# Full Stack JavaScript

Poznaj technologie Backbone.js, Node.js i MongoDB

—

Wydanie II

—

Azat Mardan

Helion

Apress

Tytuł oryginału: Full Stack JavaScript: Learn Backbone.js, Node.js, and MongoDB, Second Edition

Tłumaczenie: Agnieszka Górczyńska

ISBN: 978-83-283-5750-1

Original edition copyright © 2018 by Azat Mardan
All rights reserved.

Polish edition copyright © 2020 by Helion SA
All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Helion SA dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Helion SA nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
*ftp://ftp.helion.pl/przyklady/fulljs.zip*

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
*http://helion.pl/user/opinie/fulljs*
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

# Spis treści

**O autorze** ..... **9**

**Podziękowania** ..... **11**

**Wstęp** ..... **13**

**Wprowadzenie** ..... **15**

**Część I Szybki start** ..... **21**

**Rozdział 1 Podstawy** ..... **23**

Definicje front-endu ..... 23

Cykl żądania internetowego ..... 24

Programowanie na urządzeniach mobilnych ..... 25

Język HTML ..... 26

Kaskadowe arkusze stylów ..... 28

JavaScript ..... 29

Metody zwinne ..... 37

Scrum ..... 37

Programowanie sterowane testami ..... 38

Ciągłe wdrażanie i integracja ..... 38

Programowanie w parach ..... 38

Definicje back-endu ..... 39

Node.js ..... 39

NoSQL i MongoDB ..... 40

Przetwarzanie w chmurze ..... 40

Żądania HTTP i odpowiedzi na nie ..... 41

API RESTful ..... 41

Podsumowanie ..... 42

Kup książkę Poleć książkę

**Rozdział 2 Przygotowania ... 43**
Konfiguracja lokalna ... 43
Katalog roboczy ... 44
Przeglądarka WWW ... 45
Środowisko IDE i edytor tekstu ... 46
System kontroli wersji ... 47
Lokalne serwery HTTP ... 47
Baza danych — MongoDB ... 48
Wymagane komponenty ... 50
Instalowanie Node.js ... 51
Biblioteki JavaScriptu dla przeglądarki WWW ... 52
Aplikacja Less ... 52
Konfiguracja chmury ... 52
Klucze SSH ... 52
GitHub ... 53
Microsoft Azure ... 54
Heroku ... 56
Podsumowanie ... 56

**Część II Prototypowanie front-endu ... 57**

**Rozdział 3 Pobieranie danych z back-endu za pomocą biblioteki jQuery i platformy Parse ... 59**
Definicje ... 60
Format JSON ... 60
AJAX ... 61
Wywołania między domenami ... 61
Funkcje jQuery ... 62
Bootstrap ... 63
Less ... 65
Zmienna w języku Less ... 66
Domieszka w języku Less ... 66
Operacja w języku Less ... 67
Przykład użycia zewnętrznego API (OpenWeatherMap) i biblioteki jQuery ... 68
Parse ... 75
Aplikacja Message Board i ogólne omówienie Parse ... 81
Aplikacja Message Board wraz z Parse — wersja oparta na API REST i jQuery ... 82
Przekazanie aplikacji do serwisu GitHub ... 88
Wdrożenie do Microsoft Azure ... 89
Wdrażanie aplikacji Weather do Heroku ... 89
Uaktualnianie i usuwanie wiadomości ... 91
Podsumowanie ... 92



Kup książkę Poleć książkę

**Rozdział 4 Wprowadzenie do Backbone.js ... 93**
Konfigurowanie Backbone.js — aplikacja zupełnie od zera ... 94
Zależności Backbone.js ... 94
Praca z kolekcją Backbone.js ... 96
Dołączanie zdarzenia Backbone.js ... 100
Widoki i podwidoki Backbone.js z użyciem Underscore.js ... 104
Refaktoryzacja kodu Backbone.js ... 111
AMD i Require.js dla programowania z użyciem Backbone.js ... 115
Require.js w środowisku produkcyjnym Backbone.js ... 120
Bardzo prosty zestaw startowy Backbone.js ... 125
Podsumowanie ... 125

**Rozdział 5 Backbone.js i Parse ... 127**
Aplikacja Message Board i Parse
— wersja oparta na SDK JavaScript i Backbone.js ... 127
Dalsza rozbudowa aplikacji Message Board ... 138
Podsumowanie ... 139

**Część III Prototypowanie back-endu ... 141**

**Rozdział 6 Wprowadzenie do Node.js ... 143**
Utworzenie aplikacji typu „Witaj, świecie!" w Node.js ... 143
Podstawowe moduły Node.js ... 145
http ... 145
util ... 145
querystring ... 146
url ... 146
fs ... 146
Menedżer pakietów Node.js ... 146
Wdrożenie w PaaS aplikacji typu „Witaj, świecie!" ... 148
Wdrażanie w Microsoft Azure ... 148
Wdrażanie w Heroku ... 149
Message Board w Node.js
— wersja wykorzystująca magazyn danych w pamięci ... 150
Testy jednostkowe w Node.js ... 150
Podsumowanie ... 156

**Rozdział 7 Wprowadzenie do MongoDB ... 157**
Powłoka MongoDB ... 157
Identyfikator obiektu w formacie BSON ... 159
Natywny sterownik MongoDB ... 160
MongoDB i Heroku — MongoLab ... 161
Aplikacja Message Board — wersja wykorzystująca MongoDB ... 164
Podsumowanie ... 167



Kup książkę Poleć książkę

**Rozdział 8 Połączenie front-endu i back-endu ..... 169**
Stosowanie techniki CORS podczas wdrażania aplikacji ..... 170
Interfejs użytkownika aplikacji Message Board ..... 171
API aplikacji Message Board ..... 175
Wdrażanie w Heroku ..... 178
Wdrażanie serwera w tej samej domenie ..... 179
Wdrażanie w Amazon Web Services ..... 182
Podsumowanie ..... 185

**Rozdział 9 Podsumowanie ..... 187**

**Dodatek A Zasoby pomocne w dalszej nauce ..... 189**
Zasoby dotyczące JavaScriptu i Node.js ..... 189
Dobre książki o języku JavaScript ..... 189
Dobre książki o Node.js ..... 190
Kursy interaktywne ..... 190
Inne książki i blogi ..... 190



Kup książkę Poleć książkę

ROZDZIAŁ 1

# Podstawy

„Sądzę, że każdy powinien nauczyć się programować, ponieważ to uczy myślenia. Postrzegam informatykę jako pewien rodzaj sztuki, której każdy powinien się nauczyć”.

— Steve Jobs

W tym rozdziale:

- Ogólne przedstawienie składni HTML, CSS i JavaScriptu
- Krótkie wprowadzenie do metod zwinnych
- Prezentacja zalet obliczeń w chmurze, Node.js i MongoDB
- Omówienie koncepcji żądania i odpowiedzi w protokole HTTP oraz API RESTful

Zacznę od przedstawienia podstawowych koncepcji, a dopiero później przejdę dalej. Jeżeli jesteś doświadczonym programistą internetowym, możesz pominąć ten rozdział. Natomiast jeśli dopiero zaczynasz karierę programisty internetowego, zwróć szczególną uwagę na zaprezentowany tutaj materiał. Dlaczego? Być może wcześniej spotkałeś się już z niektórymi użytymi tu pojęciami, ale zastanawiasz się, co tak naprawdę znaczą. Tutaj znajdziesz ich wyjaśnienie. Ponadto w tym rozdziale zamieściłem wprowadzenie do API RESTful przygotowane pod kątem początkujących. Technologia REST jest stosowana praktycznie w każdej nowoczesnej architekturze internetowej i będziesz z niej dość często korzystać w książce. Jest jeszcze jeden powód: podczas spotkania z kolegami i szefem będziesz mógł zabłysnąć, posługując się akronimami z dziedziny programowania internetowego.

## Definicje front-endu

*Front-end* to pojęcie pochodzące ze świata przeglądarek WWW. Przeglądarka WWW jest nazywana klientem, ponieważ w trakcie pracy w sieci stosowana jest komunikacja klient – serwer. Użytkownik pracuje z klientem i wykonuje żądania do serwera, który z kolei udziela na nie odpowiedzi. Dlatego pojęcie front-end odnosi się do przeglądarki WWW lub innej aplikacji klienta. Klientem może być również aplikacja mobilna.

Kup książkę Poleć książkę

Obecnie niezwykle rzadko, wśród starej daty architektów Javy, pojęcie front-end jest używane do definiowania aplikacji *serwera*. To jest bardzo dezorientujące. Jedyne wytłumaczenie, jakie jestem w stanie znaleźć dla takiego sposobu użycia pojęcia front-end, wiąże się z tym, że aplikacje serwera mają do czynienia z żądaniami przeglądarek WWW jako pierwsze, jeszcze przed innymi aplikacjami serwerowymi. Ewentualnie, w zależności od kontekstu, te aplikacje serwera działają w charakterze statycznych serwerów WWW dla aplikacji w przeglądarce WWW. Aby w tej książce wszystko było jasne i dokładnie zdefiniowane, przyjąłem założenie, że **gdy wspominam o front-endzie, to mam na myśli aplikacje przeglądarki WWW i ich kod źródłowy**.

