

# Notatki z kursu Programowanie dla WWW

Małgorzata Dymek

2019/20, semestr zimowy

## 1 Wstęp.

- **Frontend**

- **HTML** - podstawowy **język znaczników** dla opisu stron www
- **CSS** (Cascading Style Sheets) - Kaskadowe arkusze stylów. Język do definiowania atrybutów, które wpływają na wygląd strony.
- **JavaScript** - de-facto standard dla programowania po stronie klienta. Umożliwia tworzenie interaktywnych stron.
- **jQuery** - jedna z najczęściej stosowanych bibliotek JavaScript do manipulowania elementami strony WWW.
- Pozostałe - to bardziej rozbudowane biblioteki lub frameworki do budowy aplikacji WWW.

- **Backend**

- **Java** - jeden z najpopularniejszych i najbardziej uniwersalnych języków programowania.
- **Serwlety** - niewielkie aplikacje (najczęściej w Javie) rozszerzające możliwości serwera WWW. Odpowiadają na żądania, które serwer otrzymuje od klienta. Najczęściej generują dynamiczne strony WWW lub zwracają dane np. pobrane z bazy danych.
- **Java Server Pages (JSP)** - technologia **generowania stron** WWW. W specyficznym języku znaczników będącym mieszanką HTML-a i Javy tworzone są szablony stron, które są następnie tłumaczone na serwlety w Javie.
- **Spring MVC** - framework będący częścią Springa pozwalający ułatwiający tworzenie aplikacji webowych bazujących na serwletach w modelu **Model-View-Controller**
- **Node.js** - wieloplatformmowe **środowisko uruchomieniowe** (runtime) **JavaScript** działające po stronie **serwera**.
- **Django - framework w Pythonie** do szybkiego tworzenia aplikacji internetowych bazujących na wzorcu model-template-view
- **Ruby on Rails - framework w Ruby** do tworzenia aplikacji internetowych
- **ASP.NET - framework od Microsoftu** do tworzenia **dynamicznych aplikacji** internetowych przy użyciu m.in. języków C# i VB.NET
- **Go** - język stworzony w firmie Google. Doskonale nadaje się do tworzenia aplikacji internetowych (po stronie serwera).
- **Swift** - język stworzony przez Apple dla iOS itp. Na jego bazie istnieją frameworki umożliwiające tworzenie aplikacji webowych

Czym jest aplikacja www?	
Klasyczne serwisy WWW	Single Page Applications
<ul style="list-style-type: none"> <li>Ładowane są <b>całe strony</b>.</li> <li>Strony są <b>generowane</b> przez skrypty CGI, serwlety Java, itp.</li> <li><b>Zmiana zawartości</b> strony wymaga załadowania <b>całej</b> strony</li> <li>Cała <b>logika</b> aplikacji (lub jej większa część) działa po stronie <b>serwera</b> (backend)</li> </ul>	<ul style="list-style-type: none"> <li>Cała strona(y) jest <b>ładowana jednorazowo</b> na początku</li> <li>Cała <b>logika</b> aplikacji działa po stronie <b>klienta</b> (js).</li> <li>Różne fragmenty aplikacji są wyświetlane i/lub tworzone <b>dynamicznie</b> w czasie działania aplikacji</li> <li>Potrzebne dane pobierane z serwera przy pomocy wywołań <b>AJAX</b>.</li> </ul>

## 2 URL i HTTP.

Internet i WWW to nie to samo	
Internet	WWW
The Internet is a <b>global system of interconnected computer networks that interchange data</b> by packet switching using the standardized Internet Protocol Suite (TCP/IP).	The World Wide Web (WWW, or simply Web) is an <b>information space</b> in which the items of interest, referred to as <b>resources, are identified by global identifiers</b> called Uniform Resource Identifiers (URI)

### Jak działa WWW?

1. Użytkownik wpisuje w polu adresu przeglądarki adres (URL)
2. Przeglądarka analizuje wpisany adres sprawdzając protokół (HTTP) oraz nazwę hosta.
3. Przeglądarka łączy się z serwerem DNS i uzyskuje adres IP hosta.
4. Przeglądarka nawiązuje połączenie TCP/IP serwerem WWW działającym pod danym adresem.
5. Przeglądarka wysyła w ramach protokołu HTTP żądanie (request) pobrania zasobu: GET /home.html HTTP/1.1
6. Serwer lokalizuje żądany zasób statyczny lub generuje go dynamicznie, a następnie zwraca w postaci odpowiedzi (response) HTTP zawierającej treść strony w postaci HTML.
7. Przeglądarka wyświetla zawartość HTML w postaci graficznej.

Powstało w roku **1990**.

### 2.1 URI

**URI (Uniform Resource Identifier)** - ciąg znaków **jednoznacznie identyfikujący** dowolny abstrakcyjny lub fizyczny zasób.

- **Uniform** - pozwala w **jednorodny** sposób odwoływać się do różnych rodzajów zasobów niezależnie od sposobu dostępu do nich. Definiuje ogólną postać identyfikatora pozwalając na rozszerzanie i uzupełnianie definicji dla różnych typów zasobów. (http, ftp, isbn)
- **Resource** -dowolny abstrakcyjny lub fizyczny **zasób**, który może być identyfikowany przez URI.
- **Identifier** - identyfikator pozwalający na **odróżnienie** jednego zasobu od drugiego. Nie musi zawierać informacji o sposobie uzyskania dostępu do tego zasobu.

URI definiuje jedynie ogólne ramy schematów nazewnictwa (**naming schemes**), podczas gdy szczegóły składni są definiowane w specyfikacjach poszczególnych schematów.

Każde URI może być **zaklasyfikowane** jako **lokalizator URL**, **nazwa URN** lub **oba** na raz.

- **URL** to URI określające również **lokalizację** zasobu (np. adres sieciowy).
- **URN**, to identyfikator nadający zasobowi "**nazwę**" - **identyfikator**, który pozostaje **globalnie unikalny** i trwały nawet jeśli sam zasób przestanie istnieć lub stanie się niedostępny.

Sam schemat nie przesądza o tym, czy URI jest URN. Np. przestrzenie nazw w XML są URN-ami, chociaż na ogół korzystają ze schematu http.

### 2.1.1 Składnia URI

```
scheme: [//[user[:password]@]host[:port]] [/path] [?query] [#fragment]
```

- **scheme** - schemat, np. http, https, mailto, ftp, file. Odpowiada często **protokołowi dostępu** do zasobu, ale nie zawsze tak być musi. Np. dla file nie ma konkretnego protokołu.
- **authority** - składa się z:
  - opcjonalnej sekcji autentykacji: username:password zakończonej znakiem (@)
  - **host** - w postaci nazwy hosta, adresu IPv4 lub IPv6,
  - opcjonalnego portu oddzielonego od hosta dwukropkiem (:).
- **path** - ścieżka do zasobu. Jeśli w URI występowała sekcja authority, to ścieżka musi zaczynać się pojedynczym ukośnikiem. Nie może się zaczynać podwójnym ukośnikiem.
- **query** - opcjonalne zapytanie jest oddzielone od wcześniejszych części pytajnikiem (?). Na ogół składa się z **zestawu par** <atrybut>=<wartość> rozdzielonych ampersandem (&) lub średnikiem (;).
- **fragment** - opcjonalny fragment oddzielony od wcześniejszych części haszem (#). Zawiera **identyfikator zasobu podrzędnego**. Może to być np. sekcja w dokumencie. W przypadku dokumentu HTML jest to na ogół identyfikator pewnego elementu w dokumencie, do którego przeglądarka przewija dokument podczas wyświetlania. Uwaga: Fragment jest obsługiwany tylko w przeglądarce i **nie jest wysyłany do serwera**.

