

Yakov **Fain**, Anton **Moiseev**



Angular

Programowanie z użyciem języka **TypeScript**

Wydanie II

Helion 

Tytuł oryginału: Angular Development with Typescript, Second Edition

Tłumaczenie: Lech Lachowski

ISBN: 978-83-283-5666-5

Original edition copyright © 2019 by Manning Publications Co.
All rights reserved.

Polish edition copyright © 2019 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 jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

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

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

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

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/anpro2.zip>

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/anpro2>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

Spis treści

<i>Przedmowa</i>	11
<i>Podziękowania</i>	13
<i>O książce</i>	15
<i>O autorach</i>	19
<i>Rozdział 1. Wprowadzenie do frameworku Angular</i>	21
1.1. Dlaczego do tworzenia aplikacji internetowych wybierać Angular?	22
1.2. Dlaczego tworzyć aplikacje w TypeScriptie, a nie w JavaScriptcie	23
1.3. Przegląd frameworku Angular	24
1.4. Wprowadzenie do CLI Angulara	28
1.4.1. Generowanie nowego projektu Angulara	29
1.4.2. Kompilacje programistyczne i produkcyjne	31
1.5. Porównanie kompilacji JIT i AOT	33
1.5.1. Tworzenie paczek za pomocą opcji -prod	34
1.5.2. Generowanie paczek na dysku	35
1.6. Wprowadzenie do przykładowej aplikacji ngAuction	35
Podsumowanie	38
<i>Rozdział 2. Główne artefakty aplikacji napisanych w Angularze</i>	39
2.1. Komponenty	39
2.2. Usługi	42
2.3. Dyrektywy	43
2.4. Potoki	45
2.5. Moduły	46
2.5.1. Moduły funkcyjne	46
2.6. Wiązanie danych	50
2.6.1. Wiązanie właściwości i zdarzeń	50
2.6.2. Jedno- i dwukierunkowe wiązanie danych w działaniu	51
2.7. Część praktyczna: rozpoczynamy tworzenie aplikacji ngAuction	55
2.7.1. Wstępna konfiguracja projektu dla aplikacji ngAuction	55
2.7.2. Generowanie komponentów dla aplikacji ngAuction	57
2.7.3. Komponent aplikacji	57
2.7.4. Komponent paska nawigacyjnego	59
2.7.5. Komponent wyszukiwania	60
2.7.6. Komponent stopki	61
2.7.7. Komponent karuzeli	61
2.7.8. Komponent strony głównej	64
Podsumowanie	65

Rozdział 3. Podstawy routera Angulara	67
3.1. Podstawy routingu	68
3.2. Strategie lokalizacji	69
3.2.1. Nawigacja oparta na znaku kratki	70
3.2.2. Nawigacja oparta na interfejsie <i>History API</i>	70
3.3. Bloki konstrukcyjne nawigacji po stronie klienta	71
3.4. Nawigacja do tras za pomocą metody <code>navigate()</code>	76
3.5. Przekazywanie danych do tras	77
3.5.1. Wyodrębnianie parametrów z <i>ActivatedRoute</i>	78
3.5.2. Przekazywanie do trasy parametrów zapytania	80
3.6. Trasy podrzędne	81
3.7. Część praktyczna: dodanie nawigacji do aplikacji aukcji internetowych	86
3.7.1. Usługa <i>ProductService</i>	87
3.7.2. Komponent <i>ProductItemComponent</i>	88
3.7.3. Komponent <i>HomeComponent</i>	89
3.7.4. Komponent <i>StarsComponent</i>	91
3.7.5. Komponent <i>ProductDetailComponent</i>	94
Podsumowanie	96
Rozdział 4. Zaawansowana konfiguracja routera	97
4.1. Strzeżenie tras	98
4.1.1. Implementowanie strażnika <i>CanActivate</i>	99
4.1.2. Implementowanie strażnika <i>CanDeactivate</i>	101
4.1.3. Implementowanie strażnika <i>Resolve</i>	103
4.2. Tworzenie aplikacji SPA z wieloma gniazdami routera	107
4.2.1. Moduły <i>leniwe ładowane</i>	109
4.2.2. Ładowarki wstępne	113
Podsumowanie	114
Rozdział 5. Wstrzykiwanie zależności we frameworku Angular	115
5.1. Wzorzec Wstrzykiwanie Zależności	116
5.2. Korzyści płynące ze wstrzykiwania zależności w aplikacjach napisanych w Angularze	117
5.2.1. Luźne powiązania i wielokrotne wykorzystywanie	117
5.2.2. Testowalność	119
5.3. Wstrzykiwacze i dostawcy	119
5.4. Prosta aplikacja ze wstrzykiwaniem zależności frameworku Angular	122
5.4.1. Wstrzyknięcie usługi produktowej	122
5.4.2. Wstrzyknięcie usługi <i>HttpClient</i>	125
5.5. Ułatwione przełączanie wstrzykiwaczy	127
5.6. Deklarowanie dostawców za pomocą właściwości <code>useFactory</code> i <code>useValue</code>	131
5.6.1. Korzystanie z klasy <i>InjectionToken</i>	133
5.6.2. Wstrzykiwanie zależności w aplikacji zmodularyzowanej	134
5.7. Dostawcy w modułach ładowanych leniwie	134
5.8. Dostawcy w modułach ładowanych gorliwie	136

5.9.	Część praktyczna: użycie komponentów biblioteki Angular Material w aplikacji ngAuction	138
5.9.1.	Krótki przegląd biblioteki Angular Material	139
5.9.2.	Dodanie biblioteki AM do projektu	141
5.9.3.	Dodanie modułu funkcyjnego z komponentami AM	142
5.9.4.	Modyfikacja wyglądu komponentu NavBarComponent	143
5.9.5.	Modyfikacja interfejsu użytkownika komponentu SearchComponent	146
5.9.6.	Zastąpienie karuzeli obrazem	148
5.9.7.	Kolejne poprawki odstępów	148
5.9.8.	Użycie mat-card w komponencie ProductItemComponent	148
5.9.9.	Dodanie stylów do komponentu HomeComponent	149
	Podsumowanie	149
	Rozdział 6. Programowanie reaktywne we frameworku Angular	151
6.1.	Obsługa zdarzeń bez strumieni obserwowalnych	152
6.2.	Przekształcanie zdarzeń DOM w strumienie obserwowalne	154
6.3.	Obsługa zdarzeń obserwowalnych za pomocą Forms API	156
6.4.	Odrzucanie wyników niechcianych żądań HTTP za pomocą operatora switchMap	158
6.5.	Korzystanie z potoku AsyncPipe	161
6.6.	Strumienie obserwowalne i router	165
	Podsumowanie	168
	Rozdział 7. Tworzenie układu stron za pomocą biblioteki Flex Layout	169
7.1.	Biblioteka Flex Layout i usługa ObservableMedia	170
7.1.1.	Korzystanie z dyrektyw biblioteki Flex Layout	171
7.1.2.	Usługa ObservableMedia	175
7.2.	Część praktyczna: przepisanie kodu aplikacji ngAuction	177
7.2.1.	Po co przepisywać aplikację ngAuction od zera?	177
7.2.2.	Generowanie nowej aplikacji ngAuction	180
7.2.3.	Tworzenie niestandardowego motywu Angular Material za pomocą Sass	180
7.2.4.	Dodawanie paska narzędzi do komponentu najwyższego poziomu	183
7.2.5.	Tworzenie usługi produktowej	186
7.2.6.	Tworzenie modułu strony głównej	188
7.2.7.	Konfigurowanie tras	193
7.2.8.	Uruchomienie aplikacji ngAuction	193
	Podsumowanie	194
	Rozdział 8. Implementowanie komunikacji komponentów	195
8.1.	Komunikacja między komponentami	196
8.2.	Właściwości wejściowe i wyjściowe	196
8.2.1.	Właściwości wejściowe	197
8.2.2.	Właściwości wyjściowe i zdarzenia niestandardowe	199
8.3.	Implementowanie wzorca projektowego Mediator	203
8.3.1.	Użycie jako mediatora wspólnego komponentu nadrzędnego	203
8.3.2.	Użycie jako mediatora wstrzykiwalnej usługi	208
8.4.	Udostępnianie interfejsu API komponentu potomnego	213

