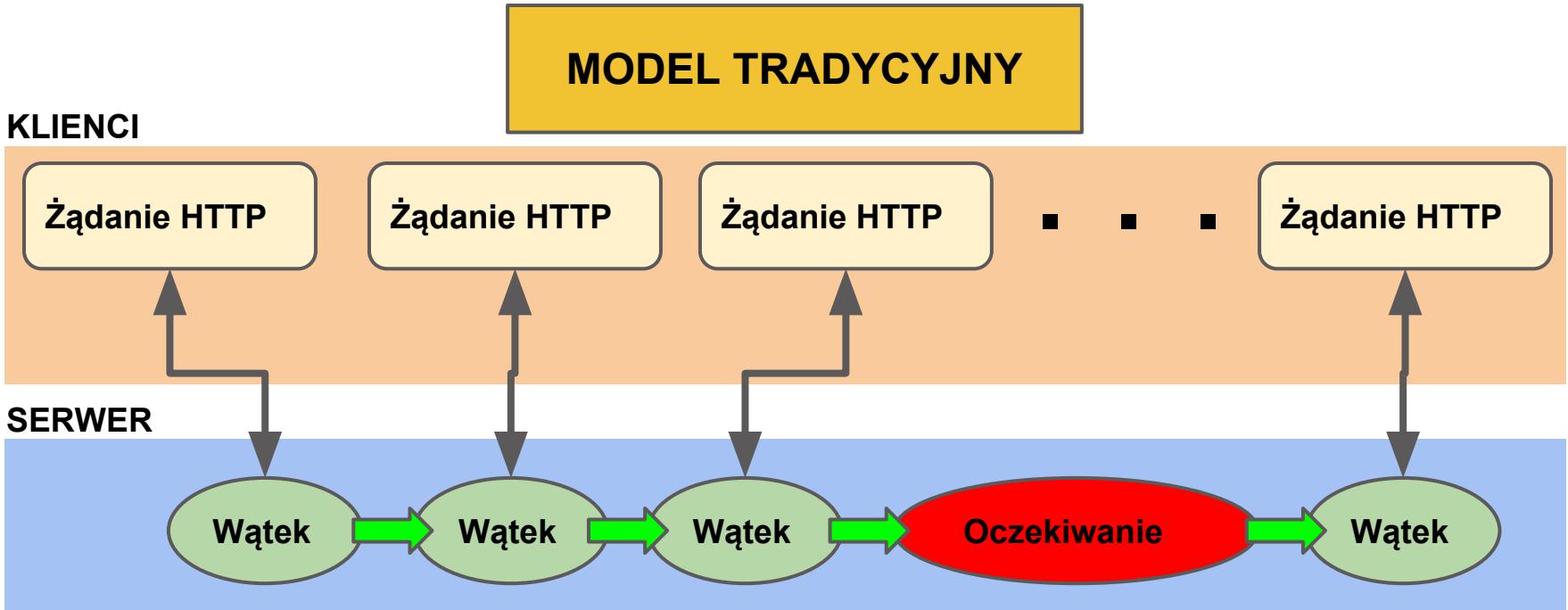
HTTP

Obsługa żądań



W tym modelu serwer do obsługi każdego żadania musi stworzyć osobny wątek z ograniczonej puli jaką może obsłużyć CPU. Powoduje to że przy dużej ilości żądań niektóre z nich muszą czekać w “kolejce” na zrealizowanie. W wyniku czego serwer zaczyna zaczyna wolniej działać co wydłuża czas oczekiwania na odpowiedź.

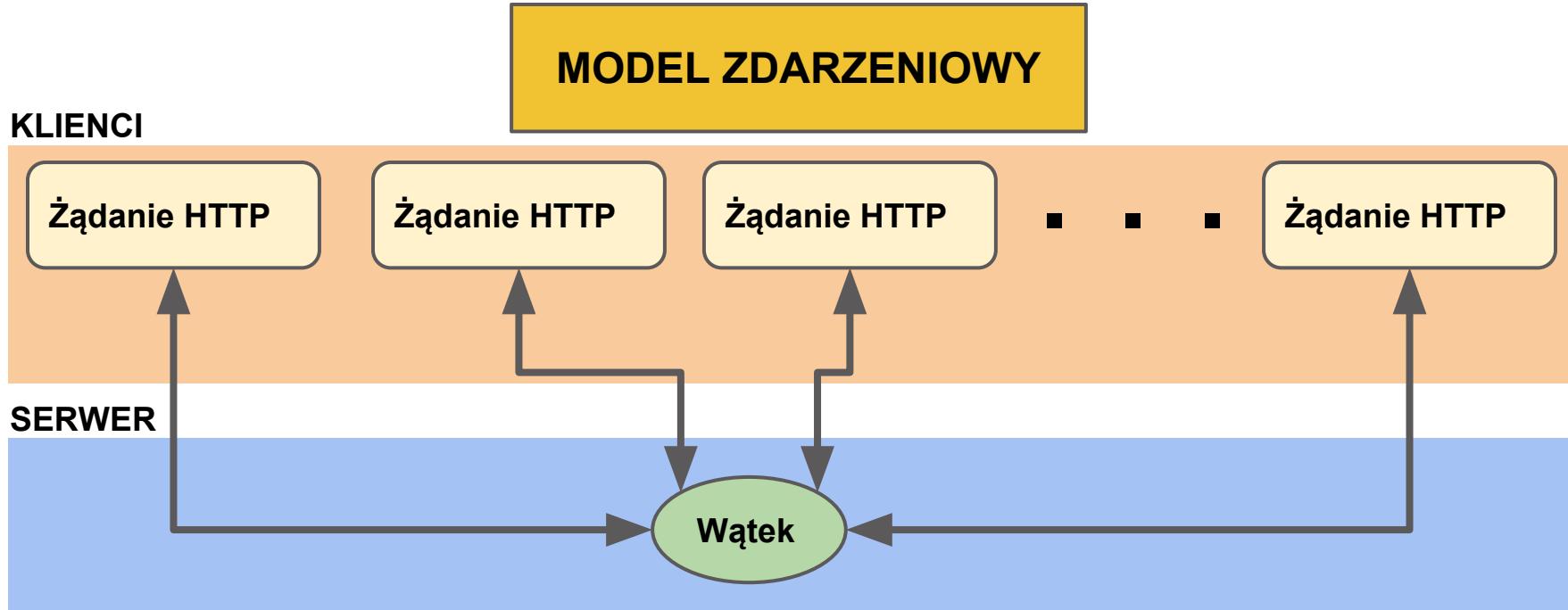
Obsługa żądań



Przykład:

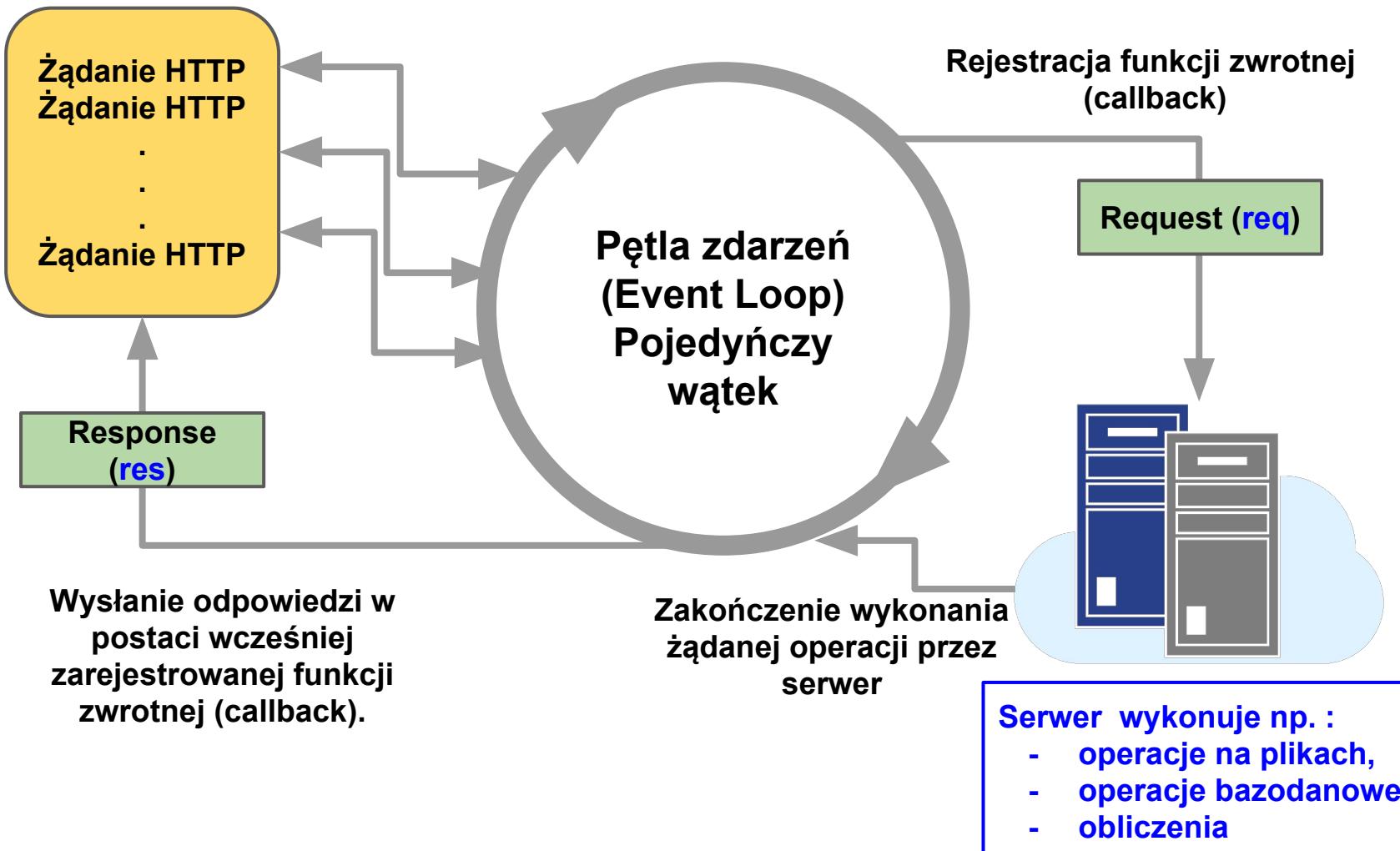
Dla systemu z 8GB pamięci RAM przydzielającego 2MB pamięci na wątek możemy obsłużyć maksymalnie w tym samym czasie **4000 żądań** (w rzeczywistości jest to mniej ponieważ zużywamy jeszcze pamięć na inne operacje).

Obsługa żądań



W modelu zdarzeniowym Node.js wykorzystuje tylko jeden wątek do obsługi wielu żądań, oraz “pętlę zdarzeń” co powoduje że aplikacja taka jest bardzo wydajna i skalowalna. W praktyce przy żadaniach które nie wymagają złożonych operacji obliczeniowych można obsłużyć nawet do **1 miliona żądań jednocześnie**.