## 2.2 Protokół HTTP

- **Hypertext Transfer Protocol** - protokół do **wymiany danych w sieci** Internet. HTTP jest **niesymetrycznym** protokołem request-response klient-serwer.
- Protokół HTTP jest **bezstanowy**, co znaczy, że serwer nie jest w stanie bez dodatkowej informacji, zawartej w żądaniu, stwierdzić że poszczególne żądania należą do danej konwersacji między klientem a serwerem. **Serwer nie wie, co działo się w poprzednich żądaniach**.
- HTTP pozwala klientowi i serwerowi na **negocjacje** typów danych i reprezentacji przesyłanych informacji.

HTTP request	HTTP response
<ul style="list-style-type: none"><li>• linię <b>żądania</b> (request line),</li><li>• <b>nagłówki</b> (headers) - przesyłane w postaci <b>par</b> nazwa:wartość, oddzielone przecinkami.</li><li>• <b>(pusta linia)</b></li><li>• opcjonalnie <b>treść</b> żądania (request body)</li></ul>	<ul style="list-style-type: none"><li>• linię <b>statusu</b> (status line),</li><li>• <b>nagłówki</b> (headers),</li><li>• <b>(pusta linia)</b></li><li>• opcjonalnie <b>treść</b> odpowiedzi (response body)</li></ul>

HTTP METHODS	
Method	Description
<b>GET</b>	The GET method is used to <b>retrieve information</b> from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
<b>HEAD</b>	Same as GET, but it transfers the <b>status line and the header section only</b> . (Eg. check if current cached version is correct by checking the "last-modified" header).
<b>POST</b>	A POST request is used to <b>send data</b> to the server.
<b>PUT</b>	<b>Replaces</b> all the current representations of the target resource with the uploaded content.
<b>DELETE</b>	<b>Removes</b> all the current representations of the target resource given by URI.
<b>CONNECT</b>	<b>Establishes a tunnel</b> to the server identified by a given URI.
<b>OPTIONS</b>	<b>Describe the communication options</b> for the target resource.
<b>TRACE</b>	Ask the server to return a <b>diagnostic trace</b> of the actions it takes.
<b>PATCH</b>	<b>Modify part</b> of the content (instead of replace as in PUT).

PORÓWNANIE GET I POST		
	GET	POST
<b>Przycisk Wstecz/ Odświeżenie strony</b>	Niegroźne	Dane zostaną powtórnie przesłane. (przeglądarka powinna ostrzec użytkownika)
<b>Zakładki</b>	Można dodać do zakładek	Nie można dodać do zakładek
<b>Cachowanie</b>	Można cachować	Nie można cachować
<b>Encoding type</b>	application/x-www-form-urlencoded	application/x-www-form-urlencoded lub multipart/form-data. Dla danych binarnych używamy multipart encoding
<b>Historia</b>	Parametry pozostają w historii przeglądarki	Parametry nie są przechowywane w historii przeglądarki
<b>Ograniczenia na długość przekazywanych danych</b>	Metoda GET dodaje parametry do URL-a, a wiele przeglądarek ogranicza długość URLa (np. do 2048 znaków)	Brak ograniczeń
<b>Ograniczenia na przesyłane dane</b>	Tylko znaki ASCII	Brak ograniczeń (również binarne)
<b>Bezpieczeństwo</b>	GET mniej bezpieczny niż POST bo dane są przekazywane jako część URLa. Nigdy nie należy przysyłać haseł i innych wrażliwych danych przy pomocy GET	POST jest <b>tylko odrobinę bezpieczniejszy</b> niż GET, ponieważ dane nie są przechowywane w historii przeglądarki ani w logach serwera.

STATUSY ODPOWIEDZI		
<b>1xx</b>	<b>Informational</b>	<b>Request received, server is continuing the process.</b>
100	Continue	The server received the request and is in the process of giving the response.
<b>2xx</b>	<b>Success</b>	<b>The request was successfully received, understood, accepted and serviced.</b>
200	OK	The request is fulfilled.
<b>3xx</b>	<b>Redirection</b>	<b>Further action must be taken in order to complete the request.</b>
301	Move Permanently	The resource requested for has been permanently moved to a new location. The URL of the new location is given in the response header called Location. The client should issue a new request to the new location. Application should update all references to this new location.
302	Found & Redirect (or Move Temporarily)	Same as 301, but the new location is temporarily in nature. The client should issue a new request, but applications need not update the references.
304	Not Modified	In response to the If-Modified-Since conditional GET request, the server notifies that the resource requested has not been modified.
<b>4xx</b>	<b>Client Error</b>	<b>The request contains bad syntax or cannot be understood.</b>
400	Bad Request	Server could not interpret or understand the request, probably syntax error in the request message.
401	Authentication Required	The requested resource is protected, and require client's credential (username/password). The client should re-submit the request with his credential (username/password).
403	Forbidden	Server refuses to supply the resource, regardless of identity of client.
404	Not Found	The requested resource cannot be found in the server.
405	Method Not Allowed	The request method used, e.g., POST, PUT, DELETE, is a valid method. However, the server does not allow that method for the resource requested.
408	Request Timeout	
414	Request URI too Large	
<b>5xx</b>	<b>Server Error</b>	<b>The server failed to fulfill an apparently valid request.</b>
500	Internal Server Error	Server is confused, often caused by an error in the server-side program responding to the request.
501	Method Not Implemented	The request method used is invalid (could be caused by a typing error, e.g., "GET" misspell as "Get").
502	Bad Gateway	Proxy or Gateway indicates that it receives a bad response from the upstream server.
503	Service Unavailable	Server cannot response due to overloading or maintenance. The client can try again later.
504	Gateway Timeout	Proxy or Gateway indicates that it receives a timeout from an upstream server.

### 2.2.1 Metody bezpieczne i dempotentne.

**Theorem 2.1** *Metoda bezpieczna (safe) - metoda, która nie generuje efektów ubocznych. W praktyce są to metody nie modyfikujące danych zasobów na serwerze*

**Theorem 2.2** *Metoda idempotentna - to metoda, której wielokrotne wykonanie daje takie same efekty, jak wykonanie jej jeden raz. Klient może tak metodę wykonać wielokrotnie i oczekuje, że jej efekt w stosunku do zasobów serwera będzie taki sam jak w przypadku wykonania jednorazowego, chociaż zwrócona odpowiedź może się różnić.*

Metoda	Bezpieczna?	Idempotentna?
GET	Tak	Tak
HEAD	Tak	Tak
OPTIONS	Tak	Tak
TRACE	Tak	Tak
PUT	Nie	Tak
DELETE	Nie	Tak
POST	Nie	Nie

Prawidlowa implementacja metod leży po stronie twórcy oprogramowania. Twórca/projektant backendu musi zadbać aby jego obsługa metod była zgodna z powyższymi wymaganiami.

### 2.2.2 Nagłówki.

Ogólne	
Connection	umożliwia tworzenie trwałych połączeń (od wersji HTTP/1.1). Wysłanie Connection: close wymusza zakończenie połączenia.
Nagłówki żądania	
Cookie	przekazuje zwrotnie do serwera HTTP Cookie (ciasteczko) przesłane wcześniej z serwera w nagłówku Set-Cookie
User-Agent	identyfikuje rodzaj aplikacji klienckiej (przeglądarkę), system operacyjny, klienta, itp.
Host	nazwa domenowa serwera. W wersji HTTP/1.1 (i 2.0) nagłówek obowiązkowy. Umożliwia Virtual hosting
Accept-Language	język(i) akceptowane/preferowane przez klienta
Nagłówki odpowiedzi	
Content-Type	typ MIME zawartości (np. text/html, text/css, image/png)
Content-Length	rozmiar ciała (body) odpowiedzi. (W przypadku metody HEAD, rozmiar ciała, które byłoby wysłane dla metody GET).
Set-Cookie	HTTP Cookie umożliwia między innymi zapamiętywanie stanu (sesji).

