



# Angular

Profesjonalne techniki  
programowania

Wydanie II

—

Adam Freeman

**Helion** 

apress®

Tytuł oryginału: Pro Angular, 2nd Edition

Tłumaczenie: Robert Górczyński

ISBN: 978-83-283-4231-6

Original edition copyright © 2017 by Adam Freeman.  
All rights reserved.

Polish edition copyright © 2018 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 Wydawnictwo HELION 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 Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION  
ul. Kościuszki 1c, 44-100 GLIWICE  
tel. 32 231 22 19, 32 230 98 63  
e-mail: [helion@helion.pl](mailto: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/angup2.xz>

Drogi Czytelniku!  
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres  
<http://helion.pl/user/opinie/angup2>  
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)



# Spis treści

<b>O autorze</b> .....	<b>17</b>
<b>O recenzencie technicznym</b> .....	<b>19</b>
<b>Część I Zaczynamy</b> .....	<b>21</b>
<b>Rozdział 1. Rozpoczęcie pracy</b> .....	<b>23</b>
Co powinieneś wiedzieć? .....	23
Jaka jest struktura niniejszej książki? .....	24
Część I. Zaczynamy .....	24
Część II. Praca z Angular .....	24
Część III. Zaawansowane funkcje Angular .....	24
Czy w książce znajdę wiele przykładów? .....	24
Gdzie znajdę przykładowe fragmenty kodu? .....	26
Jak skonfigurować środowisko programistyczne? .....	26
Kontakt z autorem .....	26
Podsumowanie .....	26
<b>Rozdział 2. Pierwsza aplikacja w Angular</b> .....	<b>27</b>
Przygotowanie środowiska programistycznego .....	27
Instalowanie Node.js .....	27
Instalowanie pakietu angular-cli .....	28
Instalowanie narzędzia git .....	28
Instalowanie edytora tekstu .....	29
Instalowanie przeglądarki WWW .....	30
Utworzenie i przygotowanie projektu .....	30
Utworzenie projektu .....	30
Uaktualnienie pliku package.json .....	30
Uruchamianie serwera .....	32
Edytowanie pliku HTML .....	33
Dodanie frameworka Angular do projektu .....	35
Przygotowanie pliku HTML .....	35
Utworzenie danych modelu .....	36

Utworzenie szablonu .....	38
Uaktualnienie komponentu .....	39
Połączenie wszystkiego w całość .....	41
Dodawanie funkcji do przykładowej aplikacji .....	42
Dodanie tabeli wyświetlającej listę rzeczy do zrobienia .....	43
Utworzenie dwukierunkowego mechanizmu dołączania danych .....	46
Dodawanie zadań .....	47
Podsumowanie .....	49
<b>Rozdział 3. Umieszczenie frameworka Angular w kontekście .....</b>	<b>51</b>
Sytuacje, w których Angular sprawdza się doskonale .....	52
Aplikacje dwukierunkowe oraz w postaci pojedynczej strony .....	52
Wzorzec MVC .....	54
Model .....	56
Kontroler (komponent) .....	58
Widok (szablon) .....	58
Usługi typu RESTful .....	59
Najczęstsze pułapki podczas stosowania wzorca MVC .....	61
Umieszczenie logiki w nieodpowiednim miejscu .....	61
Przyjęcie formatu danych stosowanego w magazynie danych .....	62
Wystarczająca ilość wiedzy, aby wpaść w tarapaty .....	62
Podsumowanie .....	62
<b>Rozdział 4. Krótkie wprowadzenie do HTML i CSS .....</b>	<b>63</b>
Utworzenie przykładowego projektu .....	63
Język HTML .....	64
Element samozamykający się .....	65
Atrybut .....	66
Stosowanie atrybutu bez wartości .....	66
Cytowanie literałów w atrybucie .....	66
Treść elementu .....	67
Struktura dokumentu .....	67
Framework Bootstrap .....	68
Stosowanie podstawowych klas Bootstrap .....	69
Użycie frameworka Bootstrap do nadawania stylu tabeli .....	72
Użycie frameworka Bootstrap do tworzenia formularzy HTML .....	74
Użycie frameworka Bootstrap do utworzenia układu opartego na siatce .....	75
Podsumowanie .....	80
<b>Rozdział 5. Wprowadzenie do języków JavaScript i TypeScript — część 1 .....</b>	<b>81</b>
Utworzenie przykładowego projektu .....	82
Utworzenie plików HTML i JavaScript .....	83
Konfiguracja kompilatora TypeScript .....	84
Uruchomienie przykładowego projektu .....	84
Element <script> .....	85
Użycie procedury wczytującej moduł JavaScript .....	85
Podstawowy sposób pracy .....	86

Używanie poleceń .....	87
Definiowanie i używanie funkcji .....	88
Definiowanie funkcji z parametrami .....	89
Definiowanie funkcji zwracającej wartość .....	91
Używanie funkcji jako argumentu innej funkcji .....	91
Używanie zmiennych i typów .....	92
Używanie typów podstawowych .....	94
Używanie operatorów JavaScript .....	95
Używanie konstrukcji warunkowych .....	96
Operator równości kontra operator identyczności .....	97
Jawna konwersja typu .....	97
Praca z tablicą .....	99
Użycie literału tablicy .....	100
Odczyt i modyfikacja zawartości tablicy .....	100
Wyświetlenie zawartości tablicy .....	100
Używanie wbudowanych metod przeznaczonych do pracy z tablicą .....	101
Podsumowanie .....	103
<b>Rozdział 6. Wprowadzenie do języków JavaScript i TypeScript — część 2 .....</b>	<b>105</b>
Utworzenie przykładowego projektu .....	105
Praca z obiektami .....	106
Używanie literału obiektu .....	107
Używanie funkcji jako metod .....	107
Zdefiniowanie klasy .....	108
Praca z modułami JavaScript .....	111
Utworzenie modułu .....	111
Import z modułu JavaScript .....	112
Użyteczne funkcje języka TypeScript .....	115
Używanie adnotacji typu .....	115
Używanie krotki .....	120
Używanie typów indeksowanych .....	120
Używanie modyfikatorów dostępu .....	121
Podsumowanie .....	122
<b>Rozdział 7. SportsStore — rzeczywista aplikacja .....</b>	<b>123</b>
Utworzenie przykładowego projektu .....	123
Utworzenie struktury katalogów .....	124
Instalowanie dodatkowych pakietów npm .....	124
Utworzenie usługi sieciowej typu RESTful .....	125
Utworzenie pliku HTML .....	127
Uruchomienie przykładowej aplikacji .....	128
Uruchomienie usługi sieciowej typu RESTful .....	128
Przygotowanie funkcji projektu Angular .....	129
Uaktualnienie komponentu głównego .....	129
Uaktualnienie modułu głównego .....	129
Analiza pliku typu bootstrap .....	130

Utworzenie danych modelu .....	131
Utworzenie klas modelu .....	131
Utworzenie fikcyjnego źródła danych .....	132
Utworzenie repozytorium modelu .....	133
Utworzenie modułu funkcjonalnego .....	134
Rozpoczęcie pracy nad utworzeniem sklepu internetowego .....	135
Utworzenie szablonu i komponentu sklepu internetowego .....	135
Utworzenie modułu funkcjonalnego dla sklepu .....	136
Uaktualnienie komponentu i modułu głównego .....	137
Dodawanie funkcji związanych z produktem .....	138
Wyświetlanie szczegółów produktu .....	138
Dodawanie możliwości wyboru kategorii .....	139
Dodawanie stronicowania produktów .....	141
Utworzenie własnej dyrektywy .....	143
Podsumowanie .....	147
<b>Rozdział 8. SportsStore — zamówienia i zakupy .....</b>	<b>149</b>
Utworzenie przykładowego projektu .....	149
Utworzenie koszyka na zakupy .....	149
Utworzenie modelu koszyka na zakupy .....	150
Utworzenie komponentów podsumowania koszyka na zakupy .....	151
Integracja koszyka na zakupy ze sklepem internetowym .....	153
Zaimplementowanie routingu .....	155
Utworzenie komponentów zawartości koszyka i procesu składania zamówienia .....	156
Utworzenie i zastosowanie konfiguracji routingu .....	157
Nawigacja po aplikacji .....	158
Zabezpieczanie tras .....	160
Ukończenie funkcji obsługi zawartości koszyka .....	163
Przetwarzanie zamówienia .....	165
Rozbudowa modelu .....	165
Pobieranie szczegółów zamówienia .....	168
Używanie usługi sieciowej typu RESTful .....	171
Zastosowanie źródła danych .....	172
Podsumowanie .....	173
<b>Rozdział 9. SportsStore — administracja .....</b>	<b>175</b>
Utworzenie przykładowej aplikacji .....	175
Utworzenie modułu .....	175
Konfigurowanie systemu routingu .....	178
Nawigacja do administracyjnego adresu URL .....	179
Implementowanie uwierzytelniania .....	179
Poznajemy system uwierzytelniania .....	180
Rozbudowa źródła danych .....	181
Konfigurowanie usługi uwierzytelniania .....	182
Włączenie uwierzytelniania .....	183

Rozbudowa źródła danych i repozytoriów .....	185
Utworzenie struktury funkcji administracyjnych .....	189
Utworzenie komponentów w postaci miejsc zarezerwowanych .....	190
Przygotowanie wspólnej treści i modułu funkcjonalnego .....	190
Zaimplementowanie funkcji obsługi produktu .....	193
Zaimplementowanie funkcji obsługi zamówienia .....	196
Podsumowanie .....	198
<b>Rozdział 10. SportsStore — wdrożenie .....</b>	<b>199</b>
Przygotowanie aplikacji do wdrożenia .....	199
Umieszczenie aplikacji SportsStore w kontenerze .....	199
Instalowanie narzędzia Docker .....	200
Przygotowanie aplikacji .....	200
Utworzenie kontenera .....	201
Uruchamianie aplikacji .....	202
Podsumowanie .....	203
<b>Część II Praca z Angular .....</b>	<b>205</b>
<b>Rozdział 11. Utworzenie projektu Angular .....</b>	<b>207</b>
Przygotowanie projektu Angular opartego na języku TypeScript .....	208
Utworzenie struktury katalogów projektu .....	208
Utworzenie i udostępnianie dokumentu HTML .....	208
Przygotowanie konfiguracji projektu .....	209
Dodawanie pakietów .....	209
Uruchamianie procesu obserwatora .....	217
Rozpoczęcie programowania Angular z użyciem TypeScript .....	218
Utworzenie modelu .....	221
Utworzenie szablonu i modułu głównego .....	224
Utworzenie modułu Angular .....	225
Utworzenie pliku typu bootstrap dla aplikacji .....	225
Konfigurowanie procedury wczytywania modułu JavaScript .....	226
Uaktualnianie dokumentu HTML .....	229
Uruchamianie aplikacji .....	231
Podsumowanie .....	233
<b>Rozdział 12. Mechanizm dołączania danych .....</b>	<b>235</b>
Utworzenie przykładowego projektu .....	236
Jednokierunkowe dołączanie danych .....	237
Cel dla operacji dołączania danych .....	239
Wyrażenie dołączania danych .....	239
Nawias kwadratowy .....	241
Element HTML .....	242
Używanie standardowego dołączania właściwości i atrybutu .....	242
Używanie standardowego dołączania właściwości .....	242
Używanie dołączania danych w postaci interpolacji ciągu tekstowego .....	243
Używanie dołączania atrybutu .....	245

Przypisywanie klas i stylów .....	246
Używanie dołączania klasy .....	246
Używanie dołączania stylu .....	250
Uaktualnienie danych w aplikacji .....	254
Podsumowanie .....	256
<b>Rozdział 13. Używanie wbudowanych dyrektyw .....</b>	<b>257</b>
Utworzenie przykładowego projektu .....	258
Używanie wbudowanej dyrektywy .....	259
Używanie dyrektywy ngIf .....	260
Używanie dyrektywy ngSwitch .....	262
Używanie dyrektywy ngFor .....	264
Używanie dyrektywy ngTemplateOutlet .....	273
Ograniczenia jednokierunkowego dołączania danych .....	275
Używanie wyrażeń idempotentnych .....	275
Kontekst wyrażenia .....	278
Podsumowanie .....	280
<b>Rozdział 14. Używanie zdarzeń i formularzy .....</b>	<b>281</b>
Utworzenie przykładowego projektu .....	282
Dodawanie modułu obsługi formularzy .....	282
Przygotowanie komponentu i szablonu .....	284
Używanie dołączania zdarzenia .....	285
Poznajemy właściwości zdefiniowane dynamicznie .....	287
Używanie danych zdarzenia .....	289
Używanie zmiennej odwołania w szablonie .....	290
Używanie dwukierunkowego dołączania danych .....	292
Używanie dyrektywy ngModel .....	294
Praca z formularzem HTML .....	295
Dodawanie formularza do przykładowej aplikacji .....	295
Dodawanie weryfikacji danych formularza .....	297
Weryfikacja danych całego formularza .....	306
Używanie formularza opartego na modelu .....	312
Włączenie funkcji tworzenia formularza opartego na modelu .....	312
Zdefiniowanie klas modelu formularza .....	313
Używanie modelu do weryfikacji danych .....	316
Generowanie elementów na podstawie modelu .....	319
Utworzenie własnych reguł weryfikacji formularza .....	320
Zastosowanie własnej reguły weryfikacji .....	321
Podsumowanie .....	323
<b>Rozdział 15. Tworzenie dyrektywy atrybutu .....</b>	<b>325</b>
Utworzenie przykładowego projektu .....	326
Utworzenie prostej dyrektywy atrybutu .....	328
Zastosowanie własnej dyrektywy .....	329



Uzyskanie w dyrektywie dostępu do danych aplikacji .....	330
Odczyt atrybutów elementu HTML .....	331
Utworzenie właściwości dołączania danych wejściowych .....	333
Reagowanie na zmianę właściwości danych wejściowych .....	335
Utworzenie własnego zdarzenia .....	337
Dołączanie do własnego zdarzenia .....	339
Utworzenie operacji dołączania danych w elemencie HTML .....	340
Używanie dwukierunkowego dołączania danych w elemencie HTML .....	341
Wyeksportowanie dyrektywy do użycia w zmiennej szablonu .....	345
Podsumowanie .....	347
<b>Rozdział 16. Tworzenie dyrektywy strukturalnej .....</b>	<b>349</b>
Utworzenie przykładowego projektu .....	350
Utworzenie prostej dyrektywy strukturalnej .....	350
Implementowanie klasy dyrektywy strukturalnej .....	352
Włączanie dyrektywy strukturalnej .....	354
Używanie związanej składni dyrektywy strukturalnej .....	356
Utworzenie iteracyjnej dyrektywy strukturalnej .....	357
Dostarczanie dodatkowych danych kontekstu .....	359
Używanie związanej składni dyrektywy strukturalnej .....	361
Zmiany danych na poziomie właściwości .....	362
Zmiany danych na poziomie kolekcji .....	363
Pobieranie treści elementu HTML .....	373
Wykonywanie zapytań do wielu elementów potomnych w treści .....	376
Otrzymywanie powiadomień o zmianie zapytania .....	378
Podsumowanie .....	379
<b>Rozdział 17. Poznajemy komponent .....</b>	<b>381</b>
Utworzenie przykładowego projektu .....	382
Strukturyzacja aplikacji z użyciem komponentów .....	382
Utworzenie nowych komponentów .....	384
Definiowanie szablonu .....	387
Zakończenie restrukturyzacji komponentu głównego .....	396
Używanie stylów komponentu .....	398
Definiowanie zewnętrznych stylów komponentu .....	398
Używanie zaawansowanych funkcji stylów .....	400
Pobieranie treści szablonu .....	406
Podsumowanie .....	408
<b>Rozdział 18. Tworzenie i używanie potoku .....</b>	<b>409</b>
Utworzenie przykładowego projektu .....	410
Instalowanie biblioteki typu polyfill .....	412
Poznajemy potok .....	414
Utworzenie własnego potoku .....	415
Rejestrowanie własnego potoku .....	416
Zastosowanie własnego potoku .....	417
Łączenie potoków .....	418
Utworzenie potoku nieczystego .....	419

