MEAN Stack - silnik szablonów Pug/Jade

Tworzenie serwisów Web 2.0

dr inż. Robert Perliński
rperlinski@icis.pcz.pl

Politechnika Częstochowska
Instytut Informatyki Teoretycznej i Stosowanej

23 marca 2019

Plan prezentacji

1. express-generator
2. Silniki szablonów
   - EJS
   - Handlebars
   - Hogan.js
3. Pug/Jade
   - Dziedziczenie szablonów
4. Źródła

express-generator

- narzędzie generujące szkielet aplikacji do frameworka express.
- instalacja: `npm install express-generator -g`

Użycie: `express [opcje] [katalog]`
Opcje:

```
    --version        wyświetla numer wersji narzędzia
-e, --ejs            wsparcie dla silnika szablonów ejs
    --pug            wsparcie dla silnika szablonów pug
    --hbs            wsparcie dla silnika szablonow handlebars
-H, --hogan          wsparcie dla silnika szablonow hogan.js
-v, --view <engine>  wsparcie dla silnika szablonów <engine>, dostępne
                     są (dust|ejs|hbs|hjs|jade|pug|twig|vash),
                     domyślnie ciągle jest jade
-c, --css <engine>   wsparcie dla silnika stylów <engine>, dostępne są
                     (less|stylus|compass|sass) - prekompilatory css,
                     domyślnie wygląd określamy w zwykłych plikach css
    --git            tworzy plik .gitignore z potrzebnymi wpisami
-f, --force          wymuszenie utworzenie projektu w niepustym katalogu
-h, --help           output usage information
```

Domyślny silnik szablonów to jade ale w przyszłości pewnie się to zmieni.

express-generator

Tworzenie struktury aplikacji: `express -v pug w03`

Następnie:

- instalacja zależności: `cd w03 && npm install`
- uruchimienie aplikacji: `DEBUG=w03:* npm start`
- sprawdzamy wersję silnika pug: `npm list pug`

  `pug@2.0.0-beta11` Jest już wersja 2.0.3

Jade jest w wersji 1.11.0.

Zmienne globalne dla całej aplikacji

Zmienne zawierające katalog i nazwę silinka szablonów wykorzystywanego w aplikacji:

- `view` - nazwa katalogu (String) albo tablicy katalogów (Array) gdzie mieszczą się szablony widoków. W przypadku tablicy katalogi są przeglądanie w kolejności występowania w tablicy. Domyślna wartość to: `process.cwd() + '/views'`
- `view engine` - nazwa określa domyślny silnik, który będzie użyty jeśli nie określono jawnie później.

```
// ustawienie silnika dla widoków
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');
app.enable('view cache');
// app.disable('view cache');
// app.set('view cache', true);
```

Silniki szablonów - EJS

EJS - <% Effective JavaScript %>

http://ejs.co/

Wybrane cechy:

- wyrażenia umieszczanie w znacznikach <% %>
- wstawianie zmiennych do szablonu:
  - modyfikacja znaczników HTML: <%= %>
  - brak modyfikacji znaczników, wyjście nieprzetworzone: <%- %>
- posiada tryb usuwania znaków nowej linii, kończy się znacznikiem -%>
- posiada tryb usuwania białych znaków, ograniczniki: <%_ _%>
- dwolne ograniczniki, np. użycie <? ?> zamiast <% %>
- wspiera dziedziczenie szablonów,
- wsparcie szablonów po stronie serwera i klienta (przeglądarka),
- statyczne buforowanie pośredniego JavaScriptu,
- statyczne buforowanie szablonów,
- kompiluje się z systemem widoków Expressa.

