



Angular

Profesjonalne techniki programowania

Wydanie IV

—

Adam Freeman

Tytuł oryginału: Pro Angular 9: Build Powerful and Dynamic Web Apps, 4th Edition

Tłumaczenie: Robert Górczyński

ISBN: 978-83-283-7543-7

First published in English under the title Pro Angular 9: Build Powerful and Dynamic Web Apps by Adam Freeman, edition: 4

Copyright © Adam Freeman, 2020

This edition has been translated and published under licence from APress Media, LLC, part of Springer Nature. APress Media, LLC, part of Springer Nature takes no responsibility and shall not be made liable for the accuracy of the translation.

Polish edition copyright © 2021 by Helion S.A.
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 jakiegokolwiek 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 wydawca 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 wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion S.A.
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:
<https://ftp.helion.pl/przyklady/angup4.zip>

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/angup4>
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	17
	O korektorze merytorycznym	19
Część I	Zaczynamy	21
Rozdział 1.	Rozpoczęcie pracy	23
	Co powinieneś wiedzieć?	24
	Jaka jest struktura niniejszej książki?	24
	Część I. Zaczynamy	24
	Część II. Praca z Angularem	24
	Część III. Zaawansowane funkcje Angulara	24
	Czego nie znajdę w książce?	24
	Co jest potrzebne do programowania z użyciem Angulara?	25
	Jak skonfigurować środowisko programistyczne?	25
	Co zrobić w przypadku problemów podczas wykonywania przykładów?	25
	Co zrobić w sytuacji, gdy znajdę błąd w książce?	26
	Czy w książce znajdę wiele przykładów?	26
	Gdzie znajdę przykładowe fragmenty kodu?	27
	Jak mogę skontaktować się z autorem?	27
	Podsumowanie	28
Rozdział 2.	Pierwsza aplikacja w Angularze	29
	Przygotowanie środowiska programistycznego	29
	Instalowanie Node.js	29
	Instalowanie pakietu angular-cli	31
	Instalowanie edytora tekstu	32
	Instalowanie przeglądarki WWW	32
	Utworzenie i przygotowanie projektu	32
	Utworzenie projektu	32
	Uruchamianie narzędzi programistycznych	33

Dodanie frameworka Angular do projektu	35
Utworzenie danych modelu	36
Utworzenie klasy listy rzeczy do zrobienia	38
Wyświetlanie danych użytkownikowi	38
Uaktualnienie komponentu	39
Nadawanie stylu elementom HTML	43
Wyświetlenie listy rzeczy do zrobienia	45
Utworzenie dwukierunkowego mechanizmu dołączania danych	48
Dodawanie zadań	51
Wyświetlanie wykonanych zadań	53
Podsumowanie	56
Rozdział 3. Umieszczenie Angulara w kontekście	57
Sytuacje, w których Angular sprawdza się doskonale	58
Aplikacje dwukierunkowe oraz w postaci pojedynczej strony	58
Porównanie Angulara z Reactem i Vue.js	59
Wzorzec MVC	60
Model	62
Kontroler (komponent)	64
Widok (szablon)	64
Usługi typu RESTful	65
Najczęstsze pułapki podczas stosowania wzorca MVC	67
Umieszczenie logiki w nieodpowiednim miejscu	67
Przyjęcie formatu danych stosowanego w magazynie danych	68
Wystarczająca ilość wiedzy, aby wpaść w tarapaty	68
Podsumowanie	69
Rozdział 4. Krótkie wprowadzenie do HTML i CSS	71
Utworzenie przykładowego projektu	71
Język HTML	72
Element samozamykający się	73
Atrybut	74
Stosowanie atrybutu bez wartości	74
Cytowanie literałów w atrybucie	74
Treść elementu	75
Struktura dokumentu	75
Framework Bootstrap	77
Stosowanie podstawowych klas Bootstrap	78
Użycie frameworka Bootstrap do nadawania stylu tabeli	81
Użycie frameworka Bootstrap do tworzenia formularzy HTML	82
Użycie frameworka Bootstrap do utworzenia układu opartego na siatce	84
Podsumowanie	89
Rozdział 5. Wprowadzenie do języków JavaScript i TypeScript — część 1.	91
Utworzenie przykładowego projektu	92
Podstawowy sposób pracy	94
Używanie poleceń	95

Definiowanie i używanie funkcji	95
Definiowanie funkcji z parametrami	96
Definiowanie funkcji zwracającej wartość	98
Używanie funkcji jako argumentu innej funkcji	98
Używanie zmiennych i typów	99
Używanie zmiennej domknięcia	101
Używanie typów podstawowych	101
Używanie operatorów JavaScript	104
Używanie konstrukcji warunkowych	104
Operator równości kontra operator identyczności	105
Jawna konwersja typu	106
Praca z tablicą	108
Użycie literału tablicy	108
Odczyt i modyfikacja zawartości tablicy	109
Wyświetlenie zawartości tablicy	109
Używanie operatora rozwinięcia	110
Używanie wbudowanych metod przeznaczonych do pracy z tablicą	111
Podsumowanie	112
Rozdział 6. Wprowadzenie do języków JavaScript i TypeScript — część 2.	113
Utworzenie przykładowego projektu	114
Praca z obiektami	114
Używanie literału obiektu	115
Używanie funkcji jako metod	115
Zdefiniowanie klasy	116
Praca z modułami JavaScript	120
Utworzenie i stosowanie modułu	120
Użyteczne funkcje języka TypeScript	123
Używanie adnotacji typu	124
Używanie krotki	128
Używanie typów indeksowanych	129
Używanie modyfikatorów dostępu	130
Podsumowanie	131
Rozdział 7. SportsStore — rzeczywista aplikacja	133
Utworzenie przykładowego projektu	134
Instalowanie dodatkowych pakietów npm	134
Utworzenie usługi sieciowej typu RESTful	135
Utworzenie pliku HTML	138
Utworzenie struktury katalogów	138
Uruchomienie przykładowej aplikacji	138
Uruchomienie usługi sieciowej typu RESTful	139
Przygotowanie funkcji projektu w Angularze	139
Uaktualnienie komponentu głównego	140
Uaktualnienie modułu głównego	140
Analiza pliku typu bootstrap	141

Utworzenie danych modelu	142
Utworzenie klas modelu	142
Utworzenie fikcyjnego źródła danych	143
Utworzenie repozytorium modelu	144
Utworzenie modułu funkcjonalnego	145
Rozpoczęcie pracy nad utworzeniem sklepu internetowego	145
Utworzenie szablonu i komponentu sklepu internetowego	146
Utworzenie modułu funkcjonalnego dla sklepu	147
Uaktualnienie komponentu i modułu głównego	148
Dodawanie funkcji związanych z produktem	149
Wyświetlanie szczegółów produktu	149
Dodawanie możliwości wyboru kategorii	151
Dodawanie stronicowania produktów	153
Utworzenie własnej dyrektywy	156
Podsumowanie	159
Rozdział 8. SportsStore — zamówienia i zakupy	161
Utworzenie przykładowego projektu	161
Utworzenie koszyka na zakupy	161
Utworzenie modelu koszyka na zakupy	162
Utworzenie komponentów podsumowania koszyka na zakupy	163
Integracja koszyka na zakupy ze sklepem internetowym	165
Zaimplementowanie routingu	168
Utworzenie komponentów zawartości koszyka i procesu składania zamówienia	169
Utworzenie i zastosowanie konfiguracji routingu	170
Nawigacja po aplikacji	171
Zabezpieczanie tras	174
Ukończenie funkcji obsługi zawartości koszyka	176
Przetwarzanie zamówienia	178
Rozbudowa modelu	179
Pobieranie szczegółów zamówienia	181
Używanie usługi sieciowej typu RESTful	185
Zastosowanie źródła danych	186
Podsumowanie	188
Rozdział 9. SportsStore — administrowanie	189
Utworzenie przykładowej aplikacji	189
Utworzenie modułu	190
Konfigurowanie systemu routingu	192
Nawigacja do administracyjnego adresu URL	193
Implementowanie uwierzytelniania	195
Poznajemy system uwierzytelniania	195
Rozbudowa źródła danych	196
Konfigurowanie usługi uwierzytelniania	197
Włączenie uwierzytelniania	198
Rozbudowa źródła danych i repozytoriów	200

Utworzenie struktury funkcji administracyjnych	203
Utworzenie komponentów w postaci miejsc zarezerwowanych	204
Przygotowanie wspólnej treści i modułu funkcjonalnego	205
Zaimplementowanie funkcji obsługi produktu	207
Zaimplementowanie funkcji obsługi zamówienia	210
Podsumowanie	213
Rozdział 10. SportsStore — wdrożenie	215
Utworzenie przykładowej aplikacji	215
Dodawanie funkcjonalności progresywnych	215
Instalowanie pakietu PWA	216
Buforowanie danych	216
Reakcja na zmiany dotyczące połączenia z internetem	217
Przygotowanie aplikacji do wdrożenia	219
Utworzenie pliku danych	219
Utworzenie serwera	220
Zmiana adresu URL usługi sieciowej w klasie repozytorium	222
Zbudowanie i przetestowanie aplikacji	222
Testowanie funkcjonalności progresywnej	223
Umieszczenie aplikacji SportsStore w kontenerze	224
Instalowanie narzędzia Docker	225
Przygotowanie aplikacji	225
Utworzenie kontenera	225
Uruchamianie aplikacji	227
Podsumowanie	228
Część II Praca z Angulariem	229
Rozdział 11. Poznajemy projekt i narzędzia Angulara	231
Utworzenie nowego projektu w Angularze	231
Struktura projektu tworzonego w Angularze	233
Katalog kodu źródłowego	234
Katalog pakietów	235
Używanie narzędzi programistycznych	239
Programistyczny serwer HTTP	240
Proces kompilacji	240
Paczka aplikacji	242
Paczka skryptów typu polyfills	243
Paczka stylów	243
Stosowanie lintera	245
Sposób działania aplikacji zbudowanych z użyciem Angulara	247
Dokument HTML	248
Przygotowanie aplikacji	249
Moduł główny aplikacji zbudowanej z użyciem Angulara	250
Komponent Angulara	251
Wyświetlanie treści	252