Używanie wbudowanych potoków .....	423
Formatowanie wartości liczbowych .....	423
Formatowanie wartości walutowych .....	425
Formatowanie wartości procentowych .....	428
Formatowanie wartości daty i godziny .....	430
Zmiana wielkości znaków ciągu tekstowego .....	433
Serializowanie danych jako JSON .....	434
Podział danych tablicy .....	435
Podsumowanie .....	436
<b>Rozdział 19. Poznajemy usługę .....</b>	<b>437</b>
Utworzenie przykładowego projektu .....	438
Poznajemy problem związany z przekazywaniem obiektów .....	439
Prezentacja problemu .....	439
Wykorzystanie mechanizmu wstrzykiwania zależności do rozprowadzania obiektu jako usługi .....	444
Zadeklarowanie zależności w innych elementach konstrukcyjnych .....	449
Problem izolacji testu .....	455
Izolowanie komponentów za pomocą usług i mechanizmu wstrzykiwania zależności .....	455
Dokończenie zastosowania usług w aplikacji .....	458
Uaktualnienie komponentu głównego i szablonu .....	459
Uaktualnianie komponentów potomnych .....	459
Podsumowanie .....	461
<b>Rozdział 20. Poznajemy dostawcę usługi .....</b>	<b>463</b>
Utworzenie przykładowego projektu .....	464
Używanie dostawcy usługi .....	465
Używanie dostawcy klasy .....	468
Używanie dostawcy wartości .....	475
Używanie dostawcy fabryki .....	477
Używanie dostawcy istniejącej usługi .....	480
Używanie dostawcy lokalnego .....	481
Ograniczenia pojedynczego obiektu usługi .....	481
Utworzenie dostawcy lokalnego w dyrektywie .....	482
Utworzenie dostawcy lokalnego w komponencie .....	483
Kontrolowanie spełniania zależności .....	489
Podsumowanie .....	491
<b>Rozdział 21. Używanie i tworzenie modułu .....</b>	<b>493</b>
Utworzenie przykładowego projektu .....	494
Moduł główny .....	495
Właściwość imports .....	497
Właściwość declarations .....	498
Właściwość providers .....	498
Właściwość bootstrap .....	498

Utworzenie modułu funkcjonalnego .....	501
Utworzenie modułu modelu .....	502
Utworzenie modułu narzędziowego .....	506
Utworzenie modułu wraz z komponentami .....	511
Podsumowanie .....	514
<b>Część III Zaawansowane funkcje Angular .....</b>	<b>515</b>
<b>Rozdział 22. Utworzenie przykładowego projektu .....</b>	<b>517</b>
Rozpoczęcie pracy nad przykładowym projektem .....	517
Dodawanie i konfigurowanie pakietów .....	518
Konfigurowanie TypeScript .....	519
Konfigurowanie programistycznego serwera HTTP .....	519
Konfigurowanie procedury wczytującej moduł JavaScript .....	520
Utworzenie modułu modelu .....	520
Utworzenie typu danych produktu .....	520
Utworzenie źródła danych i repozytorium .....	520
Zakończenie pracy nad modulem modelu .....	522
Utworzenie modułu core .....	522
Utworzenie współdzielonej usługi informacji o stanie .....	522
Utworzenie komponentu tabeli .....	523
Utworzenie komponentu formularza .....	524
Zakończenie pracy nad modulem core .....	526
Utworzenie modułu messages .....	527
Utworzenie modelu i usługi .....	527
Utworzenie komponentu i szablonu .....	528
Zakończenie pracy nad modulem messages .....	528
Zakończenie pracy nad projektem .....	529
Utworzenie pliku typu bootstrap .....	529
Utworzenie modułu Reactive Extensions .....	529
Utworzenie dokumentu HTML .....	530
Uruchomienie przykładowego projektu .....	531
Podsumowanie .....	531
<b>Rozdział 23. Poznajemy bibliotekę Reactive Extensions .....</b>	<b>533</b>
Utworzenie przykładowego projektu .....	534
Poznajemy problem .....	535
Rozwiązanie problemu za pomocą biblioteki Reactive Extensions .....	537
Klasa Observable .....	538
Klasa Observer .....	539
Klasa Subject .....	540
Używanie potoku async .....	542
Używanie potoku async wraz z niestandardowym potokiem .....	543
Skalowanie w górę modułów funkcjonalnych aplikacji .....	544
Wyjście poza podstawy .....	547
Filtrowanie zdarzeń .....	547
Transformowanie zdarzeń .....	548

Otrzymywanie jedynie odmiennych zdarzeń .....	551
Pobieranie i pomijanie zdarzeń .....	553
Podsumowanie .....	554
<b>Rozdział 24. Wykonywanie asynchronicznych żądań HTTP .....</b>	<b>555</b>
Utworzenie przykładowego projektu .....	556
Konfigurowanie procedury wczytywania modułu JavaScript .....	557
Konfigurowanie modułu funkcjonalności modelu .....	558
Uaktualnienie komponentu formularza .....	558
Uruchomienie przykładowego projektu .....	559
Poznajemy usługę sieciową typu RESTful .....	560
Zastąpienie statycznego źródła danych .....	561
Utworzenie usługi nowego źródła danych .....	561
Konfigurowanie źródła danych .....	564
Używanie źródła danych typu REST .....	564
Zapisywanie i usuwanie danych .....	566
Konsolidowanie żądań HTTP .....	569
Wykonywanie żądań między domenami .....	570
Używanie żądań JSONP .....	571
Konfigurowanie nagłówków żądania .....	573
Obsługa błędów .....	575
Wygenerowanie komunikatów przeznaczonych dla użytkownika .....	576
Faktyczna obsługa błędu .....	578
Podsumowanie .....	579
<b>Rozdział 25. Routing i nawigacja — część 1 .....</b>	<b>581</b>
Utworzenie przykładowego projektu .....	582
Wyłączenie wyświetlania zdarzenia zmiany stanu .....	584
Rozpoczęcie pracy z routingiem .....	586
Utworzenie konfiguracji routingu .....	586
Utworzenie komponentu routingu .....	588
Uaktualnienie modułu głównego .....	588
Zakończenie konfiguracji .....	589
Dodawanie łączy nawigacyjnych .....	589
Efekt zastosowania routingu .....	592
Dokończenie implementacji routingu .....	594
Obsługa zmiany trasy w komponencie .....	594
Używanie parametrów trasy .....	597
Nawigacja w kodzie .....	602
Otrzymywanie zdarzeń nawigacyjnych .....	604
Usunięcie dołączania zdarzeń i obsługującego je kodu .....	606
Podsumowanie .....	608
<b>Rozdział 26. Routing i nawigacja — część 2 .....</b>	<b>609</b>
Utworzenie przykładowego projektu .....	609
Dodawanie komponentów do projektu .....	612

Używanie znaków wieloznacznych i przekierowań .....	615
Używanie znaków wieloznacznych w trasie .....	615
Używanie przekierowania w trasie .....	618
Nawigacja w komponencie .....	618
Reakcja na zmiany w routingu .....	620
Nadawanie stylu łącza aktywnej trasy .....	622
Poprawienie przycisku Wszystkie .....	625
Utworzenie trasy potomnej .....	626
Utworzenie outletu trasy potomnej .....	627
Uzyskanie dostępu do parametrów z poziomu tras potomnych .....	629
Podsumowanie .....	632
<b>Rozdział 27. Routing i nawigacja — część 3 .....</b>	<b>633</b>
Utworzenie przykładowego projektu .....	633
Zabezpieczanie tras .....	634
Opóźnienie nawigacji za pomocą resolvera .....	635
Uniemożliwienie nawigacji dzięki użyciu strażników .....	642
Dynamiczne wczytywanie modułów funkcjonalnych .....	654
Utworzenie prostego modułu funkcjonalnego .....	655
Dynamiczne wczytywanie modułu .....	656
Zabezpieczanie dynamicznie wczytywanego modułu .....	659
Odwołania do nazwanych outletów .....	661
Utworzenie dodatkowych elementów <router-outlet> .....	662
Nawigacja podczas użycia wielu outletów .....	664
Podsumowanie .....	666
<b>Rozdział 28. Animacje .....</b>	<b>667</b>
Utworzenie przykładowego projektu .....	668
Dodanie skryptu typu polyfill zapewniającego obsługę animacji .....	668
Wyłączenie opóźnienia HTTP .....	670
Uproszczenie szablonu tabeli i konfiguracji routingu .....	671
Rozpoczęcie pracy z animacjami frameworka Angular .....	672
Utworzenie animacji .....	673
Zastosowanie animacji .....	676
Przetestowanie animacji .....	679
Poznajemy wbudowane stany aplikacji .....	680
Poznajemy transformację elementu .....	681
Utworzenie transformacji dla wbudowanych stanów .....	681
Kontrolowanie animacji transformacji .....	684
Poznajemy grupy stylów animacji .....	689
Zdefiniowanie najczęściej używanych stylów w grupie przeznaczonej do wielokrotnego użycia .....	689
Używanie transformacji elementu .....	691
Zastosowanie stylów frameworka CSS .....	692
Poznajemy zdarzenia wyzwalacza animacji .....	694
Podsumowanie .....	697

<b>Rozdział 29. Testy jednostkowe w Angular</b> .....	<b>699</b>
Utworzenie przykładowego projektu .....	701
Dodawanie pakietów przeznaczonych do wykonywania testów .....	701
Utworzenie prostego testu jednostkowego .....	705
Uruchamianie narzędzi .....	705
Praca z frameworkiem Jasmine .....	706
Testowanie komponentu Angular .....	708
Praca z klasą TestBed .....	708
Testowanie operacji dołączania danych .....	712
Testowanie komponentu wraz z zewnętrznym szablonem .....	714
Testowanie zdarzeń komponentu .....	716
Testowanie właściwości danych wyjściowych .....	718
Testowanie właściwości danych wejściowych .....	719
Testowanie operacji asynchronicznej .....	721
Testowanie dyrektywy Angular .....	723
Podsumowanie .....	725
<b>Skorowidz</b> .....	<b>727</b>

## ROZDZIAŁ 7.

# SportsStore — rzeczywista aplikacja

W rozdziale 2. utworzyliśmy niewielką i prostą aplikację Angular. Dzięki tego rodzaju przykładom mogliśmy skoncentrować się na konkretnych funkcjach Angular, choć były one pozbawione kontekstu. Teraz zbudujemy prostą, ale realistyczną aplikację typu e-commerce.

Budowana tutaj aplikacja o nazwie SportsStore będzie oparta na klasycznym podejściu stosowanym podczas tworzenia sklepów internetowych. Przygotujemy katalog produktów, które klienci będą mogli przeglądać według kategorii. Aplikacja będzie obsługiwała koszyk na zakupy, do którego klienci będą mogli dodawać produkty (lub usuwać je z niego). Oczywiście aplikacja będzie zawierać stronę pozwalającą klientom na finalizację zakupu i podanie danych potrzebnych do realizacji zamówienia. Opracujemy także obszar administracyjny, aby zapewnić sobie możliwość zarządzania katalogiem produktów (przeprowadzanie operacji typu CRUD). Wspomniany obszar będzie chroniony i tylko użytkownicy zalogowani jako administratorzy będą mogli wprowadzać zmiany. Na końcu pokażę, jak przygotować aplikację do wdrożenia, a następnie ją wdrożyć.

Celem przyświecającym mi w tym oraz kolejnych rozdziałach jest pokazanie na maksymalnie rzeczywistym przykładzie, jak faktycznie wygląda programowanie z wykorzystaniem Angular. Ponieważ koncentruję się na frameworku Angular, to uproszczona została integracja z zewnętrznymi systemami, takimi jak magazyn danych, a niektóre (na przykład przetwarzanie płatności) wręcz pominąłem.

Przykład aplikacji SportsStore wykorzystałem w kilku moich książkach, ponieważ pokazuje on, w jaki sposób można wykorzystać różne frameworki, języki i style programowania do osiągnięcia tego samego efektu. Nie musisz czytać innych moich książek, aby zrozumieć materiał przedstawiony w tym rozdziale. Jednak ich lektura może pokazać interesujące różnice w implementacji budowanej tutaj aplikacji.

Funkcje Angular wykorzystane w aplikacji SportsStore będą szczegółowo omówione w późniejszych rozdziałach książki. Zamiast powielać przedstawione treści, podam jedynie objaśnienia niezbędne do zrozumienia działania przykładowej aplikacji, a po szczegółowe informacje odeślę Cię do innych rozdziałów. Rozdziały, w których budujemy aplikację SportsStore, możesz czytać albo dokładnie od początku do końca, aby się dowiedzieć, jak działa Angular, albo też przechodzić od razu do wskazanych rozdziałów w celu uzyskania dokładniejszych informacji o danej funkcji. Niezależnie od przyjętego podejścia nie oczekuj, że wszystko od razu zrozumiesz. Angular to dość rozbudowany framework, a aplikacja SportsStore ma pokazać wiele jego możliwości, ale chwilowo bez zbytniego zagłębiania się w szczegóły, ponieważ będą one omówione w pozostałej części książki.

## Utworzenie przykładowego projektu

W celu utworzenia projektu *SportsStore* przejdź do powłoki, wybierz katalog przeznaczony dla projektu, a następnie wydaj poniższe polecenie:

---

**\$ ng new SportsStore**

---

Pakiet `angular-cli` wygeneruje nowy projekt aplikacji Angular wraz z plikami konfiguracyjnymi, miejscem zarezerwowanym na treść oraz z innymi narzędziami programistycznymi. Przygotowanie projektu wymaga nieco czasu, ponieważ w trakcie tego procesu pobierana jest i instalowana duża liczba pakietów npm.

## Utworzenie struktury katalogów

Bardzo ważnym aspektem podczas przygotowywania każdej aplikacji Angular jest utworzenie struktury katalogów. Polecenie `ng new` powoduje wygenerowanie struktury projektu, w którym wszystkie pliki aplikacji zostają umieszczone w katalogu `src`, przy czym pliki Angular trafiają do katalogu `src/app`. W celu dodania dodatkowej struktury w projekcie utwórz kolejne katalogi wymienione w tabeli 7.1.

**Tabela 7.1.** Dodatkowe katalogi dla aplikacji *SportsStore*

Katalog	Opis
<code>SportsStore/src/app/model</code>	To jest katalog przeznaczony na kod modelu.
<code>SportsStore/src/app/store</code>	To jest katalog przeznaczony na kod podstawowej funkcjonalności sklepu internetowego.
<code>SportsStore/src/app/admin</code>	To jest katalog przeznaczony na kod funkcjonalności administracyjnej.

## Instalowanie dodatkowych pakietów npm

Projekt *SportsStore* wymaga pewnych pakietów dodatkowych poza dodanymi standardowo przez `angular-cli`. Przeprowadź edycję pliku `package.json` w katalogu *SportsStore* i dodaj pakiety przedstawione na listingu 7.1.

**Listing 7.1.** Zawartość pliku `package.json` w katalogu *SportsStore*

```
{
  "name": "sports-store",
  "version": "0.0.0",
  "license": "MIT",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e",
    "json": "json-server data.js -p 3500 -m authMiddleware.js"
  },
  "private": true,
  "dependencies": {
    "@angular/common": "^4.0.0",
    "@angular/compiler": "^4.0.0",
    "@angular/core": "^4.0.0",
    "@angular/forms": "^4.0.0",
    "@angular/http": "^4.0.0",
    "@angular/platform-browser": "^4.0.0",
    "@angular/platform-browser-dynamic": "^4.0.0",
    "@angular/router": "^4.0.0",
    "core-js": "^2.4.1",
    "rxjs": "^5.1.0",
```



```

"zone.js": "^0.8.4",
"bootstrap": "4.0.0-alpha.4",
"font-awesome": "4.7.0"
},
"devDependencies": {
"@angular/cli": "1.0.0",
"@angular/compiler-cli": "^4.0.0",
"@types/jasmine": "2.5.38",
"@types/node": "~6.0.60",
"codelyzer": "~2.0.0",
"jasmine-core": "~2.5.2",
"jasmine-spec-reporter": "~3.2.0",
"karma": "~1.4.1",
"karma-chrome-launcher": "~2.0.0",
"karma-cli": "~1.0.1",
"karma-jasmine": "~1.1.0",
"karma-jasmine-html-reporter": "^0.2.2",
"karma-coverage-istanbul-reporter": "^0.2.0",
"protractor": "~5.1.0",
"ts-node": "~2.0.0",
"tslint": "~4.5.0",
"typescript": "~2.2.0",
"json-server": "0.8.21",
"jsonwebtoken": "7.1.9"
}
}

```

- 
- **Ostrzeżenie** Aby otrzymać oczekiwane dane wyjściowe, we wszystkich przykładach przedstawionych w książce bardzo ważne jest użycie wersji pakietów podanych na listingach. Jeżeli napotkasz problemy z przykładami przedstawionymi w książce, spróbuj wykorzystać kod źródłowy przygotowanych przeze mnie aplikacji, który możesz pobrać ze strony <ftp://ftp.helion.pl/przyklady/angup2.xz>. Jeżeli dojdiesz do ściany i nie będziesz mógł sobie poradzić, napisz do mnie na adres [adam@adam-freeman.com](mailto:adam@adam-freeman.com), a ja spróbuję Ci pomóc.
- 

Aby pobrać i zainstalować pakiety wymagane podczas pracy nad tym projektem, zapisz plik, a następnie z poziomu katalogu *SportsStore* wydaj poniższe polecenie:

```
$ npm install
```

Menedżer npm wyświetli listę pakietów po zainstalowaniu wszystkich niezbędnych. W trakcie procesu instalacji zwykle pojawiają się pewne ostrzeżenia, które jednak można bezpiecznie zignorować.