Programowanie front-endu oznacza użycie wielu różnych technologii. Każda z nich osobno nie należy do zbyt skomplikowanych, choć ich ilość może być przytłaczająca dla początkującego. Na przykład stosowane technologie obejmują m.in.: Cascading Style Sheets (CSS), Hypertext Markup Language (HTML), Extensible Markup Language (XML), JavaScript (JS), JavaScript Object Notation (JSON), Uniform Resource Identifier (URI), Hypertext Transfer Protocol (HTTP), Secure Sockets Layer (SSL), Transport Layer Security (TLS), Transmission Control Protocol/Internet Protocol (TCP/IP), Internet Relay Chat (IRC), Remote Procedure Call (RPC), GraphQL, ES i wiele innych. (Moja kolejna książka będzie zatytułowana *Poruszanie się po morzu akronimów*).

Poza wymienionymi technologiami działającymi na niskim poziomie istnieje jeszcze pewna liczba frameworków, narzędzi i bibliotek: React, jQuery, Backbone.js, Angular.js, Webpack, Grunt itd. Nie należy mylić frameworków front-endu z frameworkami back-endu: te pierwsze działają w przeglądarce WWW, a drugie w serwerze.

Opracowanie aplikacji internetowej wymaga użycia przez programistę wielu elementów. Spójrz na listę komponentów niezbędnych programiście internetowemu:

1. HTML lub szablony kompilowane na postać kodu HTML;
2. arkusze stylów nadające ładny wygląd kodowi HTML;
3. skrypty JavaScriptu zapewniające interaktywność lub pewną logikę biznesową aplikacji działającej w przeglądarce WWW;
4. hosting, np. AWS, Apache, Heroku;
5. skrypty kompilacji umożliwiające przygotowanie kodu, zarządzanie zależnościami i wykonywanie praktycznie wszelkich innych zadań;
6. logika odpowiedzialna za nawiązanie połączenia z serwerem, zwykle za pomocą żądań XHR i API RESTful.

Teraz już wiesz, na czym polega praca programisty front-endu. Doskonałą nagrodą za opanowanie tych narzędzi jest możliwość wyrażenia własnej kreatywności poprzez tworzenie ładnych i użytecznych aplikacji internetowych.

Jednak zanim przystąpisz do pracy nad aplikacjami, poznaj cykl żądania internetowego.

## Cykl żądania internetowego

Ta sekcja jest szczególnie ważna dla początkujących programistów internetowych. Cała sieć WWW (ang. *World Wide Web*), czyli po prostu internet, opiera się na komunikacji między klientami i serwerami. Ta komunikacja jest prowadzona przez wykonywanie żądań i otrzymywanie odpowiedzi. Typowa przeglądarka WWW (najpopularniejszy klient internetowy) wykonuje żądania do serwera. W tle serwer wykonuje własne żądania do innych serwerów, te żądania są podobne do wykonywanych przez przeglądarkę WWW. Językiem używanym podczas obsługi tych żądań i odpowiedzi na nie jest HTTP(S). Pozwól, że znacznie dokładniej wyjaśnię proces wykonywania żądania przez przeglądarkę WWW.



Kup książkę Poleć książkę

W trakcie tego procesu wykonywanych jest kilka wymienionych tutaj kroków:

1. Użytkownik wpisuje adres URL lub klika łącze w przeglądarce WWW określanej mianem klienta.
2. Przeglądarka WWW wykonuje żądanie HTTP do serwera.
3. Serwer przetwarza żądanie i jeśli w ciągu tekstowym połączenia bądź w żądaniu znajdują się jakiekolwiek parametry, zostają uwzględnione w żądaniu.
4. Serwer uaktualnia, pobiera i przekształca dane w bazie danych.
5. Serwer udziela odpowiedzi na żądanie HTTP, dostarczając dane w formatach HTML, JSON i innych.
6. Przeglądarka WWW otrzymuje odpowiedź HTTP.
7. Przeglądarka WWW generuje tę odpowiedź HTTP użytkownikowi, stosując format HTML lub inny, np. JPEG, XML, JSON.

Aplikacje mobilne działają w dokładnie taki sam sposób jak zwykłe witryny internetowe, ale zamiast przeglądarki WWW używana jest aplikacja natywna. Aplikacje mobilne (natywne lub HTML5) to po prostu jeszcze inny rodzaj klienta. Inne mniejsze różnice między aplikacjami mobilnymi i internetowymi wiążą się z ograniczeniami w transferze danych ze względu na restrykcje nakładane przez operatorów telekomunikacyjnych. Różnice wynikają także z ekranów o mniejszej wielkości w urządzeniach mobilnych oraz z bardziej efektywniejszego użycia lokalnej pamięci masowej. Być może jesteś programistą internetowym, który chciałby spróbować swoich sił w dziedzinie aplikacji internetowych dla urządzeń mobilnych. Dzięki technologiom JavaScript i HTML5 to jest możliwe. Dlatego warto nieco bliżej zapoznać się z tematem programowania na urządzeniach mobilnych.

## Programowanie na urządzeniach mobilnych

Czy platformy mobilne zdominują platformy biurowe i internetowe? Być może, ale zbliżamy się do 2020 roku i ruch sieciowy nadal ma około 50% udziału w rynku platform. Co więcej, programowanie na platformach mobilnych wciąż jest trudne i odbywa się znacznie wolniej w porównaniu do platform internetowych. To nie stanowi problemu, jeśli zajmujesz się natywnym programowaniem na platformie mobilnej, ale w większości przypadków tak nie jest. W porównaniu z internetem odczuwalny jest wyraźny brak talentów. Sama luka coraz bardziej się zmniejsza. Dzięki React Native można kod utworzyć raz w języku JavaScript, a następnie ponownie go wykorzystać na platformach iOS i Android. Framework JavaScript Electron pozwala tworzyć aplikacje działające w systemach Windows i macOS. Istnieją jeszcze inne wykorzystujące JavaScript podejścia w zakresie budowania aplikacji mobilnych i stacjonarnych.

Oto dostępne rozwiązania w zakresie programowania na platformach mobilnych (każde z nich ma swoje wady i zalety):

1. Podejście *natywne* — aplikacje natywne iOS, Android i Blackberry utworzone w językach Objective-C, Swift i Java.
1. Podejście *abstrakcyjnie natywne* — aplikacje natywne zbudowane za pomocą takich narzędzi jak m.in. JavaScript, React Native (*https://facebook.github.io/react-native/*), NativeScript, Appcelerator (*https://www.appcelerator.com/*), Xamarin (*https://visualstudio.microsoft.com/xamarin/*), Smartface (*https://www.smartface.io/smartface/*), a następnie skompilowane na natywny kod Objective-C lub Javy.
3. Podejście *responsywne* — mobilne witryny internetowe przystosowane do ekranów o małej wielkości i charakteryzujące się responsywnym projektem. Używane są przy tym frameworki CSS, czyli Bootstrap (*https://getbootstrap.com/*) i Foundation (*https://foundation.zurb.com/*), zwykły kod CSS i różne szablony. Można wykorzystać także pewne frameworki JavaScriptu, np. Backbone.js, Angular.js, Ember.js i React.js.



Kup książkę Poleć książkę

4. Podejście *hybrydowe* — aplikacje HTML5 składające się z kodu HTML, CSS i JavaScriptu, zwykle zbudowane z użyciem frameworków Sencha Touch (*https://www.sencha.com/products/touch/*), Trigger.io (*https://trigger.io/*) lub Ionic (*https://ionicframework.com/*), a następnie opakowane natywną aplikacją za pomocą PhoneGap (*https://phonegap.com/*). Podobnie jak w przypadku poprzedniego podejścia, także w tym prawdopodobnie zechcesz zastosować pewne frameworki JavaScriptu podczas programowania, np. Backbone.js, Angular.js, Ember.js i React.js.

Moimi faworytami są podejścia drugie i czwarte, czyli abstrakcyjnie natywne i hybrydowe. Drugie nie wymaga użycia innej bazy kodu. Minimalną wersję produktu można przygotować dla wielu platform współdzielących dużą ilość tego samego kodu. Gorąco zachęcam do użycia React Native. Jeżeli chcesz szybko rozpocząć programowanie mobilne z użyciem podejścia abstrakcyjnego natywnie, zapoznaj się z inną moją książką, *React Native Quickly: Start Learning Native iOS Development with JavaScript*, i kursem *React Native Quickly* (*https://node.university/p/react-native-quickly*).

Z kolei czwarte podejście charakteryzuje się znacznie potężniejszymi możliwościami i zapewnia bardziej skalowany (w sensie programistycznym) interfejs użytkownika. To rozwiązanie lepiej sprawdza się w przypadku skomplikowanych aplikacji. Ponowne użycie kodu na różnych platformach mobilnych i internetowych jest łatwe, ponieważ w większości przypadków tworzysz kod w języku JavaScript.