EJS - przykład

Przykładowy szablon z instrukcją warunkową:

```
<% if (user) { %>
  <h2><%= user.name %></h2>
<% } %>
```

Przykładowy szablon z pętlą forEach:

```
<ul>
  <% users.forEach(function(user){ %>
    <%- include('user/show', {user: user}); %>
  <% }); %>
</ul>
```

Silniki szablonów - Handlebars

Handlebars - `http://handlebarsjs.com/`

Cechy:

- w dużej mierze kompatybilny z Mustache (`http://mustache.github.io/`), zawiera dodatkową funkcjonalność
- wyrażenia umieszczamy w parze znaków `{{ ... }}`
- komentarze w szablonie: `{{!-- --}}` albo `{{! }}`
- kompilacja szablonu w przeglądarce albo wcześniejsza prekompilacja (szybsze działanie strony)
- zawiera przekształcanie znaczników HTML (dla bezpieczeństwa, `{{napis}}`), można je wyłączyć korzystając z zapisu `{{{napis}}}`,
- przy użyciu bloku rozszerzeń można dodać obsługę pętli, instrukcji warunkowych,...
- w szablonach można używać ścieżek - dostęp do zagnieżdżonych pól obiektu,
- pozwala na używanie literałów,
- pozwala na tworzenie szablonów cześciowych wielokrotnego użytku,
- zintegrowany z Express.

Handlebars - przykład

Dane:

```
var context = {
  title: "My First Blog Post!",
  author: {
    id: 47,
    name: "Yehuda Katz"
  },
  body: "My first post. Wheeeee!"
};
```

Szablon:

```
<div class="entry">
  <h1>{{title}}</h1>
  <h2>By {{author.name}}</h2>

  <div class="body">
    {{body}}
  </div>
</div>
```

Silniki szablonów - Hogan.js

`http://twitter.github.io/hogan.js/`

Wybrane cechy:

- kompilator dla silnika szablonów Mustache (`http://mustache.github.io/`),
- pozwala na tworzenie układów (layout) często występujących na stronie - coś jak szablon rodzica,
- wspiera podział szablonów na cześci (działa podobnie do include),
- wspiera buforowanie - aplikacja działa szybciej unikając niepotrzebnego renderowania,
- inaczej niż Handlebars, Hogan.js nie zawiera dodatkowych funkcjonalności, np. pętli czy instrukcji warunkowych,
- pozwala na tworzenie własnych filtrów,
- zintegrowany z Express.

Hogan.js - przykład

Przykładowy szablon:

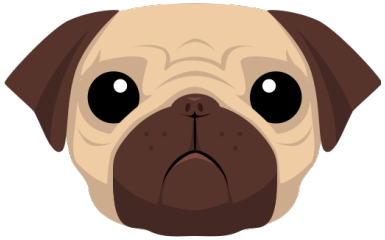
```
<div class="timeline">

  <!-- load more button -->
  <button>{{message}}</button>

  <!-- tweet object -->
  {{#tweets}}
    {{> tweet}}
  {{/tweets}}

</div>
```

Silniki szablonów - Pug (Jade)



Pug, wcześniej Jade (zastrzeżony znak towarowy)

`https://pugjs.org` Dokumentacja do Jade - `http://jade-lang.com/`

Cechy Pug/Jade:

- słowo Pug to mops :)
- silnik szablonów wysokiej wydajności,
- zaimplementowany w JavaScript dla Node.js i dla przeglądarek,
- pełna integracja z Express.js jako jeden z wspieranych silników szablonów,
- zespół pracował nad niekompatybilną wersją Jade 2.0.0 kiedy zmieniono nazwę na Pug,
- instalacja Pug oznacza używanie nowej wersji 2.0,
- nazwa `jade` jest ciągle wspierana w repozytoriach npm.

Silniki szablonów - Pug/Jade

Możliwości szablonów Jade/Pug:

- są wrażliwe na liczbę spacji określających wcięcia dla zagnieżdżonych znaczników, trzeba mieć edytor pokazujący znaki niedrukowane,
- pozwalają na różne rodzaje dołączania treści ze zmiennych czy obiektów do szablonu,
- zawierają: instrukcje warunkowe, instrukcję case, obsługę pętli (each, while),
- do szablonów można dołączać kod JavaScript,
- posiadają komentarze jednoliniowe i blokowe, przenoszące się lub nie do wynikowego kodu HTML,
- można dołączyć jeden szablon wewnątrz innego - include,
- wsparcie dla dziedziczenia poprzez bloki (`block`) albo zwyczajne dziedziczenie (`extends`),
- można tworzyć bloki z określoną nazwą i wielokrotnie ich używać.