## Utworzenie usługi sieciowej typu RESTful

Aplikacja *SportStore* będzie używała asynchronicznych żądań HTTP w celu pobrania danych modelu dostarczanych przez usługę sieciową typu RESTful. REST, jak to dokładnie przedstawię w rozdziale 24., to po prostu podejście stosowane podczas projektowania usług sieciowych używających metod HTTP do określenia operacji oraz adresu URL do wskazania obiektów danych, w których będzie wykonana operacja.

W pliku *package.json* wymieniłem między innymi *json-server*, czyli doskonały pakiet pozwalający na szybkie utworzenie usługi sieciowej na podstawie danych JSON lub kodu JavaScript. Aby upewnić się o istnieniu pewnego stanu projektu, do którego zawsze będzie można powrócić, wykorzystam zaletę funkcji pozwalającej usłudze sieciowej typu RESTfull na dostarczanie danych za pomocą kodu JavaScript. Oznacza to, że ponowne uruchomienie usługi sieciowej wyzeruje dane aplikacji. W katalogu *SportStore* utwórz plik o nazwie *data.js* i umieść w nim kod przedstawiony na listingu 7.2.

**Listing 7.2.** Zawartość pliku *data.js* w katalogu *SportsStore*

```

module.exports = function () {
  return {
    products: [
      { id: 1, name: "Kajak", category: "Sporty wodne",
        description: "Łódka przeznaczona dla jednej osoby.", price: 275 },
      { id: 2, name: "Kamizelka ratunkowa", category: "Sporty wodne",
        description: "Chroni i dodaje uroku.", price: 48.95 },
      { id: 3, name: "Piłka", category: "Piłka nożna",
        description: "Zatwierdzone przez FIFA rozmiar i waga.", price: 19.50 },
      { id: 4, name: "Flagi narozne", category: "Piłka nożna",
        description: "Nadadzą twojemu boisku profesjonalny wygląd.",
        price: 34.95 },
      { id: 5, name: "Stadion", category: "Piłka nożna",
        description: "Składany stadion na 35 000 osób.", price: 79500 },
      { id: 6, name: "Czapka", category: "Szachy",
        description: "Zwiększa efektywność mózgu o 75%.", price: 16 },
      { id: 7, name: "Niestabilne krzesło", category: "Szachy",
        description: "Zmniejsza szanse przeciwnika.",
        price: 29.95 },
      { id: 8, name: "Ludzka szachownica", category: "Szachy",
        description: "Przyjemna gra dla całej rodziny.", price: 75 },
      { id: 9, name: "Błyszczący król", category: "Szachy",
        description: "Pokryty złotem i wysadzany diamentami król.", price: 1200 }
    ],
    orders: []
  }
}

```

- 
- **Wskazówka** Podczas tworzenia plików konfiguracyjnych trzeba koniecznie zwrócić uwagę na ich nazwy. Niektóre z nich mają rozszerzenie *.json*, czyli zawierają dane statyczne sformatowane w postaci JSON. Z kolei inne mają rozszerzenie *.js*, co oznacza, że zawierają kod JavaScript. Każde narzędzie niezbędne podczas programowania z użyciem frameworka Angular ma pewne wymagania dotyczące jego pliku konfiguracyjnego.
- 

Powyższy fragment kodu definiuje dwie kolekcje danych dostarczane przez usługę sieciową typu RESTful. Pierwsza kolekcja, *products*, zawiera wszystkie produkty oferowane klientom w sklepie internetowym. Druga, *orders*, będzie zawierała zamówienia złożone przez klientów — ta kolekcja jest aktualnie pusta.

Dane przechowywane w usłudze sieciowej typu RESTful muszą być chronione, aby zwykły użytkownik nie mógł modyfikować katalogu produktów bądź stanu zamówień. Pakiet *json-server* nie oferuje żadnych wbudowanych funkcji uwierzytelniania, więc w katalogu *SportStore* utwórz plik o nazwie *authMiddleware.js* i umieść w nim kod przedstawiony na listingu 7.3.

**Listing 7.3.** Zawartość pliku *authMiddleware.js* w katalogu *SportsStore*

```

const jwt = require("jsonwebtoken");

const APP_SECRET = "appsekret";
const USERNAME = "admin";
const PASSWORD = "sekret";

module.exports = function (req, res, next) {
  if (req.url == "/login" && req.method == "POST") {
    if (req.body != null && req.body.name == USERNAME
      && req.body.password == PASSWORD) {
      let token = jwt.sign({ data: USERNAME, expiresIn: "1h" }, APP_SECRET);

```

```

        res.json({ success: true, token: token });
    } else {
        res.json({ success: false });
    }
    res.end();
    return;
} else if ((req.url.startsWith("/products") && req.method != "GET")
    || (req.url.startsWith("/orders") && req.method != "POST")) {
    let token = req.headers["authorization"];
    if (token != null && token.startsWith("Bearer<")) {
        token = token.substring(7, token.length - 1);
        try {
            jwt.verify(token, APP_SECRET);
            next();
            return;
        } catch (err) {}
    }
    res.statusCode = 401;
    res.end();
    return;
}
next();
}
}

```

Ten kod analizuje żądanie HTTP wykonane do usługi sieciowej typu RESTful i implementuje pewne podstawowe funkcje bezpieczeństwa. Jest to działający po stronie serwera kod niepowiązany bezpośrednio z programowaniem z użyciem frameworka Angular. Dlatego też nie przejmuj się, jeśli jego przeznaczenie pozostaje dla Ciebie niezrozumiałe. Proces uwierzytelniania i autoryzacji omówię dokładnie w rozdziale 9., w którym dowiesz się między innymi, jak uwierzytelniać użytkowników w aplikacji Angular.

- 
- **Ostrzeżenie** Kodu przedstawionego na listingu 7.3 nie używaj w aplikacji innej niż SportsStore. Zawiera on słabe hasła na stałe zdefiniowane w kodzie. Takie rozwiązanie jest wystarczające w projekcie aplikacji SportsStore, ponieważ tutaj nacisk położyłem na użycie frameworka Angular podczas programowania po stronie klienta. Przedstawione podejście jest jednak niedopuszczalne w rzeczywistych projektach.
- 

## Utworzenie pliku HTML

Każda aplikacja internetowa Angular ma dokument HTML wczytywany przez przeglądarkę WWW i używany do uruchomienia danej aplikacji. W katalogu *SportsStore/src* przeprowadź edycję pliku o nazwie *index.html* w celu usunięcia istniejącego w nim kodu i umieszczenia kodu przedstawionego na listingu 7.4.

**Listing 7.4.** Zawartość pliku *index.html* w katalogu *SportsStore/src*

```

<!DOCTYPE html>
<html lang="pl">
<head>
  <base href="/" />
  <title>SportsStore</title>
  <meta charset="utf-8" />
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link href="node_modules/bootstrap/dist/css/bootstrap.min.css"
    rel="stylesheet" />
</head>
<body class="m-a-1">

```

```
<app>Miejsce na treść sklepu SportsStore.</app>
</body>
</html>
```

Ten dokument HTML zawiera element `<link>` wczytujący arkusze stylów Bootstrap CSS oraz element `<app>` działający w charakterze miejsca zarezerwowanego na funkcjonalność sklepu internetowego SportsStore.

- 
- **Wskazówka** Przedstawiony dokument zawiera również element `<base>` wymagany przez funkcje routingu frameworka Angular, które dodam do projektu SportsStore w rozdziale 8.
- 

## Uruchomienie przykładowej aplikacji

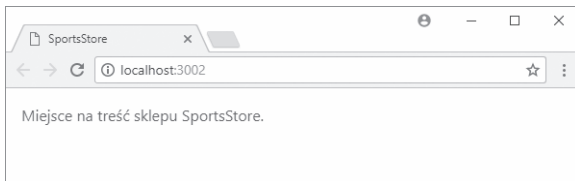
Upewnij się, że zapisałeś wszystkie pliki, a następnie z poziomu katalogu *SportsStore* wydaj poniższe polecenie:

---

```
$ ng serve --port 3000 --open
```

---

To polecenie uruchomi łańcuch skonfigurowanych przez `angular-cli` narzędzi programistycznych, które przeprowadzą automatyczną kompilację kodu i plików z treścią umieszczonych w katalogu *src*. Kompilacja będzie się odbywała po wykryciu zmiany w dowolnym z wymienionych plików. Na ekranie zostanie otworzone nowe okno przeglądarki WWW wraz z treścią, jak pokazałem na rysunku 7.1.



**Rysunek 7.1.** Efekt uruchomienia przykładowej aplikacji

Programistyczny serwer HTTP będzie nasłuchiwał na porcie 3000, więc adres URL aplikacji ma postać `http://localhost:3000`. Nie ma potrzeby podawania nazwy dokumentu HTML, ponieważ `index.html` to nazwa domyślna, na którą odpowiada serwer.

## Uruchomienie usługi sieciowej typu RESTful

W celu uruchomienia usługi sieciowej typu RESTful przejdź do nowego okna powłoki, następnie do katalogu *SportsStore* i wydaj poniższe polecenie:

---

```
$ npm run json
```

---

Usługa sieciowa typu RESTful została skonfigurowana do działania na porcie 3500. Aby przetestować żądanie do usługi sieciowej, w przeglądarce WWW wpisz adres URL `http://localhost:3500/products/1`. Przeglądarka WWW powinna wyświetlić informacje dotyczące jednego z produktów zdefiniowanych na listingu 7.2, jak pokazałem poniżej.

---

```
{
  "id": 1,
```

---

```

"name": "Kajak",
"category": "Sporty wodne",
"description": "Łódka przeznaczona dla jednej osoby.",
"price": 275
}

```

## Przygotowanie funkcji projektu Angular

Każdy projekt Angular wymaga pewnych przygotowań w celu osiągnięcia stanu, w którym aplikacja może być wczytana i uruchomiona przez przeglądarkę WWW. W kolejnych sekcjach zaprezentuję podstawy, na których zostanie utworzona aplikacja SportsStore.

### Uaktualnienie komponentu głównego

Pracę rozpoczynamy od komponentu głównego, który jest elementem konstrukcyjnym Angular przeznaczonym do zarządzania treścią elementu `<app>` zdefiniowanego w dokumencie HTML. Wprawdzie aplikacja może zawierać wiele komponentów, ale zawsze będzie komponent główny odpowiedzialny za treść najwyższego poziomu wyświetlaną użytkownikowi.

W katalogu `SportsStore/src/app` przeprowadź edycję pliku o nazwie `app.component.ts` i istniejący w nim kod zastąp przedstawionym na listingu 7.5.

**Listing 7.5.** Zawartość pliku `app.component.ts` w katalogu `SportsStore/src/app`

```

import { Component } from "@angular/core";

@Component({
  selector: "app",
  template: `<div class="bg-success p-a-1 text-xs-center">
    To jest aplikacja SportsStore.
  </div>`
})
export class AppComponent {}

```

Dekorator `@Component` informuje Angular, że klasa `AppComponent` jest komponentem, a jej właściwości konfigurują sposób zastosowania tego komponentu. W rozdziale 17. dokładnie omówię pełny zbiór właściwości komponentu, trzy z nich użyte na listingu 7.9 to najprostsze i najczęściej stosowane. Właściwość `selector` wskazuje frameworkowi Angular, jak komponent ma być zastosowany w dokumencie HTML, natomiast właściwość `template` definiuje treść wyświetlaną przez ten komponent. Podobnie jak w omawianym tutaj przykładzie, komponent ma możliwość zdefiniowania osadzonego szablonu bądź też korzystania z zewnętrznych plików HTML, które mogą ułatwić zarządzanie skomplikowaną treścią.

Klasa `AppComponent` nie zawiera kodu, ponieważ komponent główny w projekcie Angular istnieje po prostu w celu zarządzania treścią wyświetlaną użytkownikowi. Początkowo będę ręcznie zarządzać treścią wyświetlaną przez komponent główny, natomiast w rozdziale 8. funkcji o nazwie **routing URL** użyję do automatycznego zarządzania treścią na podstawie akcji podejmowanych przez użytkownika.

### Uaktualnienie modułu głównego

Mamy dwa rodzaje modułów Angular: funkcjonalny i główny. Moduł funkcjonalny jest używany w celu grupowania powiązanej funkcjonalności aplikacji, aby ułatwić zarządzanie nią. Tego rodzaju moduły utworzę w każdym z najważniejszych obszarów aplikacji, między innymi w modelu, w interfejsie sklepu wyświetlanym użytkownikom oraz w interfejsie administracyjnym.

Moduł główny jest używany do opisanego aplikacji frameworkowi Angular. Wspomniany opis zawiera nazwy modułów zwykłych wymaganych do uruchomienia aplikacji, funkcji koniecznych do wczytania oraz nazwę komponentu głównego. Zgodnie z konwencją nazwa pliku komponentu głównego to *app.module.ts*. W katalogu *SportsStore/src/app* przeprowadź edycję pliku o nazwie *app.module.ts* i istniejący w nim kod zastąp przedstawionym na listingu 7.6.

**Listing 7.6.** Zawartość pliku *app.module.ts* w katalogu *SportsStore/app*

```
import { NgModule } from "@angular/core";
import { LOCALE_ID } from '@angular/core';
import { BrowserModule } from "@angular/platform-browser";
import { AppComponent } from "../app.component";

@NgModule({
  imports: [BrowserModule],
  declarations: [AppComponent],
  bootstrap: [AppComponent],
  providers: [{ provide: LOCALE_ID, useValue: "pl-PL" }]
})
export class AppModule {}
```

Podobnie jak w przypadku komponentu głównego, także klasa modułu głównego nie zawiera kodu. Ten moduł tak naprawdę istnieje jedynie w celu dostarczenia informacji za pomocą dekoratora `@NgModule`. Właściwość `imports` nakazuje frameworkowi Angular wczytanie funkcji `BrowserModule`, która zawiera całą podstawową funkcjonalność wymaganą przez aplikację internetową.

Właściwość `declarations` wskazuje Angular, że powinien być wczytany komponent główny, z kolei właściwość `bootstrap` wskazuje ten komponent główny — to jest klasa `AppModule`. Kolejne informacje do tych właściwości dekoratora będą dodawał później wraz z implementowaniem poszczególnych funkcjonalności aplikacji *SportsStore*. Ta podstawowa konfiguracja wystarczy do uruchomienia aplikacji.

## Analiza pliku typu bootstrap

Kolejnym krokiem jest utworzenie pliku typu bootstrap przeznaczonego do uruchamiania aplikacji. W tej książce koncentruję się na użyciu frameworka Angular do tworzenia aplikacji internetowych działających w przeglądarce WWW, ale platforma Angular może być wykorzystana do budowy aplikacji przeznaczonych również dla innych środowisk. Plik typu bootstrap używa platformy przeglądarki WWW w celu wczytania modułu głównego i uruchomienia aplikacji. Nie ma konieczności wprowadzania jakichkolwiek zmian w pliku o nazwie *main.ts* znajdującym się w katalogu *SportsStore/src*. Zawartość wymienionego pliku przedstawiłem na listingu 7.7.

**Listing 7.7.** Zawartość pliku *main.ts* w katalogu *SportsStore/src*

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from '../app/app.module';
import { environment } from '../environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule);
```

Narzędzia programistyczne wykrywają wprowadzanie zmian w plikach projektu, co prowadzi do automatycznej kompilacji kodu, odświeżenia strony w przeglądarce WWW i wyświetlenia treści pokazanej na rysunku 7.2.



**Rysunek 7.2.** Uruchomiona aplikacja SportsStore

Jeżeli przeanalizujesz wygenerowany przez przeglądarkę WWW model DOM, wówczas zobaczysz, że treść pochodząca z szablonu komponentu głównego framework Angular umieścił w elemencie `<app>`, jak pokazałem poniżej.