8.5.	Rzutowanie szablonów w czasie działania aplikacji za pomocą dyrektywy ngContent	216
8.5.1.	Tryby hermetyzacji widoków	218
8.5.2.	Rzutowanie na wiele obszarów	220
	Podsumowanie	222
Rozdział 9. Mechanizm wykrywania zmian i cykl życia komponentu		223
9.1.	Ogólny przegląd działania mechanizmu wykrywania zmian	224
9.1.1.	Strategie wykrywania zmian	225
9.1.2.	Profilowanie wykrywania zmian	227
9.2.	Cykl życia komponentów	227
9.2.1.	Przechwytywanie zmian w zaczepie ngOnChanges	230
9.2.2.	Przechwytywanie zmian w zaczepie ngDoCheck	233
9.3.	Część praktyczna: dodanie widoku produktu do aplikacji ngAuction	236
9.3.1.	Tworzenie komponentów i modułu produktu	236
9.3.2.	Implementowanie komponentu produktu	238
9.3.3.	Implementowanie komponentu szczegółów produktu	241
9.3.4.	Implementowanie komponentu sugerowanych produktów	242
	Podsumowanie	244
Rozdział 10. Wprowadzenie do interfejsu API formularzy		245
10.1.	Dwa interfejsy API formularzy	246
10.2.	Formularze oparte na szablonach	246
10.2.1.	Dyrektywy formularzy	246
10.2.2.	Zastosowanie opartego na szablonach API do formularzy HTML	248
10.3.	Formularze reaktywne	251
10.3.1.	Model formularza	252
10.3.2.	Dyrektywy formularzy reaktywnych	254
10.3.3.	Zastosowanie reaktywnego API do formularzy HTML	256
10.3.4.	Dynamiczne dodawanie kontroltek do formularza	258
10.4.	Podsumowanie dyrektyw Forms API	259
10.5.	Aktualizacja na podstawie danych	261
10.6.	Korzystanie z klasy FormBuilder	263
	Podsumowanie	263
Rozdział 11. Walidacja formularzy		265
11.1.	Korzystanie z wbudowanych walidatorów	266
11.2.	Kontrolowanie momentu rozpoczęcia walidacji	269
11.3.	Walidatory niestandardowe w formularzach reaktywnych	270
11.4.	Walidacja grupy kontroltek	273
11.5.	Sprawdzanie statusu i prawidłowości kontrolki formularza	276
11.5.1.	Dotknięte i niedotknięte kontrolki formularza	276
11.5.2.	Pola czyste i brudne	277
11.5.3.	Pola oczekujące	277
11.6.	Dynamiczne zmienianie walidatorów w formularzach reaktywnych	277
11.7.	Walidatory asynchroniczne	279
11.8.	Niestandardowe walidatory w formularzach opartych na szablonach	281

11.9.	Część praktyczna: dodanie formularza wyszukiwania do aplikacji ngAuction	282
11.9.1.	Komponent formularza wyszukiwania	283
11.9.2.	Komponent wyników wyszukiwania	287
11.9.3.	Refaktoryzacja pozostałego kodu	288
	Podsumowanie	291
Rozdział 12. Interakcja z serwerami przy użyciu protokołu HTTP		293
12.1.	Przegląd usługi HttpClient	294
12.2.	Odczytywanie pliku JSON za pomocą usługi HttpClient	295
12.3.	Tworzenie serwera WWW za pomocą frameworków Node i Express oraz języka TypeScript	298
12.3.1.	Tworzenie prostego serwera WWW	299
12.3.2.	Serwowanie danych w formacie JSON	301
12.4.	Łączenie frameworku Angular i serwera Node	303
12.4.1.	Zasoby statyczne na serwerze	303
12.4.2.	Konsumowanie danych JSON w aplikacjach Angulara	305
12.4.3.	Konfigurowanie serwera proxy klienta	308
12.4.4.	Subskrybowanie strumieni obserwowalnych za pomocą potoku async	309
12.4.5.	Wstrzykiwanie HttpClient do usługi	310
12.4.6.	Wdrażanie aplikacji Angulara na serwerze za pomocą skryptów npm	312
12.5.	Przesyłanie danych na serwer	315
12.5.1.	Tworzenie serwera do obsługi żądań post	315
12.5.2.	Tworzenie klienta do wysyłania żądań post	316
12.6.	Przechwytywanie HTTP	319
12.7.	Zdarzenia postępu	323
	Podsumowanie	325
Rozdział 13. Interakcja z serwerami przy użyciu protokołu WebSocket		327
13.1.	Porównanie protokołów HTTP i WebSocket	328
13.2.	Wysyłanie danych z serwera Node do zwykłego klienta	330
13.3.	Użycie gniazd WebSocket w klientach Angulara	333
13.3.1.	Opakowywanie strumienia obserwowalnego w usługę	334
13.3.2.	Komunikacja usługi Angulara z serwerem WebSocket	336
13.4.	Część praktyczna: serwer Node z obsługą protokołu WebSocket	341
13.4.1.	Uruchamianie aplikacji ngAuction w trybie programistycznym	344
13.4.2.	Przegląd kodu serwera aplikacji ngAuction	344
13.4.3.	Zmiany w kodzie klienta aplikacji ngAuction	351
	Podsumowanie	357
Rozdział 14. Testowanie aplikacji Angulara		359
14.1.	Testy jednostkowe	360
14.1.1.	Wprowadzenie do frameworku Jasmine	361
14.1.2.	Pisanie skryptu testowego dla klasy	363
14.2.	Uruchamianie skryptów Jasmine za pomocą testera Karma	365
14.2.1.	Plik konfiguracyjny Karma	369
14.2.2.	Testowanie w wielu przeglądarkach	370

14.3.	Korzystanie z biblioteki testowej frameworku Angular	371
14.3.1.	Testowanie komponentów	372
14.3.2.	Testowanie usług	376
14.3.3.	Testowanie komponentów wykorzystujących routing	379
14.4.	Przeprowadzanie testów end-to-end przy użyciu biblioteki Protractor	382
14.4.1.	Podstawy Protractora	383
14.4.2.	Testy generowane przez CLI Angulara	386
14.4.3.	Testowanie strony logowania	387
14.5.	Część praktyczna: dodawanie testów e2e do aplikacji ngAuction	392
14.5.1.	Przeprowadzanie testów e2e przepływu pracy dla wyszukiwania produktów	392
	Podsumowanie	398
Rozdział 15. Utrzymywanie stanu aplikacji za pomocą biblioteki ngrx		399
15.1.	Od sklepu ogólnospżywczego do architektury Redux	400
15.1.1.	Co to jest Redux?	400
15.1.2.	Dlaczego ważne jest przechowywanie stanu aplikacji w jednym miejscu?	402
15.2.	Wprowadzenie do biblioteki ngrx	403
15.2.1.	Poznajemy magazyn, akcje i reduktory	404
15.2.2.	Poznajemy efekty i selektory	410
15.2.3.	Refaktoryzacja aplikacji mediatora za pomocą ngrx	412
15.2.4.	Monitorowanie stanu za pomocą rozszerzenia DevTools magazynu ngrx	420
15.2.5.	Monitorowanie stanu routera	424
15.3.	Używać ngrx albo nie używać	427
15.3.1.	Porównanie ngrx z usługami Angulara	428
15.3.2.	Problemy z mutacją stanu	429
15.3.3.	Kod ngrx jest trudniejszy do odczytania	429
15.3.4.	Krzywa uczenia się	430
15.3.5.	Wnioski	430
15.4.	Część praktyczna: stosowanie ngrx w aplikacji ngAuction	431
15.4.1.	Dodawanie obsługi stanu routera do modułu aplikacji	433
15.4.2.	Zarządzanie stanem w module głównym	434
15.4.3.	Testy jednostkowe reduktorów ngrx	442
	Podsumowanie	443
	Angular 6, 7 i nie tylko	444
Dodatek A. Przegląd specyfikacji ECMAScript		447
Dodatek B. Podstawy TypeScriptu		481
Dodatek C. Korzystanie z menedżera pakietów npm		509
Dodatek D. Podstawy biblioteki RxJS		517
Skorowidz		539

7

Tworzenie układu stron za pomocą biblioteki Flex Layout

W tym rozdziale:

- implementacja projektowania responsywnego przy użyciu biblioteki Flex Layout;
- korzystanie z usługi ObservableMedia;
- zmiana układu na podstawie rozmiaru rzutni.