Proces kompilowania dla środowiska produkcyjnego	253
Wcześniejsza kompilacja	254
Wczytywanie źróźnicowane	254
Uruchomienie produkcyjnej wersji aplikacji	255
Rozpoczęcie programowania w Angularze z użyciem TypeScriptu	256
Utworzenie modelu	256
Utworzenie komponentu i szablonu	260
Konfiguracja modułu głównego Angulara	261
Podsumowanie	262
Rozdział 12. Mechanizm dołączania danych	263
Utworzenie przykładowego projektu	264
Jednokierunkowe dołączanie danych	265
Cel dla operacji dołączania danych	267
Wyrażenie dołączania danych	268
Nawias kwadratowy	269
Element HTML	271
Używanie standardowego dołączania właściwości i atrybutu	271
Używanie standardowego dołączania właściwości	271
Używanie dołączania danych w postaci interpolacji ciągu tekstowego	273
Używanie dołączania atrybutu	274
Przypisywanie klas i stylów	275
Używanie dołączania klasy	275
Używanie dołączania stylu	280
Uaktualnienie danych w aplikacji	283
Podsumowanie	286
Rozdział 13. Stosowanie wbudowanych dyrektyw	287
Utworzenie przykładowego projektu	288
Używanie wbudowanej dyrektywy	290
Używanie dyrektywy ngIf	291
Używanie dyrektywy ngSwitch	293
Używanie dyrektywy ngFor	295
Używanie dyrektywy ngTemplateOutlet	305
Ograniczenia jednokierunkowego dołączania danych	307
Używanie wyrażeń idempotentnych	307
Kontekst wyrażenia	310
Podsumowanie	312
Rozdział 14. Używanie zdarzeń i formularzy	313
Utworzenie przykładowego projektu	314
Importowanie modułu obsługi formularzy	315
Przygotowanie komponentu i szablonu	315
Używanie dołączania zdarzenia	316
Używanie danych zdarzenia	320
Używanie zmiennej odwołania w szablonie	322

Używanie dwukierunkowego dołączania danych	324
Używanie dyrektywy ngModel	326
Praca z formularzem HTML	327
Dodawanie formularza do przykładowej aplikacji	328
Dodawanie weryfikacji danych formularza	330
Weryfikacja danych całego formularza	340
Używanie formularza opartego na modelu	346
Włączenie funkcji tworzenia formularza opartego na modelu	346
Zdefiniowanie klas modelu formularza	347
Używanie modelu do weryfikacji danych	350
Generowanie elementów na podstawie modelu	354
Utworzenie własnych reguł weryfikacji formularza	355
Zastosowanie własnej reguły weryfikacji	356
Podsumowanie	358
Rozdział 15. Tworzenie dyrektywy atrybutu	359
Utworzenie przykładowego projektu	360
Utworzenie prostej dyrektywy atrybutu	363
Zastosowanie własnej dyrektywy	364
Uzyskanie w dyrektywie dostępu do danych aplikacji	365
Odczyt atrybutów elementu HTML	365
Utworzenie właściwości dołączania danych wejściowych	367
Reagowanie na zmianę właściwości danych wejściowych	370
Utworzenie własnego zdarzenia	372
Dołączanie do własnego zdarzenia	375
Utworzenie operacji dołączania danych w elemencie HTML	376
Używanie dwukierunkowego dołączania danych w elemencie HTML	377
Wyeksportowanie dyrektywy do użycia w zmiennej szablonu	381
Podsumowanie	383
Rozdział 16. Tworzenie dyrektywy strukturalnej	385
Utworzenie przykładowego projektu	386
Utworzenie prostej dyrektywy strukturalnej	387
Implementowanie klasy dyrektywy strukturalnej	389
Włączanie dyrektywy strukturalnej	391
Używanie zwięzłej składni dyrektywy strukturalnej	392
Utworzenie iteracyjnej dyrektywy strukturalnej	393
Dostarczanie dodatkowych danych kontekstu	396
Używanie zwięzłej składni dyrektywy strukturalnej	398
Zmiany danych na poziomie właściwości	399
Zmiany danych na poziomie kolekcji	401
Pobieranie treści elementu HTML	410
Wykonywanie zapytań do wielu elementów potomnych w treści	415
Otrzymywanie powiadomień o zmianie zapytania	416
Podsumowanie	418

Rozdział 17. Poznajemy komponent	419
Utworzenie przykładowego projektu	420
Strukturyzacja aplikacji z użyciem komponentów	421
Utworzenie nowych komponentów	423
Definiowanie szablonu	426
Zakończenie restrukturyzacji komponentu głównego	437
Używanie stylów komponentu	437
Definiowanie zewnętrznych stylów komponentu	439
Używanie zaawansowanych funkcji stylów	440
Pobieranie treści szablonu	447
Podsumowanie	449
Rozdział 18. Tworzenie i używanie potoku	451
Utworzenie przykładowego projektu	452
Poznajemy potok	455
Utworzenie własnego potoku	456
Rejestrowanie własnego potoku	458
Zastosowanie własnego potoku	459
Łączenie potoków	460
Utworzenie potoku nieczystego	461
Używanie wbudowanych potoków	465
Formatowanie wartości liczbowych	466
Formatowanie wartości walutowych	468
Formatowanie wartości procentowych	471
Formatowanie wartości daty i godziny	472
Zmiana wielkości znaków ciągu tekstowego	476
Serializowanie danych jako JSON	478
Podział danych tablicy	478
Formatowanie par klucz – wartość	480
Pobieranie wartości	481
Generowanie liczby mnogiej dla pewnej wartości	483
Podsumowanie	484
Rozdział 19. Poznajemy usługę	485
Utworzenie przykładowego projektu	486
Poznajemy problem związany z przekazywaniem obiektów	487
Prezentacja problemu	488
Wykorzystanie mechanizmu wstrzykiwania zależności do rozprowadzania obiektu jako usługi	492
Zadeklarowanie zależności w innych elementach konstrukcyjnych	499
Problem izolacji testu	504
Izolowanie komponentów za pomocą usług i mechanizmu wstrzykiwania zależności	505
Dokończenie zastosowania usług w aplikacji	509
Uaktualnienie komponentu głównego i szablonu	509
Uaktualnianie komponentów potomnych	510
Podsumowanie	512

Rozdział 20. Poznajemy dostawcę usługi	513
Utworzenie przykładowego projektu	515
Używanie dostawcy usługi	516
Używanie dostawcy klasy	518
Używanie dostawcy wartości	526
Używanie dostawcy fabryki	528
Używanie dostawcy istniejącej usługi	531
Używanie dostawcy lokalnego	532
Ograniczenia pojedynczego obiektu usługi	532
Utworzenie dostawcy lokalnego w komponencie	534
Alternatywa dla dostawcy	536
Kontrolowanie spełniania zależności	541
Podsumowanie	543
Rozdział 21. Używanie i tworzenie modułu	545
Utworzenie przykładowego projektu	546
Moduł główny	548
Właściwość imports	549
Właściwość declarations	550
Właściwość providers	550
Właściwość bootstrap	550
Utworzenie modułu funkcjonalnego	553
Utworzenie modułu modelu	554
Utworzenie modułu narzędziowego	559
Utworzenie modułu wraz z komponentami	564
Podsumowanie	569
Część III Zaawansowane funkcje Angulara	571
Rozdział 22. Utworzenie przykładowego projektu	573
Rozpoczęcie pracy nad przykładowym projektem	573
Dodawanie i konfigurowanie pakietu Bootstrap CSS	574
Utworzenie struktury projektu	574
Utworzenie modułu modelu	574
Utworzenie typu danych produktu	574
Utworzenie źródła danych i repozytorium	575
Zakończenie pracy nad modulem modelu	576
Utworzenie modułu core	577
Utworzenie współdzielonej usługi informacji o stanie	577
Utworzenie komponentu tabeli	577
Utworzenie komponentu formularza	579
Zakończenie pracy nad modulem core	581
Utworzenie modułu messages	581
Utworzenie modelu i usługi	582
Utworzenie komponentu i szablonu	582
Zakończenie pracy nad modulem messages	583
Zakończenie pracy nad projektem	583
Podsumowanie	585