## Język HTML

HTML (ang. *Hypertext Markup Language*) nie jest prawdziwym językiem programowania. To zbiór znaczników opisujących treść i przedstawiających ją w pewien ustrukturyzowany i sformatowany sposób. W kodzie HTML nie umieścisz zbyt wiele logiki. Nie masz do dyspozycji zmiennych i pętli. HTML jest językiem internetu, ponieważ jest wszechobecny i wykorzystywany przez wszystkie klienty (przeglądarki WWW) do interpretowania danych dla użytkowników.

Znacznik HTML składa się z jego nazwy umieszczonej w nawiasach ostrych (<>). W większości przypadków para znaczników otwierającego i zamykającego obejmuje treść. Przed nazwą znacznika zamykającego umieszczony jest ukośnik. Znaczniki tworzą hierarchię treści. Każdy znacznik ma znaczenie, przeznaczenie i domyślną reprezentację w przeglądarce WWW. Istnieją znaczniki generujące nagłówki, akapity, listy, obrazy, łącza itd.

W przedstawionym tutaj fragmencie kodu każdy wiersz jest elementem HTML:

```
<h2>Ogólny opis języka HTML</h2>
<div>HTML to...</div>
<link rel="stylesheet" type="text/css" href="style.css" />
```

Dokument HTML sam w sobie jest znacznikiem `<html>`, a wszystkie pozostałe, czyli `<head>`, `<body>`, `<h2>` i `<p>`, są znacznikami potomnymi znacznika `<html>`. Znacznik `<head>` jest przeznaczony dla metadanych strony internetowej, czyli niewidocznych dla użytkownika informacji o samej stronie. Z kolei znacznik `<body>` zawiera treść strony wyświetlaną użytkownikowi. Programiści używają składającego się z czterech spacji wcięcia do oznaczania zagnieżdżonych elementów. W kolejnym przykładzie element `<link>` jest zagnieżdżony o dwa poziomy głębiej niż `<html>`, a jego działanie polega na importowaniu stylów CSS:

```
<!DOCTYPE html>
<html lang="pl">
    <head>
        <link rel="stylesheet" type="text/css" href="style.css"/>
    </head>
    <body>
        <h2>Ogólny opis języka HTML</h2>
        <p>HTML to...</p>
    </body>
</html>
```



Kup książkę Poleć książkę

Zwróć uwagę, że znacznik zamykający ma ukośnik (/) umieszczony przed nazwą, np. `</html>`. Ten ukośnik jest bardzo ważny dla prawidłowego generowania (interpretowania i wyświetlania) elementów w przeglądarce WWW.

Istnieje kilka odmian i wersji HTML, np. DHTML, XHTML1.0, XHTML1.1, XHTML2, HTML4 i HTML5. Komiks zamieszczony w artykule *Misunderstanding Markup: XHTML2/HTML5* i opublikowany na stronie *https://www.smashingmagazine.com/2009/07/misunderstanding-markup-xhtml-2-comic-strip/* doskonale wyjaśnia różnice między nimi. Przed wprowadzeniem specyfikacji HTML5 programiści musieli używać odpowiedniej wersji języka. Obecnie wystarczy podanie znacznika `<!DOCTYPE html>`, a nowoczesna przeglądarka WWW będzie wiedziała, w jaki sposób zinterpretować język znaczników.

Element HTML może mieć atrybuty. Wcześniej pokazałem, że znacznik `<link>` używa atrybutów `rel`, `type` i `href`. Atrybuty zwykle dostarczają informacje dodatkowe niewidoczne bezpośrednio dla użytkownika. Nie stanowią one treści i pod tym względem różnią się od elementów zagnieżdżonych, które dostarczają treść. Najważniejsze atrybuty stosowane w praktycznie wszystkich elementach i znacznikach to `class`, `id`, `style` i `data-name`. Dostępne są jeszcze atrybuty zdarzeń: `onclick`, `onmouseover`, `onkeyup` itd.

## class

Atrybut `class` definiuje klasę, która zostanie użyta podczas nadawania stylów za pomocą CSS lub przeprowadzania operacji w obiektowym modelu dokumentu (ang. *Domain Object Model* — DOM):

```
<p class="normal">...</p>
```

## id

Atrybut `id` definiuje identyfikator, którego przeznaczenie jest podobne jak klasy, ale z tą różnicą, że musi być unikatowy:

```
<div id="footer">...</div>
```

## style

Atrybut `style` pozwala zdefiniować osadzony styl CSS nadający wygląd elementowi:

```
<font style="font-size:20px">...</font>
```

## title

Atrybut `title` definiuje informacje dodatkowe, które w większości przeglądarek WWW są wyświetlane w podpowiedziach:

```
<a title="Odpowiedź na pytanie">...</a>
```

## data-name

Atrybut `data-name` umożliwia przechowywanie metadanych w obiektowym modelu dokumentu:

```
<tr data-token="fa10a70c–21ca–4e73-aaf5-d889c7263a0e">...</tr>
```

## onclick

Atrybut `onclick` wywołuje osadzony kod JavaScriptu po wystąpieniu zdarzenia `click`:

```
<input type="button" onclick="validateForm();">...</a>
```



Kup książkę Poleć książkę

### onmouseover

Atrybut `onmouseover` jest podobny do `onclick`, ale dotyczy obsługi zdarzenia `hover` wywoływanego przez mysz:

```
<a onmouseover="javascript:
  this.setAttribute('css','color:red')">
  ...
</a>
```

Dostępne są jeszcze inne atrybuty elementów HTML dla osadzonego kodu JavaScriptu, m.in.:

- `onfocus` — dotyczy sytuacji, gdy przeglądarka WWW aktywuje element.
- `onblur` — dotyczy sytuacji, gdy przeglądarka WWW opuszcza element.
- `onkeydown` — dotyczy sytuacji, gdy użytkownik naciska klawisz.
- `ondblclick` — dotyczy sytuacji, gdy użytkownik dwukrotnie klika myszą.
- `onmousedown` — dotyczy sytuacji, gdy użytkownik naciska przycisk myszy.
- `onmouseup` — dotyczy sytuacji, gdy użytkownik zwalnia przycisk myszy.
- `onmouseout` — dotyczy sytuacji, gdy użytkownik przesuwa kursor myszy poza obszar elementu.
- `oncontextmenu` — dotyczy sytuacji, gdy użytkownik wyświetla menu kontekstowe.

Pełną listę takich zdarzeń i tabelę ich zgodności z różnymi przeglądarkami WWW znajdziesz w dokumencie *Event compatibility tables* opublikowanym na stronie *https://www.quirksmode.org/dom/events/index.html*.

W przykładach przedstawionych w książce będę korzystał z klas definiowanych przez framework Bootstrap (*https://getbootstrap.com/*). Stosowanie osadzonego kodu CSS i JavaScriptu jest ogólnie rzecz biorąc kiepskim rozwiązaniem i warto tego unikać. Mimo wszystko dobrze jest znać nazwy zdarzeń JavaScriptu, ponieważ są one używane w wielu miejscach w jQuery, Backbone.js i oczywiście w samym JavaScripcie. Aby skonwertować listę atrybutów na listę zdarzeń JavaScriptu, wystarczy usunąć z nich prefiks `on` i tak na przykład atrybut `onclick` oznacza zdarzenie `click`.

Więcej informacji na ten temat znajdziesz w dokumencie *JavaScript basics* opublikowanym na stronie *https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics*.

## Kaskadowe arkusze stylów

Kaskadowe arkusze stylów (ang. *Cascading Style Sheets* — CSS) zapewniają możliwość formatowania i prezentowania treści. Dokument HTML może mieć dołączany zewnętrzny arkusz stylów wskazany przez znacznik `<link>`, jak pokazałem we wcześniejszych przykładach. Inna możliwość to osadzenie kodu CSS bezpośrednio w znaczniku `<style>`:

```
<style>
  body {
    padding-top: 60px; /* 60px, aby pozostawić trochę miejsca */
  }
</style>
```

Każdy element może mieć atrybuty `id`, `class` lub oba:

```
<div id="main" class="large">
  Lorem ipsum dolor sit amet,
  Duis sit amet neque eu.
</div>
```



Kup książkę Poleć książkę

W kaskadowych arkuszach stylów dostęp do elementu odbywa się za pomocą jego identyfikatora, klasy, nazwy znacznika, a także w pewnych sytuacjach poprzez związek element nadrzędny – potomny lub wartość atrybutu elementu.

Następne polecenie definiuje kolor tekstu wszystkich akapitów (znaczniki `<p>`) jako szary — jego wartość to `#999999`:

```
p {
  color: #999999;
}
```

W kolejnym fragmencie kodu następuje zdefiniowanie dopełnienia elementu `<div>` wraz z atrybutem `id` o wartości `main`:

```
div#main {
  padding-bottom: 2em;
  padding-top: 3em;
}
```

Następne polecenie definiuje wielkość czcionki na 14 pikseli dla wszystkich elementów wraz z atrybutem `class` o wartości `large`:

```
.large {
  font-size: 14pt;
}
```

Niniejsze polecenie powoduje ukrycie tych znaczników `<div>`, które są bezpośrednimi elementami potomnymi `<body>`:

```
body > div {
  display: none;
}
```

Kolejne polecenie definiuje szerokość 150 pikseli dla znaczników `<input>`, których atrybut `name` ma wartość `email`:

```
input[name="email"] {
  width: 150px;
}
```

Więcej informacji na temat CSS znajdziesz w artykule Wikipedii (*https://pl.wikipedia.org/wiki/Kaskadowe_arkusze_stylów*) oraz w portalu MDN (*https://developer.mozilla.org/en-US/docs/Web/CSS*).