Jeśli chodzi o tworzenie aplikacji internetowej, musisz zdecydować, czy będziesz mieć oddzielne wersje aplikacji na komputery stacjonarne i urządzenia mobilne, czy też użyjesz tego samego kodu na wszystkich urządzeniach. Pierwsze podejście pozwala korzystać z natywnych kontrolek na urządzeniach mobilnych, więc interfejs użytkownika wygląda bardziej naturalnie, ale trzeba utrzymywać oddzielne wersje kodu dla każdej wersji aplikacji. Drugie podejście polega na użyciu jednej bazy kodu i implementacji techniki **responsywnego projektowanie stron internetowych** (ang. *responsive web design* — RWD), aby układ interfejsu użytkownika dostosowywał się do rozmiaru ekranu urządzenia.

UWAGA Pojęcie responsywnego projektowania stron zostało wymyślone przez Ethana Marcotte'a i użyte w artykule „Responsive Web Design”, dostępnym pod adresem <https://alistapart.com/article/responsive-webdesign>.

Istnieje jeszcze trzecie podejście: oprócz aplikacji internetowej, która działa na komputerach stacjonarnych, można opracować aplikację **hybrydową**, która jest aplikacją internetową działającą w przeglądarkach urządzeń mobilnych, ale może także wywoływać natywny interfejs API urządzenia mobilnego.

Z tego rozdziału dowiesz się, jak sprawić, by Twoja aplikacja dobrze wyglądała i była funkcjonalna na dużych i małych ekranach dzięki zastosowaniu podejścia RWD. W rozdziale 6. omówione zostały strumienie obserwowalne, które mogą wysyłać powiadomienia, gdy w aplikacji mają miejsce ważne zdarzenia. Zobaczmy, czy można użyć strumieni obserwowalnych do powiadamiania o zmianach rozmiaru ekranu użytkownika i zmieniać układ interfejsu użytkownika na podstawie szerokości rzutni urządzenia użytkownika. Użytkownicy smartfonów i użytkownicy urządzeń z dużymi monitorami powinni zobaczyć różne układy tej samej aplikacji.

Pokażemy Ci, jak korzystać z biblioteki Flex Layout do implementacji RWD i jak używać usługi ObservableMedia, aby oszczędzić sobie pisania dużej ilości kodu CSS.

Na koniec zaczniemy przepisywać aplikację ngAuction, ilustrując wiele technik, których się nauczyłeś, a głównym celem będzie usunięcie z aplikacji biblioteki Bootstrap i stosowanie tylko bibliotek Angular Material i Flex Layout.

7.1. Biblioteka Flex Layout i usługa ObservableMedia

Wyobraź sobie, że przygotowałeś interfejs użytkownika dla swojej aplikacji i wygląda on świetnie na monitorze o rozdzielczości poziomej 1200 pikseli lub większej. A jeśli użytkownik otworzy tę aplikację na smartfonie o szerokości ekranu 640 pikseli? W zależności od urządzenia wyrenderowana zostanie tylko część interfejsu aplikacji i dodany zostanie poziomy pasek u dołu lub interfejs użytkownika zostanie przeskalowany w dół, aby zmieścił się na małym ekranie, co utrudni korzystanie z aplikacji. Można też rozważyć inny scenariusz: użytkownicy z dużymi monitorami, którzy zmniejszają szerokość okna przeglądarki, ponieważ muszą zmieścić na ekranie monitora jeszcze jedną aplikację.

Aby zaimplementować RWD, można użyć zapytań o media (ang. *media queries*) CSS, reprezentowanych przez regułę @media. W kodzie CSS aplikacji można dołączyć zestaw zapytań o media oferujących różne układy dla różnych szerokości ekranu. Przeglądarka stale sprawdza bieżącą szerokość okna i gdy tylko ta szerokość przekroczy pewną **wartość progową** (ang. *breakpoint*) ustawioną w regułach @media (na przykład szerokość staje się mniejsza niż 640 pikseli), zastosowany zostaje nowy układ strony.

Innym sposobem wdrożenia elastycznych układów jest użycie modułu Flexbox CSS z zapytaniami o media (zobacz <http://mng.bz/6B42>). Interfejs użytkownika Twojej aplikacji ma postać zestawu elastycznych pudełek i jeśli przeglądarka nie może dopasować zawartości Flexbox poziomo (lub pionowo), zawartość jest renderowana w następnym wierszu (lub kolumnie).

Można także zaimplementować RWD za pomocą systemu CSS Grid, czyli siatki CSS (zobacz <http://mng.bz/k29F>). Zarówno Flexbox, jak i CSS Grid wymagają dobrego zrozumienia reguł @media.

Biblioteka Flex Layout Angulara (zobacz <https://github.com/angular/flex-layout>) to silnik układu interfejsu użytkownika do implementacji RWD bez konieczności pisania zapytań o media w plikach CSS. Biblioteka ta zawiera zestaw prostych dyrektyw Angulara, które wewnętrznie aplikują reguły układu flexbox i oferują usługę ObservableMedia, powiadamiającą aplikację o aktualnej szerokości rzutni na urządzeniu użytkownika.

Biblioteka Flex Layout frameworku Angular ma przewagę nad standardowym interfejsem API CSS w następującym zakresie:

- tworzy CSS kompatybilny z różnymi przeglądarkami;
- zapewnia przyjazny dla Angulara interfejs do obsługi zapytań o media przy użyciu dyrektyw i strumieni obserwowalnych.

UWAGA W tym podrozdziale zapewnimy minimalny opis biblioteki Flex Layout, abyś mógł szybko zacząć z niej korzystać. Więcej szczegółowych informacji i przykładów znajdziesz w dokumentacji Flex Layout na stronie <https://github.com/angular/flex-layout/wiki>.

Biblioteka Flex Layout udostępnia dwa interfejsy API: statyczny i responsywny. Styczny interfejs API pozwala używać dyrektyw do określania atrybutów układu dla kontenerów i ich elementów potomnych. Responsywny interfejs API ulepsza dyrektywy statycznego API, umożliwiając implementację RWD, żeby układy aplikacji zmieniały się dla różnych rozmiarów ekranu.

7.1.1. Korzystanie z dyrektyw biblioteki Flex Layout

W bibliotece Flex Layout istnieją dwa typy dyrektyw: jeden dla kontenerów, a drugi dla ich elementów potomnych. Dyrektywy kontenera są używane do wyrównania jego elementów potomnych. Dyrektywy potomne są stosowane do elementów potomnych kontenera zarządzanego przez Flex Layout. Dzięki dyrektywom potomnym można określić kolejność każdego elementu potomnego, zajmowaną przez niego ilość miejsca i niektóre inne właściwości, tak jak pokazano w tabeli 7.1.

Tabela 7.1. Najczęściej używane dyrektywy biblioteki Flex Layout

Dyrektywa	Opis
<i>Dyrektywy kontenera</i>	
fxLayout	Instruuje element, że należy użyć modułu Flexbox CSS do układania elementów potomnych.
fxLayoutAlign	Wyrównuje elementy potomne w określony sposób (do lewej, do dołu, rozkłada równomiernie itd.). Dozwolone wartości zależą od wartości fxLayout dołączonej do elementu tego samego kontenera — więcej informacji znajdziesz w dokumentacji biblioteki Flex Layout frameworku Angular.
fxLayoutGap	Kontroluje odstęp między elementami potomnymi.
<i>Dyrektywy elementów potomnych</i>	
fxFlex	Kontroluje, ile miejsca element potomny zajmuje w kontenerze nadrzędnym.
fxFlexAlign	Umożliwia selektywną zmianę wyrównania elementów potomnych w kontenerze nadrzędnym na podstawie dyrektywy fxLayoutAlign.
fxFlexOrder	Umożliwia zmianę kolejności elementów potomnych w kontenerze nadrzędnym. Można jej użyć na przykład do przeniesienia ważnego komponentu w widoczny obszar przy przełączaniu z ekranu komputera stacjonarnego na ekran urządzenia mobilnego.

UWAGA Dyrektywy elementów potomnych muszą być umieszczane wewnątrz elementu HTML z dołączoną dyrektywą kontenera.

Spójrzmy, jak skorzystać z biblioteki Flex Layout, aby wyrównać dwa elementy `<div>` obok siebie w rzędzie. Najpierw musisz dodać do projektu bibliotekę Flex Layout i jej zależność równorzędną `@angular/cdk`:

```
npm i @angular/flex-layout @angular/cdk
```

Następnym krokiem jest dodanie modułu `FlexLayoutModule` do dekoratora głównego `@NgModule()`, tak jak przedstawiono w listingu 7.1.