Pug - kompilacja i renderowanie

Pug pozwala na dwie najważniejsze operacje:

- kompilacja - kompiluje kod szablonu Pug do funkcji JavaScript, która przyjmuje obiekt jako dane podstawiane do szablonu, wykorzystanie skompilowanej funkcji generuje kod HTML (renderowanie) z wartościami danych przekazanych w obiekcie,
- renderowanie - wykonuje kompilację i renderowanie w jako jeden proces - wpływa negatywnie na wydajność chyba, że jest włączona opcja `cache`.

```
//- template.pug
p Właściwielem jest #{imie}!

// index.js
const pug = require('pug');

router.get('/', function(req, res, next) {
  const compiledFun = pug.compileFile('views/template.pug');
  res.send(compiledFun({imie: 'Wojciech'}));
});
// <p>Właściwielem jest Wojciech!</p>

router.get('/', function(req, res, next) {
  res.render('template', { imie: 'Karol' });
});
// <p>Właściwielem jest Karol!</p>
```

Pug - kompilacja i renderowanie

index.pug

```
block content
  h1= tytul
  p Witaj w #{tytul}
```

pugtest.js

```
var express = require('express');
var pug = require('pug');
var router = express.Router();

const funkcjaDoRenderowania = pug.compileFile('views/index.pug');
const funkcjaDoRenderowania2 = pug.compile('p Witaj w #{tytul}');
console.log( funkcjaDoRenderowania2({ tytul: 'Inny przykład' }) );

router.get('/just-render', function(req, res, next) {
  res.send(funkcjaDoRenderowania({tytul: 'Moja aplikacja-tylko renderowanie'}));
});

router.get('/compile-render', function(req, res, next) {
  res.render('index', {tytul: 'Moja aplikacja-kompilacja i renderowanie'});
});
module.exports = router;
```

Znaczniki

- Domyślnie tekst na początku linii, albo występujący tylko po białych znakach, tworzy znacznik
- Znaczniki samozamykające się są znane
- Można dodać dowolny znacznik XML
- Białe znaki są usuwane z początku i końca znaczników - mamy kontrolę nad tym, czy się one stykają czy nie

```
p
div
img
h1
input
hr
book
br
foo/
foo(bar='baz')/
```

```
<p></p>
<div></div>
<img/>
<h1></h1>
<input/>
<hr/>
<book></book>
<br/>
<foo/>
<foo bar="baz"/>
```

```
<p></p><div></div><img><h1></h1><input><hr><book></book><br><foo/><foo bar="baz"/>
```

Cztery sposoby na wprowadzanie zwykłego tekstu

- Pug dostarcza cztery sposoby na wprowadzanie zwykłego tekstu, który przechodzi niemal nieprzetwarzany do kodu HTML.
- Znaki HTML w tekście nie są zastępowane - można ręcznie wprowadzać znaczniki.
- Jest prowadzona interpolacja zmiennych do tekstu.

Cztery sposoby:

- w linii znacznika
- bezpośredni kod HTML
- | - pionowy separator dla większych obszarów tekstu
- . - kropka przy większych blokach tekstu

Tekst w linii znacznika i przy separatorze |

- Po nazwie znacznika i pojedynczej spacji umieszczamy tekst będący jego zawartością
- Większe obszary tekstu umieszczamy w kolejnych liniach po wcięciu i znaku | (pionowa kreska)
- Każda dodatkowa spacja (powyżej pierwszej) po znaku | oznacza dodatkową spację w kodzie HTML

```
p Hello, World!
p
  | Hello again
  | and again
  | and again
  |  and ...
h2 some text
  | on the same line
  |  and on the next line
```

```
<p>Hello, World!</p>
<p>Hello again
and again
and again
 and ...</p>
<h2>some texton the same line
 and on the next line</h2>
```

Tekst w większych blokach po .