```
<body class="m-a-1">
  <app>
    <div class="bg-success p-a-1 text-xs-center">
      To jest aplikacja SportsStore.
    </div>
  </app>
</body>
```

## Utworzenie danych modelu

Najlepszym miejscem do rozpoczęcia pracy nad nowym projektem jest jego model dostarczający dane. Chciałbym dotrzeć do punktu, w którym będziesz mógł zobaczyć w działaniu pewne funkcje frameworka Angular. Dlatego też zamiast definiować dane modelu od początku do końca, wykorzystam istniejącą funkcjonalność w celu dostarczenia fikcyjnych danych. Następnie będę używać tych danych do utworzenia funkcji przeznaczonych dla użytkownika. Do danych modelu jeszcze powrócę w rozdziale 8., w którym powiążę je z usługą sieciową typu RESTful.

## Utworzenie klas modelu

Wszystkie dane modelu wymagają klas opisujących rodzaj danych, które będą znajdowały się w tym modelu. W przypadku aplikacji SportsStore oznacza to klasy opisujące produkty sprzedawane w sklepie oraz zamówienia składane przez klientów.

Na początku pracy z aplikacją SportsStore możliwość opisanie produktów będzie wystarczająca. Kolejne klasy modelu przeznaczone do obsługi funkcjonalności sklepu będą tworzyć podczas implementacji wspomnianych funkcji. W katalogu `SportsStore/src/app/model` utwórz nowy plik o nazwie `product.model.ts` i umieść w nim kod przedstawiony na listingu 7.8.

**Listing 7.8.** Zawartość pliku `product.model.ts` w katalogu `SportsStore/src/app/model`

```
export class Product {
  constructor(
    public id?: number,
    public name?: string,
    public category?: string,
    public description?: string,
    public price?: number) {}
}
```

Klasa `Product` definiuje konstruktora akceptującego właściwości `id`, `name`, `category`, `description` i `price`, odpowiadające strukturze danych użytej podczas tworzenia usługi sieciowej typu RESTful we wcześniejszej części rozdziału. Znak zapytania umieszczony po nazwie parametru oznacza, że jest to parametr opcjonalny, który może być pominięty podczas tworzenia nowego egzemplarza obiektu na podstawie klasy `Product`. To jest użyteczna możliwość w trakcie budowy aplikacji, w której właściwość obiektu modelu ma przypisywaną wartość na podstawie danych pochodzących z formularza HTML.

## Utworzenie fikcyjnego źródła danych

Aby przygotować się na przejście od danych fikcyjnych do rzeczywistych, zamierzam wykorzystać źródło danych. Pozostała część aplikacji nie wie, skąd pochodzą dane. Takie rozwiązanie umożliwi bezproblemowe przejście do pobierania danych za pomocą żądań HTTP.

W katalogu `SportsStore/src/app/model` utwórz nowy plik o nazwie `static.datasource.ts` i umieść w nim kod przedstawiony na listingu 7.9.

**Listing 7.9.** Zawartość pliku `static.datasource.ts` w katalogu `SportsStore/src/app/model`

```
import { Injectable } from "@angular/core";
import { Product } from "../product.model";
import { Observable } from "rxjs/Observable";
import "rxjs/add/observable/from";

@Injectable()
export class StaticDataSource {
  private products: Product[] = [
    new Product(1, "Produkt 1", "Kategoria 1", "Produkt 1 (Kategoria 1)", 100),
    new Product(2, "Produkt 2", "Kategoria 1", "Produkt 2 (Kategoria 1)", 100),
    new Product(3, "Produkt 3", "Kategoria 1", "Produkt 3 (Kategoria 1)", 100),
    new Product(4, "Produkt 4", "Kategoria 1", "Produkt 4 (Kategoria 1)", 100),
    new Product(5, "Produkt 5", "Kategoria 1", "Produkt 5 (Kategoria 1)", 100),
    new Product(6, "Produkt 6", "Kategoria 2", "Produkt 6 (Kategoria 2)", 100),
    new Product(7, "Produkt 7", "Kategoria 2", "Produkt 7 (Kategoria 2)", 100),
    new Product(8, "Produkt 8", "Kategoria 2", "Produkt 8 (Kategoria 2)", 100),
    new Product(9, "Produkt 9", "Kategoria 2", "Produkt 9 (Kategoria 2)", 100),
    new Product(10, "Produkt 10", "Kategoria 2", "Produkt 10 (Kategoria 2)", 100),
    new Product(11, "Produkt 11", "Kategoria 3", "Produkt 11 (Kategoria 3)", 100),
    new Product(12, "Produkt 12", "Kategoria 3", "Produkt 12 (Kategoria 3)", 100),
    new Product(13, "Produkt 13", "Kategoria 3", "Produkt 13 (Kategoria 3)", 100),
    new Product(14, "Produkt 14", "Kategoria 3", "Produkt 14 (Kategoria 3)", 100),
    new Product(15, "Produkt 15", "Kategoria 3", "Produkt 15 (Kategoria 3)", 100),
  ];

  getProducts(): Observable<Product[]> {
    return Observable.from([this.products]);
  }
}
```

Klasa `StaticDataSource` definiuje metodę o nazwie `getProducts()` zwracającą fikcyjne dane. Wynikiem wywołania tej metody jest `Observable<Product[]>`, czyli egzemplarz typu `Observable` tworzący tablicę obiektów `Product`.

Klasa `Observable` jest dostarczana przez pakiet biblioteki Reactive Extensions używanej przez framework Angular do obsługi zmiany stanu w aplikacji. W rozdziale 23. dokładnie omówię klasę `Observable`, natomiast w tym miejscu wystarczy wiedzieć, że tego rodzaju obiekt przypomina obietnicę JavaScript — reprezentuje zadanie asynchroniczne, które w przyszłości wygeneruje pewien wynik. Framework Angular wykorzystuje obiekty typu `Observable` dla niektórych funkcji, między innymi do wykonywania żądań HTTP, i dlatego metoda `getProducts()` zwraca `Observable<Product[]>`, zamiast po prostu asynchronicznie dostarczać dane lub wykorzystać obietnicę.



Dla klasy `StaticDataSource` został zastosowany dekorator `@Injectable`. Ten dekorator wskazuje frameworkowi Angular, że oznaczona nim klasa będzie używana w charakterze usługi. Dlatego też inne klasy będą mogły za pomocą mechanizmu wstrzykiwania zależności uzyskać dostęp do funkcjonalności oferowanej przez usługę. Mechanizm wstrzykiwania zależności dokładnie omówię w rozdziałach 19. i 20. Natomiast sposób działania usługi stanie się jasny, gdy aplikacja nabierze kształtu.

- 
- **Wskazówka** Zwróć uwagę na konieczność zaimportowania `Injectable` z modułu JavaScript `@angular/core`, aby można było zastosować dekorator `@Injectable`. Nie będę tutaj omawiał wszystkich klas Angular koniecznych do zaimportowania w budowanej aplikacji `SportsStore`. Więcej informacji na ich temat znajdziesz w rozdziałach, w których będę przedstawiał powiązane z nimi funkcje.
- 

## Utworzenie repozytorium modelu

Źródło danych jest odpowiedzialne za dostarczanie aplikacji wymaganych przez nią danych. Jednak dostęp do tych danych zwykle odbywa się za pomocą tak zwanego **repozytorium**, które staje się odpowiedzialne za dostarczanie danych poszczególnym komponentom aplikacji, nie ujawniając przy tym szczegółów związanych ze sposobem uzyskania danych. W katalogu `SportsStore/src/app/model` utwórz nowy plik o nazwie `product.repository.ts` i umieść w nim kod przedstawiony na listingu 7.10.

**Listing 7.10.** Zawartość pliku `product.repository.ts` w katalogu `SportsStore/src/app/model`

```
import { Injectable } from "@angular/core";
import { Product } from "../product.model";
import { StaticDataSource } from "../static.datasource";

@Injectable()
export class ProductRepository {
  private products: Product[] = [];
  private categories: string[] = [];

  constructor(private dataSource: StaticDataSource) {
    dataSource.getProducts().subscribe(data => {
      this.products = data;
      this.categories = data.map(p => p.category)
        .filter((c, index, array) => array.indexOf(c) == index).sort();
    });
  }

  getProducts(category: string = null): Product[] {
    return this.products
      .filter(p => category == null || category == p.category);
  }

  getProduct(id: number): Product {
    return this.products.find(p => p.id == id);
  }

  getCategories(): string[] {
    return this.categories;
  }
}
```

Gdy framework Angular będzie musiał utworzyć nowy egzemplarz repozytorium, przeanalizuje tę klasę i ustali, że w celu wywołania konstruktora `ProductRepository` i utworzenia nowego obiektu potrzebny jest obiekt `StaticDataSource`.

Konstruktor repozytorium wywołuje metodę `getProducts()` źródła danych, a następnie w celu otrzymania danych produktu używa metody `subscribe()` oferowanej przez zwrócony obiekt `Observable`. W rozdziale 23. dokładnie wyjaśnię sposób działania obiektu `Observable`.

## Używanie prostych struktur danych

Do przechowywania danych modelu wykorzystałem tablicę, ponieważ zgodnie z ogólną regułą najprostsza z możliwych struktur danych daje najlepsze wyniki w aplikacji Angular. Podczas generowania treści w elemencie HTML framework Angular wykonuje wyrażenie w procesie dołączania danych. Oznacza to, że bardziej skomplikowane struktury — takie jak oparta na klasie `Map` i dostarczająca kolekcji typu klucz-wartość w JavaScript ES6 — muszą nieustannie przeprowadzać konwersje aż do chwili ustabilizowania się stanu aplikacji Angular. Dlatego też im prostsza struktura danych, tym mniejsza ilość pracy jest konieczna do wykonania, aby dostarczyć frameworkowi Angular wymagane przez niego dane.

Kolejnym powodem użycia prostych struktur danych są ograniczone możliwości w zakresie obsługi nowych funkcji JavaScript w starszych wersjach przeglądarek WWW. Na przykład w przypadku klasy `Map`, gdy kompilator jest używany do wygenerowania kodu JavaScript przeznaczonego do wykonania w starszych przeglądarkach WWW, TypeScript ogranicza sposób, w jaki treść `map` może być wykorzystana.

W efekcie staram się korzystać z prostych struktur danych, zwłaszcza tablic, i tworzyć nieco bardziej skomplikowane klasy przeznaczone do zarządzania danymi w tablicy. Przykład tego rodzaju podejścia zobaczysz, gdy w rozdziale 9 zacznę dodawać do klasy repozytorium produktu funkcje administracyjne. Wspomniane nowe funkcje będą musiały przeszukiwać tablicę, aby znaleźć obiekty, w których zostanie wykonana dana operacja. Takie rozwiązanie jest nieefektywne, ale te operacje są przeprowadzane znacznie rzadziej w porównaniu z częstotliwością, z jaką Angular wykonuje wyrażenia podczas procesu dołączania danych.

## Utworzenie modułu funkcjonalnego

Przystępuję teraz do zdefiniowania modelu pozwalającego na łatwe wykorzystanie jego funkcjonalności w każdym miejscu aplikacji. W katalogu `SportsStore/src/app/model` utwórz nowy plik o nazwie `model.module.ts` i umieść w nim kod przedstawiony na listingu 7.11.

**Listing 7.11.** Zawartość pliku `model.module.ts` w katalogu `SportsStore/src/app/model`

```
import { NgModule } from "@angular/core";
import { ProductRepository } from "../product.repository";
import { StaticDataSource } from "../static.datasource";

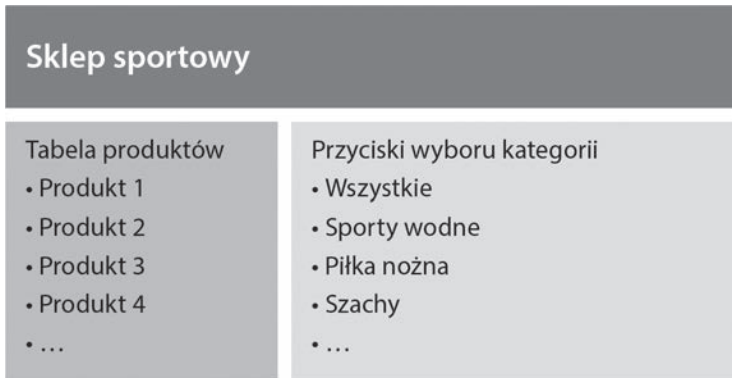
@NgModule({
  providers: [ProductRepository, StaticDataSource]
})
export class ModelModule {}
```

- **Wskazówka** Nie przejmuj się, jeśli te wszystkie nazwy plików wydają Ci się podobne i mylące. Wraz z poznawaniem materiału przedstawionego w następnych rozdziałach książki przywykniesz do struktury aplikacji Angular. Wkrótce tylko spojrzysz na pliki znajdujące się w projekcie aplikacji Angular i od razu będziesz wiedział, do czego są przeznaczone.

Dekorator `@NgModule` jest używany do tworzenia zwykłych modułów, a jego właściwości wskazują frameworkowi Angular, jak moduł powinien być użyty. W przedstawionym powyżej module istnieje tylko jedna właściwość, `providers`, wskazująca Angular, które klasy powinny być używane w charakterze usług dla mechanizmu wstrzykiwania zależności, co zostanie dokładnie omówione w rozdziałach 19. i 20. Z kolei w rozdziale 21. omówię zwykłe moduły i dekorator `@NgModule`.

## Rozpoczęcie pracy nad utworzeniem sklepu internetowego

Mając przygotowane dane modelu, można przystąpić do budowy funkcjonalności sklepu internetowego, która pozwoli użytkownikom na przeglądanie katalogu produktów oraz składanie zamówień. Podstawowa struktura sklepu internetowego to składający się z dwóch kolumn układ strony. Wyświetlane w lewej kolumnie przyciski kategorii pozwalają na filtrowanie listy produktów wyświetlanej w tabeli znajdującej się w prawej kolumnie, jak pokazałem na rysunku 7.3.



**Rysunek 7.3.** Ogólna struktura strony w naszym sklepie internetowym

W kolejnych sekcjach wykorzystam funkcje frameworka Angular i dane modelu w celu utworzenia układu pokazanego na rysunku 7.3.

## Utworzenie szablonu i komponentu sklepu internetowego

Gdy lepiej poznasz framework Angular, zobaczysz, że można łączyć jego funkcje i ten sam problem rozwiązywać na wiele różnych sposobów. W celu zaprezentowania ważnych funkcji Angular spróbuję wprowadzić w projekcie SportsStore pewną odmienną. Jednak w tym momencie stawiam na prostotę, tak aby jak najszybciej dotrzeć do punktu pozwalającego na uruchomienie projektu.

Punktem wyjścia do opracowania funkcjonalności sklepu internetowego będzie nowy komponent. Ten komponent to po prostu klasa dostarczająca dane i logikę szablonowi HTML, który wykorzystuje mechanizm dołączania danych w celu dynamicznego generowania treści. W katalogu `SportsStore/src/app/store` utwórz nowy plik o nazwie `store.component.ts` i umieść w nim kod przedstawiony na listingu 7.12.

**Listing 7.12.** Zawartość pliku `store.component.ts` w katalogu `SportsStore/src/app/store`

```
import { Component } from "@angular/core";
import { Product } from "../model/product.model";
import { ProductRepository } from "../model/product.repository";

@Component({
```

```

    selector: "store",
    moduleId: module.id,
    templateUrl: "store.component.html"
  })
  export class StoreComponent {
    constructor(private repository: ProductRepository) {}

    get products(): Product[] {
      return this.repository.getProducts();
    }

    get categories(): string[] {
      return this.repository.getCategories();
    }
  }

```

Dla klasy `StoreComponent` został zastosowany dekorator `@Component` wskazujący Angular, że mamy do czynienia z komponentem. Właściwości dekoratora określają frameworkowi Angular, jak zastosować ten komponent dla treści HTML (za pomocą elementu o nazwie `<store>`) i gdzie znajduje się szablon komponentu (w pliku o nazwie `store.component.html`).

Klasa `StoreComponent` dostarcza logikę przeznaczoną do obsługi treści umieszczonej w szablonie.

Konstruktor klasy otrzymuje argument w postaci obiektu `ProductRepository`, dostarczony za pomocą mechanizmu wstrzykiwania zależności, który dokładnie omówię w rozdziałach 19. i 20. W komponencie zostały zdefiniowane właściwości `products` i `categories` używane do wygenerowania w szablonie treści HTML na podstawie danych pochodzących z repozytorium.

Aby przygotować szablon dla budowanego tutaj komponentu, w katalogu `SportsStore/src/app/store` utwórz nowy plik o nazwie `store.component.html` i umieść w nim kod przedstawiony na listingu 7.13.

**Listing 7.13.** Zawartość pliku `store.component.html` w katalogu `SportsStore/src/app/store`