CSS3 to nowsza wersja CSS, oferująca nowe sposoby wykonywania pewnych zadań, takich jak zaokrąglanie rogów, definiowanie obramowania i wypełniania — wiele nich było możliwych w zwykłym CSS jedynie za pomocą obrazów w formacie PNG i GIF bądź innych sztuczek.

Więcej informacji o CSS3 znajdziesz w witrynie CSS3.info (*http://www.css3.info/*) oraz w artykule *CSS3 vs. CSS: A Speed Benchmark* opublikowanym przez portal Smashing na stronie *https://www.smashingmagazine.com/2011/04/css3-vs-css-a-speed-benchmark/*.

## JavaScript

Język JavaScript został opracowany przez Netscape w 1995 roku jako LiveScript. W tym samym roku powstała również Java. LiveScript okazał się tak popularny, że programiści zdecydowali się zmienić jego nazwę na JavaScript. Jednak języki Java i JavaScript są odmienne, podobnie jak koń i końcówka. JavaScript ma taki sam związek z Javą jak koń z końcówką. Dlatego nie myl JavaScriptu i Javy. JavaScript jest interpretowany i wykonywany przez silnik JavaScriptu, np. Google Chrome V8, Microsoft Chakra lub



Kup książkę Poleć książkę

SpiderMonkey. Z kolei kod Javy jest kompilowany na postać kodu bajtowego uruchamianego następnie przez wirtualną maszynę Javy. Między tymi językami istnieją różnice w składni, użyciu pamięci, typach i praktycznie we wszystkich pozostałych aspektach.

Dla większości programistów łatwiejsze będzie rozpoczęcie nauki z JavaScriptem niż jakimkolwiek innym językiem. JavaScript jest niezwykle ekspresyjny i narzuca niewielkie obciążenie — wystarczy uruchomić przeglądarkę WWW i już można rozpocząć tworzenie kodu. Ponadto jest jedynym językiem natywnie działającym w przeglądarkach WWW, o ile nie skorzystasz z WebAssembly, ale kto by tego chciał. To wszystko prowadzi do ogromnej popularności JavaScriptu wyrażonej liczbą środowisk uruchomieniowych. Co więcej, JavaScript jest wszechobecny i może być używany praktycznie wszędzie.

Obecnie JavaScript jest stosowany w programowaniu po stronie zarówno klienta, jak i serwera, a także do tworzenia aplikacji biurowych czy aplikacji dronów i internetu rzeczy (ang. *Internet of Things* — IoT) itd. Ten język stanowi główny temat niniejszej książki, ponieważ za jego pomocą można programować na wszystkich warstwach.

Jeżeli jesteś początkującym programistą, wystarczy po prostu poznać JavaScript i nie będzie potrzeby sięgania po inne języki. JavaScript można wykorzystać do wszystkiego: front-endu, back-endu, programowania baz danych, DevOps itd. To wszystko powoduje, że stajesz się prawdziwym programistą JavaScriptu.

Rozpocznę od przedstawienia kodu JavaScriptu osadzonego w HTML. Umieszczenie kodu JavaScriptu w znaczniku `<script>` to najłatwiejszy sposób na użycie JavaScriptu w dokumencie HTML.

```
<script type="text/javascript" language="javascript">
  alert("Witaj, Świecie!") // Powoduje wyświetlenie prostego okna dialogowego
</script>
```

Pamiętaj, że łączenie kodów napisanych w HTML i JavaScripcie jest kiepskim pomysłem. Dobrze jest przenieść kod JavaScriptu do pliku zewnętrznego, a następnie dołączać go za pomocą atrybutu *scr="nazwa_pliku.js"* w znaczniku `<script>`. Przykład takiego rozwiązania pokazałem dla zasobu *app.js*:

```
<script src="js/app.js" type="text/javascript"
  language="javascript">
</script>
```

Zwróć uwagę, że znacznik zamykający `</script>` jest niezbędny nawet w przypadku pustego elementu, za pomocą którego jest dołączany plik zasobu zewnętrznego. Innymi słowy niewystarczające jest użycie znacznika `<script src="js/app.js"...>`.

Dawno, dawno temu, gdy po ziemi chodziły jeszcze dinozaury, przeglądarki WWW potrafiły przetwarzać i uruchamiać VBScript (skrypt Microsoft Visual Basic, taki sam jak w skoroszytach Excela). Dlatego programiści musieli podawać typ skryptu: JavaScript, VB lub coś innego, np. Java, Flash i inni przegrani front-endu. Na szczęście nowoczesne przeglądarki WWW domyślnie zakładają użycie JavaScriptu, ponieważ to jedyny rodzaj skryptu, który potrafią uruchamiać i który jest najczęściej stosowany przez programistów. Stąd na przestrzeni lat atrybuty `type` i `language` stały się opcjonalne w *nowoczesnych* przeglądarkach WWW ze względu na przytłaczającą dominację JavaScriptu.

Mamy jeszcze inne sposoby na uruchamianie kodu JavaScriptu:

- omówione już podejście osadzonego kodu;
- konsole FireBug i narzędzia programistyczne WebKit;
- interaktywna powłoka Node.js.

Jedną z zalet języka JavaScript jest luźne podejście do stosowania typów. Jest to przeciwieństwem ścisłego stosowania typów (*https://pl.wikipedia.org/wiki/Typowanie_silne*) w językach C i Java i powoduje, że JavaScript lepiej nadaje się do prototypowania. W kolejnych sekcjach przedstawię wprowadzenie do najważniejszych typów obiektów i klas JavaScriptu. Użyłem pojęcia obiektów i klas, ponieważ JavaScript



Kup książkę Poleć książkę

nie ma typowych klas. W JavaScripcie obiekt dziedziczy po innym obiekcie, co jest nazywane dziedziczeniem prototypowanym. Dezorientujące? Poczekaj, aż poznasz inne rodzaje dziedziczenia, ponieważ istnieje wiele różnych sposobów na jego implementację.

Wróćmy do typów. Typy proste w JavaScripcie mają opakowanie w postaci obiektów i klas dostarczających funkcjonalność dodatkową i metody statyczne. Każdy typ prosty ma wspomniane opakowanie.

## Zwykłe liczby

Zwykłe liczby to po prostu standardowe wartości liczbowe:

```
const num = 1
```

Do zdefiniowania zmiennej używane jest słowo kluczowe `var`, `const` lub `let`. Dwa ostatnie respektują zasięg utworzony przez blok logiczny (funkcje, pętle i warunki), podczas gdy `var` nie. Deklaracja `const` nie pozwala na ponowne przypisanie wartości. Jeżeli programista pominie słowo kluczowe, mogą zdarzyć się złe rzeczy, np. wyciek zmiennej do zasięgu globalnego i powstanie kolizji nazw.

Wcześniej stosowane rozwiązanie polegało na użyciu słowa kluczowego `var`. Gdy w trakcie rozmowy kwalifikacyjnej słyszę od kandydata o użyciu `var`, natychmiast zapala mi się czerwona kontrolka. To słowo kluczowe jest odpowiedzialne za wiele dziwnych błędów, więc stosują je jedynie niedoświadczeni programiści, nieświadomi istnienia specyfikacji ES2015/ES6, a także ci, którzy poznawali JavaScript z portalu w3schools.com (*https://www.w3schools.com/*). Jeżeli chcesz poznać 10 najważniejszych funkcji specyfikacji ES2015/ES6, które powinien znać każdy programista, zapoznaj się ze zwięzłym postem opublikowanym na stronie *https://webapplog.com/es6/*. Jeśli chodzi o funkcje specyfikacji ES7 i ES8, zachęcam do zapoznania się z postem opublikowanym na stronie *https://node.university/blog/15982/es7es8*.