- Nadaje się, kiedy potrzebujemy większych bloków tekstu wewnątrz znacznika
- Przykłady: kod JavaScript albo CSS w znacznikach `script` i `style`
- Dodajemy . po nazwie znacznika albo nawiasie zamykającym jeśli są atrybuty, między znacznikiem a kropką nie ma przerwy
- Blok tekstu musi z dodatkowym wcięciem

```
p Hello, World!
script.
  if (usingPug)
    console.log('super!')
  else
    console.log('używaj pug')
h2
  strong Important info...
  .
    some text
    on the same line
     and on the next line
```

```
<p>Hello, World!</p>
<script>if (usingPug)
  console.log('super!')
else
  console.log('używaj pug')</script>
<h2>
<strong>Important info...</strong>some text
on the same line
 and on the next line</h2>
```

Bezpośredni kod HTML

- Całe linie są traktowane jako tekst, jeśli zaczynają się od znaku mniejszości <
- Czasem takie rozwiązanie może być użyteczne

```
<body>
<p>Hello, World!
</p>
script.
  if (usingPug)
    console.log('super!')
  else
    console.log('używaj pug')
h2
  strong Important info...
  .
    some text
    on the same line
     and on the next line
</body>
```

```
<body>
<p>Hello, World!</p>
<script>if (usingPug)
  console.log('super!')
else
  console.log('używaj pug')</script>
<h2>
<strong>Important info...</strong>some text
on the same line
 and on the next line</h2></body>
```

Zagnieżdżanie znaczników

- Zagnieżdżone znacznki uzyskujemy dodając je od nowej linii po wcięciu

```
div
  p Cześć Stefan!
table
  tr
    th imie
    th nazwisko
  tr
    td Jan
    td Kowalski
  tr
    td Zofia
    td Nowak
span
  h2 Rozdział 2
  p Kolejny akapit
```

```
<div>
  <p>Cześć Stefan!</p>
</div>
<table>
  <tr>
    <th>imie</th>
    <th>nazwisko</th>
  </tr>
  <tr>
    <td>Jan</td>
    <td>Kowalski</td>
  </tr>
  <tr>
    <td>Zofia</td>
    <td>Nowak</td>
  </tr>
</table>
<span>
  <h2>Rozdział 2</h2>
  <p>Kolejny akapit</p>
</span>
```

Rozwinięcie bloku

- Rozwinięcie bloku pozwala dodać zagnieżdżanie zanczniki wykorzystując tylko jedną linię
- Zagnieżdżamy dodatkowy znacznki poprzedzając go dwukropkiem : i spacją
- Można tak zapisać więcej znaczników ale tylko ostatni może mieć zawartość

```
div
  p Hello


div: p Hello


div: p: span Hello




div: p : span Hello


div: p Treść... : span Hello
```

```
<div>
  <p>Hello</p>
</div>
<div>
  <p>Hello</p>
</div>
<div>
  <p>
    <span>Hello</span>
  </p>
</div>
<div>
  <p>: span Hello</p>
</div>
<div>
  <p>Treść... : span Hello</p>
</div>
```