Pętla zdarzeń (Event loop)





Przykład prostego kodu (z strony nodejs.org)

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');

console.log('Server running at http://127.0.0.1:1337/');
```

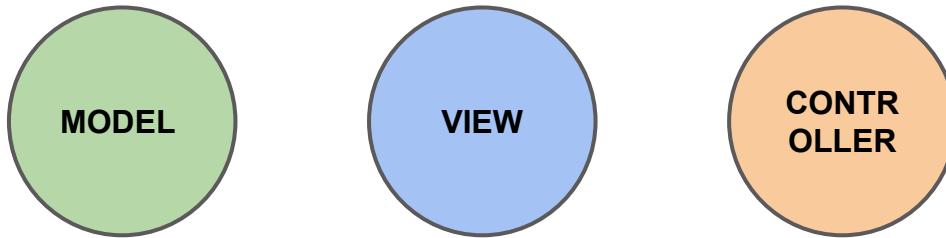
Prosty serwer przyjmujący żądania http

Założymy że powyższy kod jest zapisany w pliku “serwer.js”

Model-View-Controller

Model-View-Controller (MVC) [Model-Widok-Kontroler] - jest to wzorzec projektowy (podejście które jest bazą w oparciu o którą tworzymy aplikację), dzielący projektowaną aplikację na trzy warstwy:

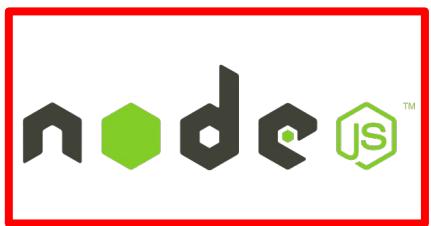
- **Model (dane / logika)**
- **Widok (prezentacja danych)**
- **Kontroler (interakcja z użytkownikiem + sterowanie aplikacją)**



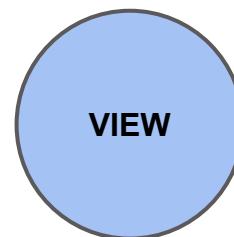
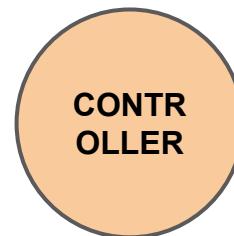
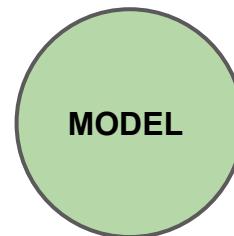
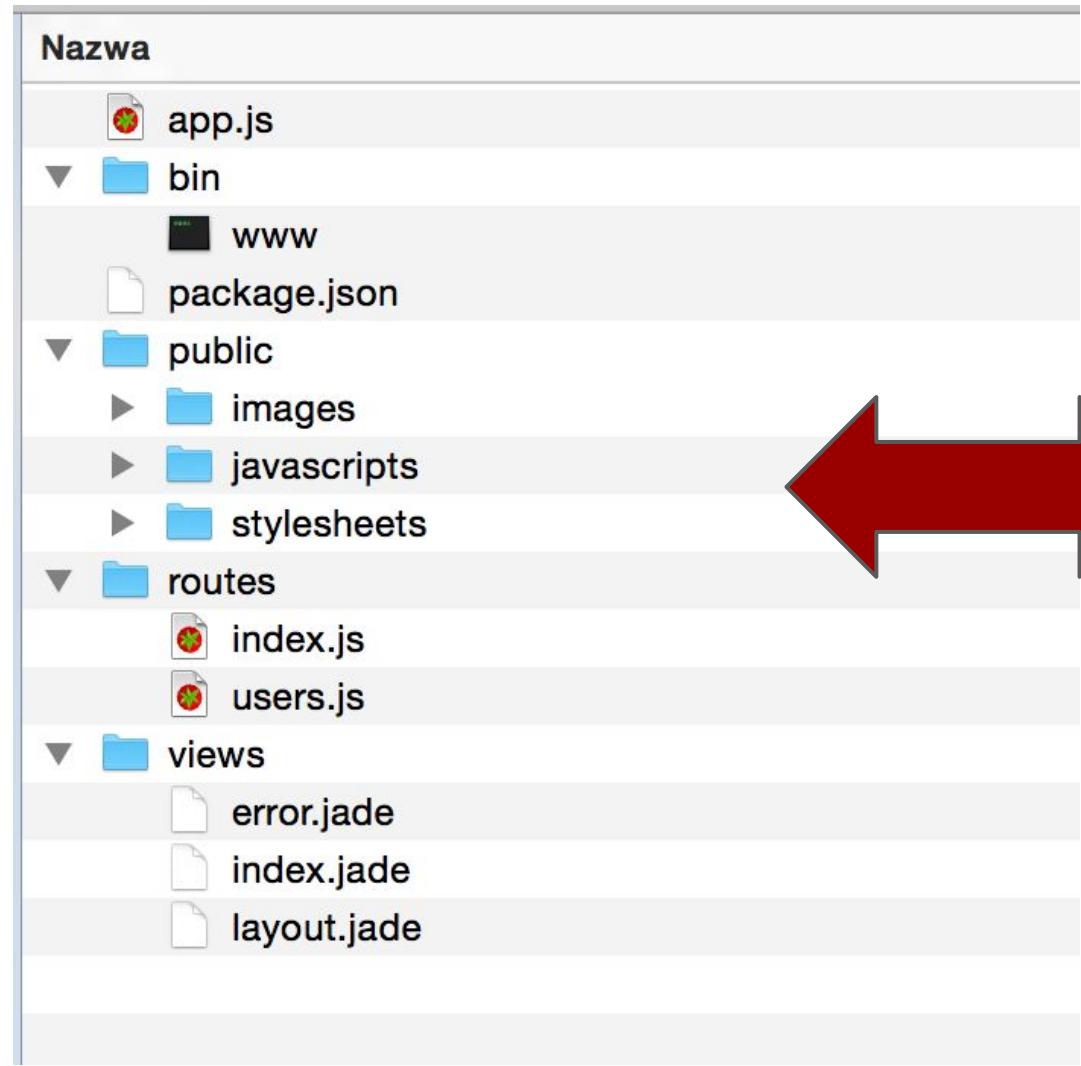
Można go zaimplementować bez użycia bibliotek czy specjalistycznych platform programistycznych, stosując jasne reguły podziału na konkretne komponenty w kodzie źródłowym. W ten sposób każdy komponent aplikacji można niezależnie od siebie rozwijać, implementować i testować.

Express JS

Express.js jest środowiskiem które pozwala na tworzenie aplikacji internetowych w formie jednostronnicowych oraz wielostronowych witryn dostosowanych do wyświetlania na urządzeniach mobilnych oraz normalnych komputerach.



Express JS - struktura aplikacji





Express JS - Uruchamianie



Strona widoczna po uruchomieniu aplikacji “0”



Struktura aplikacji

Funkcje kontrolera odpowiedzialne za realizację konkretnych funkcjonalności znajdują się w kartotece routes. W wersji “0” plik zawiera tylko funkcję obsługującą wyświetlanie strony głównej aplikacji:

Plik: routes/index.js

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

Funkcje routera

```
.....  
:router.post('/formularz', function(req, res) {  
  res.render('index', { title: req.body.fname });  
});  
.....
```

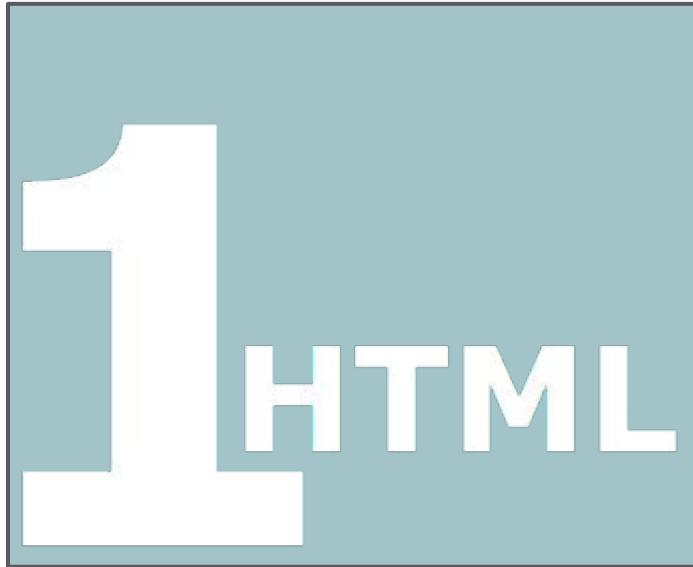
W podobny sposób możemy korzystać z metody “POST” do przekazywania danych wpisanych w formularzu. Adres URI stanowiący pierwszy argument tej metody jest tzw. akcją która ma zostać wykonane po przesłaniu formularza.

Wszystkie dane zebrane z pól przesyłanego formularza są przekazane przez obiekt “req. body.NAZWA_POLA”. Aby pobrać dane posługujemy się notacją obiektową.

Formularz którego funkcja sterująca podana jest wyżej mógłby wyglądać następująco:

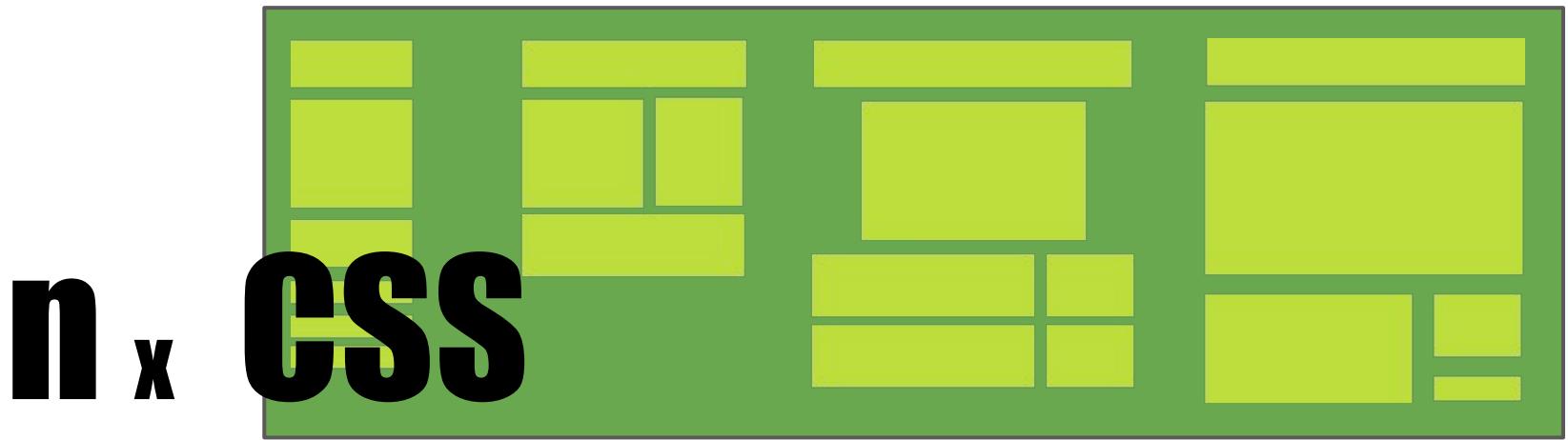
```
.....  
<form method='POST' action='/formularz'>  
  <input type='text' name='fname'>  
  <input type=submit name='Wyslij'>  
</form>  
.....
```

Struktura modułowa HTML



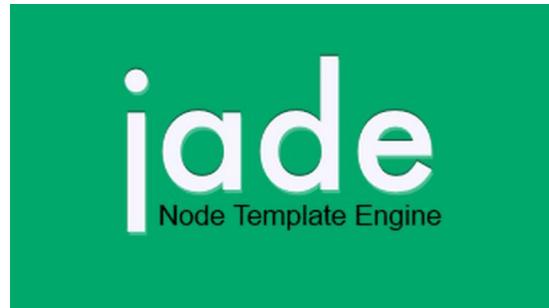
ZASADA:

W ogólności zasada jaka powinna przyświecać tworzeniu stron zgodnie z metodologią RWD jest tworzenie jednego pliku HTML, a dla formatowania jego wygądu wiele “layoutów” w CSS które będą zawierać odpowiednie reguły wyświetlania strony w zależności od rozdzielczości ekranu.



JADE

W środowisku Express.js w celu uproszczenia formy oraz usprawnienia tworzenia widoków wprowadzono nowy język pisania szablonów tzw. JADE (node template engine). Jest bardzo blisko językowi html, jednak w JADE nie występują znaczniki, a jedynie nazwy określające dane znacznik.



<http://jade-lang.com/>

```
<div>
  Element blokowy
</div>
```



```
div
| Element blokowy
```

JADE

Dziedziczenie widoków:

```
<!doctype html>
<html>
  <head>
    <title>Article
Title</title>
  </head>
  <body>
    <h1>My Article</h1>
  </body>
</html>
```