## 3 HTML - HyperText Markup Language.

### 3.1 XHTML

Znacznie bardziej restrykcyjny, bo **zgodny z XML**.

- Znaczniki i atrybuty w muszą być pisane **małymi literami**.
- Wszystkie elementy muszą być **poprawnie zakończone** (np. <br/>).
- Wszystkie wartości atrybutów muszą być w **cudzysłowach** lub **apostrofach**.

- Każdy atrybut musi mieć **wartość**.
- Bardziej **surowe zasady zagnieżdżania** (np. zakazane <a> w <a>, <form> w <form> itd).
- Znaki <> & ' " muszą być reprezentowane przez **encje** (np. &amp) nawet w wartościach atrybutów.
- Dla znaczników a, form, map itd należy używać **id zamiast name**.
- Dokument musi zaczynać się od **deklaracji** <!DOCTYPE html PUBLIC ... XHTML ...
- Dokument musi zawierać elementy <html>, <head>, <title> i <body>.

## 3.2 HTML 5

- Dużo **wymogów** dotyczących **poprawności dokumentów** i ich dokładniejsze sprawdzanie.
- Możliwość zastosowania **narzędzi pochodzących z xml**
- Wsparcie **przestrzeni nazw**
- Zintegrowane **wsparcie dla multimedii** niewymagające użycia wtyczek.
- **Brak wersji** (Strict, Frames, Transitional).
- Dobrze tolerowany przez stare przeglądarki.
- **Nowe elementy**, np.: article, aside, audio, footer, header.
- **Nowe API**
  - **GeoLocation API** - lokalizacja użytkownika
  - **Web Storage API** - wygodniejsze składowanie informacji po stronie klienta
  - **Web Socket API** - full-duplex komunikacja z serwerem.
  - **Web Worker API** - umożliwia uruchamianie zadań w Javascriptcie w innym wątku niż interfejs przeglądarki.
  - **Drag & Drop API** - wsparcie dla "drag and drop"
- **Nowe atrybuty**: charset (dla elementu meta), async (dla elementu script)
- **Globalne atrybuty** (które mogą być zastosowane do dowolnego elementu): id, tabindex, hidden, data-\* (custom data attributes)
- Usunięto niektóre **zdeprecjonowane elementy**: acronym, applet, basefont, big, center, dir, font, frame, frameset, isindex, noframes, strike, tt
- **Zmiana znaczenia/użycia** niektórych istniejących atrybutów i elementów (np <i> i <small> - zmiana znaczenia na bardziej semantyczne)
- **Nowe typy elementów formularzy**: dates and times, email, url, search, range, tel, color.
- **Content type**
  - Dla zawartości w standardzie HTML: text/html
  - Dla zawartości w standardzie XHTML: application/xhtml+xml (ewentualnie application/xml lub text/xml)

**Interpretacja przez przeglądarkę.** W zależności od Content type oraz <!doctype> przeglądarka interpretuje zawartość w różny sposób.

<b>Content-Type</b>	text/html	text/html	application/xhtml+xml
<b>doctype</b>	brak/stary	poprawny	bez znaczenia
<b>Wynik</b>	HTML(quirks mode)	HTML 4.01 / HTML 5	XHTML

W przypadku trybu html nie zauważy np. zamknięcia elementu typu <strong/>.

## 4 CSS.

- **Kaskadowe arkusze stylów** (ang. Cascading Style Sheets) - język opisu sposobu prezentacji (wyglądu) dokumentu HTML.
- CSS umożliwia **rozdzielenie struktury** i zawartości dokumentu od **sposobu jego prezentacji**

- Określanie **czcionek, marginesów, odstępów** między liniami; **rozmiarów i położenia** poszczególnych elementów strony; **obramowań i wypełnień**.
- Zmiana sposobu **prezentacji standardowych elementów** (np. punktory w listach)

## 4.1 Selektory

- Selektor decyduje do którego elementu strony są stosowane właściwości
- Kryterium wyboru może być m.in. typ elementu, atrybuty (w szczególności id i class) oraz położenie elementu względem innych elementów.
- Pseudo-elementy dają dostęp m.in. do fragmentów elementów (::first-letter)
- Można także używać pseudo-klas dających dostęp do informacji nie zawartych bezpośrednio w dokumencie (:hover, :visited)

Selector	Example	Example description
<b>.class</b>	.intro	Selects all elements with class="intro"
<b>#id</b>	#firstname	Selects the element with id="firstname"
<b>*</b>	*	Selects all elements
<b>element</b>	p	Selects all <p> elements
<b>element,element</b>	div, p	Selects all <div> elements and all <p> elements
<b>element element</b>	div p	Selects all <p> elements inside <div> elements
<b>[attribute]</b>	[target]	Selects all elements with a target attribute
<b>[attribute=value]</b>	[target=_blank]	Selects all elements with target="_blank"

## 4.2 Priorytety

Priority	CSS source type	Description
1	Importance	The !important annotation overwrites the previous priority types
2	Inline	A style applied to an HTML element via HTML 'style' attribute
3	Media Type	A property definition applies to all media types, unless a media specific CSS is defined
4	User defined	Most browsers have the accessibility feature: a user defined CSS
5	Selector specificity	A specific contextual selector overwrites generic definition
6	Rule order	Last rule declaration has a higher priority
7	Parent inheritance	If a property is not specified, it is inherited from a parent element
8	CSS property definition in HTML document	CSS rule or CSS inline style overwrites a default browser value
9	Browser default	The lowest priority: browser default value is determined by W3C initial value specifications

Style związane z fontami są dziedziczone z elementu rodzica. Większość pozostałych stylów nie jest dziedziczona.

## 4.3 Specyficzność.

Specyficzność określa "**szczegółowość**" selektora:

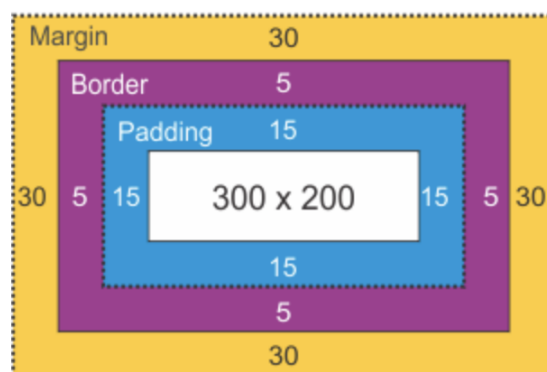
0, 0, 0, 1	prosty selektor (selektor <b>elementu</b> )	(p)
0, 0, 1, 0	selektor <b>klasy</b> ma specyficzność	(.head-button)
0, 1, 0, 0	selektor oparty na <b>id</b> ma specyficzność	(#save-button)
1, 0, 0, 0	style <b>inline</b> ma specyficzność	



Selector "body #content .data img:hover" ma specyficzność równą 0,1,2,2.

## 4.4 Box Model

- Każdy element HTML można traktować jako **prostokąt**.
- Zawartość elementu (**content**) jest otoczona wypełnieniem (**padding**), obramowaniem (**border**) oraz marginesem (**margin**).
- W zależności od wartości box-sizing (**content-box** lub **border-box**) obliczany jest całkowity rozmiar elementu.



## 4.5 Dodatki

- **CSS Transforms** - pozwala na transformacje takie jak przesunięcie, skalowanie, obracanie, itp.
- **CSS Transitions** - pozwala na płynną zmianę wartości.
- **SS Flexbox Layout**
  - Elastyczny **menadżer układu elementów** na stronie.
  - Elementy mogą być typu flex-container i/lub flex-item.
  - Dla **flex-container** możemy określić **kierunek i sposób ułożenia** elementów flex-item (pionowo/poziomo, prawo/lewo, zawijanie wierszy, wykorzystanie wolnej przestrzeni).
  - Dla **flex-item** możemy określić **zdolność elementu** do powiększania/zmniejszania, wypełniania pustej przestrzeni, bazowy rozmiar, kolejność wyświetlania, itp.
  - Stosunkowo łatwo wykonać **podział strony** na kilka **kolumn** tej samej wysokości.