Listing 7.1. Dodawanie modułu `FlexLayoutModule`

```
import { AppComponent } from './app.component';

@NgModule({
  imports: [
    FlexLayoutModule
    // ...
  ])
export class AppModule {}
```

Kod w listingu 7.2 tworzy komponent, który wyświetla elementy `<div>` obok siebie od lewej do prawej.

Listing 7.2. Plik `flex-layout/app.component.ts`

```
@Component({
  selector: 'app-root',
  styles: [
    .parent { height: 100px; }
    .left { background-color: cyan; }
    .right { background-color: yellow; }
  ],
  template: `
    <div class="parent" fxLayout="row">
      <div fxFlex class="left">Lewy</div>
      <div fxFlex class="right">Prawy</div>
    </div>
  `
})
export class AppComponent {}
```

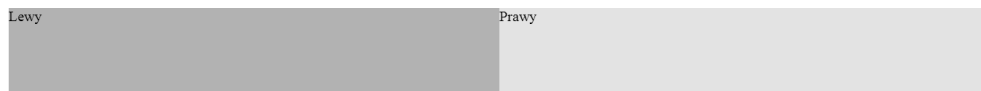
Dyrektywa `fxLayout` przekształca `<div>` w kontener flex-layout, w którym elementy potomne są rozmieszczane poziomo (w rzędzie).

Dyrektywa `fxFlex` instruuje poszczególne elementy potomne, żeby podzieliły po równo między siebie przestrzeń wewnątrz kontenera nadrzędnego.

Aby zobaczyć tę aplikację w działaniu, uruchom następujące polecenie:

```
ng serve --app flex-layout -o
```

Rysunek 7.1 pokazuje, jak przeglądarka wyrenderuje elementy potomne. Każdy element potomny zajmuje 50% dostępnej szerokości kontenera.



Rysunek 7.1. Dwa elementy wyrównane w rzędzie

Aby prawy div zajmował więcej miejsca niż lewy, można przypisać wymagane wartości przestrzeni do dyrektyw `flexFlex` podrzędnych. Poniższy szablon używa dyrektywy poziomego podrzędnego `flexFlex`, aby przeznaczyć 30% dostępnej szerokości dla lewego elementu potomnego, a 70% dla prawego:

```
<div flexLayout="row" class="parent">
  <div flexFlex="30%" class="left">Lewy</div>
  <div flexFlex="70%" class="right">Prawy</div>
</div>
```

Teraz interfejs użytkownika jest renderowany tak, jak pokazano na rysunku 7.2.



Rysunek 7.2. Prawy element zajmuje więcej miejsca niż lewy

Aby ułożyć elementy potomne kontenera w pionie, zmień orientację układu kontenera z wierszy na kolumny za pomocą `flexLayout="column"`:

```
<div flexLayout="column" class="parent">
  <div flexFlex="30%" class="left">Lewy</div>
  <div flexFlex="70%" class="right">Prawy</div>
</div>
```

Na rysunku 7.3 pokazano, jak elementy potomne są renderowane w pionie.



Rysunek 7.3. Układ kolumnowy elementów kontenera

Żałujemy, że na dużym ekranie masz wystarczająco miejsca, aby wyrenderować komponenty lewy i prawy poziomo obok siebie, ale jeśli użytkownik otworzy tę samą aplikację na mniejszym ekranie, należałoby automatycznie zmienić układ na pionowy, żeby prawy komponent był wyświetlany pod lewym.

Każda dyrektywa w bibliotece Flex Layout może opcjonalnie mieć **przyrostek** (alias dla reguły zapytań o media), który określa, jaki rozmiar ekranu trzeba zastosować. Dyrektywa `flexLayout.sm` ma na przykład przyrostek `.sm`, co oznacza, że powinna być stosowana tylko wtedy, gdy szerokość ekranu jest mała. Te aliasy odpowiadają wartościom progowym dla szerokości ekranu, zdefiniowanym w wytycznych Material Design (zobacz <http://mng.bz/RmLN>):

- `xs` — bardzo mały ekran (mniej niż 599 pikseli);
- `sm` — mały ekran (560 – 959 pikseli);
- `md` — średni ekran (960 – 1279 pikseli);
- `lg` — duży ekran (1280 – 1919 pikseli);
- `xl` — bardzo duży ekran (1920 – 5000 pikseli).

Kod w listingu 7.3 modyfikuje aplikację w taki sposób, że jej kontener nadrzędny układa elementy potomne w poziomie na średnich i dużych ekranach oraz pionowo na małych urządzeniach.

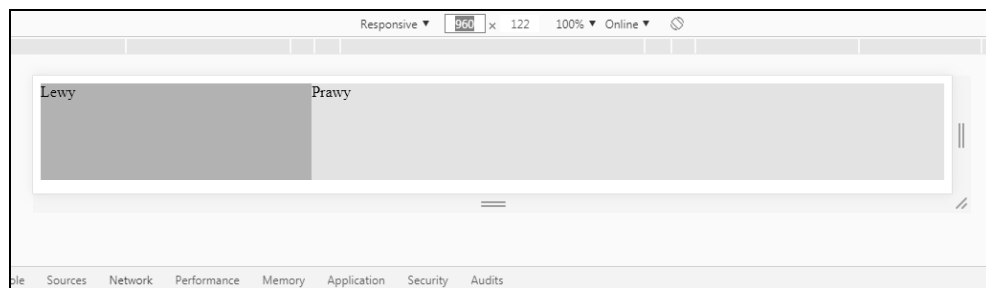
Listing 7.3. Dodanie przyrostka .sm

```
<div class="parent"
  fxLayout="row"
  fxLayout.sm="column" >
  <div fxFlex="30%" class="left">Lewy</div>
  <div fxFlex="70%" class="right">Prawy</div>
</div>
```

Domyślnie elementy potomne są wyrównywane w rzędzie.

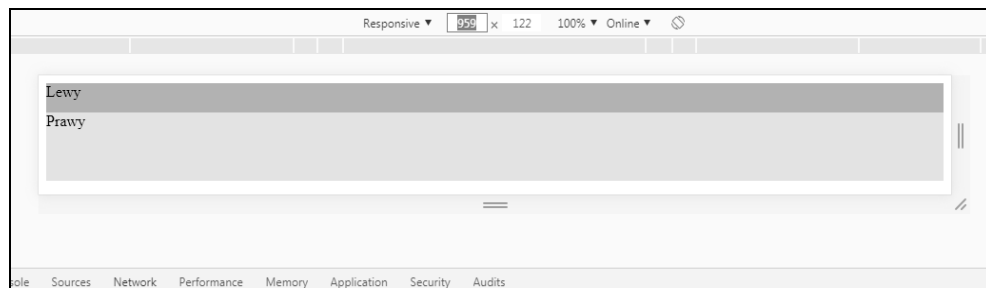
Na ekranach o małych rozmiarach elementy potomne są układane pionowo.

Aby zilustrować, jak zmieni to układ, użyjemy narzędzi dla programistów przeglądarki Chrome, w których po lewej stronie paska narzędzi znajduje się ikona *Toggle device toolbar*, umożliwiająca przełączanie urządzeń. Dla komputerów stacjonarnych mały rozmiar oznacza, że szerokość okna wynosi od 600 do 959 pikseli. Na rysunku 7.4 pokazano, jak renderowany jest interfejs użytkownika, jeśli szerokość wynosi 960 (nadal rozmiar średni).



Rysunek 7.4. Renderowanie na średnim urządzeniu o szerokości ekranu 960 pikseli

Przekroczmy wartość progową i zmieńmy szerokość na 959, aby emulować małe urządzenie. Na rysunku 7.5 pokazano, że układ zmienił się z poziomego na pionowy.



Rysunek 7.5. Renderowanie na małym urządzeniu o szerokości ekranu 959 pikseli

Zmiana szerokości na mniejszą niż 600 spowoduje przełączenie z powrotem do układu poziomego, ponieważ nie określiliśmy, że dla bardzo małych urządzeń (przyrostek `.xs`) układ powinien pozostać w pionie. Możemy dodać układ pionowy dla bardzo małych (`xs`) urządzeń:

```
<div fxLayout="row" class="parent"
      fxLayout.sm="column"
      fxLayout.xs="column">
```

Można także do aliasów zapytań o media zastosować przyrostki „mniejszy niż” (`lt-`) i „większy niż” (`>-`). Jeśli użyjemy na przykład aliasu `lt-md`, odpowiedni układ zostanie zastosowany do małych i bardzo małych ekranów. W naszej aplikacji możemy określić, że na każdym ekranie z szerokością mniejszą niż średnia należy zastosować układ kolumnowy:

```
<div fxLayout="row" class="parent"
      fxLayout.lt-md="column">
```

Używając wartości progowych, możemy statycznie zdefiniować w szablonie komponentu, w jaki sposób powinien być ułożony interfejs użytkownika. A jeśli będziemy chcieli nie tylko zmienić układ wewnątrz kontenera, ale także warunkowo pokazywać lub ukrywać pewne elementy potomne w zależności od rozmiaru ekranu? Aby dynamicznie decydować, co i jak powinna renderować przeglądarka w zależności od rozmiaru ekranu, użyjemy usługi `ObservableMedia`, która pochodzi z biblioteki `Flex Layout`.