```
//- layout.jade
doctype html
html
  head
    block title
      title Default title
  body
    block content
//- index.jade
extends ./layout.jade
block title
  title Article Title
block content
  h1 My Article
```

Taka organizacja widoków pozwala na stworzenie modularnej i hierarchicznej struktury strony.

Bazy danych

Jednym z istotnych aspektów tworzenia nowoczesnych aplikacji internetowych jest przechowywania i wymiana danych. Najczęściej dane które są używane w aplikacjach są przechowywane w bazach danych.

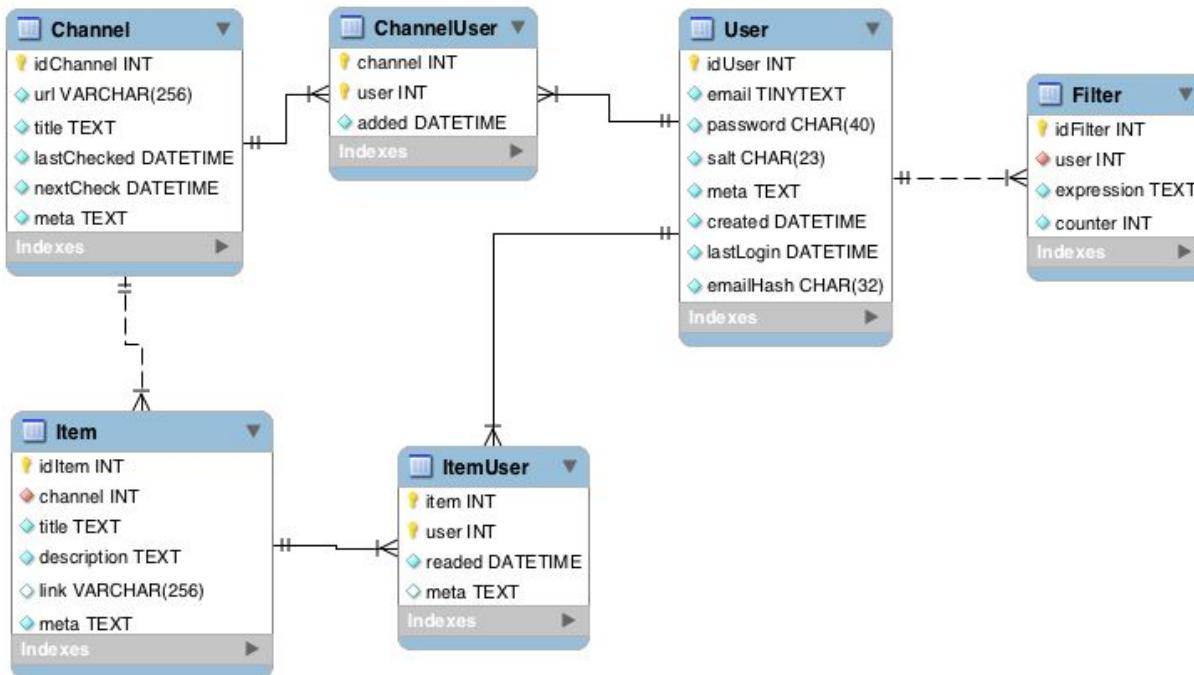
Node.js (oraz wszystkie środowiska programistyczne takie jak Express.js) posiadają interfejsy do najpopularniejszych typów baz danych:

- MySQL
- PostgreSQL
- Oracle
- MongoDB (noSQL)



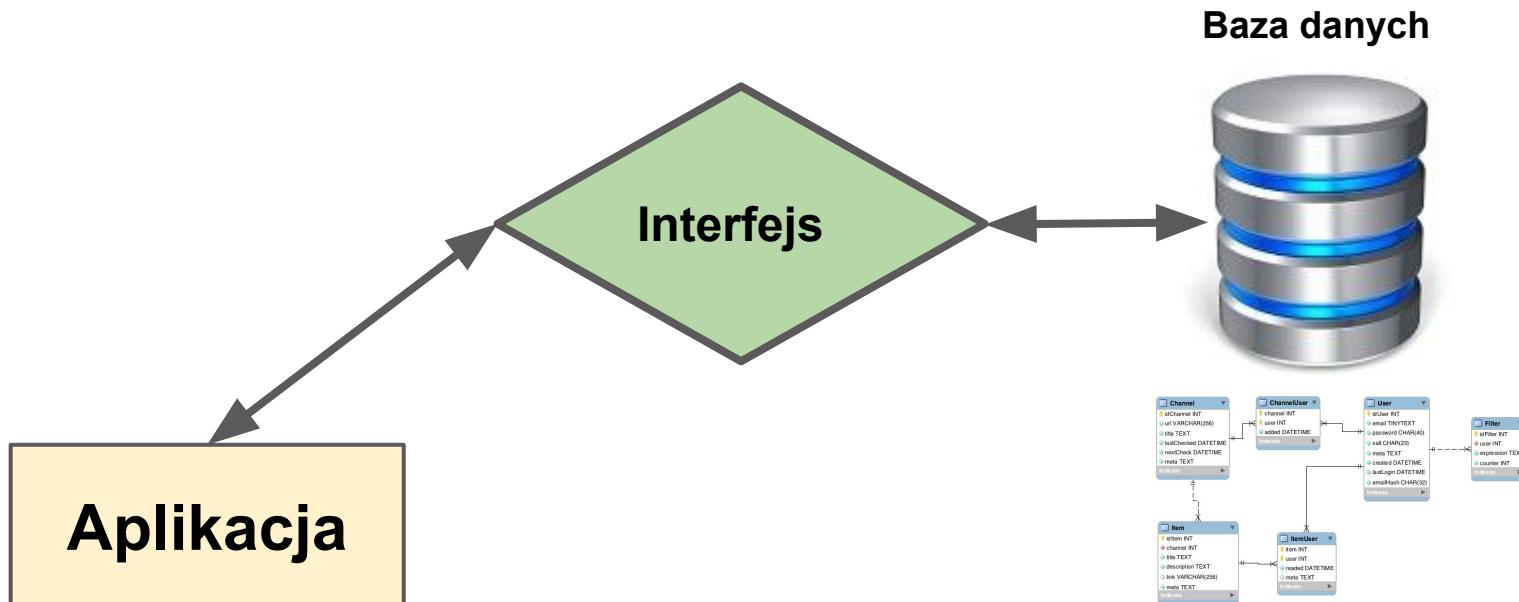
Bazy danych

Baza danych “mySQL” należy do rozwiązań OpenSource z których można korzystać bezpłatne. Typowe bazy danych posiadają wiele encji które są ze sobą połączone za pomocą relacji (np. jeden-do-jednego, jeden-do-wielu, etc.).



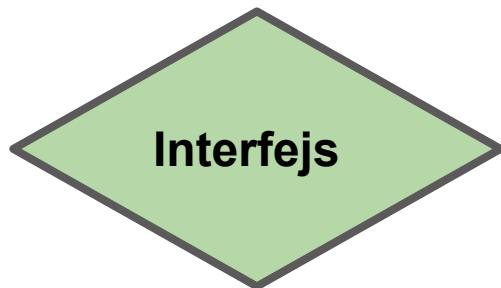
Bazy danych

W środowisku Node.js aby korzystać z baz danych konieczne jest użycie interfejsu umożliwiającego połączenie z serwerem baz danych i obsługujących komunikację pomiędzy aplikacją a serwerem baz danych.



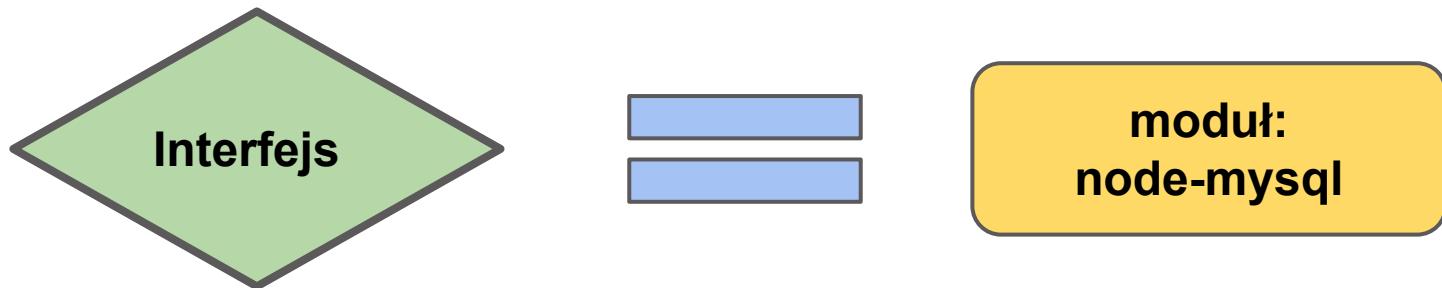
Bazy danych

W środowisku Node.js aby korzystać z baz danych konieczne jest użycie interfejsu umożliwiającego połączenie z serwerem baz danych i obsługujących komunikację pomiędzy aplikacją a serwerem baz danych.



Bazy danych

W środowisku Node.js aby korzystać z baz danych konieczne jest użycie interfejsu umożliwiającego połączenie z serwerem baz danych i obsługujących komunikację pomiędzy aplikacją a serwerem baz danych.

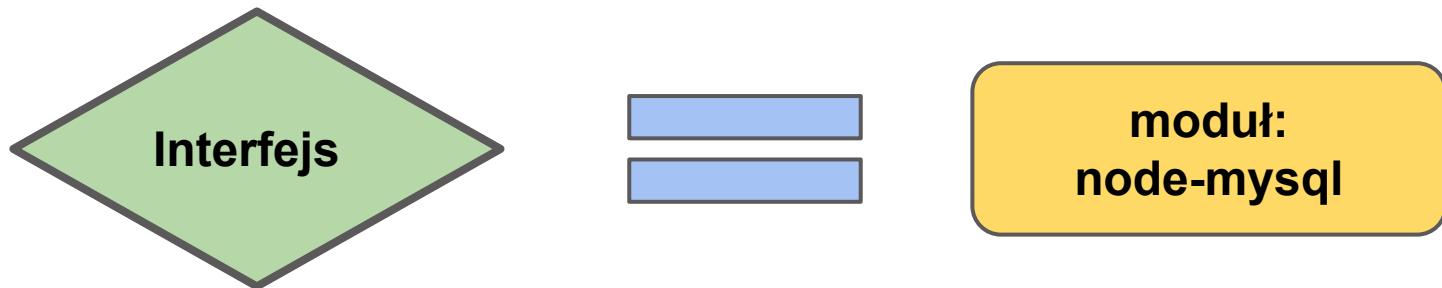


Instalacja pakietu za pomocą repozytorium NPM:

```
~/node> npm install mysql
```

Bazy danych

W środowisku Node.js aby korzystać z baz danych konieczne jest użycie interfejsu umożliwiającego połączenie z serwerem baz danych i obsługujących komunikację pomiędzy aplikacją a serwerem baz danych.



Instalacja pakietu za pomocą repozytorium NPM:

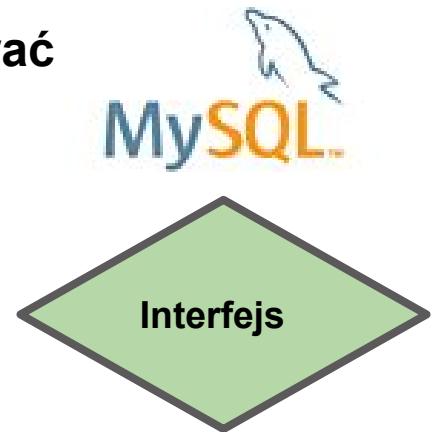
```
~/node> npm install mysql
```

Pakiet dodajemy za pomocą metody require:

```
var mysql = require('mysql');
```

Bazy danych

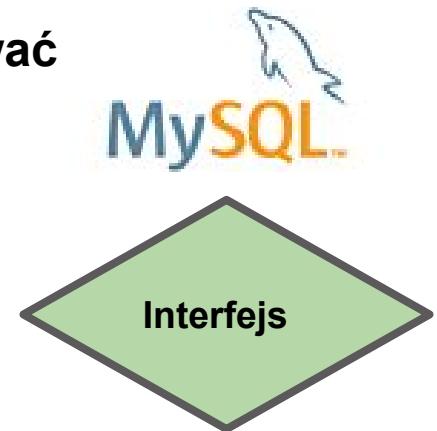
Aby połączyć się z bazą danych należy najpierw zdefiniować parametry połączennia: nazwa serwera, port na którym baza nasłuchuje żądań, nazwa użytkownika, hasło, oraz nazwa bazy danych z którą chcemy się połączyć. W tym celu interfejs “node-mysql” udostępnia metodę `.createConnection()`, która nawiązuje połączenie:



Bazy danych

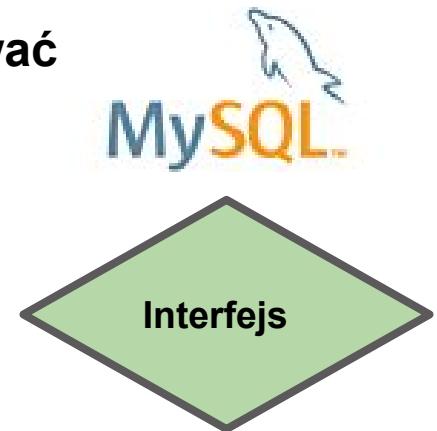
Aby połączyć się z bazą danych należy najpierw zdefiniować parametry połączennia: nazwa serwera, port na którym baza nasłuchuje żądań, nazwa użytkownika, hasło, oraz nazwa bazy danych z którą chcemy się połączyć. W tym celu interfejs “node-mysql” udostępnia metodę `.createConnection()`, która nawiązuje połączenie:

```
var connection = mysql.createConnection({
  host      : 'localhost',
  user      : 'root',
  password  : 'test',
  database  : 'meeting2015'
});
connection.connect();
```



Bazy danych

Aby połączyć się z bazą danych należy najpierw zdefiniować parametry połączennia: nazwa serwera, port na którym baza nasłuchuje żądań, nazwa użytkownika, hasło, oraz nazwa bazy danych z którą chcemy się połączyć. W tym celu interfejs “node-mysql” udostępnia metodę `.createConnection()`, która nawiązuje połączenie:



```
var connection = mysql.createConnection({  
  host      : 'localhost',  
  user      : 'root',  
  password  : 'test',  
  database  : 'meeting2015'  
});  
  
connection.connect();
```

Metoda przyjmuje jako argument obiekt JSON w którym zapisane są parametry połączenia do serwera.

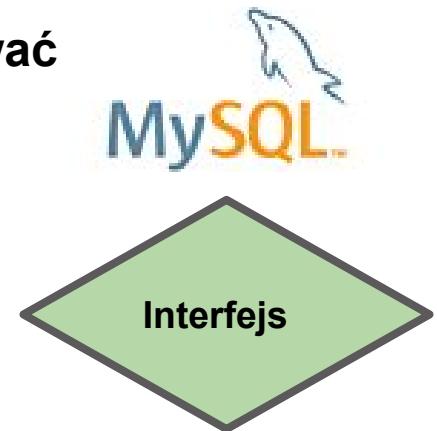
Bazy danych

Aby połączyć się z bazą danych należy najpierw zdefiniować parametry połączzenia: nazwa serwera, port na którym baza nasłuchuje żądań, nazwa użytkownika, hasło, oraz nazwa bazy danych z którą chcemy się połączyć. W tym celu interfejs “node-mysql” udostępnia metodę `.createConnection()`, która nawiązuje połączenie:

```
var connection = mysql.createConnection({  
    host      : 'localhost',  
    user      : 'root',  
    password  : 'test',  
    database  : 'meeting2015'  
});  
connection.connect();
```

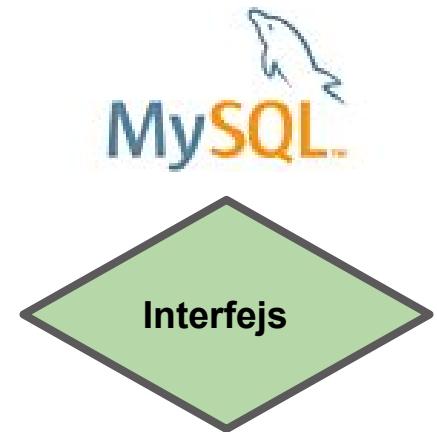
Inicjowanie połączenia

Metoda przyjmuje jako argument obiekt JSON w którym zapisane są parametry połączenia do serwera.



Bazy danych

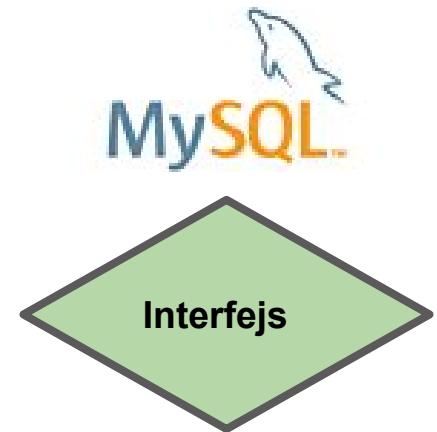
Od tego momentu mamy pełną możliwość wysyłania kwerend do serwera SQL w celu pobrania odpowiednich danych. Zapytanie formułujemy w postaci obiektu string:



Bazy danych

Od tego momentu mamy pełną możliwość wysyłania kwerend do serwera SQL w celu pobrania odpowiednich danych. Zapytanie formułujemy w postaci obiektu string:

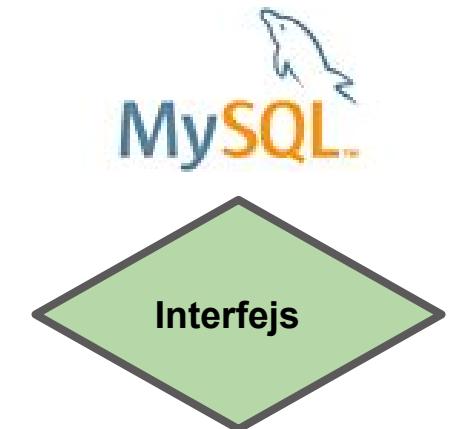
```
var zapytanie = 'select * from users';
```



Bazy danych

Od tego momentu mamy pełną możliwość wysyłania kwerend do serwera SQL w celu pobrania odpowiednich danych. Zapytanie formułujemy w postaci obiektu string:

```
var zapytanie = 'select * from users';
```



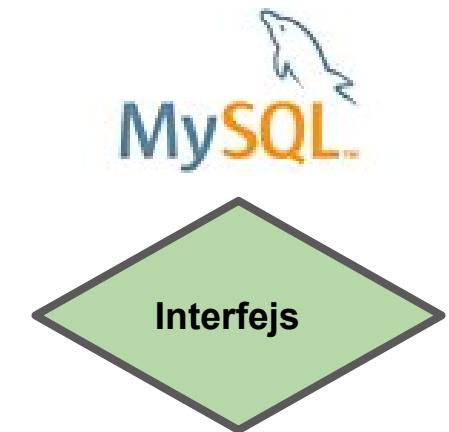
Następnie zapytanie musi zostać wysłane do serwera SQL, za pomocą metody `.query()`, która przyjmuje dwa argumenty: pierwszy zapytanie, drugi to wywołanie zwrotne w którym będziemy odbierać dane:

```
connection.query(zapytanie, function(err, rows) {
  res.render('participants', {dane: rows});
});
```

Bazy danych

Od tego momentu mamy pełną możliwość wysyłania kwerend do serwera SQL w celu pobrania odpowiednich danych. Zapytanie formułujemy w postaci obiektu string:

```
var zapytanie = 'select * from users';
```



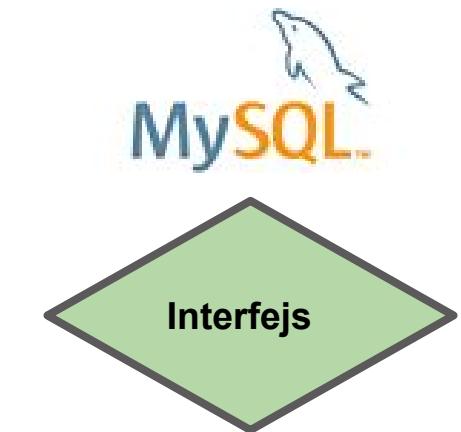
Następnie zapytanie musi zostać wysłane do serwera SQL, za pomocą metody `.query()`, która przyjmuje dwa argumenty: pierwszy zapytanie, drugi to wywołanie zwrotne w którym będziemy odbierać dane:

```
connection.query(zapytanie, function(err, rows) {
  res.render('participants', {dane: rows});
});
```

Bazy danych

Od tego momentu mamy pełną możliwość wysyłania kwerend do serwera SQL w celu pobrania odpowiednich danych. Zapytanie formułujemy w postaci obiektu string:

```
var zapytanie = 'select * from users';
```



Następnie zapytanie musi zostać wysłane do serwera SQL, za pomocą metody `.query()`, która przyjmuje dwa argumenty: pierwszy zapytanie, drugi to wywołanie zwrotne w którym będziemy odbierać dane:

```
connection.query(zapytanie, function(err, rows) {
  res.render('participants', {dane: rows});
});
```

Serwer SQL zwraca dane w postaci tablicy wyników który jest argumentem w funkcji zwrotnej: “rows” (wiersze).

Bazy danych

Tablica “rows” zawiera tyle wierszy ile zostało zwróconych w wyniku zapytania SQL do bazy. Tablica ta może zostać w całości przekazana do widoku:

```
connection.query(zapytanie, function(err, rows) {  
    res.render('participants', {dane: rows});  
});
```



Bazy danych

Tablica “rows” zawiera tyle wierszy ile zostało zwróconych w wyniku zapytania SQL do bazy. Tablica ta może zostać w całości przekazana do widoku:

```
:connection.query(zapytanie, function(err, rows) {  
    res.render('participants', {dane: rows});  
}) ;
```



W widoku możemy iterować po przekazanej tablicy i wyświetlać odpowiednia pola korzystając z notacji obiektowej:

```
#shortDescription  
ol  
each item in dane  
li  
b #{item.name} #{item.sname}  
| , #{item.affiliation}, #{item.country}
```

Bazy danych

Baza: na serwerze

lokalnym (127.0.0.1)

Nazwa bazy: Personel

Tabela w bazie mySQL: dane

id	imie	nazwisko
1	Maciej	Adamczyk
2	Jan	Kowalski
3	Anna	Nowak

```
var connection = mysql.createConnection({
  host      : 'localhost',
  user      : 'root',
  password  : 'test',
  database  : 'Personel'
});
connection.connect();

var zapytanie = 'select * from dane';