## 5 Javascript

- Jest rozwinięciem języka stworzonego przez Brendana Eich z Netscape (najpierw pod nazwą Mocha, później LiveScript i ostatecznie JavaScript).
- W **1996** pojawiła się pierwsza przeglądarka (Netscape Navigator 2.0) zawierająca obsługę JavaScriptu.
- W tym samym roku Microsoft wypuścił Internet Explorera 3.0 z obsługą własnej wersji tego języka pod nazwą **JScript**.
- W **1997** pojawił się standard tego języka pod nazwą ECMAScript\*. Obecnie Javascript, JScript (Microsoft), ActionScript (Macromedia, a potem Adobe wykorzystany we Flashu i Flexie) są po prostu implementacjami tego standardu (zawierającymi często różne rozszerzenia).
- Istnieją **kompilatory/translatorzy** (transpiler) tłumaczące kod np z ES6 na ES5.
- Istnieją **rozszerzenia** Javascriptu (ECMAScriptu): TypeScript (używany w Angular i narzędziach Microsoftu), CoffeeScript, JSX (ReactJs).

## 5.1 IIFE (Immediately Invoked Function Expression)

1. Funkcja anonimowa zwraca obiekt z funkcjami operującymi na zmiennej 'prywatnej' x
2. Funkcja anonimowa jest natychmiast wywoływana (z pustą listą argumentów) i wartość zwracana z funkcji jest przypisywana do counter (która jest obiektem zwracanym jako funkcja).

W taki między innymi sposób są realizowane w JavaScriptcie moduły (Module Pattern)

### strict mode

- Włączamy dodając "use strict"; na początku skryptu lub funkcji
- Pojawił się w ECMAScript 5 i jest wspierany w IE from version 10. Firefox from version 4. Chrome from version 13. Safari from version 5.1.+ Opera from version 12.
- Nawet jeśli nie jest wspierany, można go użyć - **nie powoduje to błędów**.
- W tym trybie **wykrywane i zabronione** są niektóre **potencjalnie niebezpieczne konstrukcje**, takie jak np. użycie zmiennej lub obiektu bez wcześniejszej deklaracji.

### this

- Słowo kluczowe this ma trochę inne znaczenie w Javascript niż w C++, czy Javie.
- Jego wartość **różni się** w zależności od **kontekstu**:
  - Na zewnątrz funkcji this oznacza pewien obiekt globalny (w przeglądarkach - window)
  - W "normalnych" funkcjach w trybie **strict** - obiekt globalny **window**, w trybie **sloppy** - **undefined**
  - W **konstruktorach** i **metodach** odnosi się do tworzonego/modyfikowanego **obektu**.

## 5.2 DOM i BOM

Kod w Javascriptcie oddziałuje na przeglądarkę i dokumenty w niej wyświetlane przy pomocy interfejsów DOM i BOM.

DOM - Document Object Model	BOM - Browser Object Model
<ul style="list-style-type: none"><li>• Odzworowuje dokument HTML lub XML w postaci <b>drzewa obiektów</b>.</li><li>• Wykrywanie elementów: <b>document</b> .getElementById, .ByTagName, .ByClassName, .querySelectorAll(css_selector).</li><li>• Zmiana elementów: <b>element</b> .attribute=, .setAttribute(attribute,value), .innerHTML=, .style.property=.</li><li>• Dodawanie/usuwanie elementów: document.createElement, appendChild, replaceChild, removeChild, write(text).</li><li>• <b>Event</b>: onclick, onmouseover, onmouseout</li><li>• Najlepiej używać metod: <b>addEventListener</b>, <b>removeEventListener</b>, które pozwalają na <b>przypisanie kilku procedur</b> obsługi zdarzeń.</li></ul>	<ul style="list-style-type: none"><li>• <b>Opisuje</b> metody i interfejsy umożliwiające <b>interakcję z przeglądarką</b></li><li>• Brak standardu - przeglądarki mogą mieć różne implementacje</li><li>• Główny obiekt - <b>window</b></li><li>• Ważniejsze obiekty i metody:<ul style="list-style-type: none"><li>– window.<b>location</b> - URL strony. Umożliwia załadowanie strony z adresu, odświeżenie, itp.</li><li>– window.<b>history</b> - historia przeglądanych stron</li><li>– window.<b>screen</b> - dane dotyczące ekranu</li><li>– <b>console</b> - konsola diagnostyczna przeglądarki</li><li>– <b>alert</b>, <b>confirm</b> itd. - wyświetlanie okien dialogowych</li></ul></li></ul>

## 6 React.

- Biblioteka w JavaScript do tworzenia **interfejsu użytkownika**

- Stworzona przez Facebooka
- Jedna z najpopularniejszych bibliotek JavaScript
- Nie jest pełnym frameworkiem (jak np. Angular)
- **Jednokierunkowy przepływ danych** (single-way data flow): "properties flow down; actions flow up"
- **Virtual DOM** (wirtualny obiektowy model dokumentu) - React tworzy w pamięci swój własny DOM w którym monitoruje wszystkie zmiany i uaktualnia widok w przeglądarce tylko o faktyczne zmiany.
- **JSX** (JavaScript Syntax eXtension) - pozwala na **wprowadzanie składni HTML**-o podobnej bezpośrednio w kodzie JavaScript
- React Native - biblioteki dla IOS-a, Androida
- **Props vs. State**
  - **props** - dane przekazane do komponentu (jak parametry wywołania funkcji)
  - **state** - stan wewnętrzny komponentu. zmieniany **asynchronicznie** przez setState().

## 7 Node.js

- **Środowisko uruchomieniowe Javascript** oparte na silniku Google Chrome V8
- Stworzony w **2009** roku przez Ryana Dahla w celu **wyeliminowania problemów ze skalowalnością** tradycyjnych serwerów HTTP
- **Open source**: licencja MIT
- Wykorzystywany przede wszystkim do tworzenia **serwerów** HTTP, ale jest też używany do tworzenia aplikacji desktopowych

### 7.1 Architektura

- Jeden wątek z **pętlą zdarzeń** (event loop)
- **Nieblokujące I/O** bazujące na zdarzeniach i wywołaniach zwrotnych (**callback**)
- Bazuje na bibliotece libuv (napisanej w języku C)

Zalety	Wady
<ul style="list-style-type: none"> <li>• <b>Duża szybkość</b>, potrafi obsłużyć znacznie więcej żądań niż tradycyjny serwer (jak np. Apache)</li> <li>• <b>Brak narzutów</b> związanych z tworzeniem i utrzymywaniem <b>wątków</b></li> <li>• Wysoka <b>skalowalność</b> związana z użyciem nieblokującego API i pojedynczej pętli zdarzeń</li> <li>• <b>Rozbudowany ekosystem</b> narzędzi i bibliotek</li> <li>• Możliwość pisania frontendu i backendu w tym samym języku</li> <li>• Jest częścią środowisk tworzenia aplikacji webowych: <ul style="list-style-type: none"> <li>– <b>MEAN</b> - MongoDB, Express, Angular, Node</li> <li>– <b>MERN</b> - MongoDB, Express, React, Node</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Niezbyt dobrze sprawdza się w zastosowaniach wymagających <b>dużej mocy obliczeniowej</b> (CPU intensive)</li> <li>• Stosunkowo <b>duża zmienność</b> całego środowiska i pakietów</li> <li>• <b>Callback hell</b> – problem z utrzymaniem <b>przejrzystości</b> kodu z związku z potrzebą stosowania dużej liczby wywołań zwrotnych.</li> </ul>