```

<div class="navbar navbar-inverse bg-inverse">
  <a class="navbar-brand">SKLEP SPORTOWY</a>
</div>
<div class="col-xs-3 bg-info p-a-1">
  {{categories.length}} kategorie
</div>
<div class="col-xs-9 bg-success p-a-1">
  {{products.length}} produktów
</div>

```

Ten szablon jest bardzo prosty i ma jedynie pozwolić na rozpoczęcie pracy. Większość elementów dostarcza strukturę dla układu sklepu internetowego i stosuje pewne style Bootstrap CSS. W tym momencie mamy jedynie dwie operacje dołączania danych wskazywane przez nawiasy `{{ i }}`. To są wiązania **interpolacji ciągu tekstowego**, które nakazują frameworkowi wykonanie wyrażenia wiązania i wstawienia jego wyniku do elementu. Wyrażenia w użytych wiązaniach powodują wyświetlenie liczby kategorii i produktów.

## Utworzenie modułu funkcjonalnego dla sklepu

W tym momencie nie mamy jeszcze zbyt dużej funkcjonalności sklepu internetowego. Konieczne jest wykonanie jeszcze dodatkowej pracy i powiązania dostępnej funkcjonalności z pozostałą częścią aplikacji. Aby utworzyć moduł funkcjonalny dla sklepu internetowego, w katalogu `SportsStore/src/app/store` utwórz nowy plik o nazwie `store.module.ts` i umieść w nim kod przedstawiony na listingu 7.14.

**Listing 7.14.** Zawartość pliku `store.module.ts` w katalogu `SportsStore/src/app/store`

```

import { NgModule } from "@angular/core";
import { BrowserModule } from "@angular/platform-browser";

```

```
import { FormsModule } from "@angular/forms";
import { ModelModule } from "../model/model.module";
import { StoreComponent } from "./store.component";

@NgModule({
  imports: [ModelModule, BrowserModule, FormsModule],
  declarations: [StoreComponent],
  exports: [StoreComponent]
})
export class StoreModule {}
```

Dekorator `@NgModule` konfiguruje moduł i za pomocą właściwości `imports` wskazuje frameworkowi Angular, że działanie modułu sklepu internetowego zależy od modułu modelu, a także modułów `BrowserModule` i `FormsModule`, które zawierają standardowe funkcje Angular przeznaczone dla aplikacji internetowych i pracy z elementami formularza HTML. Ten dekorator używa właściwości `declarations` do poinformowania frameworka Angular o klasie `StoreComponent`, natomiast właściwość `exports` wskazuje na możliwość użycia tego modułu w innych miejscach aplikacji. To jest ważne, ponieważ wymieniony moduł będzie wykorzystany przez moduł główny.

## Uaktualnienie komponentu i modułu głównego

Zastosowanie podstawowego modelu i funkcjonalności sklepu internetowego wymaga uaktualnienia modułu głównego aplikacji w celu zaimportowania dwóch przygotowanych wcześniej zwykłych modułów. Konieczne jest również uaktualnienie szablonu modułu głównego polegające na wskazaniu elementu HTML, do którego ma zastosowanie komponent w module sklepu internetowego. Na listingu 7.15 przedstawiłem zmianę niezbędną do wprowadzenia w komponencie głównym.

*Listing 7.15. Dodanie nowego elementu w pliku `app.component.ts`*

```
import { Component } from "@angular/core";

@Component({
  selector: "app",
  template: "<store></store>"
})
export class AppComponent {}
```

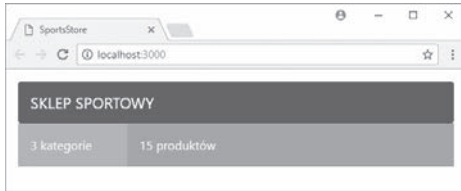
Element `<store>` zastępuje wcześniejszą treść szablonu komponentu głównego i odpowiada wartości właściwości `selector` dekoratora `@Component` na listingu 7.12. Na listingu 7.16 przedstawiłem zmiany konieczne do wprowadzenia w module głównym, aby framework Angular wczytał moduł zawierający funkcjonalność sklepu internetowego.

*Listing 7.16. Zaimportowanie modułów w pliku `app.module.ts`*

```
import { NgModule } from "@angular/core";
import { LOCALE_ID } from '@angular/core';
import { BrowserModule } from "@angular/platform-browser";
import { AppComponent } from "./app.component";
import { StoreModule } from "./store/store.module";

@NgModule({
  imports: [BrowserModule, StoreModule],
  declarations: [AppComponent],
  bootstrap: [AppComponent],
  providers: [{ provide: LOCALE_ID, useValue: "pl-PL" }]
})
export class AppModule {}
```

Po zapisaniu zmian w module głównym Angular ma wszystkie informacje niezbędne do wczytania aplikacji i wyświetlenia treści pochodzącej z modułu sklepu internetowego, jak pokazałem na rysunku 7.4.



Rysunek 7.4. Podstawowe funkcje w aplikacji sklepu internetowego

Wszystkie elementy konstrukcyjne przygotowane w poprzednich sekcjach współdziałają ze sobą i wyświetlają prostą treść. Na tym etapie otrzymujemy informacje o liczbie kategorii i produktów w sklepie.

## Dodawanie funkcji związanych z produktem

Programowanie z użyciem frameworka Angular oznacza powolny start, w trakcie którego przygotowywane są fundamenty projektu i tworzone są podstawowe elementy konstrukcyjne. Gdy ta praca zostanie wykonana, dodawanie kolejnych funkcji będzie już względnie prostym zadaniem. W kolejnych sekcjach zajmę się dodawaniem do sklepu internetowego funkcji pozwalających użytkownikowi na przeglądanie katalogu dostępnych produktów.

## Wyświetlanie szczegółów produktu

Wydaje się oczywiste, że pracę należy rozpocząć od wyświetlenia szczegółów produktów oferowanych klientowi. Na listingu 7.17 przedstawiłem elementy HTML, które trzeba umieścić w szablonie komponentu sklepu. Wyrażenia dołączania danych odpowiadają za wygenerowanie treści dla poszczególnych produktów.

Listing 7.17. Dodawanie elementów w pliku `store.component.html`

```
<div class="navbar navbar-inverse bg-inverse">
  <a class="navbar-brand">SKLEP SPORTOWY</a>
</div>
<div class="col-xs-3 bg-info p-a-1">
  {{categories.length}} kategorie
</div>
<div class="col-xs-9 p-a-1">
  <div *ngFor="let product of products" class="card card-outline-primary">
    <h4 class="card-header">
      {{product.name}}
      <span class="pull-xs-right tag tag-pill tag-primary">
        {{ product.price | currency:"PLN":true:"2.2-2" }}
      </span>
    </h4>
    <div class="card-text p-a-1">{{product.description}}</div>
  </div>
</div>
```

Większość elementów kontroluje układ i wygląd treści wyświetlanej na stronie. Najważniejszą zmianą jest dodanie wyrażenia dołączania danych Angular, jak pokazałem poniżej.

```
...
<div *ngFor="let product of products" class="card card-outline-primary">
...
```

To jest przykład dyrektywy przeprowadzającej transformację elementu HTML, do którego została zastosowana. W omawianym przykładzie mamy dyrektywę `ngFor`, której działanie polega na transformacji elementu `<div>` przez jego powielenie dla każdego obiektu zwróconego przez właściwość `products` komponentu. Angular oferuje wiele wbudowanych dyrektyw wykonujących najczęściej wymagane zadania. Więcej informacji na ten temat znajdziesz w rozdziale 13.

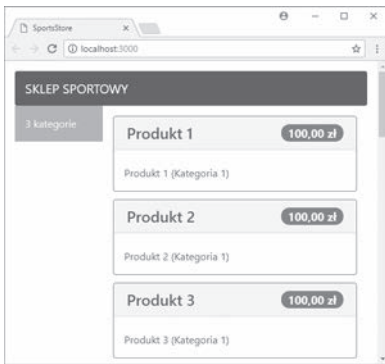
Podczas powielania elementu `<div>` bieżący obiekt zostaje przypisany zmiennej o nazwie `product`, do której można się łatwo odwoływać w innych wyrażeniach dołączania danych. Działanie przedstawionego tutaj wyrażenia powoduje wstawienie wartości właściwości `description` obiektu `product` jako treści elementu `<div>`, jak pokazałem poniżej.

```
...
<div class="card-text p-a-1">{{product.description}}</div>
...
```

Nie wszystkie dane znajdujące się w modelu aplikacji mogą być bezpośrednio wyświetlone użytkownikowi. Framework Angular oferuje funkcję o nazwie **potok**. Są to po prostu klasy używane do transformacji lub przygotowania wartości danych do użycia w procesie dołączania danych. Dostępnych jest wiele wbudowanych potoków, między innymi `currency`, który formatuje liczby jako wartości walutowe, jak pokazałem poniżej.

```
...
{{ product.price | currency:"PLN":true:"2.2-2" }}
...
```

Składnia zastosowania potoku może wydawać się nieco dziwna, ale wyrażenie w podanym poleceniu nakazuje Angular sformatowanie wartości właściwości `price` obiektu `product` z użyciem potoku `currency`, przy czym mają być zastosowane ustawienia polskiej waluty. Po zapisaniu zmian w szablonie zobaczysz długą listę produktów pobranych z modelu, jak pokazałem na rysunku 7.5.



Rysunek 7.5. Wyświetlanie informacji o produkcie

## Dodawanie możliwości wyboru kategorii

Zaimplementowanie obsługi filtrowania listy produktów według kategorii wymaga odpowiedniego przygotowania komponentu sklepu internetowego, tak aby śledził on kategorię, którą chce wyświetlić użytkownik. Konieczna będzie więc zmiana sposobu pobierania danych dla danej kategorii, jak pokazałem na listingu 7.18.

**Listing 7.18.** Zaimplementowanie w pliku `store.component.ts` filtrowania kategorii

```
import { Component } from "@angular/core";
import { Product } from "../model/product.model";
import { ProductRepository } from "../model/product.repository";
```

```

@Component({
  selector: "store",
  moduleId: module.id,
  templateUrl: "store.component.html"
})
export class StoreComponent {
  public selectedCategory = null;

  constructor(private repository: ProductRepository) {}

  get products(): Product[] {
    return this.repository.getProducts(this.selectedCategory);
  }

  get categories(): string[] {
    return this.repository.getCategories();
  }

  changeCategory(newCategory?: string) {
    this.selectedCategory = newCategory;
  }
}

```

Zmiany są proste, ponieważ opierają się na fundamentach tak pieczołowicie budowanych na początku rozdziału. Właściwości `selectedCategory` zostaje przypisana kategoria wybrana przez użytkownika (przy czym wartość `null` oznacza wszystkie kategorie). Ta wartość jest używana w metodzie `updateData()` jako argument metody `getProducts()`, delegując do źródła danych operację filtrowania produktów. Metoda `changeCategory()` wykorzystuje oba elementy składowe w metodzie, która może być wywołana, gdy użytkownik wybierze kategorię.

Na listingu 7.19 przedstawiłem zmiany konieczne do wprowadzenia w szablonie dokumentu, aby dostarczyć użytkownikowi zestaw przycisków pozwalających na zmianę kategorii i wyświetlić produkty w tej wybranej.

**Listing 7.19.** Dodanie w pliku `store.component.html` przycisków pozwalających na wybór kategorii

```

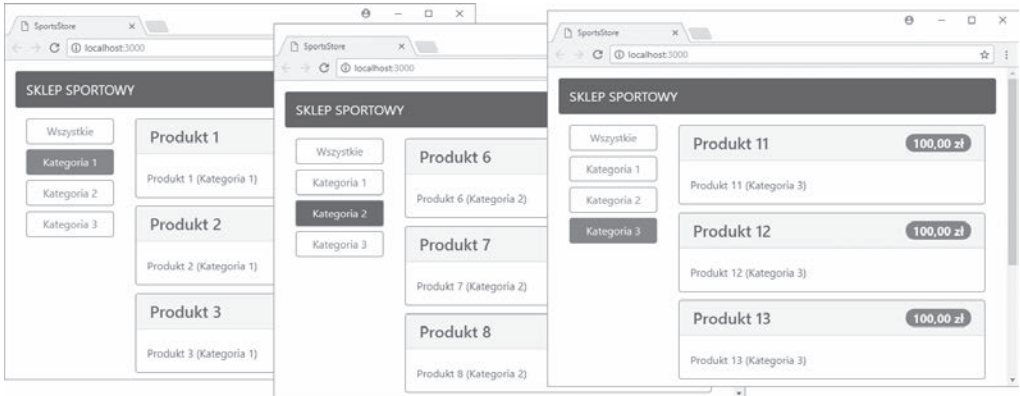
<div class="navbar navbar-inverse bg-inverse">
  <a class="navbar-brand">SKLEP SPORTOWY</a>
</div>
<div class="col-xs-3 p-a-1">
  <button class="btn btn-block btn-outline-primary" (click)="changeCategory()">
    Wszystkie
  </button>
  <button *ngFor="let cat of categories" class="btn btn-outline-primary btn-block"
    [class.active]="cat == selectedCategory" (click)="changeCategory(cat)">
    {{cat}}
  </button>
</div>
<div class="col-xs-9 p-a-1">
  <div *ngFor="let product of products" class="card card-outline-primary">
    <h4 class="card-header">
      {{product.name}}
      <span class="pull-xs-right tag tag-pill tag-primary">
        {{ product.price | currency:"PLN":true:"2.2-2" }}
      </span>
    </h4>
    <div class="card-text p-a-1">{{product.description}}</div>
  </div>
</div>

```



W szablonie pojawiły się dwa nowe elementy <button>. Pierwszy zatytułowany *Wszystkie* ma dołączone zdarzenie wywołujące metodę `changeCategory()` po kliknięciu przycisku. Metodzie nie jest przekazywany żaden argument, więc właściwość `selectedCategory` otrzymuje wartość `null`, co powoduje wyświetlenie wszystkich produktów.

Dyrektywa `ngFor` została zastosowana dla drugiego elementu <button>. Wyrażenie dyrektywy powoduje powielenie elementu dla każdej wartości w tablicy zwróconej przez właściwość `categories` komponentu. Element <button> ma przypisane zdarzenie `click`, którego wyrażenie wywołuje metodę `changeCategory()` w celu wybrania bieżącej kategorii, a tym samym wyświetlenia użytkownikowi jedynie produktów zaliczanych do danej kategorii. Mamy również wiązanie `class` dodające elementowi przycisku klasę `active`, gdy kategoria powiązana z danym przyciskiem jest wybraną kategorią. W ten sposób użytkownik otrzymuje wizualną informację o aktualnie wybranej kategorii, jak pokażalem na rysunku 7.6.



Rysunek 7.6. Wybór kategorii produktów

## Dodawanie stronicowania produktów

Filtrowanie produktów według kategorii pomaga w łatwiejszym zarządzaniu listą produktów. Jednak znacznie częściej stosowane jest podejście polegające na podzieleniu listy na mniejsze sekcje i przedstawianie ich wraz z przyciskami nawigacyjnymi pozwalającymi na poruszanie się między stronami produktów.

Na listingu 7.20 przedstawiłem usprawnioną wersję komponentu sklepu internetowego monitorującą bieżącą stronę i liczbę elementów na stronie.

Listing 7.20. Zaimplementowanie w pliku `store.component.ts` obsługi stronicowania

```
import { Component } from "@angular/core";
import { Product } from "../model/product.model";
import { ProductRepository } from "../model/product.repository";

@Component({
  selector: "store",
  moduleId: module.id,
  templateUrl: "store.component.html"
})
export class StoreComponent {
  public selectedCategory = null;
  public productsPerPage = 4;
  public selectedPage = 1;

  constructor(private repository: ProductRepository) {}
}
```

```

get products(): Product[] {
  let pageIndex = (this.selectedPage - 1) * this.productsPerPage
  return this.repository.getProducts(this.selectedCategory)
    .slice(pageIndex, pageIndex + this.productsPerPage);
}

get categories(): string[] {
  return this.repository.getCategories();
}

changeCategory(newCategory?: string) {
  this.selectedCategory = newCategory;
}

changePage(newPage: number) {
  this.selectedPage = newPage;
}

changePageSize(newSize: number) {
  this.productsPerPage = Number(newSize);
  this.changePage(1);
}

get pageNumbers(): number[] {
  return Array(Math.ceil(this.repository
    .getProducts(this.selectedCategory).length / this.productsPerPage)
    .fill(0).map((x, i) => i + 1));
}
}

```

Na listingu pojawiły się dwie nowe funkcje. Pierwsza to możliwość pobrania strony produktów, natomiast druga to zmiana wielkości strony — dzięki temu użytkownik może zdecydować o liczbie produktów umieszczonych na stronie.

Mamy tu do czynienia z pewnym dziwactwem, które trzeba będzie obejść za pomocą komponentu. Wbudowana dyrektywa `ngFor` ma ograniczenie polegające na generowaniu treści jedynie dla obiektów pochodzących z tablicy lub kolekcji, a nie na podstawie wartości licznika. Skoro konieczne jest wygenerowanie ponumerowanych przycisków nawigacji między stronami, trzeba utworzyć tablicę zawierającą te liczby. Spójrz na poniższy fragment kodu.