Komentarze

- Komentarze tworzymy poprzez podwójny slash czyli // albo //-
- Możliwe są też warunki w komentarzach
- Pug nie obsługuje rozpoznawania wnętrzy komentarza (znaczniki i inne komentarze) a Jade tak

```
// pierwszy komentarz
  na dwie linie
  bardzo długie linie...
//drugi komentarz
  h1 nagłówek w komentarzu...
  // komentarz wewnętrzny
  p paragraf w komentarzu...
hr
//if lt IE 8
  script(src="/ie-sucks.js")
p Witaj
  // komentarz w znaczniku
//- komentrzach tylko w szablonie
```

```
#### Pug
<!-- pierwszy komentarzna dwie linie
bardzo długie linie...-->
<!--drugi komentarzh1 nagłówek w komentarzu...
// komentarz wewnętrzny
p paragraf w komentarzu...-->
<hr/>
<!--if lt IE 8script(src="/ie-sucks.js")-->
<p>Witaj
  <!-- komentarz w znaczniku-->
</p>

#### Jade
...<!--drugi komentarz
<h1>nagłówek w komentarzu...</h1>
<!-- komentarz wewnętrzny-->
<p>paragraf w komentarzu...</p>-->
<hr/><!--[if lt IE 8]>
<script src="/ie-sucks.js"></script><![endif]-->
<p>Witaj
  <!-- komentarz w znaczniku-->
</p>
```

Atrybuty

- Atrybuty zaczynają się i kończą nawiasem okrągłym
- Wiele atrybutów opcjonalnie oddzielamy przecinkiem lub znakiem nowej linii
- Występowanie białych znaków nie ma znaczenia
- Nazwy atrybutów mogą posiadać dwukropek :
- Można używać wyrażeń języka JavaScript

```
input(type="text", name="log")
input(type="text" name="log")
input(type="text",
  name="log")
input(type="text"
  name="log")
element(xmlns:names="engine")
- var mobile = false
- var name = 'hi'
input(type="text"
  name=mobile?"login":name)
```

```
<input type="text" name="log"/>
<input type="text" name="log"/>
<input type="text" name="log"/>

<input type="text" name="log"/>

<element xmlns:names="engine"></element>

<input type="text" name="hi"/>
```

div, id, class

- Znaki # (hash) i . (kropka) pozwalają tworzyć atrybuty id i class znacznika
- Znaki . można wykorzystywać wielokrotnie i łączyć je z znakiem #
- Znacznik div jest bardzo powszechny w HTML - towrzony automatycznie dla znaków # i .
- Składnia `&attributes` pozwala na zamianę obiektu JavaScript w zbiór atrybutów

```
p#header
p.info
h2.big.red.notice
div.notice#menu.large
.once
#footer
.red#content.mid
p #header
p# header
-
  var att =
  {class: "big", name: "extra"}
#main(class="red")&attributes(att)
```

```
<p id="header"></p>
<p class="info"></p>
<h2 class="big red notice"></h2>
<div class="notice large" id="menu"></div>
<div class="once"></div>
<div id="footer"></div>
<div class="red mid" id="content"></div>
<p>#header</p>
<p># header</p>   <!-- BŁĄD w Node.js -->

<div class="red big" id="main"
     name="extra"></div>
```

Atrybuty - wartości zmiennych, zastępowanie znaczników

- Chcąc umieścić wartość zmiennej w atrybucie po prostu podajemy jej nazwę
- Operator + (plus) oznacza konkatenację
- Domyślnie wszystkie znaczniki HTML są zastępowane, można to wyłączyć używając !=

```
- var url="nba.com"
a(href="http://"+url)

// Stary zapis, nie działa w Pug
a(href="http://#{url}")

div(escaped="<code>")
div(unescaped!="<code>")
```

```
<a href="http://nba.com"></a>

<a href="http://#{url}"></a>

<div escaped="&lt;code&gt;"></div>
<div unescaped="<code>"></div>
```

Typ dokumentu

Do wprowadzenia typu dokumentu mamy odpowiednie skróty:

```
doctype
doctype html

doctype xml

doctype basic

doctype strict

doctype 1.1

doctype mobile

doctype frameset

doctype transitional

doctype 5

!!! 5
```

```
<!DOCTYPE html>


<?xml version="1.0" encoding="utf-8" ?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN"
     "http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.2//EN"
     "http://www.openmobilealliance.org/tech/DTD/xhtml-mobile12.dtd">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!DOCTYPE 5>     // w Jade było <!DOCTYPE html>

<!DOCTYPE html>  // Tylko w Jade
```