## 7.2 NPM - Node Package Manager.

- Podstawowe **narzędzie do zarządzania pakietami** w Node.js
- Umożliwia instalację i aktualizację pakietów z uwzględnieniem **zależności** między nimi
- Pakiety można instalować **globalnie** lub **lokalnie**
- **Synchroniczny lub asynchroniczny model programowania.**
- **Request** i **response** są strumieniami, które generują zdarzenia 'error', 'data', 'end'.
- **Plik package.json**
  - Zawiera ważne **informacje** na temat tworzonego pakietu/aplikacji
  - Zawiera **listę pakietów** od których zależy pakiet/aplikacja wraz z **numerami wersji**
  - Pozwalana na **automatyczne pobranie** wszystkich **niezbędnych zależności**

## 8 ExpressJS

Różne standardy (czyli **brak standardu**):

- **CommonJS** - serwery, Node
- **AMD** (Asynchronous Module Defintion) - przeglądarki
- **ES2015** - wbudowane w język, ale jeszcze niezbyt rozpowszechnione

**Moduły** umożliwiają

- **podział kodu** na (bardziej) **niezależne** części
- łatwiejsze **współdzielenie** i **powtórne wykorzystanie** kodu
- **ukrycie implementacji** i udostępnienie jedynie wybranych funkcji i zmiennych (interfejs)
- unikanie **konfliktów nazw**

### 8.1 CommonJS

- Plik modułu **eksportuje** funkcje i obiekty poprzez przypisując je do obiektu **module.exports**.
- Plik wykorzystujący moduł **ładuje** go przy pomocy funkcji **require()**.

### 8.2 ExpressJS

- Najpopularniejszy **framework** w Node
- Unopinionated - **nie narzuca rozwiązań** i struktury. Daje możliwość wyboru.
- Podpinanie obsługi pod **różne metody HTTP** i adresy (routing)
- Integracja z **różnymi silnikami szablonów** stron WWW
- Możliwość **rozszerzania funkcjonalności** za pomocą **middleware**: obsługa sesji, cookies, logowania użytkowników, itp.

#### 8.2.1 Routing

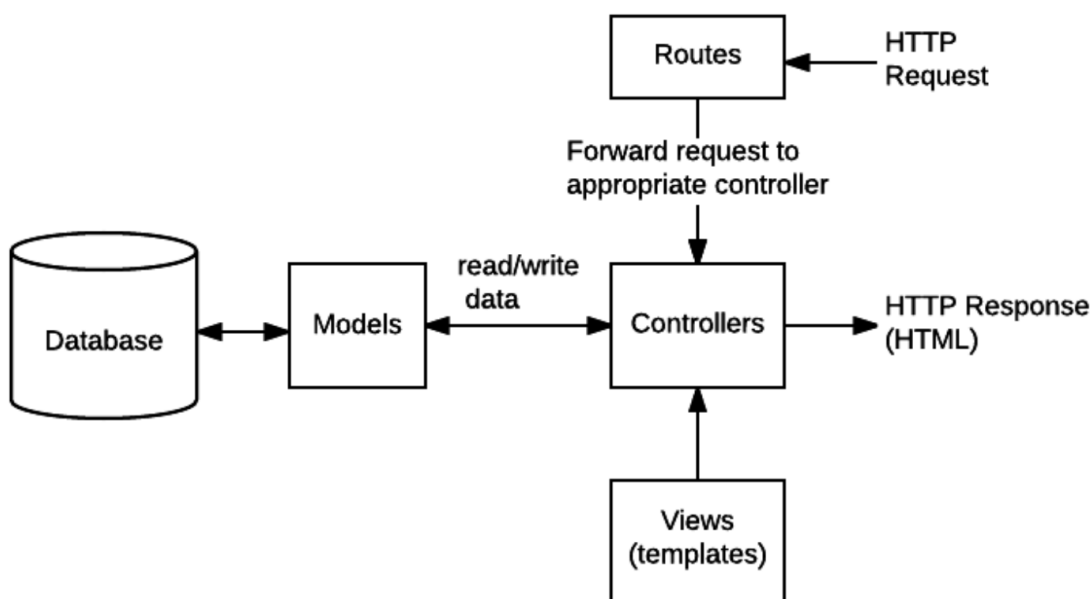
- Definiowanie obsługi **poszczególnych metod i ścieżek**: app.get(), .post(), .put(), .delete() itp.
- Definiowanie obsługi dla **dowolnej metody**: app.all().

**Express.Router** daje możliwość **podzielenia obsługi** serwisu/aplikacji webowej na części **w zależności od prefiksu** użytej ścieżki. Można **delegować** tę obsługę do **dodatkowego modułu**. Routingi zdefiniowane w `express_router.js` będą dostępne w pod adresem `/express_router` oraz `/express_router/about`.

Ścieżki używane przy routingu mogą być definiowane jako napisy z **wzorcami**, np:

- `app.get('/ab?cd', callback)` - `'/abcd'` lub `'/acd'`
- `app.get('/ab+cd', callback)` - `'/abcd'`, `'/abbc'`, `'/abbbc'`, ...
- `app.get('/ab*cd', callback)` - `'/abcd'`, `'/abXcd'`, `'/abDowolnyTekstcd'`
- `app.get(/.*info$/, callback)` - wyrażenia regularne javascript

### 8.2.2 Model MVC (Model-View-Controller)



### 8.2.3 Szablony

Express może współpracować z **różnymi systemami szablonów** HTML, m.in. Pug, Mustache, EJS.

**Pug**

- Zawieranie się elementów jest odzwierciedlone przy pomocy **wcięć**.
- Każda (prawie) linia zaczyna się od **nazwy znacznika** HTML
- Jeśli tag kończy się znakiem `=`, to dalsza część jest traktowana jako **wyrażenie Javascript**, którego wartość jest wstawiana jako zawartość generowanego elementu.
- Można używać **instrukcji warunkowych** i **pętli**
- Można również **dziedziczyć/rozszerzać** szablony: polecenia `extend` i `block`.
- Wygląda jak bardzo uproszczony html, **bez nawiasów ostrych** `<>`.

## 9 Zarządzanie sesjami w aplikacjach webowych

- Protokół HTTP jest **bezstanowy** (choć sam serwer oczywiście jakiś stan posiada).

- Każde żądanie przychodzące jest **przetwarzane niezależnie** od ewentualnych wcześniejszych żądań
- W wielu sytuacjach konieczne jest **powiązanie kolejnych żądań** przychodzących od tego samego klienta
- Ciąg takich żądań przychodzących od jednego klienta, to **sesja**
- **Grupowanie żądań** przychodzących od jednego klienta musi być **realizowane przez programistę**

**Theorem 9.1** *Sesja jest **nowa** jeśli została utworzona po stronie serwera, ale klient do niej jeszcze **nie przystąpił**.*

## 9.1 Identyfikator sesji

- Z sesją wiążemy pewien **identyfikator generowany** przez serwer (identyfikator sesji)
- Identyfikator sesji jest generowany przez serwer **przy pierwszym żądaniu** przychodzącym od klienta
- **Wszystkie następne żądania** przychodzące od tego samego klienta **zawierają** ten identyfikator

## 9.2 Techniki przekazywania identyfikatora sesji

- **Przepisywanie URL** (URL rewriting)
- **Ukryte pola** w formularzach HTML
- **Ciasteczka** (cookies)

### 9.2.1 Przepisywanie URL (URL Rewriting)

- Identyfikator sesji jest **dołączany do URL-a** każdego żądania.
- Serwer na zwracanej stronie dołącza do wszystkich URL-i **dodatkowy parametr** z identyfikatorem sesji.
- Wywołanie przez klienta takiego URL-a pozwala serwerowi na **pobranie identyfikatora sesji** z przekazanego parametru

Zalety	Wady
1. <b>Działa z każdą przeglądarką</b> niezależnie od ustawień użytkownika	1. Trzeba <b>przepisać każdy</b> używany <b>link</b> 2. W przypadku potrzeby przekazania większej liczby informacji przez parametry URL-a <b>możemy osiągnąć limit długości URL-a</b> .