7.1.2. Usługa ObservableMedia

Usługa `ObservableMedia` umożliwia subskrybowanie zmian rozmiarów ekranu i programową zmianę wyglądu i stylu aplikacji. Na dużych ekranach możemy na przykład zdecydować o wyświetleniu dodatkowych informacji. Aby uniknąć renderowania niepotrzebnych komponentów na małych ekranach, możemy subskrybować zdarzenia emitowane przez `ObservableMedia`, a jeśli rozmiar ekranu się zwiększy, możemy wyrenderować więcej komponentów.

Aby zaimplementować tę funkcjonalność, zaimportujmy usługę `ObservableMedia` i zasubskrybujmy jej obiekt `Observable`. Kod w listingu 7.4 pokazuje, jak zasubskrybować powiadomienia o zmianach wielkości ekranu za pomocą potoku `async` i wypisać bieżący rozmiar w konsoli.

Listing 7.4. Plik `observable-media/app.component.ts`

```
import {Component} from '@angular/core';
import {ObservableMedia} from '@angular/flex-layout';
import {Observable} from 'rxjs';
import {map} from 'rxjs/operators';
```

```
@Component({
  selector: 'app-root',
  template: `<h3>Zaobserwuj w konsoli komunikat o aktywowaniu wartości progowej.</h3>
    <span *ngIf="showExtras$ | async">`
```

← Pokazuje lub ukrywa tekst na podstawie wartości `showExtras$`; potok `async` subskrybuje `showExtras$`.

```

    Pokazuje dodatkowe informacje na średnich ekranach</span>
  }
}
export class AppComponent {
  showExtras$: Observable<boolean>;

  constructor(private media: ObservableMedia) {
    this.showExtras$ = this.media.asObservable()
      .pipe(map(mediaChange => {
        console.log(mediaChange.mqAlias);
        return mediaChange.mqAlias === 'md'? true: false;
      }));
  }
}

```

Wstrzykuje usługę ObservableMedia.

Subskrybuje obiekt Observable, który emituje wartości, gdy zmienia się rozmiar ekranu.

showExtras\$ emituje true, jeśli ekran ma rozmiar średni.

UWAGA Zwróć uwagę na zastosowanie dyrektywy strukturalnej `*ngIf`. Jeśli strumień obserwowalny `showExtras$` emituje wartość `true`, do drzewa DOM dodawany jest element `span`. Jeżeli emituje `false`, element `span` zostaje usunięty z DOM.

Wartości emitowane przez `media.asObservable()` mają typ `MediaChange` zawierający właściwość `mqAlias`, która przechowuje wartość reprezentującą bieżącą szerokość: `lg` dla dużych ekranów lub `md` dla średnich.

Aby zobaczyć kod z listingu 7.4 w działaniu, uruchom następujące polecenie i otwórz konsolę przeglądarki:

```
ng serve --app observable-media -o
```

Jeżeli rozmiar ekranu będzie średni (`md`), zobaczysz komunikat *Pokazuje dodatkowe informacje na średnich ekranach*. Gdy zmniejszysz szerokość okna przeglądarki do rozmiaru `sm`, ten tekst zostanie ukryty. Aby zobaczyć bieżące zapytanie o media CSS i inne właściwości klasy `mediaChange`, zmień instrukcję rejestrowania na `console.log(mediaChange);`.

W kodzie w listingu 7.4 bezpośrednio zadeklarowaliśmy obiekt obserwowalny `showExtras$` i zasubskrybowaliśmy go, ponieważ chcieliśmy monitorować `MediaChange`. Jednak ten kod można uprościć za pomocą interfejsu `API ObservableMedia.isActive()`, tak jak pokazano w listingu 7.5.

Listing 7.5. Korzystanie z interfejsu `API ObservableMedia.isActive()`

```

import {Component} from '@angular/core';
import {ObservableMedia} from '@angular/flex-layout';

@Component({
  selector: 'app-root',
  template: `<h3>Użycie interfejsu API ObservableMedia.isActive()</h3>
    <span *ngIf="this.media.isActive('md')">
      Pokazuje dodatkowe informacje na średnich ekranach</span>
  `
})
export class AppComponent {
  constructor(public media: ObservableMedia) {}
}

```

Wyświetla tekst tylko wtedy, gdy bieżącym rozmiarem rzutni jest `md`.

W części praktycznej w dalszej części tego rozdziału utworzymy nową wersję aplikacji ngAuction, która zaimplementuje RWD przy użyciu biblioteki Flex Layout i usługi ObservableMedia.

Inne opcje implementacji RWD

Biblioteka Flex Layout może być atrakcyjna dla początkujących programistów, ponieważ jest prosta w użyciu. Ale to nie jedyne rozwiązanie do tworzenia responsywnych układów w aplikacjach pisanych w Angularze. Oto kilka innych opcji:

- Pakiet CDK (ang. *component development kit*) Angulara zawiera moduł układu. Po zainstalowaniu paczki `@angular/cdk` można użyć klas `LayoutModule` i `BreakpointObserver` lub `MediaMatcher`, które monitorują zmiany w rozmiarze ekranu. Poza tym, ponieważ CDK Angulara jest zależnością równorzędną biblioteki Flex Layout, pracując bezpośrednio z CDK Angulara, będziesz używać jednej biblioteki zamiast dwóch.
- Gdy powstawał ten rozdział, biblioteka Flex Layout pozostawała w fazie beta, a jej twórcy często wprowadzali przełomowe zmiany w kolejnych wydaniach beta. Jeśli nie używasz najnowszej wersji frameworku Angular, Flex Layout może jej nie obsługiwać. Biblioteka Flex Layout nie obsługuje na przykład frameworku Angular 4.

Aby zminimalizować liczbę bibliotek wykorzystywanych w aplikacji, rozważ wdrożenie RWD przy użyciu modułu Flexbox CSS i systemu CSS Grid. Ponadto korzystanie z CSS, który jest natywnie obsługiwany przez przeglądarkę, zawsze będzie bardziej wydajne niż stosowanie jakiegokolwiek biblioteki JavaScriptu. Polecamy bezpłatny kurs wideo systemu CSS Grid autorstwa Wesa Bosa, dostępny pod adresem <https://cssgrid.io>.

7.2. Część praktyczna: przepisanie kodu aplikacji ngAuction

W tym rozdziale rozpoczniemy przepisywanie kodu aplikacji ngAuction od zera. Nowa wersja aplikacji ngAuction zostanie od samego początku napisana przy użyciu biblioteki Angular Material i będzie zawierać rzeczywiste obrazy, a nie tylko szare prostokąty. Komponent wyszukiwania będzie reprezentowany przez niewielką ikonę na pasku narzędzi i dodamy funkcjonalność koszyka. Użytkownicy będą mogli licytować produkty i kupować je, jeśli wygrają licytację.

7.2.1. Po co przepisywać aplikację ngAuction od zera?

Być może myślisz sobie: „Pracowaliśmy już nad aplikacją ngAuction w rozdziałach 2., 3. i 5. Czemu nie kontynuować po prostu budowania tej samej aplikacji?”. Celem początkowych rozdziałów było łagodne wprowadzenie do głównych artefaktów architektury Angulara, bez przeciążania Cię informacjami o architekturze aplikacji, wdrażaniu RWD i dostosowywaniu motywów.

Aplikacja ngAuction opracowana w poprzednich rozdziałach dobrze przysłużyła się temu celowi. To przepisanie kodu od zera zaprezentuje najlepsze praktyki programistyczne dla rzeczywistych aplikacji w Angularze. Oto założenia, które chcemy zrealizować:

- Utworzenie zmodularyzowanej aplikacji, w której każdy widok jest modulem leniwie ładowanym.
- Wykorzystanie biblioteki Angular Material do interfejsu użytkownika i zilustrowania dostosowywania motywów za pomocą Sass.