Rozdział 23. Poznajemy bibliotekę Reactive Extensions	587
Utworzenie przykładowego projektu	588
Poznajemy problem	589
Rozwiązanie problemu za pomocą biblioteki Reactive Extensions	592
Klasa Observable	592
Klasa Observer	594
Klasa Subject	595
Używanie potoku async	597
Używanie potoku async wraz z niestandardowym potokiem	598
Skalowanie w górę modułów funkcjonalnych aplikacji	599
Wyjście poza podstawy	602
Filtrowanie zdarzeń	602
Transformowanie zdarzeń	604
Otrzymywanie jedynie odmiennych zdarzeń	607
Pobieranie i pomijanie zdarzeń	609
Podsumowanie	611
Rozdział 24. Wykonywanie asynchronicznych żądań HTTP	613
Utworzenie przykładowego projektu	614
Konfigurowanie modułu funkcjonalności modelu	615
Utworzenie pliku danych	615
Uaktualnienie komponentu formularza	616
Uruchomienie przykładowego projektu	616
Poznajemy usługę sieciową typu RESTful	617
Zastąpienie statycznego źródła danych	618
Utworzenie usługi nowego źródła danych	619
Konfigurowanie źródła danych	621
Używanie źródła danych typu REST	621
Zapisywanie i usuwanie danych	622
Konsolidowanie żądań HTTP	626
Wykonywanie żądań między domenami	627
Używanie żądań JSONP	628
Konfigurowanie nagłówków żądania	630
Obsługa błędów	632
Wygenerowanie komunikatów przeznaczonych dla użytkownika	633
Faktyczna obsługa błędów	634
Podsumowanie	636
Rozdział 25. Routing i nawigacja — część 1.	637
Utworzenie przykładowego projektu	638
Rozpoczęcie pracy z routingiem	640
Utworzenie konfiguracji routingu	641
Utworzenie komponentu routingu	643
Uaktualnienie modułu głównego	643
Zakończenie konfiguracji	644
Dodawanie łączy nawigacyjnych	645
Efekt zastosowania routingu	647

Dokończenie implementacji routingu	649
Obsługa zmiany trasy w komponencie	650
Używanie parametrów trasy	652
Nawigacja w kodzie	659
Otrzymywanie zdarzeń nawigacyjnych	661
Usunięcie dołączania zdarzeń i obsługującego je kodu	663
Podsumowanie	665
Rozdział 26. Routing i nawigacja — część 2.	667
Utworzenie przykładowego projektu	668
Dodawanie komponentów do projektu	670
Używanie znaków wieloznacznych i przekierowań	674
Używanie znaków wieloznacznych w trasie	674
Używanie przekierowania w trasie	676
Nawigacja w komponencie	677
Reakcja na zmiany w routingu	678
Nadawanie stylu łączu aktywnej trasy	681
Poprawienie przycisku Wszystkie	684
Utworzenie trasy potomnej	684
Utworzenie outletu trasy potomnej	686
Uzyskanie dostępu do parametrów z poziomu tras potomnych	687
Podsumowanie	691
Rozdział 27. Routing i nawigacja — część 3.	693
Utworzenie przykładowego projektu	693
Zabezpieczanie tras	694
Opóźnienie nawigacji za pomocą resolvera	696
Uniemożliwienie nawigacji dzięki użyciu strażników	703
Dynamiczne wczytywanie modułów funkcjonalnych	715
Utworzenie prostego modułu funkcjonalnego	716
Dynamiczne wczytywanie modułu	717
Zabezpieczanie dynamicznie wczytywanego modułu	720
Odwołania do nazwanych outletów	723
Utworzenie dodatkowych elementów <router-outlet>	724
Nawigacja podczas użycia wielu outletów	726
Podsumowanie	728
Rozdział 28. Animacje	729
Utworzenie przykładowego projektu	730
Wyłączenie opóźnienia HTTP	731
Uproszczenie szablonu tabeli i konfiguracji routingu	732
Rozpoczęcie pracy z animacjami Angulara	733
Włączenie modułu animacji	734
Utworzenie animacji	735
Zastosowanie animacji	738
Przetestowanie animacji	740
Poznajemy wbudowane stany aplikacji	743

Poznajemy transformację elementu	744
Utworzenie transformacji dla wbudowanych stanów	744
Kontrolowanie animacji transformacji	745
Poznajemy grupy stylów animacji	751
Zdefiniowanie najczęściej używanych stylów	
w grupie przeznaczonej do wielokrotnego użycia	751
Używanie transformacji elementu	752
Zastosowanie stylów frameworka CSS	755
Podsumowanie	756
Rozdział 29. Testy jednostkowe w Angularze	759
Utworzenie przykładowego projektu	761
Utworzenie prostego testu jednostkowego	763
Praca z frameworkiem Jasmine	764
Testowanie komponentu Angulara	766
Praca z klasą TestBed	766
Testowanie operacji dołączania danych	770
Testowanie komponentu wraz z zewnętrznym szablonem	772
Testowanie zdarzeń komponentu	774
Testowanie właściwości danych wyjściowych	776
Testowanie właściwości danych wejściowych	777
Testowanie operacji asynchronicznej	780
Testowanie dyrektywy Angulara	782
Podsumowanie	784

ROZDZIAŁ 4.



Krótkie wprowadzenie do HTML i CSS

Do świata programowania aplikacji internetowych programiści docierają wieloma różnymi ścieżkami i dlatego nie zawsze mają opanowane podstawowe technologie, na których oparte są te aplikacje. W tym rozdziale przedstawię krótkie wprowadzenie do języka HTML i frameworka Bootstrap CSS, którego użyłem do nadania stylu przykładom przedstawionym w książce. Z kolei w rozdziałach 5. i 6. znajdziesz wprowadzenie do języków JavaScript i TypeScript oraz informacje niezbędne do zrozumienia przykładów zaprezentowanych w pozostałej części książki. Jeżeli jesteś doświadczonym programistą, możesz pominąć te trzy rozdziały i od razu przejść do rozdziału 7., w którym wykorzystam framework Angular do utworzenia znacznie bardziej skomplikowanej i rzeczywistej aplikacji internetowej.

Utworzenie przykładowego projektu

Materiał przedstawiony w tym rozdziale wymaga tylko prostego projektu. Pracę rozpocznij od utworzenia katalogu o nazwie *HtmlCssPrimer*, a następnie umieść w nim plik *package.json* wraz z kodem przedstawionym na listingu 4.1.

Listing 4.1. Zawartość pliku *package.json* w katalogu *HtmlCssPrimer*

```
{
  "dependencies": {
    "bootstrap": "4.4.1"
  }
}
```

Aby pobrać i zainstalować pakiety wymienione w pliku *package.json*, z poziomu katalogu *HtmlCssPrimer* wydaj poniższe polecenie:

```
$ npm install
```

Następnie w katalogu *HtmlCssPrimer* utwórz nowy plik o nazwie *index.html* i umieść w nim kod przedstawiony na listingu 4.2.

Listing 4.2. Zawartość pliku *index.html* w katalogu *HtmlCssPrimer*

```

<!DOCTYPE html>
<html>
<head>
  <title>Lista zadań</title>
  <meta charset="utf-8" />
  <link href="node_modules/bootstrap/dist/css/bootstrap.min.css"
        rel="stylesheet" />
</head>
<body class="m-1">
  <h3 class="bg-primary text-white p-3">Lista Adama</h3>

  <div class="my-1">
    <input class="form-control" />
    <button class="btn btn-primary mt-1">Dodaj</button>
  </div>

  <table class="table table-striped table-bordered">
    <thead>
      <tr>
        <th>Zadanie</th>
        <th>Wykonane</th>
      </tr>
    </thead>
    <tbody>
      <tr><td>Kupić kwiaty</td><td>Nie</td></tr>
      <tr><td>Kupić buty</td><td>Nie</td></tr>
      <tr><td>Odebrać bilety</td><td>Tak</td></tr>
      <tr><td>Zadzwoń do Janka</td><td>Nie</td></tr>
    </tbody>
  </table>
</body>
</html>

```

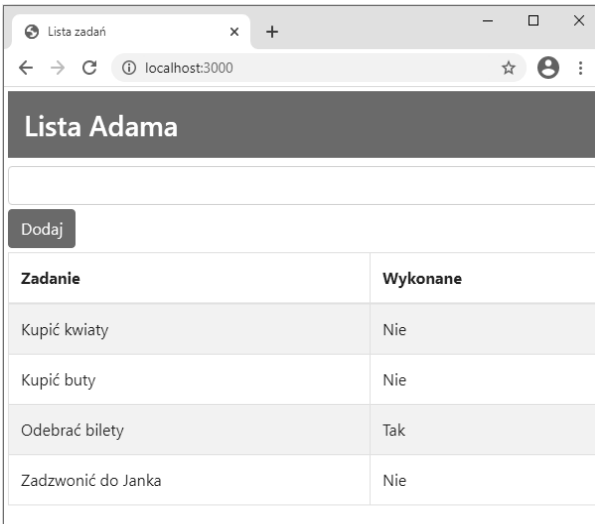
To jest kod HTML zawierający prostą listę rzeczy do zrobienia, podobną do utworzonej za pomocą frameworka Angular w rozdziale 2. Aby pobrać i uruchomić programistyczny serwer HTTP, z poziomu katalogu *HtmlCssPrimer* wydaj poniższe polecenie:

```
$ npx lite-server@2.5.4
```

Pakiet *lite-server* zawiera programistyczny serwer HTTP, który po wykryciu zmiany w plikach będzie automatycznie odświeżał stronę w przeglądarce WWW. Po pobraniu pakietu nastąpi uruchomienie serwera, a na ekranie zostanie wyświetlone nowe okno lub karta przeglądarki WWW wraz z treścią pokazaną na rysunku 4.1. (Jeżeli konfiguracja systemu uniemożliwia automatyczne otwieranie okien, możesz ręcznie wpisać adres *http://localhost:3000* w pasku adresu przeglądarki WWW).