## Obiekt Number

Obiekt `Number` (*https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number*) i jego metody zapewniają funkcjonalność przeznaczoną do pracy z wartościami liczbowymi (całkowite i zmiennoprzecinkowe). Oto przykłady utworzenia obiektu `Number` za pomocą operatora `new`:

```
const numObj = new Number('123') // Obiekt typu Number
const num = numObj.valueOf() // Zwykła liczba
const numStr = numObj.toString() // Reprezentacja w postaci ciągu tekstowego
console.log(numObj === 123) // Wartość false
```

Zwróć uwagę, że w ostatnim wierszu obiekt `Number` nie jest identyczny ze zwykłą liczbą (potrójny znak równości powoduje sprawdzenie zarówno wartości, jak i typu). Po prostu zwykła liczba i obiekt to różne typy danych. JavaScript pozwala na wygodną i automatyczną konwersję typu na podobny, co wymaga użycia podwójnego znaku równości. Oto kolejny fragment kodu:

```
const numObj = new Number('123') // Obiekt Number
console.log(numObj === 123) // Wartość false
```

Obiekt `Number` oferuje użyteczne metody, np. `parseInt()` przeznaczoną do konwersji ciągu tekstowego na postać liczby:

```
17 === "17" // Wartość false
17 === Number.parseInt("17", 10) // Wartość true
```



Kup książkę Poleć książkę

## Obiekt String

Obiekt `String` ma wiele użytecznych metod, np. `length()` i `match()`. Do utworzenia nowego egzemplarza tego typu używane jest słowo kluczowe `new`:

```
const strObj = new String("abcde") // Obiekt String
const str = strObj.valueOf() // Zwykły ciąg tekstowy
strObj.match(/ab/)
str.match(/ab/) // Oba wywołania działają
```

## Zwykły ciąg tekstowy

Zwykły ciąg tekstowy to sekwencja znaków ujętych w apostrofy lub cudzysłowy. Przykładowo do zdefiniowania zwykłego ciągu tekstowego można użyć apostrofów lub cudzysłowów:

```
const str = 'React Quickly' // Apostrofy
const str1 = "React Quickly" // Cudzysłowy.
console.log(str === str1) // Wartość true
const newStr = "abcde".substr(1,2) // Wartością newStr jest teraz bc
```

W JavaScripcie cudzysłów nie ma żadnego innego znaczenia specjalnego niż definiowanie ciągu tekstowego. Natomiast w innych językach programowania cudzysłów oznacza również interpolację. Według mnie należy pozbyć się też apostrofów i używać jedynie cudzysłowów, ponieważ to wyeliminuje wiele sporów dotyczących sposobu definiowania ciągów tekstowych. Programiści zwykle preferują apostrofy, bo dzięki temu mogą stosować cudzysłowy w wartościach atrybutów HTML. Wadą takiego rozwiązania jest brak możliwości użycia apostrofu w tekście, o ile nie będzie poprzedzony ukośnikiem:

```
'To jest apostrof \' w tekście.' // Prawidłowy ciąg tekstowy
'To jest apostrof ' w tekście ' // NIEPRAWIDŁOWY ciąg tekstowy
```

Dla wygody programisty JavaScript automatycznie opakowuje te zwykłe ciągi tekstowe metodami obiektu `String`. Dlatego można w ich przypadku stosować metody, czyli `substr()`. Jednak użycie potrójnego znaku równości daje w wyniku `false`, ponieważ obiekt `String` i zwykły ciąg tekstowy nie są tego samego typu.

## Obiekt wyrażenia regularnego

Wyrażenie regularne, określane również mianem *regex*, to wzorce znaków pozwalające wyszukiwać, dopasowywać, zastępować i testować ciągi tekstowe:

```
const pattern = /[A-Z]+/
'ab'.match(pattern) // Dane wyjściowe: null
'AB'.match(pattern) // Dane wyjściowe: ["AB"]
```

Wartością zwrotną metody `match()` jest tablica dopasowań `["AB"]`. Jeżeli chcesz otrzymać zwykłą wartość boolowską typu `true` lub `false`, wówczas użyj po prostu `pattern.test(str)`. Spójrz na kolejny przykład:

```
const str = 'A'
const pattern = /[A-Z]+/
pattern.test(str) // Dane wyjściowe: true
```



Kup książkę Poleć książkę

## Typy specjalne

Gdy podczas debugowania masz wątpliwości dotyczące typu obiektu, możesz go sprawdzić za pomocą wywołania `typeof obiekt`:

```
const obj = {}
console.log(typeof obj) // To jest obiekt
const a = 1
console.log(typeof a) // To jest liczba
```

Dostępne są jeszcze inne typy specjalne używane w JavaScripcie:

- `NaN` — nieliczba;
- `null` — brak wartości;
- `undefined` — niezadeklarowana zmienna;
- `function` — funkcja.

## JSON

Biblioteka JSON umożliwia przetwarzanie i deserializowanie obiektów JavaScriptu. Przykładowo można wziąć prawidłowy ciąg tekstowy JSON, skonwertować go na postać obiektu JavaScriptu `stringObj`, dodać nowe pole `c`, a następnie skonwertować ponownie na ciąg tekstowy `stringObj2` i jego ładniejszą postać `stringObj3` (wraz ze spacjami i znakami nowego wiersza):

```
const stringObj = '{"a": 1, "b": "cześć"}'
const obj = JSON.parse(stringObj)
obj.c = 2
const stringObj2 = JSON.stringify(obj)
console.log(stringObj2) // Ciąg tekstowy JSON {"a":1,"b":"cześć", "c":2}
const stringObj3 = JSON.stringify(obj, null, 2)
// Upiększenie ciągu tekstowego przez dodanie spacji i znaków nowego wiersza
console.log(stringObj2) // Upiększony ciąg tekstowy JSON
```

## Obiekt Array

Tablica to lista indeksowana od zera. W JavaScripcie tablica jest obiektem używającym kluczy w postaci indeksów sekwencyjnych. Istnieją dwa sposoby na utworzenie tablicy: za pomocą obiektu `Array` i literału tablicy. Oto fragment kodu:

```
const arr = new Array() // Obiekt Array
const arr = ['jabłko', 'pomarańcza', 'kiwi'] // Literał tablicy
```

Każda tablica dziedziczy wszystkie metody klasy `Array`. Egzemplarz tego typu oferuje wiele naprawdę użytecznych metod, np. `indexOf()`, `map()`, `slice()` i `join()`. Ich znajomość i umiejętność użycia może zaoszczędzić wielu godzin tworzenia kodu i jego debugowania.

## Obiekt Object

Bardzo lubię język JavaScript między innymi za łatwość, z jaką można utworzyć w nim obiekt. Na przykład w Javie programista musi zdefiniować klasę, być może interfejs, metody getter i setter, a następnie utworzyć egzemplarz danego obiektu. Natomiast w JavaScripcie wystarczy po prostu napisać `{}` i obiekt jest gotowy. Użycie składni w postaci nawiasu klamrowego nosi nazwę literału obiektu. Oto przykład pokazujący obiekt wraz z polami `name`, `url` i `price`:



Kup książkę Poleć książkę

```
const obj = {
  name: 'Gala',
  url: 'img/gala100x100.jpg',
  price: 129
}
```

Można również użyć obiektu `Object`:

```
const obj = new Object({a: 1})
```

Jednak nie zalecam takiego rozwiązania, znacznie lepszym jest literał.

Obiekt `Object` ma wiele użytecznych metod, np. `Object.keys()`, `Object.entries()` i `Object.values()`.

Obiekt jest przekazywany przez referencję. Najlepiej jest utworzyć jego kopię za pomocą wywołania `Object.assign()`, w przeciwnym razie modyfikacja pierwotnego spowoduje zmianę wszystkich obiektów odwołujących się do niego.

```
const obj1 = {a:1}
const obj2 = obj1 // Odwołanie do obiektu
console.log(obj2) // Dane wyjściowe: { a: 1 }
obj1.a = 2
console.log(obj2) // Zmiana elementu: { a: 2 }
const obj3 = Object.assign({}, obj1) // Utworzenie kopii obiektu
console.log(obj3) // Dane wyjściowe: { a: 2 }
obj1.a = 3
console.log(obj3) // Niezmieniony element { a: 2 }
```

Każdy obiekt dziedziczy po `Object`. Dziedziczenie odbywa się przez prototyp, klasę lub funkcję fabryki. Więcej informacji na temat wzorców dziedziczenia znajdziesz w dalszej części rozdziału.

## Wartości boolowskie i obiekty

Podobnie jak w przypadku obiektów `String` i `Number`, także obiekt `Boolean` oferuje alternatywę dla zwykłej wartości boolowskiej. Nie zalecam stosowania obiektu `Boolelan`, lepiej pozostać przy zwykłych wartościach boolowskich. Spójrz na przedstawiony tutaj przykład:

```
const bool1 = true
const bool2 = false
const boolObj = new Boolean(false)
console.log(bool2 === boolObj) // Dane wyjściowe: false
console.log(bool2 == boolObj) // Dane wyjściowe: true
```

## Obiekt Date

Obiekt `Date` umożliwia pracę z wartościami daty i godziny:

```
const timestamp = Date.now() // Dane wyjściowe: 1368407802561
const d = new Date() // Dane wyjściowe: Sun May 12 2013 18:17:11 GMT-0700 (PDT)
```

## Obiekt Math

Obiekt `Math` ma metody przeznaczone do pracy z matematycznymi stałymi i funkcjami, np. `floor()`, `random()`, `round()`, `sqrt()`:

```
const x = Math.floor(3.4890)
const ran = Math.round(Math.random()*100)
```



Kup książkę Poleć książkę

## Obiekty przeglądarki WWW

Obiekty przeglądarki WWW zapewniają dostęp do przeglądarki WWW i jej właściwości, czyli adres URL:

```
window.location.href = 'http://rapidprototypingwithjs.com'
console.log('test')
```

## Obiekty modelu DOM

Obiekty modelu DOM i węzły DOM są interfejsem przeglądarki WWW dla elementów modelu DOM generowanych na stronie. Mają właściwości, np. `width`, `height` i `position`, oraz oczywiście wewnętrzną treść, która może być wewnętrznym elementem lub tekstem. Aby pobrać węzeł DOM, należy użyć jego identyfikatora, na przykład tak, jak pokazałem w kolejnym fragmencie kodu:

```
const transactionsContainer = document.createElement('div')
transactionsContainer.setAttribute('id', 'main')
const content = document.createTextNode('Transactions')
transactionsContainer.appendChild(content)
document.body.appendChild(transactionsContainer)
const main = document.getElementById('main')
console.log(main, main.offsetWidth, main.offsetHeight)
```

## Metody globalne

Poza klasami tj. `String`, `Array`, `Number` i `Math`, które mają wiele użytecznych metod, można wywoływać jeszcze wymienione tutaj, które są metodami globalnymi. To oznacza, że pozostają dostępne w dowolnym miejscu kodu.