connection.query(zapytanie, function(err, rows) {
  console.log(rows[0].id + rows[0].imie + rows[0].nazwisko);
  console.log(rows[1].id + rows[1].imie + rows[1].nazwisko);
  console.log(rows[2].id + rows[2].imie + rows[2].nazwisko);
});
```



KONIEC WYKŁADU 11



UNIWERSYTET
JAGIELŁOŃSKI
W KRAKOWIE

Zaawansowane Techniki WWW (HTML, CSS i JavaScript)

Dr inż. Marcin Zieliński

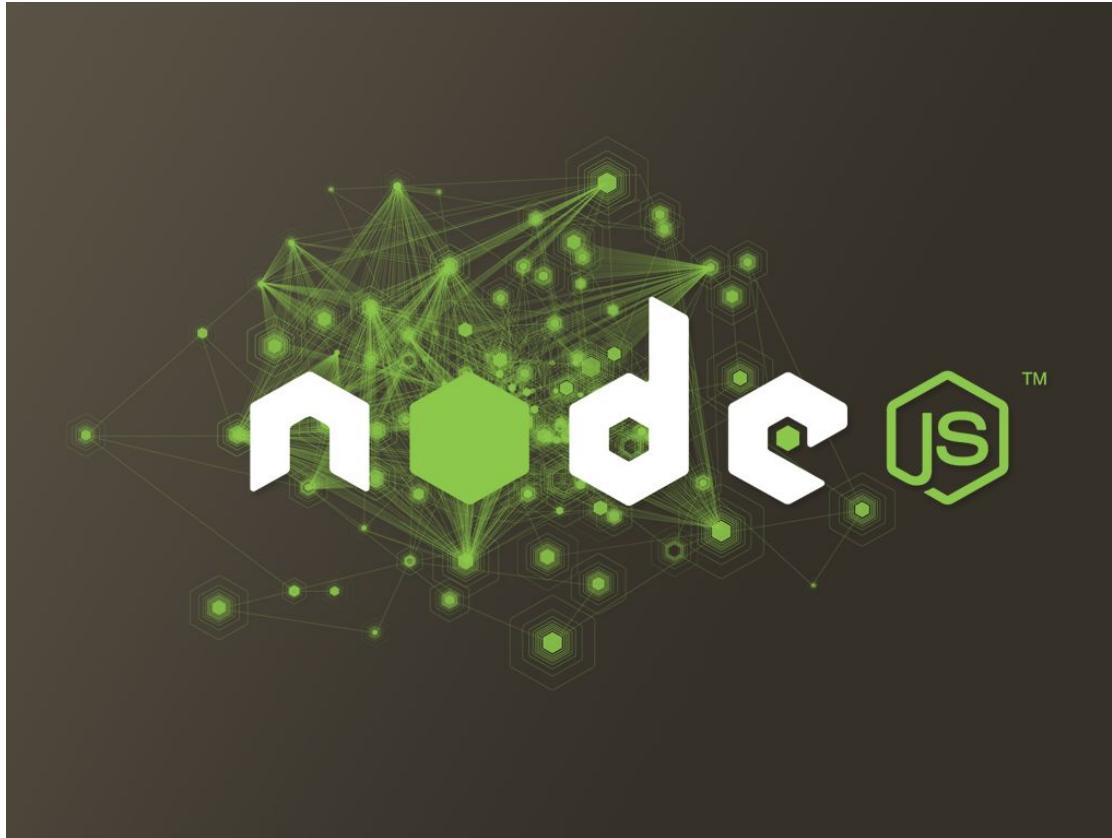
Środa 15:30 - 17:00 sala: A-1-04

WYKŁAD 12

Wykład dla kierunku: **Informatyka Stosowana II rok**

Rok akademicki: **2015/2016 - semestr zimowy**

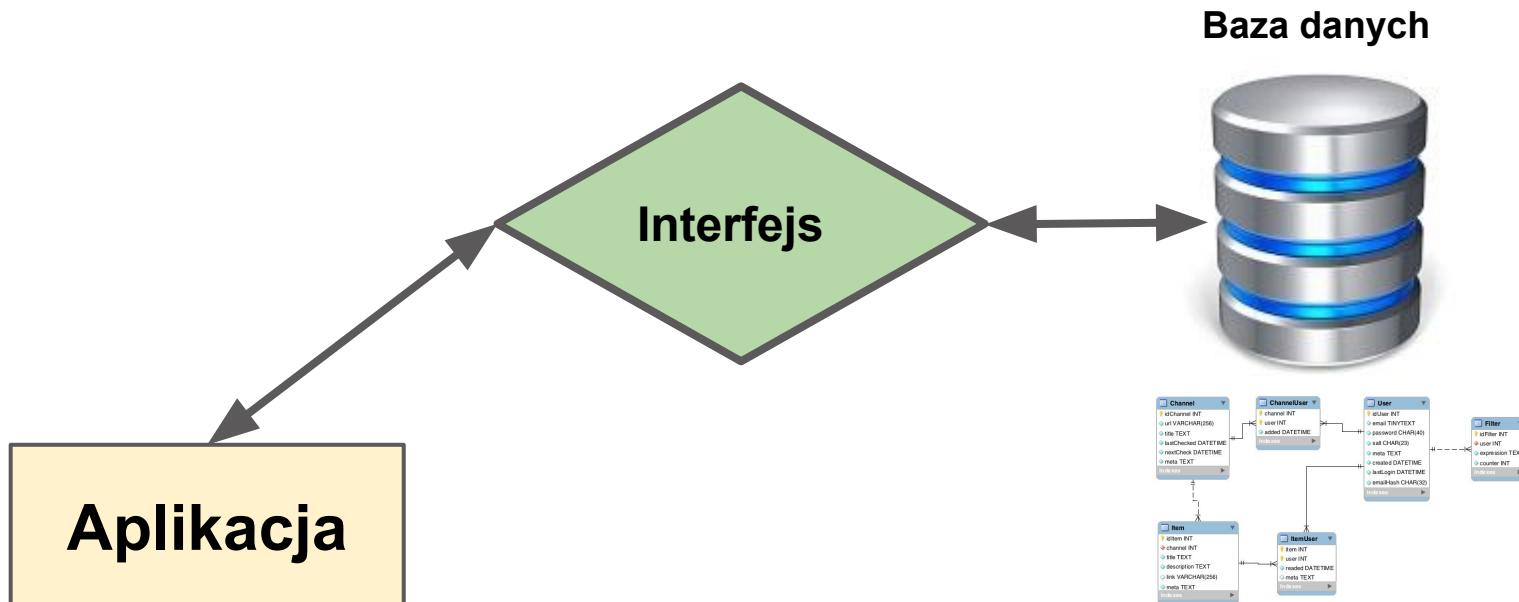
Node.js



<http://nodejs.org/>

Bazy danych

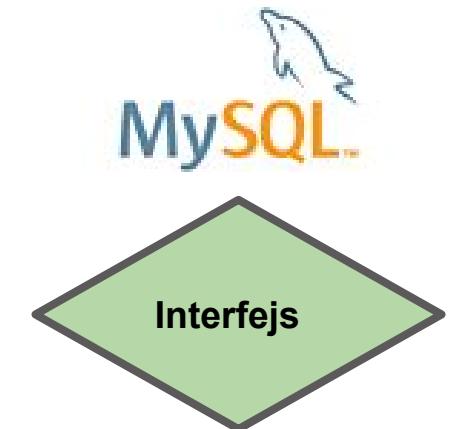
W środowisku Node.js aby korzystać z baz danych konieczne jest użycie interfejsu umożliwiającego połączenie z serwerem baz danych i obsługujących komunikację pomiędzy aplikacją a serwerem baz danych.



Bazy danych

Od tego momentu mamy pełną możliwość wysyłania kwerend do serwera SQL w celu pobrania odpowiednich danych. Zapytanie formułujemy w postaci obiektu string:

```
var zapytanie = 'select * from users';
```



Następnie zapytanie musi zostać wysłane do serwera SQL, za pomocą metody `.query()`, która przyjmuje dwa argumenty: pierwszy zapytanie, drugi to wywołanie zwrotne w którym będziemy odbierać dane:

```
connection.query(zapytanie, function(err, rows) {
  res.render('participants', {dane: rows});
});
```

Bazy danych

Tablica “rows” zawiera tyle wierszy ile zostało zwróconych w wyniku zapytania SQL do bazy. Tablica ta może zostać w całości przekazana do widoku:

```
:connection.query(zapytanie, function(err, rows) {  
    res.render('participants', {dane: rows});  
}) ;
```



W widoku możemy iterować po przekazanej tablicy i wyświetlać odpowiednia pola korzystając z notacji obiektowej:

```
#shortDescription  
ol  
each item in dane  
li  
b #{item.name} #{item.sname}  
| , #{item.affiliation}, #{item.country}
```

Bazy danych

Baza: na serwerze

lokalnym (127.0.0.1)

Nazwa bazy: Personel

Tabela w bazie mySQL: dane

id	imie	nazwisko
1	Maciej	Adamczyk
2	Jan	Kowalski
3	Anna	Nowak

```
var connection = mysql.createConnection({
  host      : 'localhost',
  user      : 'root',
  password  : 'test',
  database  : 'Personel'
});
connection.connect();

var zapytanie = 'select * from dane';