Język HTML

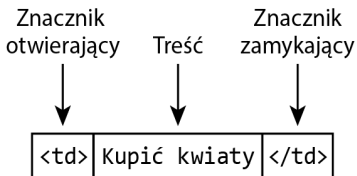
Sercem języka HTML jest **element**, który wskazuje przeglądarce WWW rodzaj treści umieszczonej w danym miejscu dokumentu HTML. Poniżej przedstawiłem element HTML pochodzący z dokumentu zaprezentowanego na listingu 4.2.



Rysunek 4.1. Efekt uruchomienia przykładowej aplikacji

```
...
<td>Kupić kwiaty</td>
...
```

Jak pokazałem na rysunku 4.2, element ten składa się z trzech części — znaczników: otwierającego i zamykającego oraz z treści.



Rysunek 4.2. Struktura prostego elementu HTML

Nazwą tego elementu (określaną również mianem **nazwy znacznika** lub po prostu **znacznikiem**) jest `<td>`. Ten znacznik wskazuje przeglądarce WWW, że umieszczona w nim treść powinna być traktowana jako komórka tabeli. Znacznik otwierający zawiera się w nawiasie ostrym (znaki `< i >`), natomiast znacznik zamykający posiada dodatkowo ukośnik, `/`, umieszczony tuż po znaku otwierającego nawiasu ostrego (`<`). Wszystko to, co znajduje się między znacznikami, jest treścią elementu — może nią być tekst, jak w omawianym przykładzie `Kupić kwiaty`, bądź też inne elementy HTML.

Element samozamykający się

W specyfikacji HTML zdefiniowano również elementy, które mogą nie zawierać treści. Tego rodzaju elementy są określane mianem **samozamykających się** (lub też *void*) i nie zawierają znacznika zamykającego, jak pokazałem poniżej.

```
...
<input />
...
```

Element samozamykający się jest zdefiniowany w jednym znaczniku, należy dodać ukośnik / przed ostatnim nawiasem ostrym (znak >). Element `<input>` jest najczęściej używanym elementem samozamykającym się i służy do tego, by umożliwić użytkownikowi podanie danych wejściowych za pomocą pola tekstowego, przycisku opcji lub pola wyboru. W dalszych rozdziałach wielokrotnie spotkasz się z wykorzystaniem tego elementu.

Atrybut

Istnieje możliwość przekazania przeglądarce WWW informacji dodatkowych poprzez dodanie do elementu tak zwanego **atrybutu**. Poniżej przedstawiłem przykład atrybutu w elemencie pochodzącym z dokumentu zdefiniowanego na listingu 4.2.

```
...
<link href="node_modules/bootstrap/dist/css/bootstrap.min.css" rel="stylesheet" />
...
```

To jest znacznik `<link>` odpowiedzialny za wczytanie treści do dokumentu HTML. Mamy tutaj dwa atrybuty zaznaczone pogrubioną czcionką, aby można było je łatwiej dostrzec. Atrybut zawsze jest definiowany jako część znacznika początkowego, a ponadto posiada **nazwę i wartość**.

W omawianym przykładzie istnieją dwa atrybuty o nazwach `href` i `rel`. W przypadku znacznika `<link>` atrybut `href` określa treść przeznaczoną do zaimportowania, natomiast atrybut `rel` wskazuje przeglądarce WWW rodzaj importowanej treści. Dlatego też przedstawione tutaj atrybuty elementu `<link>` nakazują przeglądarce WWW import pliku *bootstrap.min.css* i potraktowanie go jako arkusza stylów, ponieważ wskazany plik zawiera style CSS.

Stosowanie atrybutu bez wartości

Nie każdy atrybut wymaga wartości. W takim przypadku zdefiniowanie atrybutu stanowi dla przeglądarki WWW sygnał, że dla danego elementu oczekiwany jest pewien rodzaj zachowania. Poniżej przedstawiłem przykład elementu wraz z atrybutem niewymagającym wartości (ten element nie pochodzi z listingu 4.2).

```
...
<input class="form-control" required />
...
```

Powyższy znacznik `<input>` zawiera dwa atrybuty. Pierwszy, `class`, powoduje przypisanie wartości, podobnie jak w poprzednim przykładzie. Natomiast drugi atrybut to po prostu słowo `required`. To jest przykład atrybutu niewymagającego wartości.

Cytowanie literałów w atrybucie

Spora część funkcjonalności frameworka Angular opiera swoje działanie na atrybutach elementów HTML. W większości przypadków wartości atrybutów są podawane w postaci wyrażeń JavaScript, podobnie jak pokazałem poniżej w elemencie pochodzącym z kodu przedstawionego w rozdziale 2.

```
...
<td [ngSwitch]="item.complete">
...
```

Atrybut zastosowany dla znacznika `<td>` nakazuje Angularowi odczyt wartości właściwości o nazwie `complete` w obiekcie przypisanym zmiennej `i` tem. Będą się zdarzać sytuacje, gdy konieczne będzie przekazanie konkretnej wartości zamiast oczekiwania od Angulara odczytania wartości z danych modelu. W takim przypadku wymagane jest zastosowanie dodatkowego cytowania wskazującego frameworkowi, że ma do czynienia z literałem. Spójrz na poniższy fragment kodu.

```
...
<td [ngSwitch]="'jabłka'">
...
```

Wartość atrybutu zawiera ciąg tekstowy `jabłka` ujęty zarówno w apostrofy, jak i w cudzysłów. Podczas analizy wartości atrybutu Angular odkrywa użycie apostrofów i przetwarza tę wartość jako literał ciągu tekstowego.

Treść elementu

Element może zawierać tekst oraz inne elementy, jak pokazałem poniżej.

```
...
<thead>
  <tr>
    <th>Zadanie</th>
    <th>Zrobione</th>
  </tr>
</thead>
...
```

Elementy w dokumencie HTML tworzą naturalną hierarchię. Dlatego też element `<html>` zawiera `<body>`, a ten z kolei zawiera elementy definiujące treść strony, które dalej mogą zawierać kolejne elementy itd. W przypadku kodu przedstawionego na listingu 4.2 element `<thead>`

zawiera elementy `<tr>`, które z kolei zawierają elementy `<th>`. Zagnieżdżanie elementów w ten sposób jest kluczową koncepcją w HTML, ponieważ znaczenie zewnętrznych elementów nadaje elementom znajdującym się wewnątrz.

Struktura dokumentu

Istnieją pewne kluczowe elementy definiujące podstawową strukturę dokumentu HTML: `<!DOCTYPE>`, `<html>`, `<head>` i `<body>`. Poniżej przedstawiłem powiązania między wymienionymi elementami, przy czym usunąłem pozostałą treść z kodu strony internetowej.

```
<!DOCTYPE html>
<html>
<head>
  ...treść nagłówka...
</head>
<body>
```

```
...treść główna dokumentu...
</body>
</html>
```

Każdy z wymienionych elementów ma do odegrania konkretną rolę w dokumencie HTML. Element <DOCTYPE> informuje przeglądarkę WWW, że dany dokument to HTML, a dokładnie dokument HTML5. Wcześniejsze wersje specyfikacji HTML wymagały podania informacji dodatkowych. Na przykład poniżej przedstawiłem element <DOCTYPE> dla dokumentu HTML4:

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

Element <html> wskazuje obszar dokumentu zawierający treść HTML. Ten element zawsze zawiera dwa inne elementy o kluczowym znaczeniu dla struktury: <head> i <body>. Nie zamierzam tutaj omawiać poszczególnych elementów specyfikacji HTML — jest ich po prostu zbyt wiele. Dokładne omówienie specyfikacji HTML5 wymagało napisania ponad 850 stron w innej mojej książce zatytułowanej *HTML5. Przewodnik encyklopedyczny* również wydanej przez Helion. Dlatego w tabeli 4.1 zamieściłem jedynie krótkie omówienie elementów wykorzystanych do zdefiniowania dokumentu HTML, którego kod przedstawiłem na listingu 4.2. To powinno Ci pomóc w zrozumieniu, jak element HTML wskazuje przeglądarce WWW rodzaj znajdującej się w nim treści.

Tabela 4.1. Elementy HTML użyte w przykładowym dokumencie

Element	Opis
<DOCTYPE>	Określa typ treści w dokumencie HTML.
<body>	Wskazuje obszar dokumentu, w którym znajdują się elementy dotyczące treści.
<button>	Przycisk — bardzo często używany w celu wysłania formularza sieciowego do serwera.
<div>	Ogólny element, bardzo często wykorzystywany do nadawania struktury dokumentu w celach prezentacyjnych.
<h3>	Nagłówek trzeciego poziomu.
<head>	Wskazuje obszar dokumentu, w którym znajdują się elementy dotyczące metadanych.
<html>	Wskazuje obszar dokumentu, w którym znajduje się kod HTML (to najczęściej cały dokument).
<input>	Ten element definiuje pole używane do pobrania pojedynczych danych od użytkownika.
<link>	Za pomocą tego znacznika można zaimportować do dokumentu HTML wskazaną treść.
<meta>	To jest znacznik zawierający opisowe dane dotyczące dokumentu, na przykład zastosowane kodowanie znaków.
<table>	Tabela używana do umieszczenia treści w wierszach i kolumnach.
<tbody>	Właściwe dane tabeli (w przeciwieństwie do nagłówka i stopki).
<td>	Komórka wyświetlająca treść w wierszu tabeli.