Przypisanie wartości

- Przypisanie wartości do zmiennej wymaga znaku równości =
- Przekazanie wartości zmiennej do znacznika wymaga znaku równości zaraz po znaczniku, w przeciwnym razie będzie potraktowana jako tekst znacznika

```
- var imie = "Jarek"
nazw='Wojtek'
nazw= 'Wojtek'
nazw ='Wojtek'
nazw = 'Wojtek'

- var num = 123

p=imie
p= imie
p = imie
p =imie
p=num
```

```
<nazw>Wojtek</nazw>
<nazw>Wojtek</nazw>
<nazw>='Wojtek'</nazw>
<nazw>= 'Wojtek'</nazw>



<p>Jarek</p>
<p>Jarek</p>
<p>= imie</p>
<p>=imie</p>
<p>123</p>
```

Kod JavaScript

Pug pozwala dodawać trzy rodzaje kodu JavaScrit do szabonu

- nie przetwarzany do szablonu (unbuffered), zaczyna się od - (minus)
- przetwarzany do szablonu (buffered), zaczyna się od = (znak równości
- przetwarzany do szablonu ale znaczniki HTML nie są przetwarzane, zaczyna się od !=
- wszystkie trzy mogą być w jednej linii albo w bloku

```
p 'Jola ma'+(2+3)+' lat'
p= 'Jola ma '+(2+3)+' lat'
- for (var x = 0; x < 3; x++)
  li item

p
  = 'Kod z zastępowanymi <znacznikami>!'
  = 'Wykorzystano zapis <blokowy>'
p!= 'Brak przetwarzania <znacznikow>'
p != 'Brak przetwarzania <znacznikow>'
```

```
<p>'Jola ma'+(2+3)+' lat'</p>
<p>Jola ma 5 lat</p>
<li>item</li>
<li>item</li>
<li>item</li>

<p>Kod z zastępowanymi &lt;znacznikami&gt;!
Wykorzystano zapis &lt;blokowy&gt;</p>
<p>Brak przetwarzania <znacznikow></p>
<p>!= 'Brak przetwarzania <znacznikow>'</p>
```

Przetwarzanie zmiennych

Pug pozwala przetwarzać zmienne do szablonu na kilka sposobów:

- Cała linia przetwarzana jako kod JavaScript (znak =)
- Przetwarzanie kodu JS wewnątrz #{...} - zastępowanie znaczników HTML
- Przetwarzanie kodu JS wewnątrz !{...} - znaczniki HTML bez zmian

```
- var tytul = "Książka twoim przyjacielem";
- var autor = "John Doe";
- var wielka = "<span>ucieczka!</span>";

h1= tytul
p Napisana z radością przez #{autor}

p Bezpieczne przetwarzanie: Wielka #{wielka}


p Bezpieczne przetwarzanie:
  Wielka #{wielka.toUpperCase()}


p Niebezpieczne przetwarzanie: Wielka !{wielka}
```

```
<h1>Książka twoim przyjacielem</h1>
<p>Napisana z radością przez John Doe</p>
<p>Bezpieczne przetwarzanie:
  Wielka &lt;span&gt;ucieczka!&lt;/span&gt;
</p>
<p>Bezpieczne przetwarzanie:
  Wielka &lt;SPAN&gt;UCIECZKA!&lt;/SPAN&gt;
</p>
<p>Niebezpieczne przetwarzanie:
  Wielka <span>ucieczka!</span>
</p>
```

Wynik w przeglądarce:

Książka twoim przyjacielem

Napisana z radością przez John Doe

Bezpieczne przetwarzanie: Wielka ucieczka!

Bezpieczne przetwarzanie: Wielka <SPAN>UCIECZKA!</SPAN>

Niebezpieczne przetwarzanie: Wielka ucieczka!