connection.query(zapytanie, function(err, rows) {
  console.log(rows[0].id + rows[0].imie + rows[0].nazwisko);
  console.log(rows[1].id + rows[1].imie + rows[1].nazwisko);
  console.log(rows[2].id + rows[2].imie + rows[2].nazwisko);
});
```



Bazy danych i ORM

Większość nowoczesnych aplikacji internetowych jest pisanych w językach obiektowych (Java, JavaScript, PHP, etc), gdzie logika biznesowa i model danych jest reprezentowany przez obiekty.



Bazy danych i ORM

Większość nowoczesnych aplikacji internetowych jest pisanych w językach obiektowych (Java, JavaScript, PHP, etc), gdzie logika biznesowa i model danych jest reprezentowany przez obiekty.

Gdzie przechowywane
są obiekty ?

Bazy danych i ORM

Większość nowoczesnych aplikacji internetowych jest pisanych w językach obiektowych (Java, JavaScript, PHP, etc), gdzie logika biznesowa i model danych jest reprezentowany przez obiekty.

Gdzie przechowywane
są obiekty ?



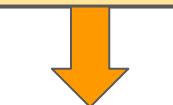
Baza danych SQL (lub noSQL)

Bazy danych i ORM

Większość nowoczesnych aplikacji internetowych jest pisanych w językach obiektowych (Java, JavaScript, PHP, etc), gdzie logika biznesowa i model danych jest reprezentowany przez obiekty.

Gdzie przechowywane są obiekty ?

Jak w bazie danych reprezentowane są dane ?



Baza danych SQL (lub noSQL)

Bazy danych i ORM

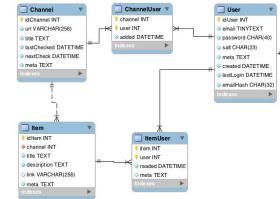
Większość nowoczesnych aplikacji internetowych jest pisanych w językach obiektowych (Java, JavaScript, PHP, etc), gdzie logika biznesowa i model danych jest reprezentowany przez obiekty.

Gdzie przechowywane są obiekty ?



Baza danych SQL (lub noSQL)

Jak w bazie danych reprezentowane są dane ?



Dane przechowywane są w postaci relacji (lub dokumentów) połączonych zależnościami.

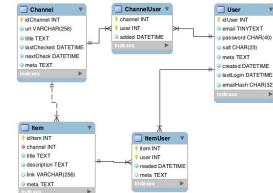
Bazy danych i ORM

Większość nowoczesnych aplikacji internetowych jest pisanych w językach obiektowych (Java, JavaScript, PHP, etc), gdzie logika biznesowa i model danych jest reprezentowany przez obiekty.

Gdzie przechowywane są obiekty ?



Jak w bazie danych reprezentowane są dane ?



Baza danych SQL (lub noSQL)

Dane przechowywane są w postaci relacji (lub dokumentów) połączonych zależnościami.

Zatem w bazie danych nie przechowujemy obiektów które dokładnie odpowiadają strukturze i modelowi danych aplikacji!!!

Bazy danych i ORM

Pojawił się pomysł przechowywania danych w tzw. “obiektowych bazach danych”, czyli strukturach o takiej samej formie jak w aplikacji bez konieczności wykonywania dodatkowych operacji zmieniających format obiektowy na relacyjny i odwrotnie.



Bazy danych i ORM

Pojawił się pomysł przechowywania danych w tzw. “obiektowych bazach danych”, czyli strukturach o takiej samej formie jak w aplikacji bez konieczności wykonywania dodatkowych operacji zmieniających format obiektowy na relacyjny i odwrotnie.

Pomysł ten jednak nie został wprowadzony w życie !

Bazy danych i ORM

Pojawił się pomysł przechowywania danych w tzw. “obiektowych bazach danych”, czyli strukturach o takiej samej formie jak w aplikacji bez konieczności wykonywania dodatkowych operacji zmieniających format obiektowy na relacyjny i odwrotnie.

Pomysł ten jednak nie został wprowadzony w życie !



Powstało natomiast rozwiązanie alternatywne nazywane:
Mapowaniem Obiektowo-Relacyjnym (ORM) [ang. *Object-Relational Mapping*]

Bazy danych i ORM

Pojawił się pomysł przechowywania danych w tzw. “obiektowych bazach danych”, czyli strukturach o takiej samej formie jak w aplikacji bez konieczności wykonywania dodatkowych operacji zmieniających format obiektowy na relacyjny i odwrotnie.

Pomysł ten jednak nie został wprowadzony w życie !



Powstało natomiast rozwiązanie alternatywne nazywane:
Mapowaniem Obiektowo-Relacyjnym (ORM) [ang. *Object-Relational Mapping*]



Do czego służy ORM ?

Bazy danych i ORM

Pojawił się pomysł przechowywania danych w tzw. “obiektowych bazach danych”, czyli strukturach o takiej samej formie jak w aplikacji bez konieczności wykonywania dodatkowych operacji zmieniających format obiektowy na relacyjny i odwrotnie.

Pomysł ten jednak nie został wprowadzony w życie !



Powstało natomiast rozwiązanie alternatywne nazywane:
Mapowaniem Obiektowo-Relacyjnym (ORM) [ang. *Object-Relational Mapping*]



Do czego służy ORM ?



Implementuje warstwę pośredniczącą w zagadnieniu komunikacji baza danych - aplikacja, która umożliwia zamianę danych z postaci tabularycznej (relacji) na reprezentację obiektową.

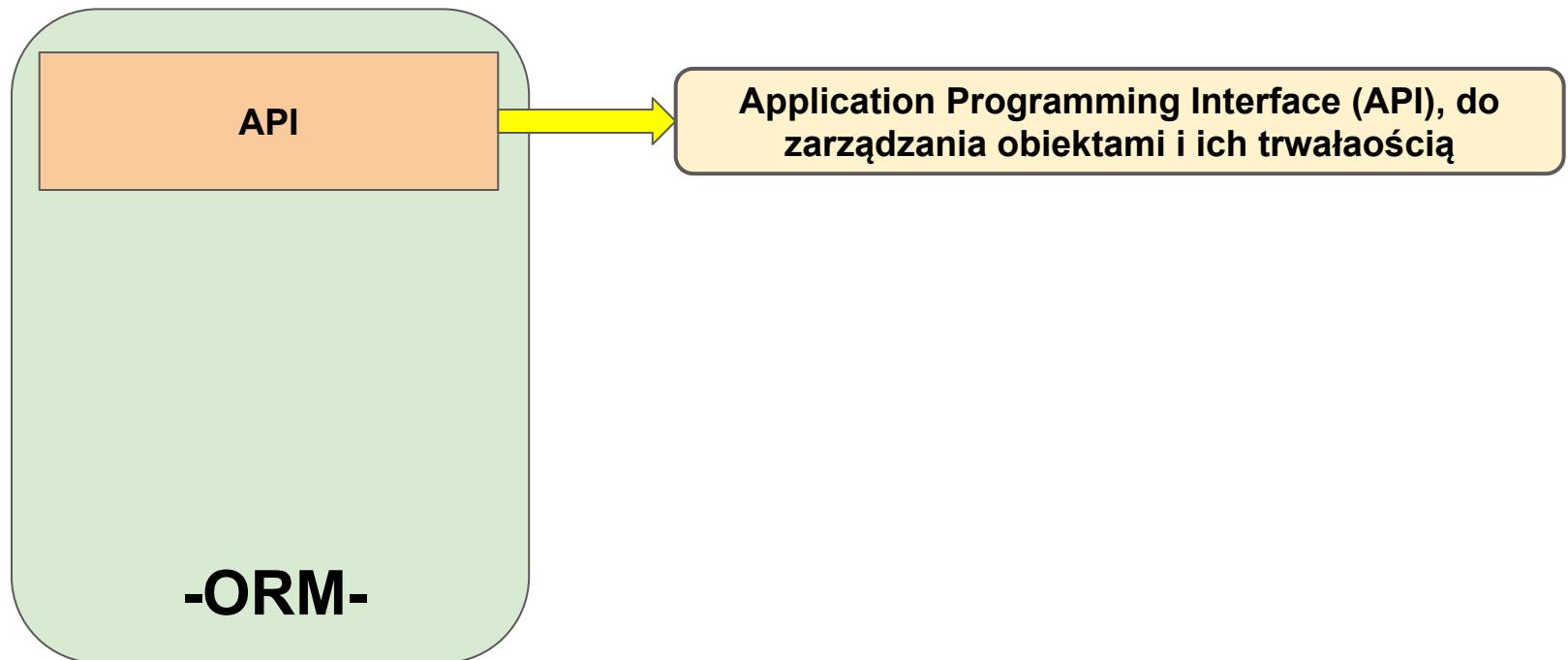
Bazy danych i ORM

Technologia ORM charakteryzuje się trzema elementami:



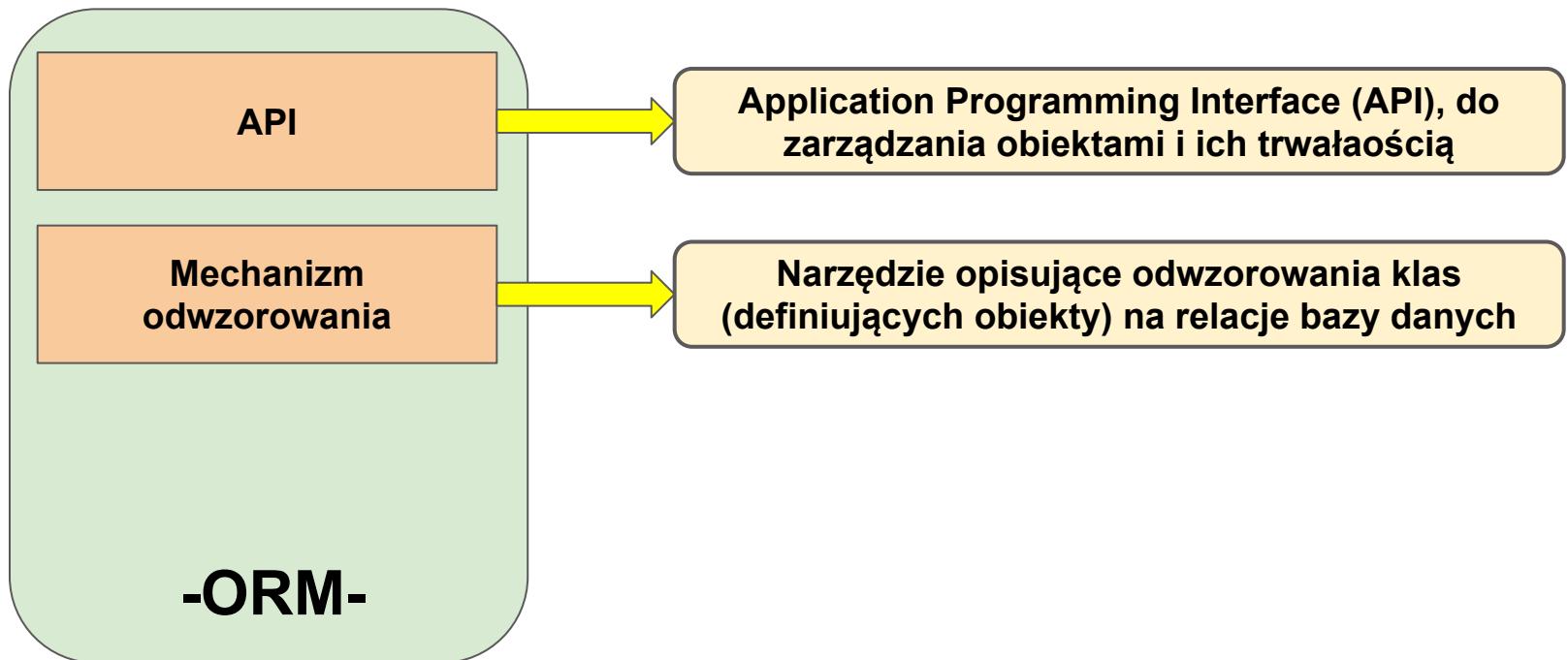
Bazy danych i ORM

Technologia ORM charakteryzuje się trzema elementami:



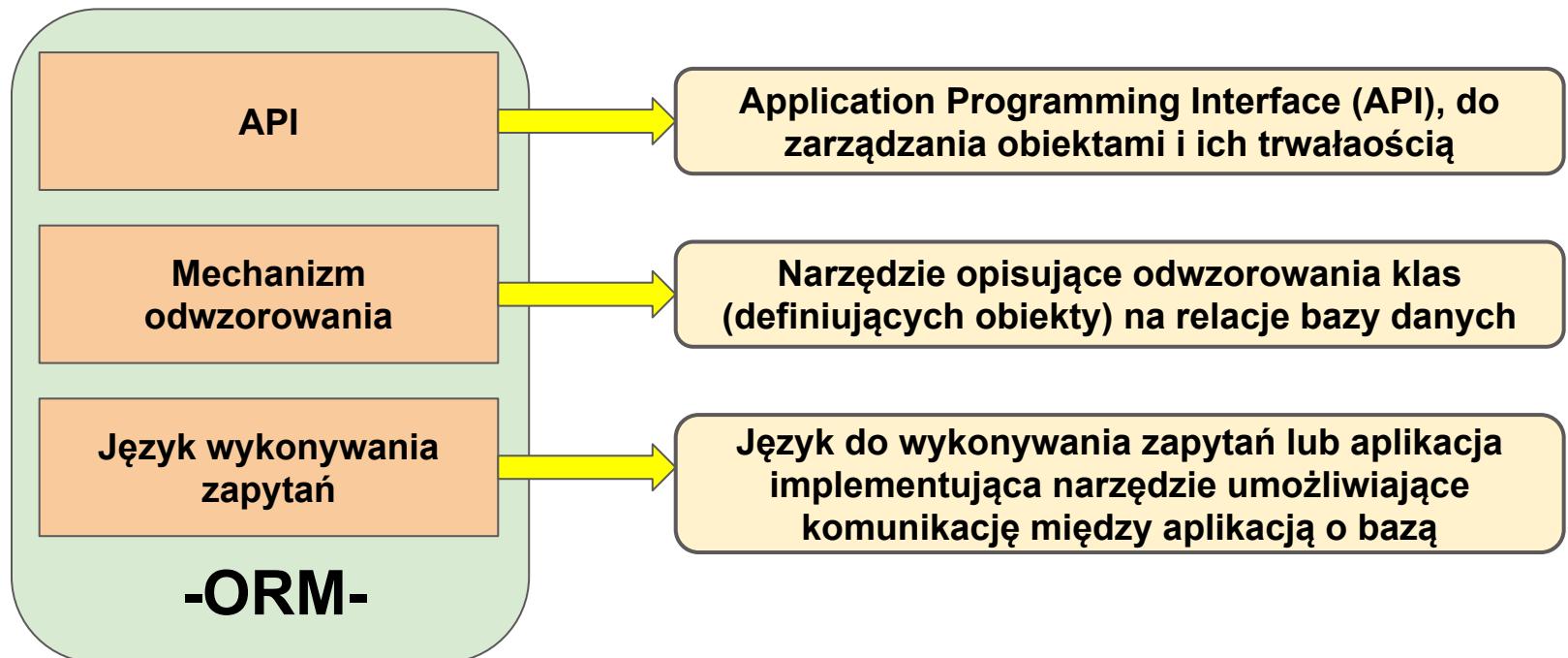
Bazy danych i ORM

Technologia ORM charakteryzuje się trzema elementami:



Bazy danych i ORM

Technologia ORM charakteryzuje się trzema elementami:



Bazy danych i ORM

Przykład - podejście klasyczne:

```
function zmienHaslo(_login, _haslo){  
    var zapytanie = 'UPDATE users SET password' + _haslo + ' WHERE  
    login=' + _login;  
    connection.query(zapytanie, function(err,rows){  
        ....  
    });  
}
```

Bazy danych i ORM

Przykład - podejście klasyczne:

```
function zmienHaslo(_login, _haslo){  
    var zapytanie = 'UPDATE users SET password'+_haslo+'WEHERE  
    login=_login;  
    connection.query(zapytanie, function(err,rows){  
        .....  
    });  
}
```

Przykład - podejście obiektowe:

```
Uzytkownik u;  
function zmienHaslo(_login, _haslo){  
    u.SetHaslo(_login,_haslo);  
    .....  
}
```



Bazy danych i ORM

Dla środowiska uruchomieniowego Node.js istnieje wiele aplikacji które wspierają model z zaimplementowaną obsługą baz danych w podejściu ORM. Jednym z najpopularniejszych a zarazem łatwych w użyciu jest pakiet:

Bazy danych i ORM

Dla środowiska uruchomieniowego Node.js istnieje wiele aplikacji które wspierają model z zaimplementowaną obsługą baz danych w podejściu ORM. Jednym z najpopularniejszych a zarazem łatwych w użyciu jest pakiet:



Sequelize

www.sequelizejs.com

Bazy danych i ORM

Dla środowiska uruchomieniowego Node.js istnieje wiele aplikacji które wspierają model z zaimplementowaną obsługą baz danych w podejściu ORM. Jednym z najpopularniejszych a zarazem łatwych w użyciu jest pakiet:



www.sequelizejs.com

Sequelize oferuje wsparcie dla najpopularniejszych relacyjnych baz danych:





Bazy danych i ORM



Sequelize

www.sequelizejs.com

Bazy danych i ORM



Sequelize

www.sequelizejs.com

Instalacja w środowisku Node.js przez repozytorium NPM:

```
~/node> npm install --save sequelize
```

Bazy danych i ORM



Sequelize

www.sequelizejs.com

Instalacja w środowisku Node.js przez repozytorium NPM:

```
~/node> npm install --save sequelize
```

oraz jeden z dodatkowych pakietów bazodanowych:

```
$ npm install --save pg pg-hstore //PostgreSQL
$ npm install --save mysql // MySQL
$ npm install --save sqlite3 // SQLite
$ npm install --save tedious // MSSQL
```

Bazy danych i ORM



Sequelize

www.sequelizejs.com

Instalacja w środowisku Node.js przez repozytorium NPM:

```
~/node> npm install --save sequelize
```

oraz jeden z dodatkowych pakietów bazodanowych:

```
$ npm install --save pg pg-hstore //PostgreSQL
$ npm install --save mysql // MySQL
$ npm install --save sqlite3 // SQLite
$ npm install --save tedious // MSSQL
```

w aplikacji:

```
var Sequelize = require('sequelize');
```

Bazy danych i ORM



Sequelize

Inicjalizacja połączenia z bazą danych dla Sequelize:

```
[ var Sequelize = require('sequelize');
var sequelize = new Sequelize('database',      //wymagana
                            'uzytkownik', //wymagany
                            'haslo',
                            { 'host', 'port' }
                          );
```

Bazy danych i ORM



Sequelize

Inicjalizacja połączenia z bazą danych dla Sequelize:

```
[ var Sequelize = require('sequelize');
var sequelize = new Sequelize('database',      //wymagana
                            'uzytkownik', //wymagany
                            'haslo',
                            { 'host', 'port' }
                          );
```

dość często (w przypadku lokalnej bazy danych) wymaga tylko:

```
[ var Sequelize = require('sequelize');
var sequelize = new Sequelize('database',      //wymagana
                            'uzytkownik', //wymagany
                          );
```



Bazy danych i ORM



Sequelize

Co dalej ?



Bazy danych i ORM



Sequelize

Co dalej ?



Musimy zdefiniować MODEL, czyli reprezentację bazy danych i ich własności

Bazy danych i ORM



Sequelize

Co dalej ?



Musimy zdefiniować MODEL, czyli reprezentację bazy danych i ich własności

Spróbujemy rozwiązać następujący problem i przedstawić go w postaci bazy danych:

Chcemy utworzyć prosty system zarządzania projektami i zadaniami w tych projektach,
tak aby do projektów można było przypisać zadania do zrealizowania.

Bazy danych i ORM



Sequelize

Co dalej ?



Musimy zdefiniować MODEL, czyli reprezentację bazy danych i ich własności

Spróbujemy rozwiązać następujący problem i przedstawić go w postaci bazy danych:

Chcemy utworzyć prosty system zarządzania projektami i zadaniami w tych projektach, tak aby do projektów można było przypisać zadania do zrealizowania.

Projekt

- tytuł (STRING)
- opis (TEXT)
- utworzony (DATA)

Zadanie

- tytuł (STRING)
- status (BOOLEAN)



Bazy danych i ORM



Sequelize

Projekt

- tytul (STRING)
- opis (TEXT)
- utworzony (DATA)

Zadanie

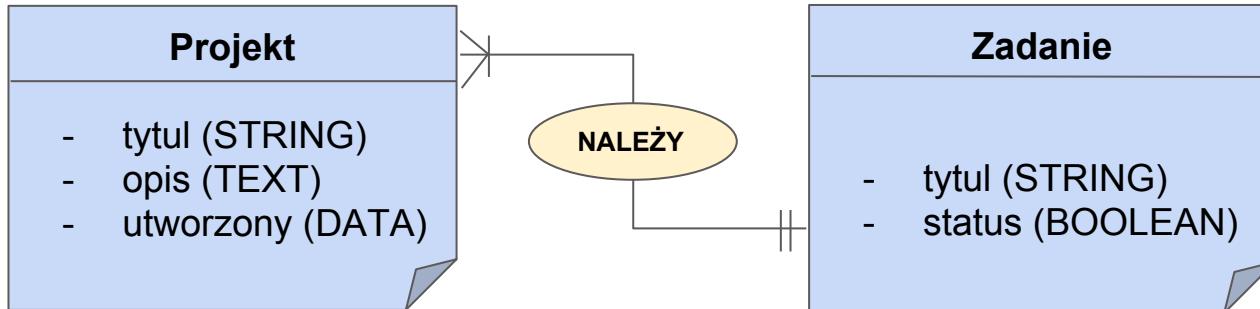
- tytul (STRING)
- status (BOOLEAN)

Jakie zależności łączą te relacje ?

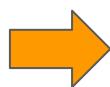
Bazy danych i ORM



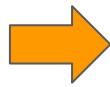
Sequelize



Jakie zależności łączą te relacje ?



Każde zadanie należy do jakiegoś projektu

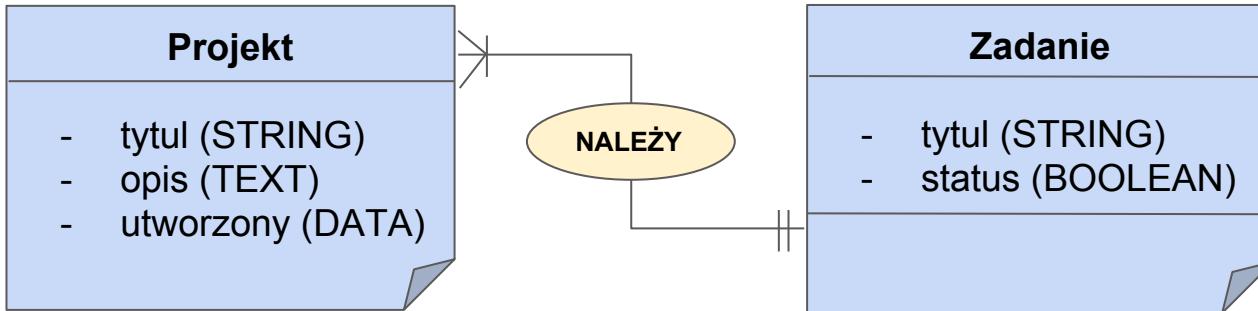


Dla jednego projektu może być zdefiniowanych wiele zadań

Bazy danych i ORM



Sequelize

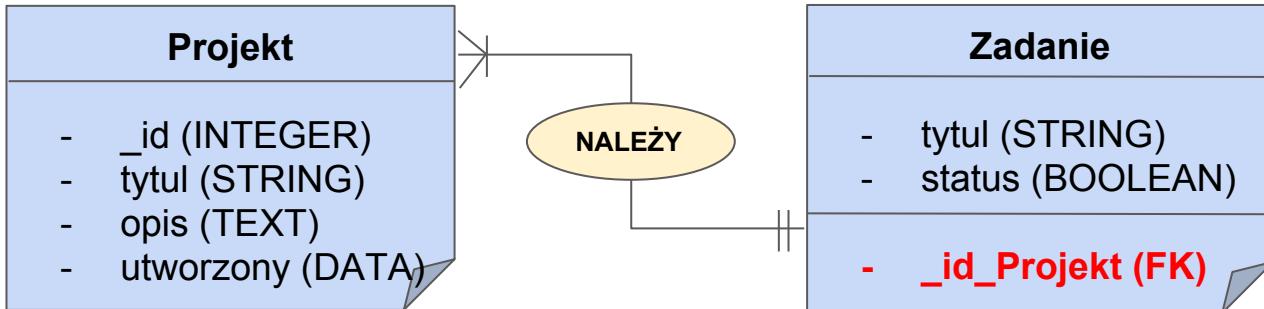


Fizycznie relacja ta oznacza że:

Bazy danych i ORM



Sequelize



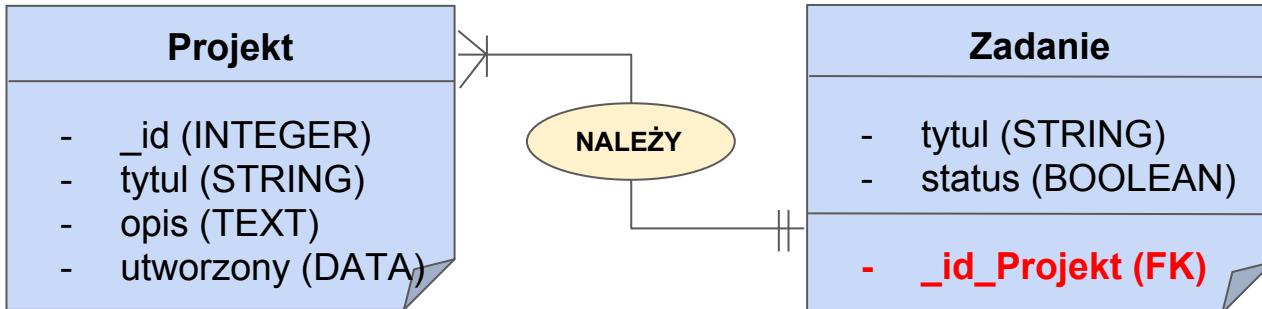
Fizycznie relacja ta oznacza że:

- ➡ każdy rekord “Zadania” musi posiadać pole jednoznacznie wskazujące projekt do którego należy
- ➡ projekt posiada wiele zadań i musi zapewniać dostęp do wszystkich zadań projektu.

Bazy danych i ORM



Sequelize



Fizycznie relacja ta oznacza że:

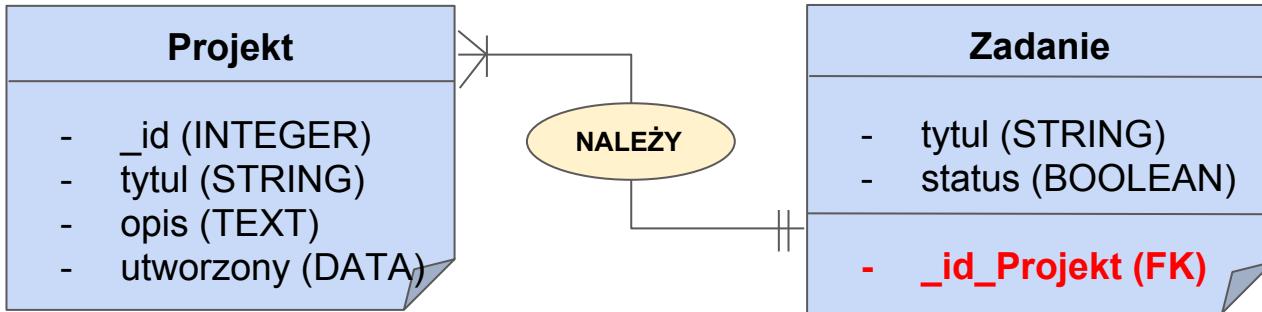
- ➡ każdy rekord “Zadania” musi posiadać pole jednoznacznie wskazujące projekt do którego należy
- ➡ projekt posiada wiele zadań i musi zapewniać dostęp do wszystkich zadań projektu.

REALIZACJA ZA POMOCĄ NARZĘDZIA SEQUELIZE:

Bazy danych i ORM



Sequelize

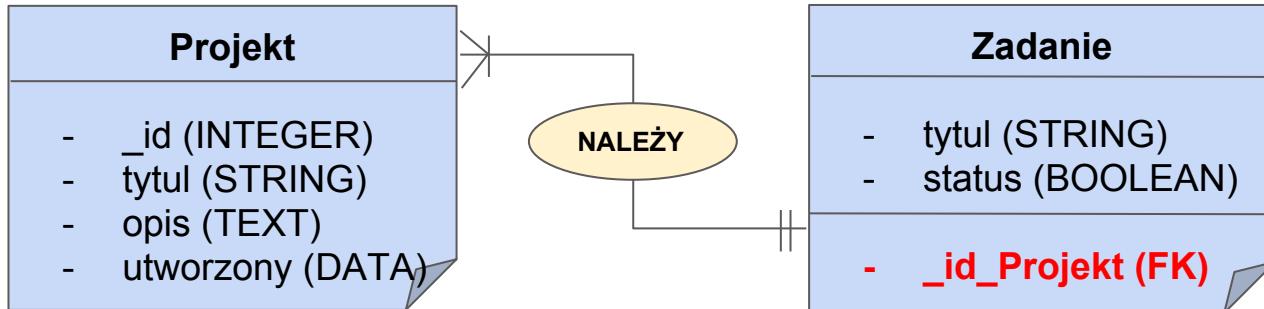


Na początek musimy zdefiniować MODEL który będzie reprezentował bazę danych:

Bazy danych i ORM



Sequelize



Na początek musimy zdefiniować MODEL który będzie reprezentował bazę danych:

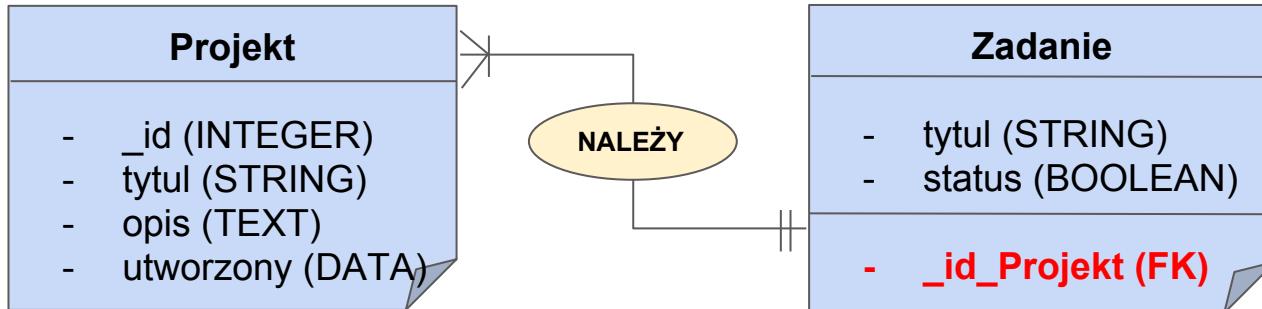
```
var Projekt = sequelize.define('Projekt', {
  tytul: Sequelize.STRING,    // VARCHAR(255)
  opis: Sequelize.TEXT,
  utworzony: Sequelize.DATE  //DATETIME
});
```