```

...
return Array(Math.ceil(this.repository.getProducts(this.selectedCategory).length
  / this.productsPerPage)).fill(0).map((x, i) => i + 1);
...

```

To polecenie tworzy nową tablicę, wypełnia ją zerami, a następnie używa metody `map()` do wygenerowania nowej tablicy wraz z sekwencją liczb. Tego rodzaju podejście sprawdza się doskonale podczas implementacji funkcji stronicowania, ale wygląda okropnie. W następnej sekcji przedstawię znacznie lepsze podejście. Na listingu 7.21 pokazałem zmiany konieczne do wprowadzenia w szablonie sklepu internetowego, aby móc wykorzystać możliwość stronicowania produktów.

**Listing 7.21.** Wykorzystanie stronicowania produktów w pliku `store.component.html`

```

<div class="navbar navbar-inverse bg-inverse">
  <a class="navbar-brand">SKLEP SPORTOWY</a>
</div>
<div class="col-xs-3 p-a-1">
  <button class="btn btn-block btn-outline-primary" (click)="changeCategory()">
    Wszystkie

```

```

</button>
<button *ngFor="let cat of categories" class="btn btn-outline-primary btn-block"
  [class.active]="cat == selectedCategory" (click)="changeCategory(cat)">
  {{cat}}
</button>
</div>
<div class="col-xs-9 p-a-1">
  <div *ngFor="let product of products" class="card card-outline-primary">
    <h4 class="card-header">
      {{product.name}}
      <span class="pull-xs-right tag tag-pill tag-primary">
        {{ product.price | currency:"PLN":true:"2.2-2" }}
      </span>
    </h4>
    <div class="card-text p-a-1">{{product.description}}</div>
  </div>
  <div class="form-inline pull-xs-left m-r-1">
    <select class="form-control" [value]="productsPerPage"
      (change)="changePageSize($event.target.value)">
      <option value="3">3 na stronie</option>
      <option value="4">4 na stronie</option>
      <option value="6">6 na stronie</option>
      <option value="8">8 na stronie</option>
    </select>
  </div>
  <div class="btn-group pull-xs-right">
    <button *ngFor="let page of pageNumbers" (click)="changePage(page)"
      class="btn btn-outline-primary" [class.active]="page == selectedPage">
      {{page}}
    </button>
  </div>
</div>

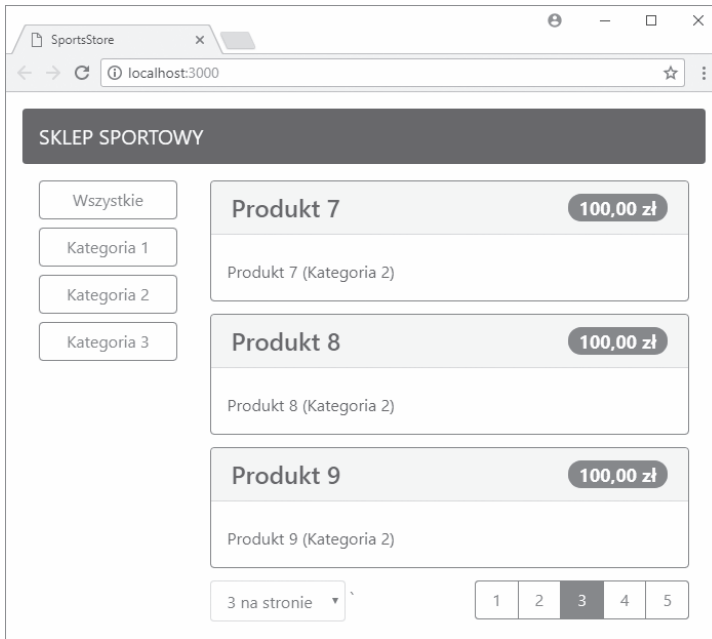
```

Nowy kod powoduje dodanie elementu `<select>` pozwalającego użytkownikowi na zmianę liczby produktów wyświetlanych na stronie oraz zestawu przycisków nawigacyjnych do poruszania się po stronach produktów. Te nowe elementy mają zdefiniowane wyrażenia dołączania danych łączące je z właściwościami i metodami dostarczonymi przez komponent. W efekcie wprowadzonych zmian otrzymujemy znacznie łatwiejszy w zarządzaniu zbiór produktów, jak pokazałem na rysunku 7.7.

- 
- **Wskazówka** Element `<select>` przedstawiony na listingu 7.21 zawiera elementy `<option>` zdefiniowane statycznie, a nie utworzone na podstawie danych komponentu. Dlatego też po przekazaniu wybranej wartości metodzie `changePageSize()` będzie ona typu `string`, stąd konieczność zmiany argumentu na typ `number` przed jego użyciem do zdefiniowania wielkości strony na listingu 7.20. Trzeba zachować szczególną ostrożność podczas używania wartości pochodzących z elementów formularza HTML i upewnić się, że są one oczekiwanego typu. Oferowane przez TypeScript adnotacje skryptu tutaj nie pomogą, ponieważ wyrażenie dołączania danych jest wykonywane w trakcie działania programu, czyli znacznie później po tym, gdy kompilator TypeScript wygenerował kod JavaScript niezawierający dodatkowych informacji o typie.
- 

## Utworzenie własnej dyrektywy

W tej sekcji zamierzam utworzyć własną dyrektywę, aby uniknąć konieczności generowania tablicy liczb potrzebnej do przygotowania przycisków nawigacji między stronami w trakcie stronicowania produktów. Framework Angular oferuje dość szeroką gamę wbudowanych dyrektyw. Prosty proces utworzenia własnej



**Rysunek 7.7.** Zaimplementowane stronicowanie produktów

dyrektywy rozwiązuje problem w naszej aplikacji, pozwala również na dodanie obsługi funkcji nieoferowanych przez wbudowane dyrektywy. W katalogu `SportsStore/src/app/store` utwórz nowy plik o nazwie `counter.directive.ts` i umieść w nim kod przedstawiony na listingu 7.22.

**Listing 7.22.** Zawartość pliku `counter.directive.ts` w katalogu `SportsStore/src/app/store`

```
import {
  Directive, ViewContainerRef, TemplateRef, Input, Attribute, SimpleChanges
} from "@angular/core";

@Directive({
  selector: "[counterOf]"
})
export class CounterDirective {
  constructor(private container: ViewContainerRef,
    private template: TemplateRef<Object>) {
  }

  @Input("counterOf")
  counter: number;

  ngOnChanges(changes: SimpleChanges) {
    this.container.clear();
    for (let i = 0; i < this.counter; i++) {
      this.container.createEmbeddedView(this.template,
        new CounterDirectiveContext(i + 1));
    }
  }
}
```

```

}

class CounterDirectiveContext {
  constructor(public $implicit: any) {}
}

```

To jest przykład dyrektywy strukturalnej, ten rodzaj dyrektyw omówię dokładnie w rozdziale 16. Dyrektywa będzie stosowana w elemencie za pomocą właściwości `counter`. Opiera ona swoje działanie na funkcjach specjalnych dostarczanych przez framework Angular na potrzeby powtarzającego się tworzenia treści, podobnie jak ma to miejsce w przypadku dyrektywy `ngFor`. W takim przypadku zamiast pobierać poszczególne obiekty z kolekcji, dyrektywa pobiera serię liczb, które następnie mogą być użyte do utworzenia przycisków nawigacji między stronami produktów.

- 
- **Wskazówka** W przypadku zmiany liczby stron ta dyrektywa powoduje usunięcie całej utworzonej przez siebie treści i rozpoczyna proces jej tworzenia zupełnie od początku. W bardziej skomplikowanych dyrektywach może to być dość kosztowny proces, dlatego w rozdziale 16. pokażę, jak można poprawić wydajność jego działania.
- 

Aby móc użyć dyrektywy, należy ją dodać do właściwości `declarations` zwykłego modułu, jak pokazałem na listingu 7.23.

**Listing 7.23.** Przykład rejestrowania własnej dyrektywy w pliku `store.module.ts`

```

import { NgModule } from "@angular/core";
import { BrowserModule } from "@angular/platform-browser";
import { FormsModule } from "@angular/forms";
import { ModelModule } from "../model/model.module";
import { StoreComponent } from "./store.component";
import { CounterDirective } from "./counter.directive";

@NgModule({
  imports: [ModelModule, BrowserModule, FormsModule],
  declarations: [StoreComponent, CounterDirective],
  exports: [StoreComponent]
})
export class StoreModule {}

```

Po zarejestrowaniu dyrektywy można ją wykorzystać w szablonie komponentu sklepu internetowego i zastąpić nią wbudowaną dyrektywę `ngFor`, jak pokazałem na listingu 7.24.

**Listing 7.24.** Zastąpienie w pliku `store.component.html` wbudowanej dyrektywy `ngFor`

```

<div class="navbar navbar-inverse bg-inverse">
  <a class="navbar-brand">SKLEP SPORTOWY</a>
</div>
<div class="col-xs-3 p-a-1">
  <button class="btn btn-block btn-outline-primary" (click)="changeCategory()">
    Wszystkie
  </button>
  <button *ngFor="let cat of categories" class="btn btn-outline-primary btn-block"
    [class.active]="cat == selectedCategory" (click)="changeCategory(cat)">
    {{cat}}
  </button>
</div>
<div class="col-xs-9 p-a-1">
  <div *ngFor="let product of products" class="card card-outline-primary">
    <h4 class="card-header">

```

```

        {{product.name}}
        <span class="pull-xs-right tag tag-pill tag-primary">
            {{ product.price | currency:"PLN":true:"2.2-2" }}
        </span>
    </h4>
    <div class="card-text p-a-1">{{product.description}}</div>
</div>
<div class="form-inline pull-xs-left m-r-1">
    <select class="form-control" [value]="productsPerPage"
        (change)="changePageSize($event.target.value)">
        <option value="3">3 na stronie</option>
        <option value="4">4 na stronie</option>
        <option value="6">6 na stronie</option>
        <option value="8">8 na stronie</option>
    </select>
</div>
<div class="btn-group pull-xs-right">
    <button *counter="let page of pageCount" (click)="changePage(page)"
        class="btn btn-outline-primary" [class.active]="page == selectedPage">
        {{page}}
    </button>
</div>
</div>
</div>

```

Nowe wyrażenie dołączania danych opiera działanie na właściwości o nazwie `pageCount` konfigurującej niestandardową dyrektywę. Na listingu 7.25 pokazałem zastąpienie tablicy liczb prostym wywołaniem dostarczającym wartość dla wyrażenia.

**Listing 7.25. Przykład użycia w pliku `store.component.ts` własnej dyrektywy**

```

import { Component } from "@angular/core";
import { Product } from "../model/product.model";
import { ProductRepository } from "../model/product.repository";

@Component({
    selector: "store",
    moduleId: module.id,
    templateUrl: "store.component.html"
})
export class StoreComponent {
    public selectedCategory = null;
    public productsPerPage = 4;
    public selectedPage = 1;

    constructor(private repository: ProductRepository) {}

    get products(): Product[] {
        let pageIndex = (this.selectedPage - 1) * this.productsPerPage
        return this.repository.getProducts(this.selectedCategory)
            .slice(pageIndex, pageIndex + this.productsPerPage);
    }

    get categories(): string[] {
        return this.repository.getCategories();
    }

    changeCategory(newCategory?: string) {
        this.selectedCategory = newCategory;
    }
}

```

```

changePage(newPage: number) {
  this.selectedPage = newPage;
}

changePageSize(newSize: number) {
  this.productsPerPage = Number(newSize);
  this.changePage(1);
}

get pageCount(): number {
  return Math.ceil(this.repository
    .getProducts(this.selectedCategory).length / this.productsPerPage)
}