### 9.2.2 Ukryte pola formularza (hidden fields)

- Identyfikator sesji (i ewentualne inne dane) są **przekazywane** z serwera do przeglądarki jako **wartości ukrytych pól formularza**
- W przeglądarce po wykonaniu **submit**, zawartość tych pól jest **dołączana automatycznie** do wartości pozostałych pól i, jako parametry metody POST lub GET, są przekazywane z powrotem na serwer.

### 9.2.3 Cookies

- W kodzie serwlet pobieramy z **request ciasteczka** (jeśli są) oraz **tworzymy nowe** i dodajemy do **response**
- Przeglądarka **automatycznie** do zwrotnego request'u **dołącza** ciasteczka przesłane z serwera
- HTTP Cookie to **niewielki zestaw informacji** przekazywany z serwera do klienta
- Klient może takie ciasteczko **zachować i odesłać** z powrotem z następnym zapytaniem do serwera
- **Session cookies** - są **usuwane przez przeglądarkę** kiedy przeglądarka jest **zamykana**. Nie mają ustawionej dyrektywy MaxAge ani Expires.
- **Permanent cookies** - **dezaktywują się** w określonym **moomencie** (dyrektywa Expires) lub po określonym **czasie** (dyrektywa MaxAge). Mogą trwać pomiędzy kolejnymi uruchomieniami klienta.
- **Inne ważniejsze dyrektywy:**
  - **Secure** - ciasteczko może być wysłane na serwer jedynie przy użyciu **bezpiecznego protokołu HTTPS**
  - **HttpOnly** - ciasteczko może być wysłane na serwer **jedynie przez przeglądarkę**. Nie jest możliwe dołączenie ciasteczka z poziomu kodu w Javascript.
  - **Domain** - domena (nazwa hosta) do której ciasteczko może być wysłane. Dopuszczalne są także poddomeny. Jeśli Domain **nie jest utawione**, to można odesłać ciasteczko jedynie do **domeny z której przyszło** (wykluczając poddomeny).

### 9.3 Http Session

- W **serwletach** mamy do dyspozycji **interfejs HttpSession**.
- Kontener serwletów używa tego iterfejsu do utworzenia **trwałej sesji** trwającej **określany prze-dział czasu** i rozciągającej się wiele requestów od tego samego użytkownika.
- Konkretna implementacja zależy od serwera i jego konfiguracji i może opierać się np. na **ciasteczkach** lub **przepisywaniu URLi**.
- Interfejs HttpSession umożliwia:
  - **Podgląd i modyfikację informacji** o sesji, takich jak identyfikator sesji, moment utworzenia, czas ostatniego dostępu, itp.
  - **Dodawanie** do sesji **obiektów** przechowujących informację dostępną podczas następnych wywołań serwletu dla tej samej sesji
  - Wykorzystuje nagłówek **Authorization**.

### 9.4 Filtr Servlet

- Filtr jest obiektem, który jest **wywoływany przed obsłużeniem** przez serwlet **żądania** klienta oraz **po** jego obsłużeniu.
- Może **odczytywać i modyfikować zawartość żądania** oraz **odpowiedzi**.
- **Zastosowania filtrów"**
  - zapisywanie do **logów**
  - **kompresja**
  - **szyfrowanie** i deszyfrowanie
  - **autoryzacja** użytkownika

- walidacja dostępu do zasobów

## 9.5 Autentykacja użytkownika

### 9.5.1 HTTP basic authentication

- Mechanizm **wbudowany** w protokół HTTP
- Schematy: Basic (base64), Bearer(OAuth 2.0), Digest (md5, sha itp)
- **Basic authentication scheme**:
  - Identyfikator użytkownika wraz z hasłem przesyłane są w kodowaniu **base64**, które jest **odwracalne**
  - **Powininen** (właściwie musi) być używany z protokołem **HTTPS** aby uchronić hasło przed dostępem osób trzecich
  - Wsparcie dla tej metody jest wbudowane zarówno w kliencie jak i na serwerze
  - **Brak możliwości wylogowania** użytkownika

### 9.5.2 Cookies

- Po przesłaniu danych autoryzacyjnych serwer generuje **identyfikator sesji autoryzowanej**.
- Jest on **przechowywany na serwerze** oraz **przekazany do klienta**
- Klient **przekazuje** poprzez **każde** żądanie dostępu do zasobów
- Serwer **sprawdza**, czy identyfikator jest **poprawny i aktywny**
- **Wylogowanie** polega na **dezaktywacji** lub **usunięciu** identyfikatora z serwera

### 9.5.3 Tokeny

- Najczęściej używane są JSON Web Tokens (**JWT**)
- **Brak możliwości wylogowania użytkownika**, ale można określić **czas ważności tokena**
- Token **nie jest przechowywany na serwerze**, a tylko u klienta (Local storage, Cookies, itp)
- **Klient** wraz z żądaniem dostępu do zasobu **wysyła** token w **nagłówku Authorization: Bearer <token>**.
- **Serwer weryfikuje poprawność** tokenu przy pomocy swojego **prywatnego klucza**
- Struktura tokena:
  - **Nagłówek** (header) - określa **typ** tokenu i **rodzaj algorytmu haszującego**.
  - **Zawartość** (payload) - Zawiera stwierdzenia najczęściej na temat **tożsamości użytkownika** (sub - subject)
  - **Podpis** (signature) - **podpis kryptograficzny** nagłówka i zawartości.

## 10 Komunikacja Frontend-Backend

### 10.1 AJAX

- AJAX (**Asynchronous JavaScript and XML**) - **technika wykonywania zapytań HTTP i pobierania** z poziomu aplikacji internetowej w przeglądarce WWW **bez potrzeby przeładowania** całej strony.
- **Bazuje** na obiekcie **XMLHttpRequest** (często używany jest skrót **XHR**).
- Implementowany przez **wszystkie przeglądarki** i dostępny z poziomu kodu Javascript.



- Umożliwia **wysłanie żądania HTTP** do serwera, **pobranie zasobów**, **przetworzenie** ich w kodzie Javascriptu bez potrzeby przeładowania całej strony.
- Oryginalnie wykorzystywany przede wszystkim do pobierania danych w **formacie XML**, obecnie bardzo popularny jest **JSON**.

### 10.1.1 Obiekt XMLHttpRequest

- **Zwróconą zawartość** można uzyskać z pomocą
  - **responseText** - w postaci tekstu
  - **responseXML** - jako obiekt dokumentu XML
- W nowszych wersjach przeglądarek pole **responseType** pozwala na **ustawienie typu** zwracanej zawartości. Możliwe są między innymi wartości:
  - **"text"** - wynik zwracany jako string
  - **"document"** - wynik zwracany jako dokument HTML lub XML
  - **"json"** - obiekt Javascript powstały na podstawie przesłanego JSON-a.
- Pole **readyState** może przyjmować następujące wartości:
  - **0** - zapytanie **niezainicjowane**
  - **1** - zapytanie **otwarte**
  - **2** - zapytanie **wysłane**
  - **3** - **odbieranie odpowiedzi**
  - **4** - zapytanie **zakończone**

Uwaga: Jeśli przekazywana treść nie jest w odpowiednim formacie otrzymamy **null**.

- Metoda **setRequestHeader()** umożliwia **ustawienie nagłówka** w wysłanym zapytaniu.
- W starych wersjach Internet Explorera (IE 5/6) obiekt XMLHttpRequest należało uzyskać poprzez wywołanie.
- Z powodów bezpieczeństwa współczesne przeglądarki standardowo pozwalają jedynie na **ładowanie zasobów z tego samego serwera** z którego była załadowana strona główna. Jednym ze sposobów na obejście tego problemu jest protokół **CORS (Cross-domain requests)**.