- `encodeURI()` — zakodowanie URI (ang. *Uniform Resource Identifier*) w celu otrzymania adresu URL, np. `encodeURI('http://www.webapplog.com/js')`.
- `decodeURI()` — dekodowanie adresu URI.
- `encodeURIComponent()` — zakodowanie URI dla parametrów URL (nie będzie zastosowana dla całego ciągu tekstowego adresu URL).
- `decodeURIComponent()` — dekodowanie fragmentu adresu URL.
- `isNaN()` — ustalenie, czy podana wartość jest liczbą.
- `JSON()` — przetworzenie (`parse()`) i serializacja (`stringify()`) danych w formacie JSON.
- `parseFloat()` — konwersja ciągu tekstowego na liczbę zmiennoprzecinkową.
- `parseInt()` — konwersja ciągu tekstowego na liczbę całkowitą.
- `Intl()` — metody porównywania ciągów tekstowych uwzględniające dany język.
- `Error()` — obiekt błędu, którego można używać do tworzenia własnych obiektów błędu, np. `throw new Error('To jest świetna książka!')`.
- `Date()` — różne metody przeznaczone do pracy z datą i godziną.

## Konwencje JavaScriptu i Node.js

JavaScript używa wielu konwencji stylu. Jedna z nich, camelCase, polega na łączeniu słów ze sobą i zapisywaniu pierwszej litery każdego z nich z wyjątkiem pierwszego słowa jako dużej.

Średniki są opcjonalne. Nazwa rozpoczynająca się od znaku podkreślenia jest prywatną metodą lub atrybutem, ale nie dlatego, że jest chroniona przez język. Znak podkreślenia jest stosowany po to, aby poinformować innych programistów, że nie powinni używać danych metod i atrybutów, ponieważ mogą się one zmienić w przyszłości.



Kup książkę Poleć książkę

JavaScript obsługuje liczby o wielkości do jedynie 53 bitów. Jeżeli musisz pracować z większymi liczbami, rozejrzyj się za umożliwiającą to biblioteką zewnętrzną.

Inną ważną cechą JavaScriptu jest fakt, że to język funkcyjny i prototypowy. Typowa składnia deklaracji funkcji przedstawia się następująco:

```
function Sum(a,b) {
  const sum = a + b
  return sum
}
console.log(Sum(1, 2))
```

Funkcje w JavaScripcie są pierwszoklasowymi elementami ze względu na funkcyjną naturę języka. Dlatego funkcja może być używana jako inna zmienna lub obiekt. To pozwala na przekazanie funkcji jako argumentu innej funkcji.

```
const f = function (str1){
  return function(str2){
  return str1 + ' ' + str2
  }
}
const a = f('Witaj')
const b = f('Żegnaj')
console.log((a('Kotek'))
console.log((b('Piesek'))
```

Kolejnym sposobem na zdefiniowanie funkcji jest zastosowanie składni grubej strzałki. W takim przypadku nie jest używana nazwa, więc programiści zwykli przechowywać taką funkcję w zmiennej:

```
const Sum = (a,b) => {
  const sum = a + b
  return sum
}
console.log(Sum(1, 2))
```

Następną cechą języka jest to, że składnia grubej strzałki zachowuje wartość `this` z zasięgu zewnętrznego. To powoduje, że ta składnia jest odpowiednikiem stosowania metody `bind(this)` w zwykłej funkcji:

```
const Sum = function(a, b) {
  const sum = a + b
  return sum
}.bind(this)
```

Oczywiście w przedstawionym tutaj przykładzie nie zostało użyte słowo kluczowe `this`, więc nie było potrzeby zastosowania pokazanego rozwiązania. Jednak gdy korzystasz z klas i dziedziczenia, bardzo często będziesz używać `this`, ponieważ w ten sposób można odwoływać się do egzemplarzy klas, ich metod, atrybutów i właściwości.

Skoro wspomniałem o egzemplarzach, klasach i dziedziczeniu, jeżeli chcesz stać się *dobrym* programistą JavaScriptu, bardzo ważna jest znajomość wielu różnych sposobów tworzenia obiektu w JavaScripcie:

- wzorzec dziedziczenia klasycznego (*http://www.crockford.com/javascript/inheritance.html*);
- wzorzec dziedziczenia pseudoklasycznego (*http://javascript.info/class-patterns*);
- wzorzec dziedziczenia funkcyjnego.

Więcej informacji na temat wzorców dziedziczenia znajdziesz w artykułach *Inheritance Patterns in JavaScript* (*http://bolinfest.com/javascript/inheritance.php*) i *Inheritance and the prototype chain* (*https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain*).



Kup książkę Poleć książkę

Portal Mozilla Developer Network ma *najlepsze* przewodniki dotyczące JavaScriptu i modelu DOM (*https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference*). Ponadto edytor Visual Studio Code oferuje automatyczne uzupełnianie kodu i podpowiedzi. (Niezależnie od wszystkiego, przestań używać edytora Notepad++).

Specyfikację ECMAScript (standard dla JavaScriptu i Node.js) znajdziesz w witrynie *http://www.ecma-international.org/*.

# Metody zwinne

W nowoczesnym programowaniu większość zespołów wykorzystuje podczas pracy tzw. metody zwinne (ang. *agile methodologies*). Metody zwinne w zakresie tworzenia oprogramowania zostały opracowane, ponieważ podejście tradycyjne okazało się niewystarczająco dobre w sytuacjach charakteryzujących się wysokim poziomem nieprzewidywalności, to znaczy gdy rozwiązanie jest nieznane (*http://www.startuplessonslearned.com/2009/03/combining-agile-development-with.html*). Metody zwinne idą w parze z metodą Scrum (sprint), programowaniem sterowanym testami, ciągłym wdrażaniem, programowaniem w parach, a także innymi praktycznymi technikami, z których wiele zostało zapożyczonych z programowania ekstremalnego.

## Scrum

W zakresie zarządzania metody zwinne często stosują podejście Scrum. Metoda Scrum to sekwencja krótkich cykli, z których każdy jest nazywany *sprintem*. Jeden sprint trwa zwykle tydzień lub dwa. Typowy sprint rozpoczyna się i kończy spotkaniem, w trakcie którego nowe zadania są przydzielane członkom zespołu. Nowe zadania nie mogą być przydzielane w czasie trwania sprintu, lecz jedynie podczas spotkań.

Istotną częścią metody Scrum są codzienne spotkania *scrum*, stąd nazwa tego podejścia. Każde spotkanie trwa od 5 do 15 minut i zwykle odbywa się na korytarzu. W trakcie spotkania członkowie zespołu odpowiadają na trzy pytania:

1. Co zrobiłeś od wczoraj?
2. Co zamierzasz zrobić dzisiaj?
3. Czy potrzebujesz czegoś od innych członków zespołu?

Podobnie jak w przypadku wielu frameworków zwinnych (Kanban, XP, SAFE), także Scrum oferuje elastyczność w zakresie zmiany wymagań projektu podczas pracy nad nim. Jest to *doskonałe* usprawnienie względem tradycyjnego modelu kaskadowego (ang. *waterfall model*), zwłaszcza w sytuacjach cechujących się wysokim poziomem niepewności, na przykład w startupach. JavaScript jest stosowany w interfejsie użytkownika, gdzie bardzo często może zachodzić wiele zmian. Zobaczysz lub już miałeś okazję zobaczyć, że wiele zespołów odpowiedzialnych za przygotowywanie front-endu adaptuje techniki Scrum i zwinne.

Zaletą metody Scrum jest efektywność w sytuacjach, w których trudno jest planować naprzód, a także gdy decyzje są podejmowane na podstawie otrzymywanych informacji zwrotnych.

Więcej informacji na temat metody Scrum znajdziesz w następujących źródłach:

- dostępny w formacie PDF przewodnik po metodzie Scrum (*https://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf*);
- witryna scrum.org (*https://www.scrum.org/*);
- książka *Succeeding with Agile* napisana przez Mike'a Cohena i wydana przez Addison-Wesley w 2010 roku.



Kup książkę Poleć książkę

## Programowanie sterowane testami

Programowanie sterowane testami (ang. *Test-Driven Development* — TDD) składa się z trzech kroków:

1. Utworzenie kończących się niepowodzeniem zautomatyzowanych pakietów testu dla nowych funkcji, zadań lub usprawnień przez wykorzystanie asercji, które będą prawdziwe lub fałszywe.
2. Utworzenie kodu pozwalającego na zaliczenie przygotowanych wcześniej pakietów testów.
3. Przeprowadzenie refaktoryzacji kodu, o ile jest niezbędna, oraz dodanie funkcjonalności, przy jednoczesnym zapewnieniu zaliczania testów.
4. Powtarzanie kroków, dopóki nie zostaną ukończone wszystkie zadania.