// get pageNumbers(): number[] {
//   return Array(Math.ceil(this.repository
//     .getProducts(this.selectedCategory).length / this.productsPerPage)
//     .fill(0).map((x, i) => i + 1);
// }
}

```

Nie powinieneś zauważyć żadnej widocznej zmiany w sposobie działania aplikacji SportsStore, chociaż w tej sekcji pokazałem, jak wbudowaną funkcjonalność frameworka Angular można uzupełnić własnym kodem dopasowanym do potrzeb danego projektu.

## Podsumowanie

W tym rozdziale rozpoczęliśmy proces budowy aplikacji SportsStore. Na początku rozdziału zająłem się przygotowaniem struktury projektu, między innymi zainstalowałem i skonfigurowałem narzędzia programistyczne, utworzyłem główne elementy konstrukcyjne aplikacji i rozpocząłem pracę nad poszczególnymi modułami. Po zakończeniu etapu przygotowań bardzo szybko byłem w stanie dodać funkcje pozwalające na wyświetlanie użytkownikowi fikcyjnych danych, a także zaimplementować stronicowanie oraz filtrowanie produktów według kategorii. Na końcu rozdziału pokazałem, jak można utworzyć własną dyrektywę oraz jak funkcje oferowane przez framework Angular uzupełnić własnym kodem. W następnym rozdziale będę kontynuował pracę nad aplikacją SportsStore.





# Skorowidz

## A

- adnotacja typu, 115, 117
- adres URL, 60, 156, 189, 629
- aktywacja trasy, 643, 644
- analiza pliku bootstrap, 130
- AngularJS, 58
- animacje, 667, 672
  - funkcje czasu, 684
  - opóźnienie początkowe, 685
  - równoczesne odtwarzanie, 688
  - style, 686, 689
  - testowanie, 679
  - transformacje, 675, 684
  - tworzenie, 673
  - wyzwalacz, 676, 694
  - zastosowanie, 676
- AOT, ahead-of-time, 714
- aplikacje
  - dwukierunkowe, 52
  - w postaci pojedynczej strony, 52
- argumenty potoku, 427
- arkusze stylów, 68
  - Bootstrap CSS, 35, 128
- asynchroniczne
  - operacje, 721
  - żądanie HTTP, 555
- atak typu XSS, 570
- atrybut, 66, 244
  - bez wartości, 66
  - cytowanie literałów, 66

- elementu, 332
- weryfikacji danych, 297

## B

- biblioteka
  - jQuery, 54
  - Reactive Extensions, 533, 537
  - typu polyfill, 412
- błąd weryfikacji danych, 303
- błędy, 575
- Bootstrap, 128, 211
  - CSS, 35, 128
  - dopełnienia, 70
  - nadawanie stylu tabeli, 72
  - tworzenie
    - formularzy HTML, 74
    - elastycznego układu, 77
  - układ oparty na siatce, 75, 78
  - użycie
    - klas kontekstu, 69
    - marginisu, 70
  - zmiana wielkości elementu, 71

## C

- cel, 238
- ciąg tekstowy, 94
  - interpolacja, 243
  - konwersja liczby, 98
  - konwersja na liczbę, 99
  - metody, 94

- szablon, 95
- zmiana wielkości znaków, 433

- CRUD, create, read, update, delete, 57, 63, 68, 123, 692

- CSSOM, css object model, 692
- cytowanie literałów, 66

## D

- dane
  - kontekstu, 273
  - modelu, 36, 131
  - widoku, 58
  - zdarzenia, 289, 290
- definicja
  - funkcji, 88–91
  - klasy, 108
  - modułu, 503, 508, 513
  - szablonu, 387
  - typu, 117
  - zewnętrznego szablonu, 388
  - zewnętrznych stylów komponentu, 398
- dekorator, 40, 82
  - @Component, 129
  - @ContentChild, 376
  - @ContentChildren, 377
  - @HostBinding, 341
  - @HostListener, 341
  - @NgModule, 497, 503
  - @Pipe, 416

- dekorator
    - @SkipSelf, 490
    - komponentu, 384
  - Docker, 200
  - dodawanie
    - formularza, 295
    - funkcji, 42, 138
    - łączy, 589
    - pakietów, 209, 518
    - stronicowania produktów, 141
    - zadań, 47
  - dokument, 67
    - HTML, 229, 530
  - dołączanie
    - atrybutów, 244
    - danych, 46, 235, 389, 535
      - cel, 239
      - do własnego zdarzenia, 339
      - dwukierunkowe, 237, 292, 341
      - interpolacja ciągu
        - tekstowego, 243
      - jednokierunkowe, 237, 275
      - testowanie, 712
      - w elemencie, 340
      - wejściowych, 333
    - klasy, 246, 247
    - stylu, 250
    - właściwości, 239–242
    - zdarzeń, 48, 285, 286, 288
  - DOM, document object model, 53, 692
  - domknięcie, 93
  - dopełnienie, 70
  - dostarczanie danych kontekstu, 273, 275
  - dostawca
    - fabryki, 477
    - tokena, 537
    - usługi, 463, 465, 480
    - wartości, 475
  - dostawcy
    - dla elementów potomnych
      - widoku, 488
    - lokalni, 481
    - wbudowani, 468
  - dostęp
    - do danych aplikacji, 330
    - do parametrów, 629
  - dwukierunkowe dołączanie
    - danych, 46, 292
      - w elemencie, 341
  - dynamiczne
    - definiowanie właściwości, 287
    - wczytywanie modułów, 654
  - dyrektywa
    - ngClass, 239, 248, 260
    - ngFor, 141, 239, 260, 264, 266, 269
    - ngIf, 239, 259–261
    - ngModel, 294
    - ngStyle, 239, 252, 260
    - ngSwitch, 239, 260, 262
    - ngSwitchCase, 239, 263
    - ngSwitchDefault, 239
    - ngTemplateOutlet, 239, 260, 273
  - dyrektywy
    - \*ngFor, 268
    - atrybutu, 325, 328
    - deklarowanie zależności, 451
    - dostawcy lokalni, 482
    - dostępu do danych, 330
    - mikroszablonów, 261
    - strukturalne, 261, 349, 350
      - dane kontekstu, 359
    - iteracyjne, 357
    - klasy, 352
    - pobieranie treści elementu, 373
    - składnia, 356, 361
    - tworzenie, 350
    - włączanie, 354
    - zmiany danych, 362, 363
    - testowanie, 723
    - wbudowane, 257, 259
  - dziedziczenie, 110
- ## E
- edycja pliku HTML, 34
  - edytor
    - produktu, 194
    - tekstu, 29
      - Atom, 29
      - Brackets, 29
      - Sublime Text, 29
      - Visual Studio Code, 29
      - WebStorm, 29
  - edytowanie pliku HTML, 33
  - eksportowanie dyrektywy, 345
  - elastyczne układy, 77
  - element, *Patrz* znacznik
  - elementy
    - aplikacji Angular, 219
    - HTML, 67, 237, 285
    - samozamykające się, 65
- ## F
- fikcyjne źródła danych, 132
  - filtrowanie
    - kategorii, 140
    - zadań, 47
    - zdarzeń, 547, 549
      - klawiszy, 292
  - formatowanie wartości
    - daty i godziny, 430
    - liczbowych, 423
    - procentowych, 428
    - walutowych, 425
  - formaty modułu, 112
  - formularze HTML, 74, 281, 295
    - klasy modelu, 313
    - oparte na modelu, 312
    - style komponentu, 526
    - utworzenie komponentu, 524
    - weryfikacja danych, 297, 299, 306, 316
      - własne reguły weryfikacji, 320
      - właściwości bezpiecznej nawigacji, 302
      - wyłączenie przycisku, 310
  - framework
    - Bootstrap, 68
    - Bootstrap CSS, 35
    - Jasmine, 706
  - funkcja, 42, 107
    - animate, 686
    - constructor, 108
    - Number, 99
    - parseFloat, 99
    - parseInt, 99
    - shadow DOM, 402

funkcje, 88  
 administracyjne, 189  
 definiowanie, 88  
 hoisting, 89  
 jako argument innej funkcji, 91  
 jako metody, 107  
 języka TypeScript, 115  
 parametry domyślne, 90  
 parametry resztowe, 90  
 polimorfizm, 90  
 strzałki, 92  
 transformacji CSS, 692  
 wyrażenie, 88  
 z parametrami, 89  
 zaawansowane, 515  
 zwracające wartość, 91

## G

generowanie  
 elementów, 319  
 formularza, 319  
 getter, 109

## H

hermetyzacja widoku, 400  
 hoisting funkcji, 89  
 HTML, 63, 64

## I

idempotentne wyrażenia, 275  
 implementacja  
 edytora produktu, 194  
 funkcji obsługi zamówienia, 196  
 klasy dyrektywy  
 strukturalnej, 352  
 routingu, 155, 594  
 uwierzytelniania, 179  
 wzorca MVC, 55  
 import  
 typów, 112, 114  
 z modułu, 112  
 informacje  
 o animacji, 668  
 o bibliotece Reactive  
 Extensions, 533

o dostawcy, 463  
 o dyrektywie atrybutu, 325  
 o dyrektywie strukturalnej, 349  
 o formularzach, 281  
 o kompetencie, 381  
 o module, 493  
 o potoku, 409  
 o routingu, 581  
 o stanie, 522  
 o testach jednostkowych, 699  
 o usłudze, 437  
 o wbudowanych  
 dyrektywach, 257  
 o zdarzeniach, 281  
 o żądaniach HTTP, 555

instalowanie

biblioteki typu polyfill, 412  
 edytora tekstu, 29  
 narzędzia Docker, 200  
 Node.js, 27  
 pakietów menedżera npm, 32  
 pakietu angular-cli, 28  
 przeglądarki WWW, 30  
 koszyka na zakupy, 153

interpolacja ciągu tekstowego, 243

izolacja testu, 455

izolowanie komponentów, 455

## J

Jasmine, 706  
 JavaScript, 81, 105  
 jawna konwersja typu, 97  
 jednokierunkowe  
 dołączanie danych, 275  
 język  
 HTML, 64  
 JavaScript, 81  
 TypeScript, 81, 115  
 JSON, 434  
 JSONP, 571  
 JWT, json web token, 180

## K

kaskadowe arkusze stylów, 68  
 katalogi, 124

klasa, 40, 108

ActivatedRoute, 595, 620  
 AnimationTransitionEvent, 694  
 btn-lg, 72  
 ComponentFixture, 710  
 DebugElement, 713  
 EventEmitter, 339  
 Map, 372  
 ng-invalid, 299  
 ng-pristine, 299  
 ng-untouched, 299  
 ng-valid, 299  
 Observable, 538  
 Observer, 539  
 Response, 563  
 Router, 602  
 Routes, 587  
 SimpleChange, 336  
 StaticDataSource, 132  
 Subject, 540  
 TestBed, 708, 710  
 Validators, 314  
 ViewContainerRef, 353  
 klasy  
 Bootstrap, 69  
 CSS, 70  
 differs, 367  
 dyrektywy strukturalnej, 352  
 dziedziczenie, 110  
 kontekstu, 69  
 modelu, 131  
 modelu formularza, 313  
 opisowe modelu, 221  
 kompilator TypeScript, 84  
 komponent, 39, 190, 381  
 formularza, 524  
 główny, 382, 396  
 routingu, 588  
 tabeli, 523  
 komponenty  
 dostawcy lokalni, 483  
 funkcje stylów, 400  
 koordynacja, 393  
 nadrzędne, 392  
 nawigacja, 618  
 potomne, 391  
 restrukturyzacja, 396  
 strukturyzacja aplikacji, 382

- komponenty
    - style, 398
    - w szablonie, 404
    - zewnętrzne, 398
  - szablony, 387
  - testowanie, 708, 714
  - testowanie zdarzeń, 716
  - treści szablonu, 406
  - tworzenie, 384
  - komunikaty
    - podsumowania weryfikacji danych, 308
    - weryfikacji danych, 301, 304
  - konfiguracja
    - języka TypeScript, 212
    - kompilatora TypeScript, 84, 213
    - nagłówków żądania, 573
    - pakietów, 518
    - pakietu Karma, 703
    - procedury wczytywania modułu, 226
    - JavaScript, 520
    - programistycznego serwera HTTP, 215, 519
    - projektu, 209
    - routingu, 157, 586, 671, 672
    - serwera HTTP, 201
    - systemu routingu, 178
    - TypeScript, 519
    - usługi uwierzytelniania, 182
    - źródła danych, 564
    - żądania HTTP, 562
  - konsolidowanie żądań HTTP, 569
  - konstrukcje warunkowe, 96
  - kontekst wyrażenia, 278
  - kontener, 199, 201
    - uruchamianie aplikacji, 202
  - kontrola wersji, 28
  - kontroler, 58
  - konwersja typu, 97
  - koordynacja
    - między komponentami, 390, 393
  - koszyk na zakupy, 149
    - podsumowanie, 151
  - krotka, 120
- L**
- liczby, 95
  - literał
    - obiektu, 107
    - tablicy, 100
  - logika modelu, 56
  - LTS, long term support, 28
- Ł**
- łącza, 589
  - łączenie potoków, 418
- M**
- mapa, 372
  - mapowanie adresu URL, 157
  - margins, 70
  - mechanizm
    - dołączania danych, 46, 235
    - wstrzykiwania zależności, 437, 444, 447, 455
  - menedżer npm, 32, 210
  - metoda, 107
    - afterEach, 707
    - beforeEach, 707
    - clear, 353
    - compileComponents, 708
    - concat, 94, 101
    - configureTestingModule, 708
    - createComponent, 708
    - createEmbeddedView, 353
    - deleteProduct, 223
    - describe, 707
    - detach, 353
    - detectChanges, 710
    - emit, 339
    - every, 102
    - expect, 707
    - filter, 102
    - find, 102
    - findIndex, 102
    - foreach, 102
    - get, 353
    - getProduct, 223
    - getProducts, 223
    - includes, 102
    - indexOf, 94, 353
    - insert, 353
    - it, 707
    - join, 101
    - map, 102
    - ngAfterContentChecked, 335
    - ngAfterContentInit, 335
    - ngDoCheck, 335
    - ngOnChanges, 335
    - ngOnDestroy, 335
    - ngOnInit, 335
    - pop, 101
    - push, 101
    - queryAll, 713
    - reduce, 102
    - remove, 353
    - replace, 94
    - reverse, 101
    - saveProduct, 223
    - shift, 101
    - slice, 94, 101
    - some, 102
    - sort, 101
    - splice, 102
    - split, 94
    - toBe, 707
    - toBeDefined, 707
    - toBeFalsy, 707
    - toBeGreaterThan, 708
    - toBeLessThan, 707
    - toBeNull, 707
    - toBeTruthy, 707
    - toBeUndefined, 707
    - toContain, 707
    - toEqual, 707
    - toExponential, 98
    - toFixed, 98
    - convertFtoC, 116
    - GET, 60
    - getStyles, 253
    - IterableDiffer.diff, 368
    - reduce, 102
    - resolve, 638
    - RouterModule.forRoot, 158
    - submitOrder, 168
    - subscribe, 538
    - toLowerCase, 94
    - toMatch, 707
    - toPrecision, 98, 119
    - toString, 98

- toUpperCase, 94
  - TriggerEventHandler, 713
  - trim, 94
  - unshift, 101, 102
  - whenStable, 710
  - metody
    - biblioteki Reactive
      - Extensions, 547
    - HTTP, 562
    - idempotentne, 60
    - Jasmin, 707
    - klasy
      - ComponentFixture, 710
      - DebugElement, 713
      - Response, 563
      - Router, 602
      - SimpleChange, 336
      - TestBed, 708
      - ViewContainerRef, 353
    - nieidempotentne, 60
    - obiektu
      - ciągu tekstowego, 94
      - Headers, 575
      - Observer, 539
    - zaczepów cyklu życiowego, 334
  - mikroszablon, 261
  - model, 56, 221
    - do weryfikacji danych, 316
    - do wygenerowania formularza, 319
  - DOM, 53
    - właściwość target, 289
    - właściwość timeStamp, 289
    - właściwość type, 289
  - formularza, 319
  - koszyka na zakupy, 150
  - model-widok-kontroler, 23, 54
  - moduły, 41, 111
    - Angular, 225, 497
    - core, 522, 526
    - główne, 41, 224, 495
    - formaty, 112
    - funkcjonalne, 134, 190, 501
    - importowanie typów, 114
    - JavaScript, 226, 497
    - messages, 527, 528
    - modelu, 502, 520
    - narzędziowe, 506, 507
    - obsługi formularzy, 282
    - Reactive Extensions, 529
    - tworzenie, 111
    - uaktualnianie, 505
    - uaktualnianie klas, 507
    - wczytywanie, 113, 557
    - z komponentami, 511
    - zmiana nazwy, 114
  - modyfikatory dostępu, 121
  - monitorowanie widoków, 369
  - MVC, model-view-controller, 23, 54
    - format danych, 62
    - umieszczanie logiki, 61
- ## N
- narzędzie
    - Docker, 200
    - git, 28
  - nawias
    - kwadratowy, 238, 241
    - okrągły, 285
  - nawigacja, 581, 602, 609, 618, 633, 642, 664
  - nawigacja po aplikacji, 158
  - nazwa importowanego modułu, 114
  - Node.js, 27
- ## O
- obiekt QueryList, 377
  - obiekty, 106
    - literał, 107
    - mapy stylu, 252
    - weryfikacji, 302
    - zdarzeń, 551
  - obietnice, 539
  - obserwator, 217
  - obsługa
    - animacji, 668
    - błędów, 575, 578
    - błędu HTTP, 579
    - formularzy, 282
    - produktu, 193
    - zamówienia, 196
    - zawartości koszyka, 163
    - zmiany trasy, 594
  - odczyt atrybutów elementu, 331
  - ograniczanie wyszukiwania dostawcy, 489
  - ograniczenia obiektu usługi, 481
  - określenie wielu typów, 119
  - opcje
    - kompilatora TypeScript, 214
    - konfiguracyjne BrowserSync, 217
  - operacje
    - asynchroniczne, 721
    - elementu, 269
    - typu CRUD, 123
  - operator
    - identyczności, 97
    - równości, 97
  - operatory JavaScript, 96
  - opóźnienie