Tabela 4.1. Elementy HTML użyte w przykładowym dokumencie (ciąg dalszy)

Element	Opis
<th>	Komórka nagłówka w wierszu tabeli.
<thead>	Nagłówek tabeli.
<title>	Tytuł dokumentu, używany przez przeglądarkę WWW w celu ustawienia tytułu okna lub karty.
<tr>	Wiersz tabeli.

Poznajemy obiektowy model dokumentu (DOM)

Gdy przeglądarka WWW wczytuje i przetwarza dokument HTML, wtedy tworzy tak zwany **obektowy model dokumentu** (ang. *document object model*, **DOM**). We wspomnianym modelu DOM obiekty JavaScript są używane do przedstawienia każdego elementu dokumentu. Dzięki temu model DOM stanowi mechanizm, za pomocą którego można w programowy sposób operować na zawartości dokumentu HTML.

W aplikacji zbudowanej na podstawie frameworka Angular bardzo rzadko zachodzi potrzeba bezpośredniej pracy z modelem DOM. Mimo wszystko bardzo ważne jest poznanie, w jaki sposób przeglądarka WWW utrzymuje „żywy” model dokumentu HTML reprezentowany przez obiekty JavaScript. Podczas modyfikacji tych obiektów przez Angular przeglądarka WWW uaktualnia wyświetlaną treść i tym samym odzwierciedla wprowadzone zmiany. To jest jedna z kluczowych koncepcji, na których opierają się aplikacje internetowe. Gdyby nie było możliwości modyfikacji modelu DOM, to niemożliwe by było utworzenie aplikacji internetowych działających po stronie klienta.

Framework Bootstrap

Elementy HTML wskazują przeglądarce WWW rodzaj znajdującej się w nich treści, ale nie zawierają żadnych informacji o sposobie, w jaki ta treść ma być wyświetlona. Informacje o sposobie wyświetlania treści są dostarczane przez kaskadowe arkusze stylów (ang. *cascading style sheets*, **CSS**). Style CSS składają się z obszernego zbioru **właściwości**, za pomocą których można skonfigurować praktycznie każdy aspekt wyglądu elementu. Ponadto zawierają zbiór **selektorów** pozwalających na stosowanie wspomnianych właściwości.

Jednym z największych problemów związanych ze stylami CSS jest to, że niektóre przeglądarki WWW nieco odmiennie interpretują właściwości. To może prowadzić do różnic w sposobie wyświetlania treści elementów HTML w poszczególnych urządzeniach. Monitorowanie i niwelowanie wspomnianych różnic okazało się trudnym zadaniem, stąd pojawienie się frameworków CSS, które mają pomagać programistom aplikacji internetowych w nadawaniu w prosty i spójny sposób stylu tworzonej przez nich treści HTML.

Jednym z frameworków, który zyskał dużą popularność, jest Bootstrap. Składa się on ze zbioru klas CSS, które można stosować w elementach i tym samym nadawać im spójny styl. Klasy można stosować także dla kodu JavaScript, co pozwala na przeprowadzenie dodatkowych usprawnień tego kodu. W książce zdecydowałem się na wykorzystanie stylów Bootstrap CSS, ponieważ pozwalają one na nadanie przykładom odpowiedniego wyglądu bez konieczności samodzielnego definiowania stylów CSS w każdym rozdziale. Natomiast nie użyłem żadnego kodu Bootstrap JavaScript, ponieważ interaktywność przykładów jest zapewniona przez framework Angular.

Nie będę się zbyt zagłębiać w szczegóły dotyczące frameworka Bootstrap, ponieważ nie stanowi on tematu książki. Jednak znajdziesz tutaj wystarczającą ilość informacji, aby zauważyć, które części przykładów są funkcjami Angulara, a które stylami Bootstrap. Framework Bootstrap oferuje znacznie więcej funkcji, niż tu przedstawiam. Więcej informacji na jego temat znajdziesz w witrynie <http://getbootstrap.com/>.

Stosowanie podstawowych klas Bootstrap

Style Bootstrap są stosowane za pomocą atrybutu `class` używanego do połączenia powiązanych ze sobą elementów. Atrybut `class` jest używany nie tylko do stosowania stylów CSS — choć jest to najczęściej spotykany sposób jego wykorzystania — ale stanowi podstawę pozwalającą na działanie frameworka Bootstrap i podobnych. Poniżej przedstawiłem przykład elementu HTML wraz z atrybutem `class`:

```
...  
<button class="btn btn-primary mt-1">Dodaj</button>  
...
```

Atrybut `class` przypisuje znacznikowi `<button>` trzy klasy, których nazwy są rozdzielone spacjami: `btn`, `btn-primary` i `mt-1`. Odpowiadają one kolekcjom stylów zdefiniowanych przez framework Bootstrap, jak to przedstawiłem w tabeli 4.2.

Tabela 4.2. Trzy klasy CSS użyte w znaczniku `<button>`

Klasa	Opis
<code>btn</code>	Ta klasa powoduje zastosowanie podstawowego stylu dla przycisku. Może być stosowana dla znaczników <code><button></code> i <code><a></code> , aby zapewnić ich spójny wygląd.
<code>btn-primary</code>	Ta klasa powoduje zastosowanie stylu kontekstu dostarczającego wskazówkę o przeznaczeniu przycisku. Więcej informacji na ten temat znajdziesz w sekcji „Użycie klas kontekstu”.
<code>mt-1</code>	Ta klasa zapewnia przestrzeń między górną krawędzią elementu i otaczającą go treścią. Więcej informacji na ten temat znajdziesz w sekcji „Użycie marginesu i dopełnienia”.

Użycie klas kontekstu

Jedną z podstawowych zalet użycia frameworka CSS takiego jak Bootstrap jest uproszczenie procesu tworzenia spójnego motywu w aplikacji. Bootstrap definiuje zbiór tak zwanych **stylów kontekstu** używanych do nadania spójnego stylu powiązanym ze sobą elementom. Wspomniane style kontekstu, wymienione w tabeli 4.3, są używane w nazwach klas powodujących nadanie stylów Bootstrap elementom.

Bootstrap oferuje klasy pozwalające na stosowanie stylów kontekstu w różnego rodzaju elementach. Poniżej pokazałem przykład nadania stylu kontekstu `primary` elementowi `<h3>`. Ten element pochodzi z pliku `index.html` utworzonego na początku rozdziału.

```
...  
<h3 class="bg-primary text-white p-3">Lista Adama</h3>  
...
```

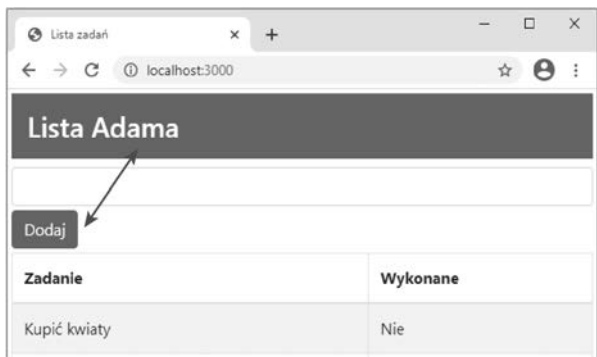
Tabela 4.3. Style kontekstu we frameworku Bootstrap

Klasa	Opis
primary	Ten kontekst służy do wskazania głównej akcji lub obszaru treści.
secondary	Ten kontekst służy do wskazania dodatkowego obszaru treści.
success	Ten kontekst służy do wskazania danych wyjściowych oznaczających sukces.
info	Ten kontekst służy do przedstawienia informacji dodatkowych.
warning	Ten kontekst służy do wyświetlania ostrzeżeń.
danger	Ten kontekst służy do wyświetlania poważnych ostrzeżeń.
muted	Ten kontekst służy do zmniejszenia znaczenia treści.
dark	Ten kontekst służy do zwiększenia kontrastu przez użycie ciemnego koloru.
white	Ten kontekst służy do zwiększenia kontrastu przez użycie koloru białego.

Jedną z klas przypisanych elementowi jest `bg-primary`. Powoduje ona zdefiniowanie koloru tła elementu na podstawie stylu koloru kontekstu. Poniżej przedstawiłem ten sam styl kontekstu zastosowany dla elementu `<button>`.

```
...
<button class="btn btn-primary mt-1">Dodaj</button>
...
```

Klasa stylu `btn-primary` powoduje nadanie elementowi `<button>` lub `<a>` stylu na podstawie kolorów użytych w stylu kontekstu. Wykorzystanie tego samego kontekstu do nadania stylu różnym elementom gwarantuje zachowanie spójności i wzajemne uzupełnianie się stylu treści, jak pokazałem na rysunku 4.3. Czerwoną strzałką wskazałem te elementy, dla których zostały zastosowane style kontekstu.