Pętle i iteracje I

- Pętle tworzymy przy użycia słowa `each` albo `for`
- Podając drugi parametr można odczytać wartość indeksu kolejnych elementów

```
ul
  each item in ['Ala', 'Jan', 'Ola']
    li= item


for zmienna in ['słoń', 'pies', 'żyrafa']
  p= zmienna



for kraj, i in ['PL', 'GB', 'LT']
  div(index=i)= kraj
    p= "index wynosi " + i




for krajj, ii in ['PL', 'GB', 'LT']
  span(index=ii)= krajj
    p= "index wynosi " + ii + " a kraj to " + krajj
```

```
<ul>
  <li>Ala</li>
  <li>Jan</li>
  <li>Ola</li>
</ul>
<p>słoń</p>
<p>pies</p>
<p>żyrafa</p>
<div index="0">PL
  <p>index wynosi 0</p>
</div>
<div index="1">GB
  <p>index wynosi 1</p>
</div>
<div index="2">LT
  <p>index wynosi 2</p>
</div>
<span index="0">PL
  <p>index wynosi 0 a kraj to PL</p>
</span>
<span index="1">GB
  <p>index wynosi 1 a kraj to GB</p>
</span>
<span index="2">LT
  <p>index wynosi 2 a kraj to LT</p>
</span>
```

Pętle i iteracje II

- Można również iterować po kluczach w obiekcie
- Możliwe jest umieszczenie klauzuli `else` w pętli - jeśli tablica czy obiekt będą puste, to wykona się instrukcja alternatywna

```
ul
  each os, iii in {1:'Jan', 2:'Ela', 'james':'James'}
    li= iii + ":" + os
  else
    li Brak danych w obiekcie
```

```
<ul>
  <li>1:Jan</li>
  <li>2:Ela</li>
  <li>james:James</li>
</ul>

...
  <li>Brak danych w obiekcie</li>
```

- Generalnie iterujemy po tablicach czy obiektach - zmienne języka JavaScript
- Dostępna jest również pętla `while`

```
- var n = 10;
ul
  while n >= 7
    li= n--
```

```
<ul>
  <li>10</li>
  <li>9</li>
  <li>8</li>
  <li>7</li>
</ul>
```

Instrukcje warunkowe

- Instrukcje warunkowe to proste wyrażenia `if/else`
- Tutaj również nie potrzeba nawiasów czy klamer

```
- var sem = "3 semestr"
if sem == "3 semestr"
  p Pamiętaj o magisterce!
  // To już jest ten czas
else
  p Jest jeszcze czas

- var deszcz = true
unless deszcz == true
  p Pójdę na spacer

- var deszcz = false
unless deszcz
  p Naprawdę pójdę na spacer
```

```
<p>Pamiętaj o magisterce!</p>
<!-- To już jest ten czas-->

<p>Naprawdę pójdę na spacer</p>
```

Dziedziczenie szablonów

- Dziedziczenie szablonów pozwala wykorzystać stworzoną strukturę w wielu miejscach
- W szablonie rodzica definiujemy poleceniem `block` obszary, które szablon dzicka może zastąpić
- Szablon dziecka dziedziczy szablon rodzica korzystając ze słowa `extends` (pierwsza linia w pliku!)

```
### layout.pug
// layout.pug, sekcja body
h1 Informacje o technologiach webowych
block content
  p Przyszła zawartość poszczególnych podstron


### index.pug
extends layout
block content
  // index.pug, sekcja body
  section Do technologi webowych zaliczamy:
    ul
      li przeglądarka internetowa
      li HTML, CSS
      li Frameworki internetowe, np. Angular
      li języki programowania np. JavaScript, Go
      li protokoły, np. HTTP czy REST
      li Web API i foramty danych, np. JSON, XML
```

```
### renderowanie szablonu layout.pug
<!-- layout.pug, sekcja body-->
<h1>Informacje o technologiach webowych</h1>
<p>Przyszła zawartość poszczególnych podstron</p>


### renderowanie szablonu index.pug
<!-- layout.pug, sekcja body-->
<h1>Informacje o technologiach webowych</h1>
<!-- index.pug, sekcja body-->
<section>Do technologi webowych zaliczamy:
<ul>
  <li>przeglądarka internetowa</li>
  <li>HTML, CSS</li>
  <li>Frameworki internetowe, np. Angular</li>
  <li>języki programowania np. JavaScript, Go</li>
  <li>protokoły, np. HTTP czy REST</li>
  <li>Web API i foramty danych, np. JSON, XML</li>
</ul>
</section>
```