    - HTTP, 670
    - nawigacji, 635
  - outlet, 661
    - trasy potomnej, 627
- ## P
- pakiet
    - angular-cli, 28, 34
    - Bootstrap CSS, 32
    - classlist.js, 211
    - concurrently, 212, 217
    - Karma, 703
    - lite-server, 212
    - reflect-metadata, 211
    - rxjs, 211
    - systemjs, 211
    - typescript, 212
    - typings, 212
    - zone.js, 211
  - pakiety
    - dodawanie, 518
    - konfigurowanie, 518
    - menedżera npm, 32
    - zależne, 211
  - parametry
    - domyślne, 90
    - resztowe, 90
    - trasy, 597–601
  - pierwsza aplikacja, 27

- plik
  - addTax.pipe.ts, 415
  - animationUtils.ts, 693
  - app.component.ts, 158
  - app.components.ts, 47
  - app.module.ts, 130, 137, 157, 312, 385, 441, 449, 467, 471, 474, 477, 480, 482, 485, 496, 501
  - app.routing.ts, 586, 617, 672
  - attr.directive.spec.ts, 724
  - attr.directive.ts, 340, 723
  - auth.component.ts, 176
  - authMiddleware.js, 126
  - cart.model.ts, 150
  - cartDetail.component.html, 163
  - cartSummary.component.html, 152
  - cartSummary.component.ts, 151
  - categoryFilter.pipe.ts, 420, 504, 507
  - cellColor.directive.ts, 373
  - cellColorSwitcher.directive.ts, 376, 378
  - cellColorSwitches.directive.ts, 482
  - checkout.component.html, 169
  - checkout.component.ts, 168
  - compilertest.ts, 214
  - component.ts, 224, 249–252, 271, 275, 279, 304, 327, 397
  - core.module.ts, 526, 541, 543, 584, 614, 652
  - counter.directive.ts, 144
  - data.js, 126
  - deploy-package.json, 200
  - discount.service.ts, 469
  - discountAmount.directive.ts, 451
  - discountEditor.component.ts, 509
  - DuplicateName.ts, 114
  - first.component.spec.ts, 711–722
  - first.component.ts, 710, 714–721
  - form.component.html, 525, 591
  - form.component.ts, 536, 538, 551–553, 598, 603
  - form.model.ts, 314
  - index.html, 64, 208
  - iterator.directive.ts, 358, 362, 363, 366, 370
  - karma-test-shim.js, 704
  - log.service.ts, 478
  - main.ts, 42, 130
  - message.module.ts, 528
  - model.module.ts, 151, 172, 503
  - model.ts, 37
  - NameAndWeather.ts, 111, 118
  - order.model.ts, 165
  - order.repository.ts, 167
  - package.json, 30, 32, 63, 83, 209, 282, 518, 669, 706
  - primer.ts, 86, 93, 120
  - product.model.ts, 131
  - product.repository.ts, 133, 187
  - productForm.component.html, 394, 447, 486
  - productForm.component.ts, 393, 490, 504, 512
  - productTable.component.html, 414
  - productTable.component.ts, 190, 384, 390, 411, 430
  - repository.model.ts, 270, 521, 567, 609
  - rest.datasource.ts, 561, 566, 573, 636, 670
  - sharedState.model.ts, 523, 537
  - static.datasource.ts, 132, 521
  - store.component.html, 138, 143
  - store.component.ts, 135, 146, 153
  - store.module.ts, 145, 156
  - systemjs.config.js, 227, 283, 583
  - table.animations.ts, 687, 693
  - table.component.html, 615, 695
  - table.component.ts, 523, 695
  - tempConverter.ts, 115, 116, 122
  - template.html, 224, 249, 251, 267, 294, 301, 311
  - typings.json, 519
  - unsaved.guard.ts, 651
- pliki
  - HTML, 33, 35, 127
  - typu bootstrap, 130, 225, 529
- po pobieraniu
  - szczegółów zamówienia, 168
  - treści elementu, 373
  - treści szablonu, 406
  - zdarzeń, 553
- podział danych tablicy, 435
- pojedyncza strona, 52
- polecenie, 87
  - import, 39, 113
- polimorfizm, 90
- pojmanie zdarzeń, 553, 554
- potok, 139, 409, 414
  - async, 542, 543
  - currency, 427
  - date, 432
  - deklarowanie zależności, 449
  - json, 434
  - łączenie, 418
  - nieczysty, 419
  - rejestrowanie, 416
  - slice, 435
  - tworzenie, 415
  - wbudowany, 423
  - zastosowanie, 417
- powiadomienia o zmianie zapytania, 378
- procedura
  - wczytująca moduł, 40, 85, 229
  - JavaScript, 520
- proces obserwatora, 217
- projekcja treści elementu, 395
- projekt
  - Angular, 207
  - dodanie frameworka Angular, 35
  - dodawanie funkcji, 42

dodawanie zadań, 47  
 JavaScriptPrimer, 82, 105  
 SportsStore, *Patrz* sklep internetowy  
 przeglądarka WWW, 30, 42  
 przekazywanie obiektów, 439  
 przekierowanie, 618, 619  
 przycisk wysyłający formularz, 310

## R

Reactive Extensions, 533, 537  
 reguły weryfikacji formularza, 320  
 rejestrowanie usług, 446, 456  
 usługi resolvera, 638  
 własnego potoku, 416  
 repozytorium, 166, 185, 520  
 modelu, 133, 222  
 resolver, 635, 639  
 REST, representational state transfer, 560  
 restrukturyzacja komponentu głównego, 396  
 routing, 155, 161, 178, 581, 586, 609, 633  
 URL, 129, 157

## S

selektory CSS, 402  
 serializowanie danych, 434  
 serwer HTTP, 201, 215  
 konfiguracja, 519  
 setter, 109  
 shadow DOM, 401  
 siatka, 75  
 skalowanie, 544  
 sklep internetowy, 123, 135, 149, 175, 199,  
 administracyjny adres URL, 179  
 dodawanie funkcji, 138  
 dodawanie nowego produktu, 196  
 edytor produktu, 194  
 funkcje administracyjne, 189

integracja koszyka na zakupy, 153  
 komponent, 135, 137  
 komponenty zawartości koszyka, 156  
 koszyk na zakupy, 149  
 moduł funkcjonalny, 136  
 moduł główny, 137  
 moduły funkcjonalne, 190  
 obsługa produktu, 193  
 zamówienia, 196  
 zawartości koszyka, 163  
 przetwarzanie zamówienia, 165  
 routing, 155  
 składanie zamówienia, 156  
 stronicowanie produktów, 141  
 struktura katalogów, 124  
 strony, 135  
 szablon, 135  
 szczegóły zamówienia, 168  
 uruchamianie aplikacji, 202  
 usługa sieciowa typu RESTful, 171  
 uwierzytelniania, 179, 182  
 wdrożenie aplikacji, 199  
 własne dyrektywy, 144  
 włączenie uwierzytelniania, 183  
 wybór kategorii, 139  
 wyświetlanie szczegółów produktu, 138  
 zarządzanie zamówieniem, 198  
 złożenie zamówienia, 170  
 składanie zamówienia, 156  
 skrypt polyfill, 668  
 słowo kluczowe any, 119  
 class, 108  
 export, 39, 112  
 extends, 110  
 function, 88, 108  
 get, 110  
 import, 39, 113  
 let, 88, 92

new, 108  
 private, 119, 121  
 protected, 121  
 public, 121  
 set, 110  
 static, 108  
 var, 93  
 void, 120  
 spełnianie zależności, 474, 489  
 kontrolowanie, 489  
 SportsStore, *Patrz* sklep internetowy  
 sprawdzanie równości, 552  
 stosowanie wzorca MVC, 61  
 strażnik, 642  
 stronicowanie produktów, 141, 144  
 struktura aplikacji, 386  
 dokumentu, 67  
 katalogu, 124, 208  
 struktury danych, 134  
 strukturyzacja aplikacji, 382  
 strzałka, 92  
 styl treści, 231  
 style, 250  
 animacji, 689  
 Bootstrap, 73  
 elementów, 299  
 komponentu, 398  
 formularza, 526  
 kontekstu, 70  
 łączy, 622  
 tabeli, 72  
 zewnętrzne komponentu, 398  
 synchronizacja przeglądarek WWW, 216  
 system kontroli wersji, 28  
 uwierzytelniania, 180  
 szablon, 38, 59, 224, 387  
 ciągu tekstowego, 95  
 komponentu, 389  
 potomnego, 391  
 tabeli, 524  
 tabeli, 671  
 zewnętrzny, 388

## Ś

ścieżka dostępu, 617  
 środowisko programistyczne, 27

## T

tablice, 99  
 literał, 100  
 metody, 101  
 modyfikacja, 100  
 odczyt, 100  
 wyświetlenie zawartości, 100

testowanie  
 animacji, 679  
 dyrektywy, 723  
 komponentu, 708, 714  
 metody równości, 272  
 operacji  
 asynchronicznej, 721  
 dołączania danych, 712  
 właściwości danych, 718, 719  
 zdarzeń komponentu, 716

testy jednostkowe, 564, 699, 705

token, 469  
 JWT, 180

transformacje  
 animacji, 675  
 elementu, 681, 691  
 zdarzeń, 548

trasa URL, 129, 157

trasy, *Patrz także* routing  
 aktywowanie, 643, 644  
 dezaktywacja, 649  
 konsolidowanie strażników,  
 646  
 potomne, 646  
 zabezpieczanie, 634

treści  
 elementu, 395  
 potomne, 373  
 szablonu, 406

tworzenie  
 animacji, 673  
 danych modelu, 36, 131  
 definicji modułu, 503, 508,  
 513  
 dokumentu HTML, 530

dostawcy lokalnego, 486  
 w dyrektywie, 482  
 w komponencie, 483

dyrektywy  
 atrybutu, 325, 328  
 strukturalnej, 349, 350

egzemplarza klasy, 108

fikcyjnego źródła danych,  
 132

formularza opartego  
 na modelu, 312

formularzy HTML, 74

iteracyjnej dyrektywy  
 strukturalnej, 357

klas modelu, 131

komponentu, 190, 384, 528  
 formularza, 524  
 koszyka, 156  
 routingu, 588  
 tabeli, 523

konfiguracji routingu, 157,  
 586

kontenera, 201

mechanizmu dołączania  
 danych, 46

modelu, 221, 527

modułu, 111  
 Angular, 225  
 core, 522  
 funkcjonalnego, 134, 136,  
 501  
 głównego, 224  
 messages, 527  
 modelu, 502, 520  
 Reactive Extensions, 529

opisowej klasy modelu, 221

pliku typu bootstrap, 225

potoku nieczystego, 419

projektu, 30

repozytorium, 520  
 modelu, 133, 222

sklepu internetowego, 135

struktury katalogów, 124

stylów komponentu  
 formularza, 526

szablonu, 38, 224, 528  
 komponentu tabeli, 524

testu jednostkowego, 705

transformacji dla  
 wbudowanych stanów, 681

trasy potomnej, 626

typu danych produktu, 520

usługi, 527  
 resolvera, 637  
 sieciowej, 125  
 współdzielonej, 522  
 źródła danych, 561

własnego  
 potoku, 415  
 zdarzenia, 337

własnej dyrektywy, 144

właściwości dołączania  
 danych, 333

źródła danych, 222, 520

typ  
 boolean, 94  
 indeksowany, 120  
 number, 95, 117  
 string, 94  
 wyliczeniowy, 401

TypeScript, 81, 115, 208  
 konfiguracja, 519  
 kompilatora, 84, 213  
 opcje kompilatora, 214  
 programowanie Angular, 218

typy, 92  
 podstawowe, 94

## U

uaktualnianie  
 adresów URL, 511  
 dokumentu HTML, 229  
 danych, 253  
 komponentu, 39, 137  
 głównego, 129, 459  
 potomnego, 459  
 modelu aplikacji, 255  
 modułu  
 głównego, 129, 137, 505,  
 510, 514  
 modelu, 167  
 odwołań modułu, 512

układ oparty na siatce, 75, 78

unia typów, 119



uniemożliwianie aktywowania trasy, 643

uruchamianie

- aplikacji, 128, 202, 231, 705
- serwera, 32
- usługi sieciowej, 128

usługa

- resolvera, 637
- sieciowa typu RESTful, 57, 59, 125, 128, 171, 173, 560
- uwierzytelniania, 182

usługi

- dostawca, 463, 465
- przygotowanie, 444, 455
- rejestrwanie, 446, 456
- zastosowania, 458

usuwanie danych, 566

uwierzytelnianie, 179, 180, 183

używanie

- frameworka Angular, 35
- funkcji, 88

## W

wartości w postaci literału, 263

wbudowane

- potoki, 423
- stany aplikacji, 680

wbudowani dostawcy, 468

wczytywanie modułów, 113, 226

- dynamiczne, 654
- JavaScript, 230

wdrożenie aplikacji, 199

weryfikacja danych formularza, 297, 299, 306

widok, 58

własna dyrektywa, 329

własne

- szablony, 381
- zdarzenia, 337

własny potok, 415

właściwości

- danych
  - wejściowych, 335, 719
  - wyjściowych, 393, 718
- dekoratora
  - @Pipe, 416
  - komponentu, 384
- dołączania danych, 333

- dostawcy, 469, 480
  - fabryki, 477
  - wartości, 475
- elementu HTML, 243
- getter i setter, 109
- klasy
  - ComponentFixture, 710
  - DebugElement, 713
  - Response, 563
  - Router, 602
  - Routes, 587
  - SimpleChange, 336
  - Validators, 314
  - ViewContainerRef, 353
- obiektów zdarzeń, 289
- stylu, 252
- zdefiniowane dynamicznie, 287

właściwość

- bootstrap, 498
- declarations, 498, 509
- errors, 302
- exports, 509
- first, 269, 270
- imports, 497, 508
- odd, 268
- providers, 498, 508
- useClass, 472

włączanie dyrektywy

- strukturalnej, 354

wspólny element nadrzędny, 443

wstrzykiwanie zależności, 444, 447, 455

wybór kategorii, 139–141

wyeksportowana funkcjonalność dyrektywy, 346

wyrażenia dołączania danych, 535

wyrażenie, 238, 285

- dołączania danych, 239
- funkcji, 88

wyszukiwanie dostawcy, 489

wyświetlanie

- komunikatów weryfikacji danych, 301, 304
- szczegółów produktu, 138
- zdarzenia zmiany stanu, 584
- zdarzeń, 542
- tabeli, 45

wyzwalacz, 676

- animacji, 694

wzorzec

- CQRS, 57
- MVC, 54
- projektowy
  - model-widok-kontroler, 23
- REST, 560

## X

XSS, cross-site scripting, 570

## Z

zabezpieczanie tras, 160, 634

zależności

- modułów RxJS, 227
- niestandardowych modułów aplikacji, 228

zapisywanie danych, 566

zapytania

- do widoków potomnych, 407
- do wielu elementów, 376

zarządzanie zamówieniem, 198

zdarzenia, 48, 281, 285

- filtrowanie, 547, 549
- klawiszy, 292
- komponentu, 716
- modelu DOM, 287
- nawigacyjne, 604
- pobieranie, 553
- pomijanie, 553
- transformowanie, 548
- własne, 337
- zmiany stanu, 584

zmiana

- adresu URL, 593
- stanu, 584
- wielkości elementu, 71, 73
- właściwości danych, 335

zmiennie, 92

- domknięcia, 93
- odwołania w szablonie, 290
- szablonu, 265

znacznik, 65

- <button>, 70
- <base>, 157
- <div>, 260

znacznik

<html>, 67  
<input>, 296, 311, 319  
<link>, 128, 231  
<router-outlet>, 662  
<script>, 85, 215  
<template>, 355

znaki wieloznaczne, 615

Ź

źródło danych, 166, 172, 185,  
222, 520  
konfiguracja, 564  
statyczne, 561  
RESTful, 181, 564

Ż

żądania  
HTTP, 555, 571  
JSONP, 571

# PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ 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>

# Funkcjonalne narzędzie dla profesjonalisty

Framework Angular 2 jest dziś uważany za jeden z najwydajniejszych sposobów rozwijania średnich i dużych aplikacji internetowych. Działają one bez problemu niezależnie od platformy. Separacja kodu interfejsu od logiki aplikacji, prosta modularyzacja aplikacji, bezproblemowe przetwarzanie danych asynchronicznych, bogaty zestaw narzędzi i ogromne ułatwienia podczas projektowania interfejsu użytkownika — to tylko kilka z wielu zalet Angulara. Nawet jednak tak świetny framework wymaga od programisty wiedzy i umiejętności prawidłowego budowania kodu.

Książka rozpoczyna się od przedstawienia wzorca MVC i jego zalet. Ta wiedza następnie przydaje się do budowy własnego projektu za pomocą Angulara. Najpierw będzie to prosta, praktyczna aplikacja, a później zostaną zaprezentowane bardziej zaawansowane funkcje. Każdy temat jasno i spójnie wyjaśniono, ze szczegółami koniecznymi do efektywnej pracy. Przedstawiono sposoby wykorzystywania różnych pożytecznych narzędzi, w tym frameworka Bootstrap, biblioteki Reactive Extensions czy frameworka Jasmine. Nie zabrakło objaśnień najczęściej występujących problemów oraz sposobów ich rozwiązywania.

W tej książce między innymi:

- solidne podstawy koncepcji MVC
- obiektowy model dokumentu (DOM)
- usługi i dostawcy usług
- routing i animacja w Angularze
- testowanie aplikacji

**Adam Freeman** — jest wyjątkowo doświadczonym programistą i architektem. Doskonale rozumie wyzwania, jakie niesie ze sobą zapewnienie bezpieczeństwa dużym systemom informatycznym. Pracował w wielu firmach między innymi Netscape i Sun Microsystems, a ostatnio zajmował stanowisko dyrektora naczelnego w międzynarodowym banku. Obecnie jest na emeryturze. Swoją czas dzieli między dwie pasje: pisanie i bieganie.

 <b>helion.pl</b>	<i>Sprawdź nasze szkolenia!</i> SZKOLENIA  AKADEMIA IT & BUSINESS WWW.SZKOLENIA.HELION.PL	<b>KOD KORZYŚCI</b> Sięgnij po więcej! ▶  ISBN 978-83-283-4231-6  9 788328 342316
INFORMATYKA W NAJLEPSZYM WYDANIU		Cena: 119,00 zł

Apress