## 10.2 Fetch API

- Nowy interfejs służący do **pobierania zasobów**, mający zastąpić XMLHttpRequest
- Oparte na **javascriptowych Promise**. Obiekt Promise reprezentuje **ewentualne zakończenie** (lub porażkę) **asynchronicznej operacji** i jej wartość.
- API **niskopoziomowe**
- Mamy **kontrolę nad wszystkimi parametrami zapytania HTTP**.
- Należy **samodzielnie ustawiać** wszystkie niezbędne **nagłówki**, ciasteczka, itp
- Jako **błąd** (Promise.reject()) traktowana jest sytuacja nie możliwości pobrania zasobu w wyniku **błędu sieciowego**.
- Nawet dla odpowiedzi z kodem HTTP 404 obsługa przebiega normalnie, tylko status ok jest ustawiany na false.

## 10.3 REST

- REST = **Representational State Transfer**
- Podejście zaproponowane przez Roya T. Fieldinga w 2000r.
- Oparty na pojęciu **zasobu identyfikowanego** przy pomocy **URI** i mogącego posiadać **różne reprezentacje** (np. XML, JSON).
- Operacje na zasobach wykorzystują **metody protokołu HTTP** (GET, POST, DELETE, itd.)
- **Statusy operacji** są zwracane jako **statusy protokołu HTTP**
- Zakładamy **bezstanowość**: każda operacja stanowi niezależną całość
- **Format** zwracanych danych powinna być **określana w nagłówku Content-type**, np. text/xml
- W praktyce **rzadko spotyka się** serwery całkowicie zgodne z wytycznymi REST
- Często **odstępstwa**:
  - Wykorzystanie **własnych kodów błędów**
  - **Przekazywanie** operacji w **URL-u** zamiast jako metody HTTP
  - **Przekazywanie formatu** zwracanych danych w **URL-u**
- Czysty REST **niezbyt dobrze nadaje się** do API **zorientowanego na operacje** zamiast na zasoby.

## 10.4 SOAP

- **Simple Object Access Protocol**
- Starszy i **bardziej skomplikowany** protokół **niż REST**
- SOAP jest **protokołem**, a **nie** jedynie **stylem budowania API**
- Oparty na **XML**
- Można go wykorzystywać do **przekazywania danych/zasobów** lub do **wywoływania procedur/operacji**
- Używany przede wszystkim w **ramach protokołu HTTP**, ale **może działać na innych** protokołach transportowych
- Wbudowana **obsługa błędów i protokołów bezpieczeństwa** (WS-Security: m.in. podpisywanie i szyfrowanie wiadomości)
- Posiada **dodatkowy język opisu udostępnianych operacji i danych (WSDL)**
- Ma **ściśle określoną** (dość skomplikowaną) **strukturę**. Wymaga przekazywania sporej ilości dodatkowych danych.
- "SOAP is like an envelope, whereas REST is like a postcard"

## 11 Spring vs JEE

### 11.1 JEE

- **Java for Enterprise Edition**
- Platforma do **tworzenia aplikacji biznesowych**
- Jest **zestawem specyfikacji**, a **nie implementacji**.
- Twórcy specyfikacji (Sun Microsystems, a następnie Oracle) udostępniają również **wzorcową implementację** specyfikacji JEE (obecnie: GlassFish lub Sun Java System Application Server)

- Podstawowe **interfejsy** programistyczne zdefiniowane w pierwszych wersjach, to
  - **JDBC** (Java Database Connectivity) - dostęp do baz danych
  - **Java Servlets** - obsługa komunikacji sieciowej (najczęściej HTTP)
  - **JSP** (Java Server Pages) - dynamiczne strony WWW
  - **EJB** (Enterprise Java Beans) - specyfikacja komponentów biznesowych po stronie serwera
- Niektóre z tych API zostały przeniesione do "zwykłej" Javy (JavaSE - Java Standard Edition), a JEE wzbogaciła się o wiele innych interfejsów
- **Problemy z JEE:**
  - Pierwsze wersje miały dużo możliwości, ale były **skomplikowane** w użyciu (szczególnie EJB)
  - Aplikacje JEE wymagały **dużego** (ciężkiego) **serwera** implementującego pełen zakres specyfikacji
  - W związku z tym pojawiły się konkurencyjne/komplementarne rozwiązania: **Spring**, Struts, Guiceł

**Serwery aplikacji** application servers) - **implementują pełną specyfikację** JEE między innymi z EJB (Java beans) i JMS (Java messaging).

## 11.2 Spring

Rozwiązuje pewne problemy z JEE:

- **Upraszcza tworzenie aplikacji biznesowych**, szczególnie tych, które nie potrzebowały dużej części interfejsów JEE
- Pozwala na pracę ze **zwykłymi obiektami Javy** (**POJO - Plain Old Java Object** zamiast skomplikowanych EJB).
- Jest **zestawem bibliotek** zawierających implementacje, co umożliwia wdrażanie aplikacji Springa na serwerach WWW nie implementujących JEE, a nawet budowanie aplikacji desktopowych.
- Opiera na **IoC (Inversion of Control)** i **Dependency Injection**, co pozwala między innymi ograniczyć stosowanie niezbyt wygodnych usług nazwicznych JNDI (Java Naming and Directory Interface)
- Różne **sposoby konfiguracji** systemu:
  - **pliki XML**
  - **adnotacje**
  - **kod Javy**
- **Dwa warianty budowy** aplikacji webowych: standardowe **serwlety** oraz nieblokujące technologie **Reactive stack**.
- Moduły Springa można wykorzystać także do budowy **aplikacji desktopowych**
- Można budować aplikacje zawierające **wbudowany serwer WWW**.
- Spring Boot umożliwiający **łatwe tworzenie różnych typów aplikacji** bazujący na domyślnych sensowych ustawieniach

### 11.2.1 Kontener IoC

- Filozofia Spring-a opiera się na **odwróceniu zależności** (IoC) i **wstrzykiwaniu zależności** (DE).
- Najważniejszą częścią architektury Springa jest kontener IoC w terminologii Spring-a zwany **kontekstem aplikacji**, implementujący interfejs ApplicationContext.

- Kontekst aplikacji **zarządza cyklem życia** komponentów (beans) tworzących aplikację.
- Komponenty to **zwykłe obiekty Javy** (POJO).

### 11.2.2 Wstrzykiwanie zależności w Springu

- przy pomocy **XML**-a
- przy pomocy **adnotacji**
- wstrzykiwanie zależności przy pomocy:
  - **konstruktora** - zalecane dla wymaganych zależności
  - metod ustawiających (**setterów**) - najlepiej dla zależności opcjonalnych
  - **pól w klasach**

### 11.2.3 Bean scopes

Ogólne zakresy.	
<b>singleton (default)</b>	tworzona tylko <b>jedna instancja</b> (w jednym ApplicattionContext)
<b>prototype</b>	Z każdym <b>żądaniem</b> pobrania bean'a zwracany jest <b>nowy egzemplarz</b>
Zakresy dostępne ApplicationContext typu webowego	
<b>request</b>	<b>Pojedyncza instancja</b> jest tworzona i dostępna podczas <b>trwania</b> jednego <b>żądania HTTP</b> (HTTP request)
<b>session</b>	<b>Pojedyncza instancja</b> jest tworzona i dostępna podczas trwania jednej <b>sesji HTTP</b> (HTTP session)
<b>application</b>	<b>Pojedyncza instancja</b> jest tworzona i dostępna podczas trwania <b>ServletCon-text</b> .
<b>websocket</b>	<b>Pojedyncza instancja</b> jest tworzona i dostępna podczas trwania <b>WebSocket</b> .