**Rysunek 4.3.** Użycie stylów kontekstu pozwala na zachowanie spójności

Użycie marginesu i dopełnienia

Bootstrap oferuje zestaw klas pomocniczych przeznaczonych do zdefiniowania dopełnienia, czyli przestrzeni między krawędzią elementu i jego treścią, oraz marginesu, czyli przestrzeni między krawędzią elementu i otaczających go elementów. Zaletą zastosowania tego rodzaju klas jest stała wielkość przestrzeni w aplikacji.

Nazwy tych klas mają doskonale zdefiniowany wzorzec. Poniżej przedstawiłem znacznik `<body>` pochodzący z pliku *index.html* utworzonego na początku rozdziału. W tym znaczniku została zastosowana klasa definiująca margines.

```
...
<body class="m-1">
...
```

Klasy definiujące margines i dopełnienie w elementach stosują doskonale opracowany schemat nazewnictwa. Najpierw mamy literę *m* oznaczającą margines (ang. *margin*) lub *p* oznaczającą dopełnienie (ang. *padding*). Następnie znajduje się myślnik. Dalej znajduje się myślnik. Na końcu mamy liczbę określającą ilość wolnej przestrzeni — 0 w przypadku braku przestrzeni bądź też 1, 2 lub 3 dla coraz większej ilości przestrzeni. Istnieje możliwość dodania litery wskazującej na dodanie wolnej przestrzeni tylko dla określonej krawędzi: *t* oznacza górną (ang. *top*) krawędź, *b* oznacza dolną (ang. *bottom*), *l* oznacza lewą (ang. *left*), *r* oznacza prawą (ang. *right*), *x* oznacza lewą i prawą, *y* zaś górną i dolną.

Aby pomóc w zrozumieniu tego schematu nazewnictwa, w tabeli 4.4 wymieniłem klasy użyte w pliku *index.html*.

Tabela 4.4. Przykładowe klasy Bootstrap definiujące margines i dopełnienie

Klasa	Opis
p-1	Ta klasa powoduje zastosowanie dopełnienia dla wszystkich krawędzi elementu.
m-1	Ta klasa powoduje zastosowanie marginesu dla wszystkich krawędzi elementu.
mt-1	Ta klasa powoduje zastosowanie marginesu dla górnej krawędzi elementu.
mb-1	Ta klasa powoduje zastosowanie marginesu dla dolnej krawędzi elementu.

Zmiana wielkości elementu

Sposób nadawania stylu pewnym elementom można zmienić przez użycie klasy wskazującej na wielkość elementu. Tego rodzaju klasa powstaje przez połączenie nazwy klasy podstawowej, myślnika i ciągu tekstowego *lg* lub *sm*. Na listingu 4.3 przedstawiłem przykład dodania do pliku *index.html* elementów `<button>` wraz z dostarczającymi przez framework Bootstrap klasami zmieniającymi wielkość elementu.

Listing 4.3. Użycie w pliku *index.html* klasy zmieniającej wielkość elementu

```
<!DOCTYPE html>
<html>
<head>
  <title>Lista zadań</title>
  <meta charset="utf-8" />
  <link href="node_modules/bootstrap/dist/css/bootstrap.min.css"
        rel="stylesheet" />
</head>
<body class="m-1">
  <h3 class="bg-primary text-white p-3">Lista Adama</h3>

  <div class="my-1">
    <input class="form-control" />
```



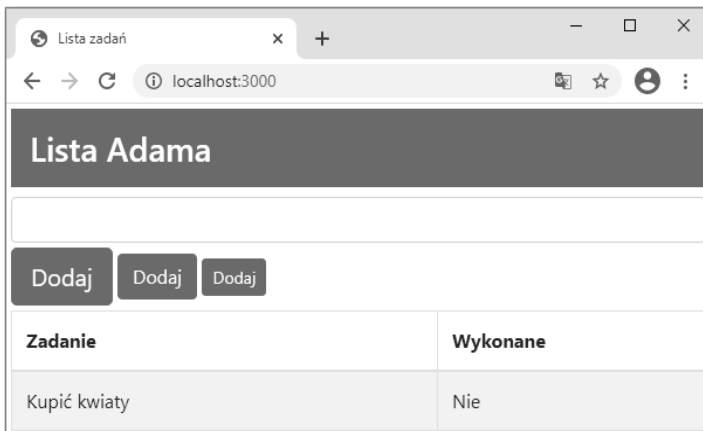
```

<button class="btn btn-lg btn-primary mt-1">Dodaj</button>
<button class="btn btn-primary mt-1">Dodaj</button>
<button class="btn btn-sm btn-primary mt-1">Dodaj</button>
</div>

<table class="table table-striped table-bordered">
  <thead>
    <tr>
      <th>Zadanie</th>
      <th>Wykonane</th>
    </tr>
  </thead>
  <tbody>
    <tr><td>Kupić kwiaty</td><td>Nie</td></tr>
    <tr><td>Kupić buty</td><td>Nie</td></tr>
    <tr><td>Odebrać bilety</td><td>Tak</td></tr>
    <tr><td>Zadzwonić do Janka</td><td>Nie</td></tr>
  </tbody>
</table>
</body>
</html>

```

Klasa `btn-lg` powoduje utworzenie dużego przycisku, natomiast klasa `btn-sm` tworzy mały przycisk. Pominięcie klasy wielkości oznacza użycie domyślnej wielkości elementu. Zwróć uwagę na połączenie klas kontekstu i wielkości. Klasy modyfikujące frameworka Bootstrap współdziałają ze sobą, aby zapewnić pełną kontrolę nad sposobem nadawania stylów elementom. Efekt wprowadzonej zmiany w pliku *index.html* pokazałem na rysunku 4.4.



Rysunek 4.4. Zmiana wielkości elementu

Użycie frameworka Bootstrap do nadawania stylu tabeli

Framework Bootstrap zawiera także style przeznaczone do użycia w tabelach. Z tej możliwości korzystam w książce. W tabeli 4.5 wymieniałem klasy CSS frameworka Bootstrap przeznaczone do nadawania stylu tabelom.

Tabela 4.5. Style Bootstrap przeznaczone do nadawania stylu tabeli

Klasa	Opis
table	Tak klasa powoduje zastosowanie ogólnych stylów dla elementów tabeli (<table>) i ich treści.
table-striped	Tak klasa powoduje zastosowanie stylu naprzemiennego w wierszach w tabeli.
table-bordered	Tak klasa powoduje zdefiniowanie obramowania dla wszystkich wierszy i kolumn.
table-hover	Tak klasa powoduje wyświetlenie innego stylu, gdy kursor myszy zostanie umieszczony nad wierszem tabeli.
table-sm	Tak klasa powoduje zmniejszenie dopełnienia w tabeli, aby tym samym utworzyć zwęższy układ.

Wszystkie wymienione klasy są stosowane bezpośrednio w elemencie <table>, jak pokazałem na listingu 4.4, na którym pogrubioną czcionką przedstawiłem style Bootstrap zastosowane w tabeli zdefiniowanej w pliku *index.html*.

Listing 4.4. Użycie klas Bootstrap do nadawania stylu tabeli

```
...
<table class="table table-striped table-bordered">
  <thead>
    <tr>
      <th>Zadanie</th>
      <th>Wykonane</th>
    </tr>
  </thead>
  <tbody>
    <tr><td>Kupić kwiaty</td><td>Nie</td></tr>
    <tr><td>Kupić buty</td><td>Nie</td></tr>
    <tr><td>Odebrać bilety</td><td>Tak</td></tr>
    <tr><td>Zadzwonić do Janka</td><td>Nie</td></tr>
  </tbody>
</table>
...
```

■ **Wskazówka** Podczas definiowania tabeli na listingu 4.4 zwróć uwagę na użycie elementu <thead>. Przeglądarka WWW automatycznie wygeneruje w <tbody> wszelkie niezbędne elementy <tr> będące bezpośrednimi elementami potomnymi <table>, o ile żaden nie został użyty. Jeśli będziesz polegał na tym zachowaniu podczas pracy z frameworkiem Bootstrap, otrzymasz dziwne wyniki, ponieważ większość klas CSS zastosowanych dla elementu <table> powoduje dodanie stylów do elementów potomnych elementu <tbody>.

Użycie frameworka Bootstrap do tworzenia formularzy HTML

Framework Bootstrap zawiera style przeznaczone do nadawania elementom formularza HTML wyglądu spójnego wraz z pozostałymi elementami w aplikacji. Na listingu 4.5 w pliku *index.html* rozbudowałem elementy formularza i tymczasowo usunąłem tabelę.