Testy mogą być podzielone na funkcjonalne i jednostkowe. Te drugie pozwalają systemowi testować poszczególne jednostki, metody i funkcje, gdy wymagane zależności są imitowane. Natomiast pierwsze z wymienionych, nazywane również testami integracji, umożliwiają systemowi przetestowanie wycinka funkcjonalności, łącznie z zależnościami.

Programowanie sterowane testami ma kilka zalet:

- mniejsza ilość błędów i defektów;
- znacznie efektywniejsza baza kodu;
- pewność, że kod działa prawidłowo i nie prowadzi do uszkodzenia istniejącej funkcjonalności.

## Ciągłe wdrażanie i integracja

Ciągłe wdrażanie (ang. *continuous deployment*) to zestaw technik pozwalających na szybkie dostarczanie klientom nowych funkcji, poprawek znalezionych błędów i usprawnień. Obejmuje zautomatyzowane testowanie i wdrażanie. W takim podejściu następuje zmniejszenie obciążenia związanego z ręcznie wykonywaną pracą i skrócenie czasu oczekiwania na informacje zwrotne. Im szybciej programista będzie w stanie otrzymać informacje od klientów, tym szybciej dostarczy produkt, co przekłada się na przewagę nad konkurencją. Wiele startupów kilkukrotnie w ciągu dnia przeprowadza wdrażanie produktu, podczas gdy cykl wydań w korporacjach i dużych firmach może wynosić nawet 6 – 12 miesięcy.

Zaletami podejścia opartego na ciągłym wdrażaniu jest skrócenie czasu oczekiwania na informacje zwrotne i zmniejszenie ilości pracy, którą trzeba wykonywać ręcznie.

Mamy ciągłe dostarczanie, ciągłe wdrażanie i ciągłą integrację. Wprawdzie istnieją pewne różnice między tymi pojęciami, ale w idealnej sytuacji dobrze jest wykorzystać te wszystkie trzy podejścia, aby zapewnić możliwość szybszego opracowywania produktu.

Oto kilka z najpopularniejszych rozwiązań w zakresie ciągłej integracji:

- *Jenkins* (*https://jenkins.io/*) — rozszerzalny i oferowany jako open source serwer ciągłej integracji;
- *CircleCI* (*https://circleci.com/*) — pozwala na szybsze dostarczanie lepszej jakości kodu źródłowego;
- *Travis CI* (*https://travis-ci.org/*) — hosting usługi ciągłej integracji dla społeczności open source.

## Programowanie w parach

Programowanie w parach polega na tym, że dwie osoby będące programistami współpracują ze sobą w jednym środowisku. Jedna z nich jest kierującym, natomiast druga obserwatorem. Pierwsza tworzy kod, druga zaś asystuje, obserwując pierwszą i sugerując rozwiązania. Następnie programiści zamieniają się rolami.



Kup książkę Poleć książkę

Kierujący ma bardziej taktyczną rolę skoncentrowania się na bieżącym zadaniu. Z kolei rola obserwatora jest bardziej strategiczna — ma patrzeć szerzej, dostrzegać błędy i znajdować sposoby na usprawnienie algorytmu.

Technika programowania w parach charakteryzuje się następującymi zaletami:

- mniejsza i efektywniejsza baza kodu, w której znajduje się mniej błędów i defektów;
- wiedza jest przekazywana między programistami, którzy ze sobą współpracują (niestety między nimi może dochodzić do konfliktów, co zdarza się nie tak rzadko).

# Definicje back-endu

Back-end to inna nazwa serwera. Jest to wszystko, co znajduje się po przeglądarce WWW. Obejmuje platformy serwerowe, takie jak PHP, Python, Java, Ruby i oczywiście Node.js, a także bazy danych i inne technologie.

Na szczęście dzięki istnieniu nowoczesnych rozwiązań z zakresu back-endu jako usługi (ang. *Back-end as a Service* — BaaS) można całkowicie pominąć etap programowania back-endu. Wystarczy dołączyć pojedynczy znacznik `<script>`, a uzyskasz dostęp do działającej w czasie rzeczywistym bazy danych wraz z możliwością umieszczenia w niej logiki kontroli poziomu dostępu (ang. *Access Level Control* — ALC), weryfikacji itd. Istnieje wiele usług oferujących odmienne poziomy BaaS. Najpopularniejsze i najłatwiejsze w użyciu to Firebase (*https://firebase.google.com/*) i Parse (*https://parseplatform.org/*).

Jeżeli będziesz musiał samodzielnie przygotować kod działający po stronie serwera, Node.js będzie Twoim narzędziem.

## Node.js

Node.js to dostępna jako open source i sterowana zdarzeniami technologia asynchronicznego wejścia-wyjścia umożliwiająca tworzenie skalowalnych i efektywnie działających serwerów WWW. Składa się z opracowanego przez Google silnika JavaScriptu o nazwie V8 oraz wielu modułów C++. Zajmująca się przetwarzaniem w chmurze firma Joyent (obecnie przejęta przez Samsunga) na początku rozwijała Node.js, a teraz pracami nad tą technologią kieruje fundacja Node.

Przeznaczeniem i wykorzystaniem Node.js jest zapewnienie nieblokujących operacji wejścia-wyjścia, które przyśpieszają działanie całego rozwiązania. Nieblokujące operacje wejścia-wyjścia nie są nowością, istniały w postaci NIO dla Javy, Twisted dla Pythona i EventMachine dla Ruby. W porównaniu do wymienionych cechą charakterystyczną Node.js jest to, że technologia została opracowana od początku jako łatwa w użyciu, podczas gdy rozwiązania zastosowane w innych językach są raczej skomplikowane.

Za dość interesujące można uznać to, że JavaScript nie był nawet pierwszym językiem dla Node.js. Oparta na JavaScripcie implementacja Node.js była dopiero trzecią, po wcześniejszych próbach wykorzystujących języki programowania Ruby i C++.

Sam w sobie Node.js nie jest frameworkiem jak Ruby on Rails. Bardziej można go porównać do pary PHP i Apache. Listę najważniejszych frameworków Node.js przedstawię w rozdziale 6.

Oto podstawowe zalety korzystania z Node.js:

- Istnieje ogromne prawdopodobieństwo, że programista zna już JavaScript, ponieważ jest to praktycznie standard w programowaniu na platformach internetowej i mobilnej.
- Używanie jednego języka do programowania front-endu i back-endu pozwala przyśpieszyć proces tworzenia kodu źródłowego. Programista nie musi pamiętać o różnych składniach i dokonywać nieustannie tzw. przełączania kontekstu. Poznawanie metod i klas odbywa się znacznie szybciej.



Kup książkę Poleć książkę

- Node.js umożliwia szybkie prototypowanie, co przekłada się na szybsze dostarczenie produktu klientom. To daje przewagę nad konkurencją, która korzysta z mniej zwinnych technologii, takich jak PHP i MySQL.
- Node.js opracowano do tworzenia aplikacji działających w czasie rzeczywistym dzięki wykorzystaniu gniazd sieciowych.

Node.js ewoluuje bardzo szybko. Więcej informacji o aktualnym stanie prac nad Node.js znajdziesz w oficjalnym blogu pod adresem *https://nodejs.org/en/blog/*.

## NoSQL i MongoDB

Oferowana przez huMONGOus MongoDB to charakteryzująca się wysoką wydajnością nierelacyjna baza danych przeznaczona do przechowywania ogromnych ilości informacji (*https://www.mongodb.com/*). Koncepcja NoSQL pojawiła się, gdy tradycyjne systemy zarządzania relacyjnymi bazami danych (ang. *Relational Database Management System* — RDBMS) nie były w stanie sprostać wyzwaniom związanym z ogromnymi ilościami danych.

Oto najważniejsze zalety stosowania MongoDB:

- *Skalowalność* — ze względu na naturę rozproszoną wiele serwerów i centrów danych może mieć powielające się dane.
- *Wysoka wydajność* — MongoDB zapewnia wysoką efektywność podczas przechowywania i pobierania danych, co po części wynika z braku relacji między elementami i kolekcjami w bazie danych.
- *Elastyczność* — magazyn typu klucz – wartość idealnie sprawdza się podczas prototypowania, ponieważ nie wymaga od programistów znajomości schematu, stosowania na stałe zdefiniowanych modeli danych ani skomplikowanych migracji.

## Przetwarzanie w chmurze

Przetwarzanie w chmurze składa się z następujących komponentów:

- infrastruktura jako usługa (ang. *Infrastructure as a Service* — IaaS), czyli m.in. Rackspace i Amazon Web Services (AWS);
- platforma jako usługa (ang. *Platform as a Service* — PaaS), czyli m.in. Heroku i Microsoft Azure;
- back-end jako usługa (ang. *Back-end as a Service* — BaaS), czyli najnowsze i najlepsze podejście obejmujące m.in. Compose i Firebase;
- oprogramowanie jako usługa (ang. *Software as a Service* — SaaS), czyli m.in. aplikacje Google i Salesforce.com (*https://www.salesforce.com/eu/?ir=1*).