### 11.2.4 MVC

- **Model** - dane aplikacji (najczęściej POJO)
- **View** - prezentuje dane z modelu dla użytkownika (w naszym wypadku generuje stronę HTML)
- **Controller** - przetwarza żądania użytkownika generując/modyfikując model i przekazuje go do widoku w celu wyświetlenia

### 11.2.5 Dispatcher Servlet

- w Spring MVC głównym **obiektem synchronizującym** te **aktywności** jest DispatcherServlet
  - **Handler mapping** - mapuje **żądania HTTP** na odpowiednie **kontrolery**
  - **Controller** - wywołuje odpowiednie **metody** obsługujące żądania GET, POST, itp. Metody te **ustawiają odpowiedni model** danych i zwracają nazwę widoku.
  - **View Resolver** - mapuje **nazwy widoków** na odpowiednie **szablony stron** (np. JSP)
  - **View** - **Renderuje** stronę na podstawie szablonu z odpowiednim modelem

## 12 Bezpieczeństwo

### 12.1 OWASP Top 10 Application Security Risks 2017

**Open Web Application Security Project** - organizacja non-profit zajmująca się **zagrożeniami i zabezpieczeniami** aplikacji webowych.

- A1:2017 - Injection
- A2:2017 - Broken Authentication
- A3:2017 - Sensitive Data Exposure
- A4:2017 - XML External Entities (XXE)
- A5:2017 - Broken Access Control
- A6:2017 - Security Misconfiguration
- A7:2017 - Cross-Site Scripting (XSS)
- A8:2017 - Insecure Deserialization
- A9:2017 - Using Components with Known Vulnerabilities
- A10:2017 - Insufficient Logging & Monitoring

## 12.2 SQL Injection

- Polega na **niedostatecznej walidacji i filtrowaniu** wprowadzonych przez użytkownika danych, które są wykorzystywane do tworzenia **zapytania SQL**
- Dane te są **interpretowane** przez bazę **jako elementy języka SQL**, a nie - zwykle dane tekstowe.
- **Zabezpieczenia:**
  - Używanie **prepared (parametrized) query** - szkielet tych zapytań jest **wstępnie parsowany**/kompilowany na serwerze bazy danych, a dopiero później uzupełniany o parametry.
  - **Czyszczenie** (sanitizing) wprowadzonych **danych z niebezpiecznych znaków** (apostrofy, cudzysłowy, itp.). Znaki te można usuwać, albo **zastępować bezpiecznymi kombinacjami (escaping)**.

## 12.3 Cross-Site Scripting (XSS)

- W treści atakowanej strony wstawiany jest kod (najczęściej Javascript) wykonujący **niepożądaną akcję**.
- Kod ten jest **wykonywany** jest z **aktualnymi uprawnieniami** użytkownika, co umożliwia mu dostęp i ewentualną zmianę danych prywatnych użytkownika
- **Zabezpieczenia:**
  - Podobnie jak w przypadku SQL Injection, **sprawdzanie odbieranych przez użytkownika danych**. Usuwanie lub zamiana na zwykły tekst wszystkich instrukcji HTML, javascript, itp.
  - Używanie **cookies** z flagą **HttpOnly** (oraz **Domain**) co zabezpiecza przed dostępem do ciasteczek z poziomu skryptów. Skrypt nie może odczytać ciasteczka i przesłać do atakującego. **Nie zabezpiecza** nas to jednak przed wykonaniem złośliwej akcji **w ramach istniejącej sesji** na serwerze na który użytkownik jest zalogowany. Przeglądarka automatycznie dołącza ciasteczka do zapytań idących do serwera z pasującym Domain.

## 12.4 Session hijacking

- Atakujący **przechwytuje identyfikator** sesji lub jest go w stanie **przewidzieć i wygenerować**.
- Posiadając identyfikator może **połączyć się** z serwerem **w ramach sesji autoryzowanej** już przez jej właściciela i wykonać złośliwe działania.
- Fatalnym pomysłem jest np. nadawanie kolejnych numerów jako identyfikatory sesji!
- **Zabezpieczenia:**
  - **Generowanie niemożliwych do przewidzenia** identyfikatorów sesji
  - Używanie **HTTPS**
  - **Umożliwienie** użytkownikowi **wylogowania się** powodujące dezaktywacja sesji i jej identyfikatora.

## 12.5 Session Fixation

- Wiele serwisów przy pierwszym połączeniu użytkownika do serwisu rozpoczyna sesję (nadaje użytkownikowi session ID), a następnie, już w ramach tej sesji, pozwala mu się zalogować do serwisu.
- W takiej sytuacji czasami możliwy jest atak Session Fixation
  1. Atakujący łączy się do strony logowania.
  2. Serwer zwraca stronę logowania z identyfikatorem sesji
  3. **Atakujący przesyła ofierze link do strony logowania z ustawionym swoim identyfikatorem sesji**
  4. Ofiara loguje się na serwer używając przekazanego identyfikatora sesji
  5. Użytkownik ładuje stronę z kontem ofiary używając **wspólnego identyfikatora sesji**, która została już wcześniej autoryzowana przez ofiarę.
- **Zabezpieczenia:**
  - **Zablokowanie** przyjmowania Session ID w URL-u i parametrach POST. Zabezpiecza przed pierwszymi dwoma typami ataków.
  - Generowanie nowej sesji (**nowy Session ID**) po zalogowaniu użytkownika.

## 12.6 Cross-Site Request Forgery

- Definicja wg. OWASP: A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.
- Przykładowy scenariusz:
  1. Atakujący chce wykonać nieautoryzowany przelew z banku, który udostępnia taką funkcjonalność przez wywołania postaci
  2. Ofiara jest zalogowana na serwer (np. do banku)
  3. Atakujący podsuwa ofierze stronę, maila, itp. zawierającą link wykonujący złośliwą operację na serwerze (np. przelew bankowy).
  4. Ofiara **uruchamia** podsunęty **link w przeglądarce**, w której **aktywna jest sesja** na serwer. W przypadku drugiego linka, ofiara nie musi nawet uruchamiać linka - zostanie on automatycznie uruchomiony w celu pobrania "obrazka"
  5. Link zawierający żądanie dla serwera zostaje przesłany na serwer wraz ze wszystkimi ciasteczkami (w tym z identyfikatorem sesji).
- **Zabezpieczenia:**
  - **Synchronizer Tokens** - Każda **operacja zmieniająca stan** aplikacji **wymaga dodatkowego unikalnego jednorazowego tokenu** (CSRF token).
  - **Dodatkowa autoryzacja** przez użytkownika - dodatkowe hasło, CAPTCHA, token jednorazowy
  - Atrybut **SameSite** w ciasteczku - zapobiega wysłaniu ciasteczka jeśli **zapytanie nie pochodzi z tej samej strony**/ serwisu, z której pochodzi ciasteczko.
  - **CORS**

## 12.7 SOP i CORS

### 12.7.1 Same Origin Policy (SOP)

- Aby zabezpieczyć się przed CSRF zasób (skrypt) pobrany z danej lokalizacji (Origin) może czytać dane i wysyłać jedynie do innych **zasobów z tej samej lokalizacji**.
- Lokalizacja = scheme + host + port
- **Problemy z SOP** - czasami chcemy zezwolić na dostęp do naszych zasobów z zewnątrz. Na przykład udostępniając **publiczne API**, fonty, skrypty.

### 12.7.2 Cross-Origin Resource Sharing (CORS)

- **Każde żądanie** w trybie CORS powoduje dodanie przez przeglądarkę do zapytania nagłówka **Origin** z adresem serwisu z którego pochodzi strona wysyłająca żądanie.
- **Nagłówek** jest **ustawiany przez przeglądarkę** i kod użytkownika nie ma możliwości zmiany tego nagłówka.
- Serwer odpowiada z nagłówkiem
- Przeglądarka sprawdza, czy zwrócona wartość pozwala na udostępnienie zasobu i przekazuje go do kodu strony tylko wtedy, gdy jest to dozwolone.
- Istnieje też możliwość **wymuszenia dodatkowej autoryzacji**.