- Użycie biblioteki Flex Layout.
- Usunięcie zależności od bibliotek Bootstrap i JQuery.
- Usunięcie pola wyszukiwania ze strony głównej, aby lepiej wykorzystać przestrzeń ekranu.
- Umieszczenie współdzielonych komponentów i usług w oddzielnym folderze.
- zilustrowanie zarządzania stanem za pomocą wstrzykiwalnych usług, a następnie zaimplementowanie go ponownie przy użyciu biblioteki NgRx.
- Utworzenie skryptów dla testów jednostkowych i testów typu *end-to-end* (e2e).

Nie zamierzamy wdrożyć wszystkiego w tym rozdziale, ale zaczniemy realizować nasz plan.

W tej aplikacji zaimplementujemy RWD przy użyciu biblioteki Flex Layout i omówionej wcześniej usługi ObservableMedia. Na dużych ekranach strona główna aplikacji ngAuction wyświetli cztery produkty w rzędzie, tak jak pokazano na rysunku 7.6.



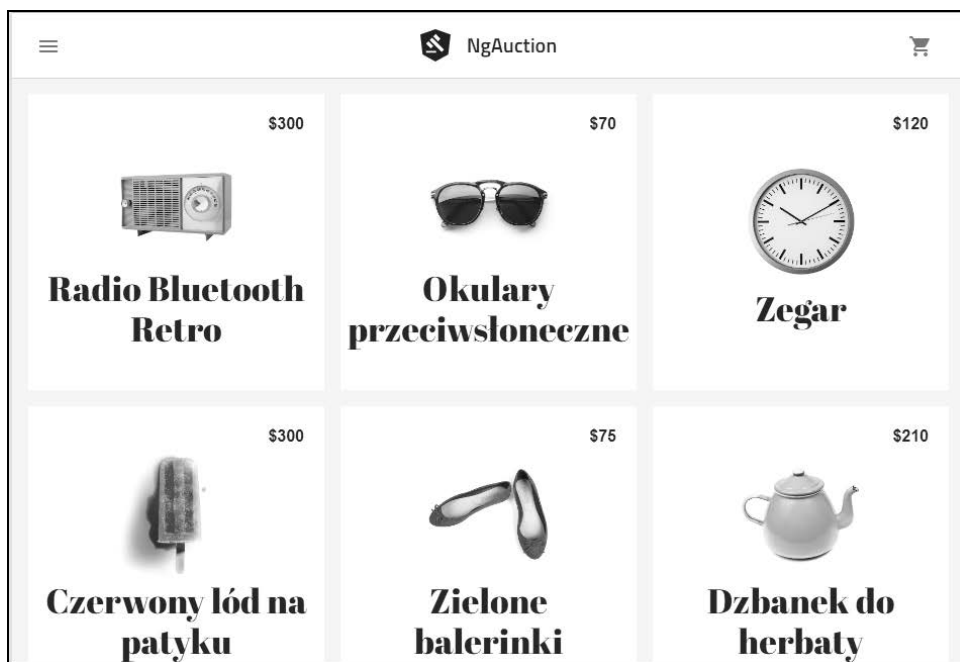
Rysunek 7.6. Renderowanie aplikacji ngAuction na dużych ekranach

UWAGA Dane i obrazy zapożyczyliśmy z aplikacji Google ilustrującej bibliotekę Polymer (zobacz <http://mng.bz/Y5d9>).

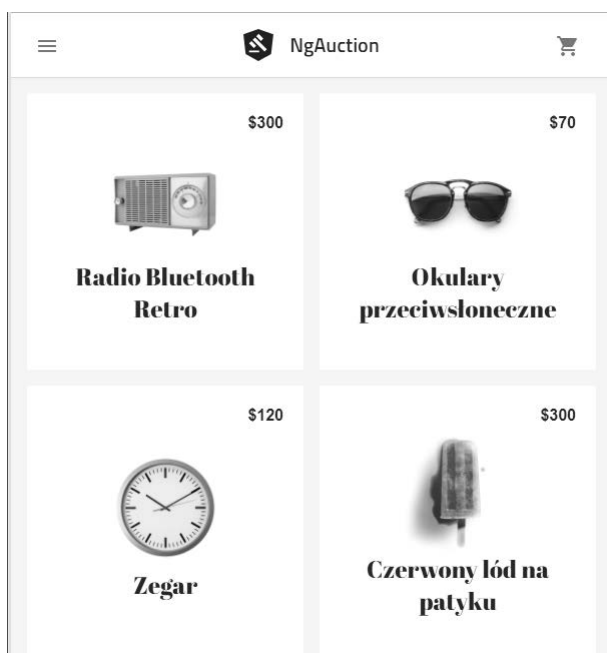
Aplikacja będzie subskrybować usługę ObservableMedia za pomocą potoku `async` i automatycznie zmieniać układ na trzy produkty w rzędzie, gdy tylko szerokość okna zmieni się na średnią, tak jak pokazano na rysunku 7.7.

Na małych ekranach aplikacja zmieni układ na dwukolumnowy, tak jak pokazano na rysunku 7.8.

Aplikacja będzie również zmieniać swój układ, gdy będzie renderowana na bardzo małym ekranie (układ jednokolumnowy) i bardzo dużym ekranie (układ pięciokolumnowy).



Rysunek 7.7. Renderowanie aplikacji ngAuction na średnich ekranach



Rysunek 7.8. Renderowanie aplikacji ngAuction na małych ekranach

7.2.2. Generowanie nowej aplikacji ngAuction

UWAGA Kod źródłowy dla tego rozdziału możesz pobrać z serwera wydawnictwa Helion, pod adresem <ftp://ftp.helion.pl/przyklady/anpro2.zip>.

Tym razem wygenerujemy projekt za pomocą polecenia `new` CLI Angulara z kilkoma opcjami. Nowa aplikacja `ngAuction` będzie używać preprocesora Sass dla stylów ze składnią SCSS. Określimy również prefiks `nga-`, aby każdy nowo wygenerowany komponent miał ten prefiks w swoim selektorze:

```
ng new ng-auction --prefix nga --style scss
```

UWAGA Korzyści wynikające z zastosowania SCSS omówimy w następnym podrozdziale.

Przejdź do katalogu `ng-auction` i uruchom następujące polecenia, aby dodać do projektu biblioteki Angular Material i Flex Layout:

```
npm install @angular/material @angular/cdk
```

```
npm i @angular/flex-layout
```

Instaluje bibliotekę Flex Layout.

Instaluje bibliotekę Angular Material oraz Component Development Kit. Biblioteka Angular Material wymaga także paczki `animations`, która została już zainstalowana przez CLI Angulara podczas generowania projektu.

Biblioteka Angular Material zawiera cztery wstępnie opracowane motywy i miałeś szansę wypróbować jeden z nich w rozdziale 5., w punkcie 5.6.1. Ale co zrobić, jeśli żaden z wbudowanych motywów nie będzie pasował do wymagań Twojego interfejsu użytkownika?

7.2.3. Tworzenie niestandardowego motywu Angular Material za pomocą Sass

Jeśli chcesz utworzyć niestandardowy motyw Angular Material dla swojej aplikacji, zapoznaj się z przewodnikiem Theming Guide na stronie <https://material.angular.io/guide/theming>. W tym punkcie przejrzymy po prostu kod plików `.scss`, które utworzyliśmy w celu dostosowania motywu dla aplikacji `ngAuction`.

Gdy wygenerowaliśmy aplikację `ngAuction`, użyliśmy opcji `--style scss`. W ten sposób poinformowaliśmy CLI Angulara, że nie zamierzamy używać plików CSS, ale zamiast tego zastosujemy arkusze Sass (ang. *Syntactically Awesome Style Sheets*; zobacz <http://sass-lang.com>). Sass jest rozszerzeniem CSS z własnym preprocesorem. Do korzyści wynikających ze stosowania z arkuszy Sass należą między innymi:

- **Zmienne** — przypisywanie stylów do zmiennych i wielokrotne używanie ich w wielu arkuszach stylów.
- **Zagnieżdżanie** — łatwa do pisania i odczytu składnia zagnieżdżonych selektorów CSS.
- **Domieszki** — bloki stylów, które mogą zawierać zmienne

Sass zapewnia dwie składnie, Sass i SCSS, a w niniejszej książce będziemy używać tej drugiej. Gdybyś zainstalował Sass osobno, musiałbyś uruchamiać pliki `.scss` przez

preprocesor, aby przed wdrożeniem skompilować je do zwykłych plików `.css`. Jednak CLI Angulara obsługuje Sass od ręki, więc preprocesor wykonuje swoje zadanie podczas kompilowania.