Platformy przetwarzania w chmurze oferują następujące korzyści:

- skalowalność, np. nowe egzemplarze można utworzyć w ciągu zaledwie paru minut;
- łatwość wdrażania, np. przekazanie projektu do Heroku sprowadza się do wydania polecenia `git push`;
- plany typu „zapłać za użyte zasoby”, które pozwalają użytkownikom na dodawanie pamięci operacyjnej i masowej wedle potrzeb i na żądanie;
- dodatki ułatwiające instalację i konfigurację baz danych, serwerów aplikacji, pakietów itd.;
- zapewnienie bezpieczeństwa i pomoc techniczna.



Kup książkę Poleć książkę

Rozwiązania typu PaaS i BaaS są idealne podczas prototypowania i tworzenia minimalnej wersji produktu oraz ogólnie na wczesnym etapie startupu.

Oto lista najpopularniejszych rozwiązań w zakresie PaaS:

- Heroku (*https://www.heroku.com/*),
- AWS Elastic Beanstalk (*https://aws.amazon.com/elasticbeanstalk/*).
- Microsoft Azure (*https://azure.microsoft.com/pl-pl/*).

## Żądania HTTP i odpowiedzi na nie

Każde żądanie HTTP i udzielona na nie odpowiedź składają się z następujących komponentów:

- *Nagłówek* — informacje o kodowaniu, wielkości treści, pochodzeniu żądania, typie treści itd.
- *Treść* — najczęściej parametry lub dane, które są przekazywane do serwera lub odsyłane klientowi.

Poza wymienionymi żądanie HTTP zawiera jeszcze następujące elementy:

- *Metoda* — istnieje kilka metod wykonywania żądania HTTP, a do najpopularniejszych zaliczamy `GET`, `POST`, `PUT` i `DELETE`.
- *Adres URL* — protokół, host, port, ścieżka, np. *https://webaplog.com/es6*.
- *Ciąg tekstowy zapytania* — wszystko to, co znajduje się po znaku zapytania w adresie URL, np. `?q=rpjs&page=20`.

## API RESTful

API RESTful (ang. *Representational State Transfer*) zyskało dużą popularność ze względu na ogromne oczekiwania, aby systemy rozproszone stały się bezstanowe, ponieważ aplikacje bezstanowe znacznie lepiej poddają się skalowaniu. Takie rozwiązanie wymaga, aby każda transakcja (żądanie) zawierała wystarczającą ilość informacji do zidentyfikowania stanu klienta. Ten standard jest uznawany za bezstanowy, gdyż żadne informacje o stanie klienta nie są przechowywane w serwerze. Tym samym istnieje możliwość, aby każde żądanie zostało obsłużone przez inny system.

Oto niektóre cechy charakterystyczne API RESTful:

- lepsza obsługa skalowalności ze względu na to, że różne komponenty mogą być niezależnie wdrażane w oddzielnych serwerach;
- łatwiejsze użycie niż SOAP (ang. *Simple Object Access Protocol*) z powodu prostszej struktury i braku konieczności podawania metody w adresie URL;
- wykorzystywanie metod HTTP: `GET`, `POST`, `DELETE`, `PUT`, `OPTIONS` itd.

W tabeli 1.1 wymieniłem przykłady prostych metod CRUD (ang. *Create*, *Read*, *Update*, *Delete*) w API RESTful pozwalających na obsługę kolekcji `Messages`.

REST nie jest protokołem, to raczej architektura, która charakteryzuje się większą elastycznością niż protokół SOAP. Dlatego adres URL API RESTful może mieć postać np. */messages/list.html* lub */messages/list.xml*, o ile chcesz zapewnić obsługę wymienionych formatów. Jednak w większości przypadków programiści używają formatu JSON bez żadnych rozszerzeń, np. */messages* i */messages/{id}*.

Metody `PUT` i `DELETE` są idempotentne, co oznacza, że jeśli serwer otrzyma dwa lub więcej podobnych żądań, efekt końcowy będzie dokładnie taki sam. Natomiast metoda `GET` nie jest idempotentna, ponieważ operacja odczytu jest uznawana za bezpieczną w przypadku jej powtórzeń. Z kolei metoda `POST` nie jest idempotentna, gdyż jej działanie wpływa na stan i może mieć efekt uboczny w przypadku jej powtórzenia.



Kup książkę Poleć książkę

***Tabela 1.1.*** *Przykłady metod CRUD API RESTful*

| Metoda | URL | Opis |
|---|---|---|
| `GET` | */messages.json* | Zwraca listę komunikatów w formacie JSON |
| `PUT` | */messages.json* | Aktualizuje lub zastępuje wszystkie komunikaty i zwraca informacje o stanie lub błędzie w formacie JSON |
| `POST` | */messages.json* | Tworzy nowy komunikat i zwraca jego identyfikator w formacie JSON |
| `GET` | */messages/{id}.json* | Zwraca komunikat o identyfikatorze `{id}` w formacie JSON |
| `PUT` | */messages/{id}.json* | Zastępuje komunikat o identyfikatorze `{id}` |
| `PATCH` | */messages/{id}.json* | Aktualizuje komunikat o identyfikatorze `{id}` |
| `DELETE` | */messages/{id}.json* | Usuwa komunikat o identyfikatorze `{id}` |

Architekturę REST wykorzystasz w następnym rozdziale podczas tworzenia back-endu w Node.js oraz klienta w Backbone.js.

# Podsumowanie

W ten sposób dotarliśmy do końca rozdziału 1. Omówiłem w nim wybrane podstawowe koncepcje programowania internetowego. Masz zatem solidne podstawy dla materiału przedstawionego w pozostałej części książki. Jestem pewien, że znasz już następujące koncepcje:

- HTML,
- CSS,
- typy i obiekty JavaScriptu,
- metody zwinne,
- Node.js,
- NoSQL,
- żądanie HTTP,
- API RESTful.

Mimo to dobrze jest je sobie podszlifować, ponieważ jest ich całkiem sporo i są obszerne. Teoria nie jest wystarczająco użyteczna i interesująca, jeśli nie wiadomo, jak ją zastosować i jakie przynosi korzyści w rzeczywistym kodzie. Dlatego szybko przejdę do konfiguracji technicznej, aby bez zbędnej zwłoki móc przystąpić do tworzenia projektów.



Kup książkę Poleć książkę


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

# Oto najlepszy komplet technologii dla JavaScriptu!

JavaScript jest dziś jednym z ulubionych języków programowania projektantów profesjonalnych aplikacji internetowych. Deweloperzy mogą korzystać z kilku znakomitych technologii do tworzenia front-endu i back-endu aplikacji. Największy potencjał mają Node.js, MongoDB, Backbone.js, Parse.com, Heroku i MS Azure. Są wciąż unowocześniane, a każda kolejna wersja dostarcza innowacyjnych rozwiązań. Mimo to wyszukiwanie informacji potrzebnych programiście w danej chwili bywa problemem: oficjalna dokumentacja i najpopularniejsze zasoby internetowe nie zawsze ułatwiają rozwiązywanie konkretnych zagadnień programistycznych.

Oto podręcznik szybkiego prototypowania oprogramowania za pomocą kilku najciekawszych technologii programowania dla internetu i urządzeń mobilnych. Zamieszczono tu sporo przykładów praktycznych i niewiele teorii, którą ograniczono do minimum pozwalającego na zrozumienie działania poszczególnych rozwiązań. Dokładnie omówiono praktyczne podstawy typowego sposobu działania aplikacji internetowej: pobieranie, wyświetlanie i dodawanie nowych danych. Przykłady zostały utworzone z zastosowaniem wielu technologii. Dzięki temu książka stanowi kolekcję starannie wybranych i przetestowanych fragmentów kodu. Poszczególne technologie zostały zaprezentowane w ich najnowszych wersjach.

W tej książce między innymi:

- przygotowanie środowiska pracy
- praca z biblioteką jQuery, Bootstrap i Less
- praktyczne wprowadzenie do Backbone.js
- wdrażanie aplikacji w PaaS, Heroku i MS Azure
- wykorzystanie MongoDB w aplikacjach

**Azat Mardan** jest doświadczonym inżynierem programowania, autorem książek i uznanym trenerem. Obecnie pracuje jako inżynier oprogramowania w firmie Indeed. Przedtem był zatrudniony w jednym z najważniejszych banków USA, a jeszcze wcześniej tworzył aplikacje dla kilku agencji rządowych w Waszyngtonie. Poza tworzeniem kodu zajmuje się działalnością edukacyjną: prowadzi zajęcia dla kilku uznanych firm i pisze bloga o technologii dostępnego pod adresem: *http://webapplog.com/*.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
Sprawdź nasze szkolenia!
SZKOLENIA
AKADEMIA IT & BUSINESS
WWW.SZKOLENIA.HELION.PL
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-283-5750-1
9 788328 357501
INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 49,00 zł
Apress