Listing 4.5. Przykład zdefiniowania w pliku `index.html` dodatkowych elementów formularza

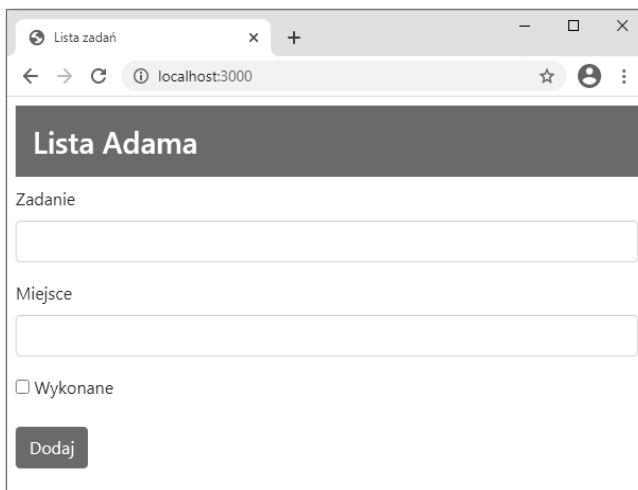
```

<!DOCTYPE html>
<html>
<head>
  <title>Lista zadań</title>
  <meta charset="utf-8" />
  <link href="node_modules/bootstrap/dist/css/bootstrap.min.css"
        rel="stylesheet" />
</head>
<body class="m-2">
  <h3 class="bg-primary text-white p-3">Lista Adama</h3>
  <form>
    <div class="form-group">
      <label>Zadanie</label>
      <input class="form-control" />
    </div>

    <div class="form-group">
      <label>Miejsce</label>
      <input class="form-control" />
    </div>
    <div class="form-group">
      <input type="checkbox" />
      <label>Wykonane</label>
    </div>
    <button class="btn btn-primary">Dodaj</button>
  </form>
</body>
</html>

```

Nadanie podstawowych stylów formularza odbywa się poprzez zastosowanie klasy `form-group` w elemencie `<div>` zawierającym znaczniki `<label>` i `<input>`, gdzie znacznik `<input>` ma przypisaną klasę `form-control`. Style frameworka Bootstrap powodują wyświetlenie znacznika `<label>` nad elementem `<input>`, który zabiera 100% dostępnej przestrzeni poziomej, jak pokazałem na rysunku 4.5.

**Rysunek 4.5.** Nadanie stylów elementom formularza

Użycie frameworka Bootstrap do utworzenia układu opartego na siatce

Framework Bootstrap zawiera klasy stylów przeznaczone do tworzenia różnego rodzaju układów opartych na siatce. Wspomniane układy mogą składać się z 1 – 12 kolumn i zapewniać obsługę układu responsywnego. (W tym przypadku „responsywny” oznacza, że układ siatki zmienia się na podstawie szerokości ekranu). Na listingu 4.6 pokazałem zmodyfikowaną wersję pliku *index.html* zawierającą układ oparty na siatce.

Listing 4.6. Użycie w pliku *index.html* układu opartego na siatce

```
<!DOCTYPE html>
<html>
<head>
  <title>Lista zadań</title>
  <meta charset="utf-8" />
  <link href="node_modules/bootstrap/dist/css/bootstrap.min.css"
        rel="stylesheet" />
  <style>
    .row > div { border: 1px solid lightgrey; padding: 10px;
                background-color: aliceblue; margin: 5px 0; }
  </style>
</head>
<body class="m-2">
  <h3>Układ oparty na siatce</h3>
  <div class="container">
    <div class="row">
      <div class="col-1">1</div>
      <div class="col-1">1</div>
      <div class="col-2">2</div>
      <div class="col-2">2</div>
      <div class="col-6">6</div>
    </div>

    <div class="row">
      <div class="col-3">3</div>
      <div class="col-4">4</div>
      <div class="col-5">5</div>
    </div>

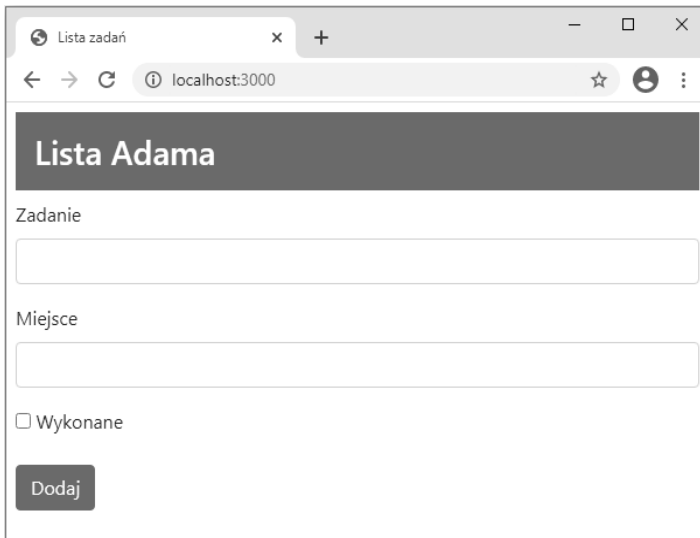
    <div class="row">
      <div class="col-6">6</div>
      <div class="col-6">6</div>
    </div>

    <div class="row">
      <div class="col-11">11</div>
      <div class="col-1">1</div>
    </div>

    <div class="row">
      <div class="col-12">12</div>
    </div>
  </div>
</body>
</html>
```

Oferowany przez framework Bootstrap układ oparty na siatce jest łatwy w użyciu. Najwyższego poziomu element `<div>` ma przypisaną klasę `container` (lub `container-fluid`, jeśli ma zabrać całą dostępną przestrzeń). Należy wskazać liczbę kolumn poprzez zastosowanie klasy `row` do elementu `<div>`. Ma to taki sam efekt jak przygotowanie układu opartego na siatce dla treści znajdującej się w danym elemencie `<div>`.

Każdy wiersz definiuje 12 kolumn. Liczbę kolumn zajmowanych przez każdy element potomny można określać przez przypisanie klasy, której nazwa to `col-liczba_kolumn`. Na przykład klasa `col-1` oznacza, że element zajmuje tylko jedną kolumnę. Z kolei `col-2` oznacza zabranie dwóch kolumn itd., aż do `col-12`. W przypadku klasy `col-12` element wypełnia cały wiersz. Na listingu 4.6 utworzyłem serię elementów `<div>` wraz z przypisanymi klasami `row`. Wspomniane elementy zawierały kolejne elementy `<div>`, w których również zastosowałem klasy `col-*`. Na rysunku 4.6 możesz zobaczyć wyświetlony w przeglądarce WWW układ oparty na siatce.



Rysunek 4.6. Utworzony za pomocą frameworka Bootstrap układ oparty na siatce

■ **Wskazówka** Framework Bootstrap nie stosuje żadnych stylów dla elementów znajdujących się w wierszu. Dlatego też na listingu 4.6 użyłem elementu `<style>` w celu zdefiniowania własnego stylu CSS określającego kolor tła, odległości między wierszami i obramowanie.

Utworzenie responsywnego układu opartego na siatce

Responsywna siatka powoduje dopasowanie układu do wielkości okna przeglądarki WWW. Podstawowym powodem tworzenia tego rodzaju układów jest umożliwienie urządzeniom mobilnym i tradycyjnym komputerom wyświetlanie tej samej treści i wykorzystanie zalet, jakie niesie ze sobą ewentualna dostępność dodatkowego miejsca na ekranie. Aby utworzyć responsywny układ oparty na siatce, klasy `col-*` w poszczególnych komórkach siatki zastąp klasami wymienionymi w tabeli 4.6.

Tabela 4.6. Klasy frameworka Bootstrap przeznaczone do utworzenia elastycznego układu opartego na siatce

Klasa	Opis
col-sm-*	Komórki siatki są wyświetlane poziomo, gdy szerokość ekranu jest większa niż 576 pikseli.
col-md-*	Komórki siatki są wyświetlane poziomo, gdy szerokość ekranu jest większa niż 768 pikseli.
col-lg-*	Komórki siatki są wyświetlane poziomo, gdy szerokość ekranu jest większa niż 992 pikseli.
col-xl-*	Komórki siatki są wyświetlane poziomo, gdy szerokość ekranu jest większa niż 1200 pikseli.

Gdy szerokość ekranu jest mniejsza niż obsługiwana przez klasę, to komórki w wierszu siatki będą układane pionowo zamiast poziomo. Na listingu 4.7 pokazałem przykład responsywnego układu opartego na siatce zdefiniowanego w pliku *index.html*.