Składnia SCSS

Oto krótkie wprowadzenie do składni SCSS:

- **Zmienne** — nazwa zmiennej rozpoczyna się od znaku dolara. Poniższy fragment kodu deklaruje zmienną `$font-stack` i jej używa:

```
$font-stack: Helvetica, sans-serif;
```

```
body {  
  font: 100% $font-stack;  
}
```

Ta zmienna może być używana w wielu miejscach, a jeśli zdecydujesz się zmienić czcionkę Helvetica na jakąś inną, robisz to w jednym miejscu zamiast dokonywać zmian w każdym pliku `.css`, w którym ją zastosowałeś.

- **Zagnieżdżanie** — łatwa do odczytania składnia do pisania zagnieżdżonych selektorów CSS. Poniższy fragment kodu pokazuje, jak zagnieżdżyć selektory stylu `ul` i `a` wewnątrz selektora `div`:

```
div {  
  ul {  
    margin: 0;  
  }  
  a {  
    display: block;  
  }  
}
```

- **Domieszki** (ang. *mixins*) — to bloki stylu Sass. Domieszkę można dodać za pomocą `@include`. Domieszki mogą również używać zmiennych i mogą być wywoływane jako funkcje z argumentami, na przykład `mat-palette($mat-red);`.
- **Pliki częściowe** (ang. *partials*) — to po prostu pliki z fragmentami kodu, które mają być importowane przez inne pliki Sass. Ich nazwy muszą rozpoczynać się od podkreślenia, na przykład `_theme.scss`. Podczas importowania pliku częściowego podkreślenia nie są wymagane, na przykład `@import './theme';`. Pliki częściowe nie są kompilowane do osobnych plików CSS, ich zawartość jest kompilowana tylko jako część plików `.scss`, które je importują.
- **Importy** — instrukcja `@import` umożliwia importowanie stylów znajdujących się w innych plikach. Chociaż CSS również ma słowo kluczowe `@import`, dodatkowo wykonuje żądanie HTTP dla każdego pliku. W przypadku Sass wszystkie importy są łączone w jeden plik CSS podczas wstępnego przetwarzania, więc do załadowania CSS potrzebne jest tylko jedno żądanie HTTP.

W naszej aplikacji *ngAuction* utworzymy katalog *styles*, przeniesiemy tam wygenerowany plik *styles.scss* i dodamy jeszcze jeden plik częściowy: *_theme.scss*. Zawartość pliku *_theme.scss* została pokazana w listingu 7.6. Używamy palety `$mat-cyan` zdefiniowanej w zaimportowanym pliku *_theming.scss*.

Listing 7.6. Plik `_theme.scss`

```

@import '~@angular/material/theming';

$nga-primary: mat-palette($mat-cyan);
$nga-accent:  mat-palette($mat-cyan, A200, A100, A400);
$nga-warn:    mat-palette($mat-red);

$nga-theme:  mat-light-theme($nga-primary, $nga-accent, $nga-warn);

$nga-background: map-get($nga-theme, background);
$nga-foreground: map-get($nga-theme, foreground);
$nga-typography: mat-typography-config();

```

Deklaruje zmienną dla palety podstawowej i inicjuje ją z paletą \$mat-cyan.

Deklaruje i inicjuje zmienną dla palety kolorów dodatkowych, określając domyślny, jaśniejszy i ciemniejszy odcień \$mat-cyan.

Deklaruje i inicjuje zmienną dla palety błędów i ostrzeżeń.

Tworzy motyw (obiekt Sass zawierający wszystkie palety).

Deklaruje i inicjuje zmienną dla palety tła.

Deklaruje i inicjuje zmienną dla palety tekstu i ikon.

Deklaruje i inicjuje zmienną dla typografii.

W pliku `_theme.scss` użyliśmy koloru niebieskozielonego dla palet podstawowych i akcentów. Ich definicje możesz znaleźć w pliku `node_modules/@angular/material/_theming.scss`.

W kodzie w listingu 7.7 dodajesz style w `styles.scss`, zaczynając od importowania powyższego `_theme.scss`.

Listing 7.7. Plik `styles.scss`

```

@import './theme';

@import url('https://fonts.googleapis.com/icon?family=Material+Icons');
@import url('https://fonts.googleapis.com/css?family=Titillium+Web:600');

@import url('https://fonts.googleapis.com/css?family=Abril+Fatface');

// Upewnij się, że tę domieszkę załączyłeś tylko raz!
@include mat-core();

@include angular-material-theme($nga-theme);

// Style globalne
html {
  -moz-osx-font-smoothing: grayscale;
  -webkit-font-smoothing: antialiased;
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
  height: 100%;
}

body {
  color: #212121;
  background-color: #f3f3f3;
}

```

Importuje ikony Google Material.

Importuje czcionki Titillium Web (użyjemy ich dla paska narzędzi i później dla wartości składanych ofert).

Importuje czcionki Abril Fatface (użyjemy ich dla nazw produktów).

Importuje style podstawowe biblioteki Angular Material, które nie są zależne od motywu.

Ładuje nasz niestandardowy motyw skonfigurowany w pliku `_theme.scss`.

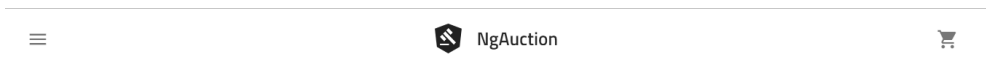
```
font-family: mat-font-family($nga-typography);
line-height: mat-line-height($nga-typography, body-1);
font-size: mat-font-size($nga-typography, body-1);
height: 100%;
margin: 0;
}
```

Pliki *styles.scss* i *_theme.scss* definiują style globalne dla całej aplikacji, a określimy je we właściwości *styles* w pliku *.angular-cli.json*. W aplikacji ngAuction będziesz także stylizować poszczególne komponenty, a *_theme.scss* będzie ponownie używany w każdym komponencie. Celowo rozbiliśmy definicję stylów na dwa pliki, żebyśmy mogli ponownie użyć pliku *_theme.scss* (zawierającego tylko definicje zmiennych) w komponentach bez duplikowania podstawowych stylów, obrazów i czcionek użytych w *styles.scss*.

Skonfigurowaliśmy nasz niestandardowy motyw i możesz rozpocząć pracę nad interfejsem użytkownika strony głównej aplikacji ngAuction.

7.2.4. Dodawanie paska narzędzi do komponentu najwyższego poziomu

Rysunek 7.6 przedstawia stronę główną aplikacji ngAuction, która zawiera pasek narzędzi biblioteki Material i komponent HomeComponent. Mówiąc ściślej, zawiera pasek narzędzi i znacznik `<router-outlet>`, gdzie będziemy renderować HomeComponent. Zaczniemy od utworzenia pierwszej wersji paska narzędzi. Ten pasek narzędzi będzie zawierał ikonę menu po lewej stronie, logo ngAuction pośrodku i ikonę koszyka po prawej. Nie będzie zawierał przycisku *Szukaj* (dodamy go w rozdziale 11., w podrozdziale 11.8) i będzie wyglądać tak, jak na rysunku 7.9.



Rysunek 7.9. Pasek narzędzi

Po lewej stronie wykorzystamy z ikony menu, a po prawej z ikony `shopping_cart` — obie pochodzą z Google Material. W przypadku logo umieszczamy ikonę `gavel` Google Material na kształcie przypominającym logo Angulara i zapisujemy ją w pliku *logo.svg* dołączonym do kodu źródłowego dla tej książki.

Jak wiesz z części praktycznej rozdziału 5., aby użyć komponentów Angular Material, należy załączyć odpowiednie moduły w sekcji `imports` modułu głównego aplikacji. W przypadku paska narzędzi potrzebujemy modułów `MatToolbarModule`, `MatButtonModule` i `MatIconModule`. Ponieważ będziesz używać biblioteki Flex Layout, musimy dodać do modułu głównego również moduł `FlexLayoutModule`. W dalszej części tego podrozdziału użyjemy usługi `HttpClient` do odczytywania danych o produktach, więc musimy dodać moduł `HttpClientModule` do modułu głównego.

Zaktualizujemy wygenerowany przez interfejs CLI plik *app.module.ts*, aby uwzględnił moduły z listingu 7.8.