Dołączanie szablonów - include

- Dziedziczenie do dobry mechanizm ale nie zawsze najbardziej porządany
- Czasem lepiej jest po prostu dołączyć fragment szablonu w różnych miejscach strony
- Słowo `include` pozwala dołączyć fragment szablonu zawarty w innym pliku

```
### layout.pug
// layout.pug, sekcja body
h1 Informacje o technologiach webowych
block content
  p Przyszła zawartość poszczególnych podstron
include footer


### footer.pug
footer
  div Copyright &copy; 2019 by Your App Team
```

```
### renderowanie szablonu layout.pug
<!-- layout.pug, sekcja body-->
<h1>Informacje o technologiach webowych</h1>
<p>Przyszła zawartość poszczególnych podstron</p>
<footer>  <div>Copyright &copy; 2019
   by Your App Team</div></footer>

### renderowanie szablonu index.pug
<!-- layout.pug, sekcja body-->
<h1>Informacje o technologiach webowych</h1>
<!-- index.pug, sekcja body-->
<section>Do technologi webowych zaliczamy:
<ul>
  <li>przeglądarka internetowa</li>
  <li>HTML, CSS</li>
  <li>Frameworki internetowe, np. Angular</li>
  <li>języki programowania np. JavaScript, Go</li>
  <li>protokoły, np. HTTP czy REST</li>
  <li>Web API i foramty danych, np. JSON, XML</li>
</ul>
</section>
<footer>  <div>Copyright &copy; 2019
  by Your App Team</div></footer>
```

Mixins w Pug I

- Domieszki (ang. *mixins*) pozwalają tworzyć bloki kodu wielokrotnego użycia
- Są kompilowane do funkcji, które mogą przyjmować argumenty

```
//- Deklaracja
mixin list
  ul
    li Ala
    li Ela
    li Ola
//- Użycie
+list
+list

mixin mebel(nazwa)
  li.mebel= nazwa
ul
  +mebel('krzesło')
  +mebel('stół')
  +mebel('fotel')
```

```
<ul>
  <li>Ala</li>
  <li>Ela</li>
  <li>Ola</li>
</ul>
<ul>
  <li>Ala</li>
  <li>Ela</li>
  <li>Ola</li>
</ul>

<ul>
  <li class="mebel">krzesło</li>
  <li class="mebel">stół</li>
  <li class="mebel">fotel</li>
</ul>
```

Mixins w Pug II

- W domieszkach można używać bloków Pug jako zawartość

```
mixin article(tytul)
  .article
    .article-wrapper
      h1= tytul
      if block
        block
      else
        p Nie dostarczono treści

+article('Witaj świecie')

+article('Witaj świecie')
  p to jest mój
  p wspaniały artykuł
```

```
<div class="article">
  <div class="article-wrapper">
  <h1>Witaj świecie</h1>
    <p>Nie dostarczono treści</p>
  </div>
</div>

<div class="article">
  <div class="article-wrapper">
  <h1>Witaj świecie</h1>
    <p>to jest mój</p>
    <p>wspaniały artykuł</p>
  </div>
</div>
```

Źródła

W wykładzie wykorzystano materiały:

- http://expressjs.com/en/starter/generator.html
- https://expressjs.com/en/guide/using-template-engines.html
- http://ejs.co/
- http://handlebarsjs.com/
- http://twitter.github.io/hogan.js/
- https://pugjs.org
- http://jade-lang.com/