```
sequelize.define( nazwa_modelu, {obiekt_wlasnosci} )
```

Bazy danych i ORM



Sequelize



Na początek musimy zdefiniować MODEL który będzie reprezentował bazę danych:

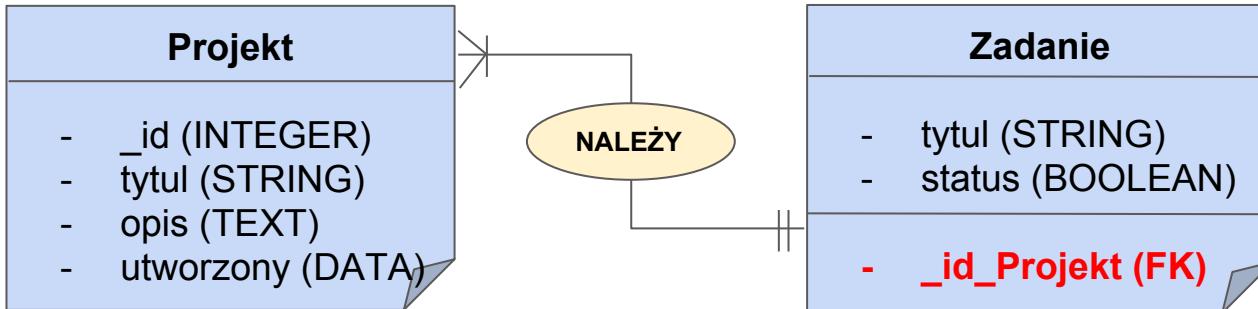
```
var Zadanie = sequelize.define('Zadanie', {
  tytul: Sequelize.STRING, // VARCHAR(255)
  status: Sequelize.BOOLEAN //TINYINT(1)
});
```

```
sequelize.define( nazwa_modelu, {obiekt_wlasnosci} )
```

Bazy danych i ORM



Sequelize



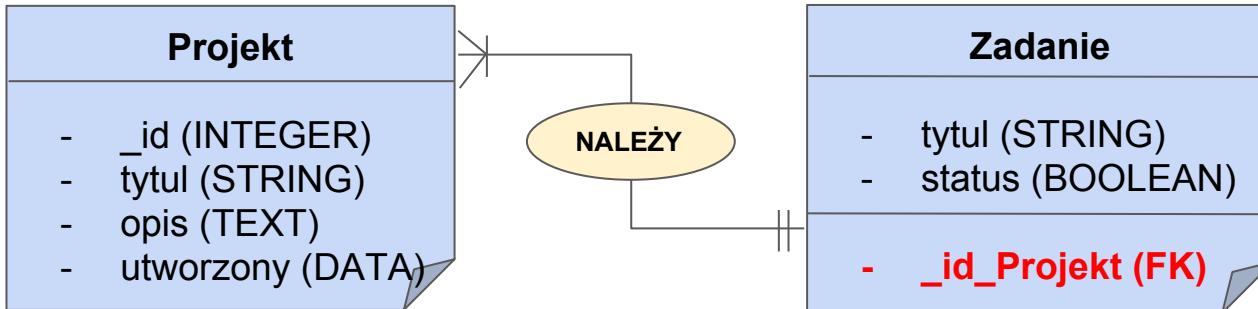
Zdefiniowanie zależności pomiędzy relacjami:

```
:
  Zadanie.belongsTo( Projekt );
  ProjekthasMany( Zadanie );
```

Bazy danych i ORM



Sequelize



Zdefiniowanie zależności pomiędzy relacjami:

```
• Zadanie.belongsTo( Projekt );
• ProjekthasMany( Zadanie );
```

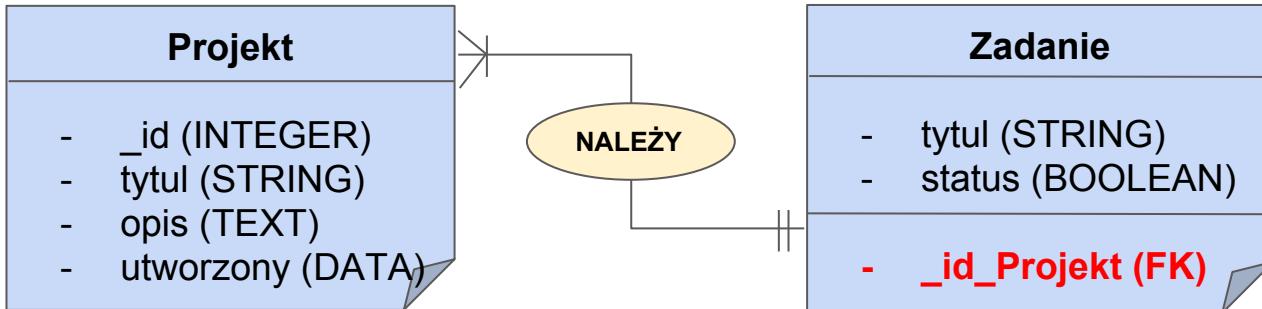
Sequelize automatycznie definiuje powiązanie pomiędzy relacjami ustawiając odpowiednie kolumny, klucze główne i indeksy:

- `.belongsTo()` powoduje, że każdy obiekt “Zadanie” posiada pole jednoznacznie wskazujące projekt do którego należy “`_id_Projekt`”.
- `.hasMany()` powoduje, że jeśli wywołamy metodę wyszukującą spowoduje zwrócenie informacji o wszystkich zadaniach danego projektu.

Bazy danych i ORM



Sequelize



Ostatnim krokiem utworzenia ORM dla bazy danych jest synchronizacja, która kontroluje czy model obiektowy zdefiniowany w aplikacji odpowiada strukturze bazy danych:

```
: sequelize.sync();
```

W trakcie budowy aplikacji często zmienia się struktura tabel, dalego synchronizacja zapewnia że za każdym razem uruchamiana aplikacja będzie dopowiadała strukturze bazy danych.

Jak teraz odczytać dane z bazy danych ?

Bazy danych i ORM



Sequelize

Jak teraz odczytać dane z bazy danych ?



```
router.get('/',function(req,res,next) {
  Projekty.findAll()
    .success(function(projects) {
      res.render('projekty',{projekty:projekt});
    })
    .error(next);
});
```

Bazy danych i ORM



Sequelize

Jak teraz odczytać dane z bazy danych ?



```
router.get('/',function(req,res,next) {  
  Projekty.findAll()  
    .success(function(projects) {  
      res.render('projekty',{projekty:projekt});  
    })  
    .error(next);  
});
```

.findAll() - zwraca wszystkie rekordy z relacji Projekty

Bazy danych i ORM



Sequelize

Jak teraz odczytać dane z bazy danych ?



```
router.get('/',function(req,res,next) {  
  Projekty.findAll()  
    .success(function(projects) {  
      res.render('projekty',{projekty:projekt});  
    })  
    .error(next);  
});
```

.findAll() - zwraca wszystkie rekordy z relacji Projekty

.sucess() - obsługuje zdarzenie w przypadku prawidłowej odpowiedzi z bazy danych

Bazy danych i ORM



Sequelize

Jak teraz odczytać dane z bazy danych ?



```
router.get('/',function(req,res,next) {  
  Projekty.findAll()  
    .success(function(projects) {  
      res.render('projekty',{projekty:projekt});  
    })  
    .error(next);  
});
```

.findAll() - zwraca wszystkie rekordy z relacji Projekty

.sucess() - obsługuje zdarzenie w przypadku prawidłowej odpowiedzi z bazy danych

.error() - obsługuje zdarzenie w przypadku błędu

Bazy danych i ORM



Sequelize

Jak odczytać wszystkie zadania dla danego projektu ?



```
router.get('/projekt/:id/tasks', function(req, res, next) {  
  Projekty.find(Number(req.params.id))  
    .success(function(projekt) {  
      projekt.getZadania()  
        .success(function(zadania) {  
          res.render('zadania', {projekt:projekt,  
                                zadania: zadania});  
        })  
    })  
    .error(next);  
});
```

`.getZadania()` - metoda automatyczna powstała w trakcie tworzenia modelu



Bazy danych i ORM



Sequelize

Inne przydatne metody:

Bazy danych i ORM



Sequelize

Inne przydatne metody:



`.findById([id])` - metoda poszukuje rekordu o konkretnym identyfikatorze

Bazy danych i ORM



Sequelize

Inne przydatne metody:



`.findById([id])` - metoda poszukuje rekordu o konkretnym identyfikatorze

`.findOne({where: { [warunki] } })` - metoda poszukuje rekordu spełniającego konkretne warunki podane jako obiekt

Bazy danych i ORM



Sequelize

Inne przydatne metody:



`.findById([id])` - metoda poszukuje rekordu o konkretnym identyfikatorze

`.findOne({where: { [warunki] } })` - metoda poszukuje rekordu spełniającego konkretne warunki podane jako obiekt

```
Projekty.findOne({  
  where: {  
    tytul: 'projekt1'  
  }  
})  
.then()
```

Bazy danych i ORM



Sequelize

Inne przydatne metody:



`.findById([id])` - metoda poszukuje rekordu o konkretnym identyfikatorze

`.findOne({where: { [warunki] } })` - metoda poszukuje rekordu spełniającego konkretne warunki podane jako obiekt

```
Projekty.findOne({  
  where: {  
    tytul: 'projekt1'  
  }  
})  
.then()
```

```
SELECT *  
FROM `Projekty`  
WHERE (  
  `Projekty`.`tytul` = 'Projekt1')  
LIMIT 1;
```

Bazy danych i ORM



Sequelize

Inne przydatne metody:



`.findById([id])` - metoda poszukuje rekordu o konkretnym identyfikatorze

`.findOne({where: { [warunki]} })` - metoda poszukuje rekordu spełniającego konkretne warunki podane jako obiekt

```
Projekty.findOne({  
  where: {  
    tytul: 'Projekt1',  
    id: {  
      $or: [  
        [1,2,3],  
        { $gt: 10 }  
      ]  
    }  
  }  
})
```

```
SELECT *  
FROM `Projekty`  
WHERE (  
  `Projekty`.`tytul` = 'Projekt1'  
  AND (`Projekty`.`id` IN (1,2,3) OR  
  `Projekty`.`id` > 10)  
)  
LIMIT 1;
```



Bazy danych i ORM



Sequelize

Jak coś dodać do bazy danych INSERT

Bazy danych i ORM



Sequelize

Jak coś dodać do bazy danych INSERT



`.build() .save()` - metoda “tworząca rekord” i zapisującą go w bazie danych

Bazy danych i ORM



Sequelize

Jak coś dodać do bazy danych INSERT



`.build().save()` - metoda “tworząca rekord” i zapisującą go w bazie danych

```
Zadanie.build({ tytul: 'foo', status: 0 })
  .save()
  .then(function(next) {
    //
  }).catch(function(error) {
  })
```

Bazy danych i ORM



Sequelize

Jak coś dodać do bazy danych INSERT



`.build().save()` - metoda “tworząca rekord” i zapisującą go w bazie danych

```
Zadanie.build({ tytul: 'foo', status: 0 })
  .save()
  .then(function(next) {
    //
  }).catch(function(error) {
  })
```

```
INSERT INTO `Zadanie`
(`title`, `status`)
VALUES
(`foo`, 0)
```

Bazy danych i ORM



Sequelize

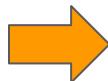
Jak coś dodać do bazy danych INSERT



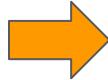
`.build().save()` - metoda “tworząca rekord” i zapisująca go w bazie danych

```
Zadanie.build({ tytul: 'foo', status: 0 })  
  .save()  
  .then(function(next) {  
    //  
  }).catch(function(error) {  
  })
```

```
INSERT INTO `Zadanie`  
(`title`, `status`)  
VALUES  
(`foo`, 0)
```



Samo stworzenie zapytania metodą “build” jest nie wystarczające!



Stworzony rekord musi zostać zapisany do bazy danych metodą
“save”



Bazy danych i ORM



Sequelize

Usuwanie rekordów z bazy danych DELETE

Bazy danych i ORM



Sequelize

Usuwanie rekordów z bazy danych DELETE



`.destroy()` - usuwa rekord z bazy danych spełniający określone warunki

Bazy danych i ORM



Sequelize

Usuwanie rekordów z bazy danych DELETE



`.destroy()` - usuwa rekord z bazy danych spełniający określone warunki

```
-----  
|  
| Projekt.findById(4)  
|   .success(function(proj) {  
|     proj.destroy()  
|       .success( )  
|       .error( )  
|   })  
|   .error();  
|  
-----
```

```
DELETE FROM `Projekt`  
WHERE `id` = 4
```



KONIEC WYKŁADU 12