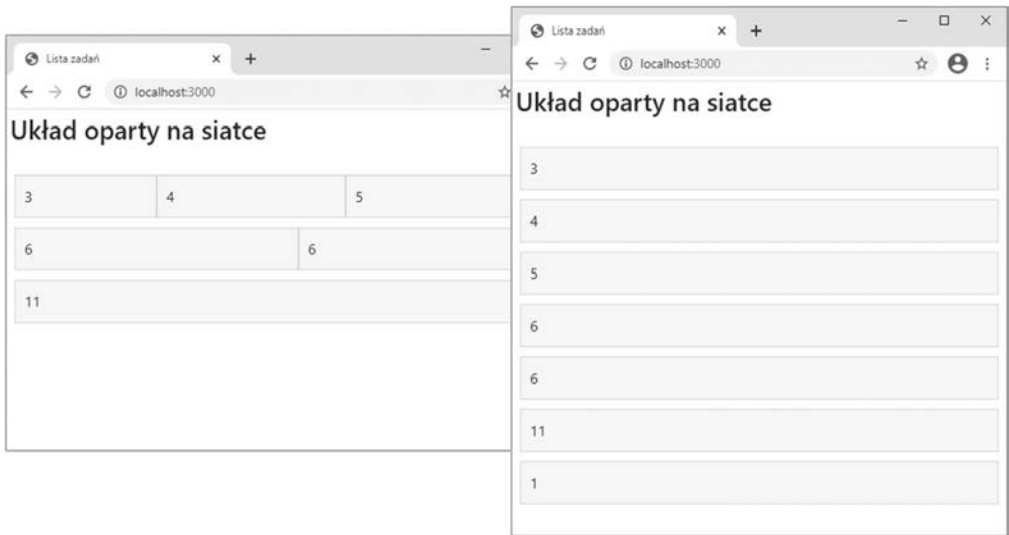
Listing 4.7. Utworzenie responsywnego układu opartego na siatce

```
<!DOCTYPE html>
<html>
<head>
  <title>Lista zadań</title>
  <meta charset="utf-8" />
  <link href="node_modules/bootstrap/dist/css/bootstrap.min.css"
        rel="stylesheet" />
  <style>
    #gridContainer {padding: 20px;}
    .row > div { border: 1px solid lightgrey; padding: 10px;
                background-color: aliceblue; margin: 5px 0; }
  </style>
</head>
<body class="m-1">
  <h3>Układ oparty na siatce</h3>
  <div id="container">
    <div class="row">
      <div class="col-sm-3">3</div>
      <div class="col-sm-4">4</div>
      <div class="col-sm-5">5</div>
    </div>

    <div class="row">
      <div class="col-sm-6">6</div>
      <div class="col-sm-6">6</div>
    </div>

    <div class="row">
      <div class="col-sm-11">11</div>
      <div class="col-sm-1">1</div>
    </div>
  </div>
</body>
</html>
```

Usunąłem pewne wiersze siatki z poprzedniego przykładu i zastąpiłem klasy col-* klasami col-sm-*. W efekcie, jeśli okno przeglądarki WWW ma szerokość większą niż 576 pikseli, to wiersze są ułożone poziomo. W przypadku mniejszej szerokości okna wiersze są układane pionowo. Efekt możesz zobaczyć na rysunku 4.7.



Rysunek 4.7. Przykład responsywnego układu opartego na siatce

Utworzenie uproszczonego układu opartego na siatce

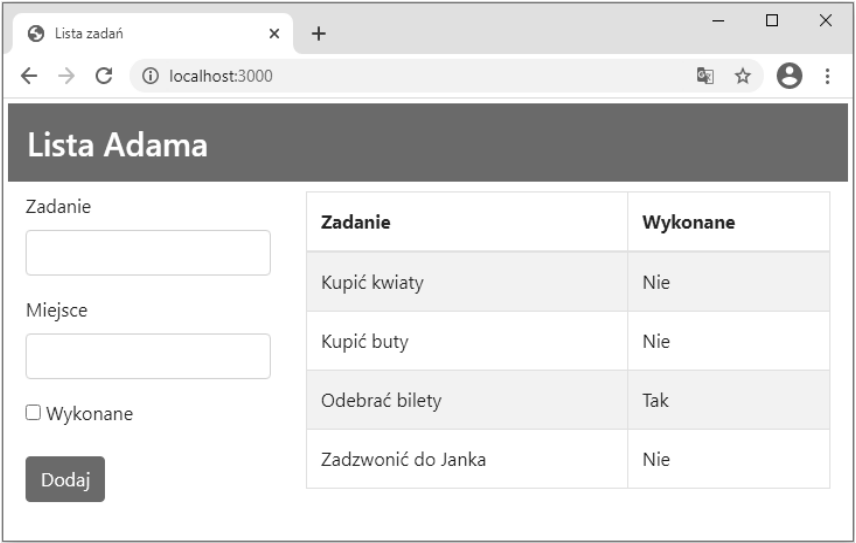
W większości przykładów w książce opierających się na siatce tworzonej za pomocą frameworka Bootstrap zdecydowałem się na zastosowanie uproszczonego podejścia. Polega ono na wyświetleniu treści w pojedynczym wierszu i podaniu jedynie liczby kolumn, jak pokazałem na listingu 4.8.

Listing 4.8. Przykład użycia w pliku `index.html` uproszczonego układu opartego na siatce

```
<!DOCTYPE html>
<html>
<head>
  <title>Lista zadań</title>
  <meta charset="utf-8" />
  <link href="node_modules/bootstrap/dist/css/bootstrap.min.css"
        rel="stylesheet" />
</head>
<body class="m-1">
  <h3 class="bg-primary text-white p-3">Lista Adama</h3>
  <div class="container-fluid">
    <div class="row">
      <div class="col-4">
        <form>
          <div class="form-group">
            <label>Zadanie</label>
            <input class="form-control" />
          </div>
          <div class="form-group">
            <label>Miejsce</label>
            <input class="form-control" />
          </div>
          <div class="form-group">
            <input type="checkbox" />
            <label>Wykonane</label>
          </div>
        </form>
      </div>
    </div>
  </div>
</body>
```

```
        <button class="btn btn-primary">Dodaj</button>
      </form>
    </div>
    <div class="col-8">
      <table class="table table-striped table-bordered">
        <thead>
          <tr>
            <th>Zadanie</th>
            <th>Opis</th>
          </tr>
        </thead>
        <tbody>
          <tr><td>Kupić kwiaty</td><td>Nie</td></tr>
          <tr><td>Kupić buty</td><td>Nie</td></tr>
          <tr><td>Odebrać bilety</td><td>Tak</td></tr>
          <tr><td>Zadzwonić do Janka</td><td>Nie</td></tr>
        </tbody>
      </table>
    </div>
  </div>
</body>
</html>
```

W kodzie przedstawionym na listingu 4.8 zastosowałem klasy col-4 i col-8 w celu wyświetlenia obok siebie elementów <div>. To pozwala na wyświetlenie w poziomie formularza oraz tabeli wraz z listą rzeczy do zrobienia, jak pokazałem na rysunku 4.8.



Rysunek 4.8. Przykład uproszczonego układu opartego na siatce

Podsumowanie

W tym rozdziale zawarłem krótkie wprowadzenie do języka HTML i frameworka CSS o nazwie Bootstrap. Jeżeli chcesz osiągać zadowalające efekty w tworzeniu aplikacji internetowych, powinieneś dobrze poznać technologie HTML i CSS. Najlepszym sposobem nauki jest praktyka. Przykłady i opisy przedstawione w tym rozdziale powinny wystarczyć na początek i zapewnić Ci solidne podstawy pomagające w zrozumieniu przykładów zaprezentowanych w dalszej części książki. W kolejnym rozdziale będę kontynuować wprowadzenie, ale tym razem zajmę się podstawowymi funkcjami języka JavaScript, używanego w tej książce.

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 

Dobre rozwiązanie dla aplikacji klienta? Z Angularem się uda!

Angular jest frameworkiem napisanym w języku TypeScript. Służy do tworzenia złożonych, nowoczesnych i wydajnych aplikacji działających po stronie klienta. Jest rozwijany przez Google i cieszy się dużym uznaniem projektantów aplikacji: ułatwia tworzenie poprawnego, czystego kodu, umożliwia bezproblemowe testowanie, a praca programisty z tym frameworkiem jest przyjemna, efektywna i satysfakcjonująca. Aby w pełni wykorzystać możliwości Angulara, konieczne trzeba się zapoznać z architekturą jego aplikacji, a także z modułami, komponentami i powiązaniem danych.

To kolejne wydanie popularnego przewodnika, będącego starannie przygotowanym połączeniem opisu architektonicznego, specyfikacji technicznej i praktycznych wskazówek. Pozwala na niemal natychmiastowe rozpoczęcie pracy w Angularze. Na początek czeka Cię zapoznanie się ze wzorcem MVC, z jego zaletami i zastosowaniem. Następnie zaczniesz pisać funkcjonalne aplikacje — stopniowo będziesz przechodzić do coraz bardziej zaawansowanych i złożonych rozwiązań. Poszczególne przykłady zostały omówione w sposób jasny, zrozumiały i spójny. Dodatkowo w książce przedstawiono mnóstwo wskazówek i szczegółów, pozwalających na efektywną pracę i uzyskiwanie najlepszych efektów. Nie zabrakło także prezentacji najczęściej spotykanych problemów i metod ich rozwiązywania.

W książce:

- dokładne omówienie wzorca MVC
- tworzenie złożonych i dynamicznych aplikacji w Angularze
- używanie narzędzi podczas tworzenia projektów w Angularze
- rozszerzanie i dostosowywanie frameworka Angular
- prowadzenie testów jednostkowych dla projektów w Angularze

Adam Freeman — jest doświadczonym programistą i autorem świetnie przyjętych książek o programowaniu w Javie. Tworzył również duże systemy rozproszone (platformy e-commerce). Zajmował stanowiska kierownicze w wielu firmach, między innymi w Netscape, Sun Microsystems, NASDAQ i bankach. Obecnie jest na zasłużonej emeryturze. Swoją czas przeznaczają na pisanie i bieganie na długich dystansach.

  helion.pl  HELION SA ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	<i>Sprawdź nasze szkolenia!</i>  SZKOLENIA AKADEMIA IT & BUSINESS HELIONSZKOLENIA.PL	KOD KORZYŚCI <i>Sięgnij po więcej ▶</i>  ISBN 978-83-283-7543-7  9 788328 375437 Cena: 149,00 zł
INFORMATYKA W NAJLEPSZYM WYDANIU		

Apress