Listing 7.8. Plik `app.module.ts`

```
import {BrowserModule} from '@angular/platform-browser';
import {NgModule} from '@angular/core';
import {MatButtonModule} from '@angular/material/button';
import {MatIconModule} from '@angular/material/icon';
import {MatToolbarModule} from '@angular/material/toolbar';
import {FlexLayoutModule} from '@angular/flex-layout';
import {HttpClientModule} from '@angular/common/http';
import {AppComponent} from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    MatButtonModule,
    MatIconModule,
    MatToolbarModule,
    FlexLayoutModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Dodaje wymagane moduły z biblioteki Angular Material.

Dodaje moduł biblioteki Flex Layout.

Dodaje HttpClientModule – będziemy używać HttpClient do pobierania danych o produktach.

Zastąp zawartość wygenerowanego pliku `app.component.html` kodem z listingu 7.9.

Listing 7.9. Plik `app.component.html`

```
<mat-toolbar class="toolbar">
  <button class="toolbar__icon-button" mat-icon-button>
    <mat-icon>menu</mat-icon>
  </button>

  <div class="toolbar__logo-title-group"
    fxLayout
    fxLayoutAlign="center center">
    <a routerLink="/">
      
    </a>
    <a class="toolbar__title"
      routerLink="/">NgAuction</a>
  </div>

  <div fxFlex></div>

  <button mat-icon-button class="toolbar__icon-button
    toolbar__shopping-cart-button">
    <mat-icon>shopping_cart</mat-icon>
```

Przycisk menu z ikoną.

Wyświetla logo pośrodku paska narzędzi.

Przekształca logo w klikalny link, który po skonfigurowaniu tras będzie wyświetlał HomeComponent.

Przekształca tekst "NgAuction" w klikalny link.

Wypełniacz, który przesuwa ikonę koszyka do prawego brzegu.

Przycisk koszyka z ikoną.

```

</button>

</mat-toolbar>

<!--<router-outlet></router-outlet>-->

```

← Gniazdo routera pozostaje wykomentowane, dopóki nie skonfigurujemy tras.

Aby pasek narzędzi wyglądał tak jak na rysunku 7.9, musimy dodać do pliku *app.component.scss* stylizację pokazaną w listingu 7.10.

Listing 7.10. Plik *app.component.scss*

```

@import '../styles/theme';

```

← Importuje nasz niestandardowy motyw.

```

:host {
  display: block;
  height: 100%;
}

```

← Używa pseudoselektora Angulara `:host` do stylizacji komponentu, który hostuje `AppComponent`.

```

.toolbar {
  background-color: mat-color($nga-background, card);
  position: relative;
  box-shadow: 0 1px mat-color($nga-foreground, divider);
}

```

← Stosuje w tym motywie to samo tło co w komponencie `card` Material (w naszym motywie jest to białe tło).

```

.toolbar__logo-title-group {
  position: absolute;
  right: 50%;
  left: 50%;
}

```

← Stylizuje nazwę logo.

```

.toolbar__logo {
  height: 32px;
  margin-right: 16px;
}

```

← Stylizuje obraz logo.

```

.toolbar__title {
  color: mat-color($nga-foreground, text);
  font-family: 'Titillium Web', sans-serif;
  font-weight: 600;
  text-decoration: none;
}

```

← Stylizuje nazwę paska narzędzi.

```

.toolbar__icon-button {
  color: mat-color($nga-foreground, icon);
}

```

← Stylizuje wygląd ikony.

```

.toolbar__shopping-cart-button {
  margin-right: 8px;
}

```

← Stylizuje przycisk koszyka.

Uruchomienie polecenia `ng serve` spowoduje wyrenderowanie aplikacji ngAuction, która wygląda tak, jak pokazano na rysunku 7.9.

Mamy wyrenderowany interfejs paska narzędzi, a teraz musimy pod nim wyświetlić produkty. Najpierw należy utworzyć usługę `ProductService`, która zapewni dane

o produktach, a następnie utworzyć komponent HomeComponent, który wyrenderuje te dane. Zaczniemy od usługi ProductService.

7.2.5. Tworzenie usługi produktowej

Usługa produktowa potrzebuje danych. W rzeczywistych aplikacjach dane są dostarczane przez serwer i zajmiemy się tym w rozdziale 12. Na razie użyjemy po prostu pliku JSON zawierającego informacje o produktach. Obrazy produktów również będą znajdować się po stronie klienta. Przykłady kodu dołączone do książki zawierają plik *src/data/products.json*, którego fragment znajduje się w listingu 7.11.

Listing 7.11. Fragment pliku *src/data/products.json*

```
[
  {
    "id": 1,
    "description" : "Czy to nie fajnie, gdy rzeczy wyglądają na stare, choć wcale nie  
→ są?... ",
    "imageUrl" : "data/img/radio.png",
    "price" : 300,
    "title" : "Radio Bluetooth retro"
  },
  {
    "id": 2,
    "description" : "Bądź optymistą. Zawsze noś ze sobą okulary przeciwsłoneczne...",
    "featured" : true,
    "imageUrl" : "data/img/sunnies.png",
    "price" : 70,
    "title" : "Okulary przeciwsłoneczne"
  }
  ...
]
```

Ten plik zawiera adresy URL obrazów produktów znajdujących się w folderze *data/img*. Jeśli postępujesz zgodnie z instrukcjami i próbujesz samodzielnie zbudować aplikację ngAuction, skopiuj do swojego projektu katalog *src/data* z przykładów kodu i dodaj linię "data" właściwości assets aplikacji w pliku *.angular-cli.json*.

Klasy ProductService będziemy używać w więcej niż jednym komponencie. Wygenerujemy ją w folderze *src/app/shared/services*. Później w tym folderze będziemy dodawać inne usługi wielokrotnego użytku (na przykład SearchService). ProductService wygenerujemy za pomocą następującego polecenia CLI Angulara:

```
ng generate service shared/services/product
```

Następnie w *app.module* dodamy dostawcę dla tej usługi:

```
...
import {ProductService} from './shared/services/product.service';

@NgModule({
  ...
  providers: [ProductService]
})
export class AppModule {}
```

Najlepsze praktyki

Instrukcja importowania usługi `ProductService` jest dość długa i wskazuje na plik, w którym ta usługa jest zaimplementowana. Wraz z rozwojem aplikacji wzrośnie liczba usług i liczba instrukcji importu w module, zanieczyszczając kod modułu.

W folderze *services* utwórz plik o nazwie *index.ts* w następujący sposób:

```
import {Product, ProductService} from './product.service';
```

```
export {Product, ProductService} from './product.service';
```

Importujemy klasy `Product` i `ProductService` i natychmiast ponownie je eksportujemy. Teraz instrukcję importu w *app.module* można uprościć w następujący sposób:

```
import {Product, ProductService} from './shared/services';
```

Jeśli masz tylko jedną ponownie eksportowaną klasę, może to wyglądać na przesadę. Ale jeśli masz wiele klas w folderze *services*, wystarczy napisać tylko jedną instrukcję importu dla wszystkich klas, funkcji lub zmiennych, które chcesz zaimportować, na przykład:

```
import { ProductService, Product, SearchService } from './shared/services';
```

Pamiętaj, że zadziała to tylko wtedy, gdy plik z takimi reeksportami będzie nazywał się *index.ts*.

Plik *product.service.ts* zawiera interfejs `Product` i klasę `ProductService`. Interfejs `Product` definiuje typ obiektów zwracanych przez metody klasy `ProductService`: `getAll()` i `getById()`. Kod usługi `ProductService` został pokazany w listingu 7.12.

Listing 7.12. Plik *product.service.ts*

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';

export interface Product { ←—— Definiuje typ Product.
  id: number;
  title: string;
  price: number;
  imageUrl: string;
  description: string;
}

@Injectable()
export class ProductService {
  constructor(private http: HttpClient) {} ←—— Wstrzykuje obiekt HttpClient.

  getAll(): Observable<Product[]> { ←—— Ta funkcja deklaruje obiekt Observable,
    return this.http.get<Product[]>('/data/products.json'); ←—— który może zwracać wszystkie obiekty Product.
  }

  getById(productId: number): Observable<Product> { ←—— Ta funkcja deklaruje obiekt
    return this.http.get<Product[]>('/data/products.json') ←—— Observable, który może zwracać
    .pipe( ←—— produkty według identyfikatora.
      map(products => <Product>products.find(p => p.id === productId)) ←——
    ); ←—— Funkcja map() znajduje identyfikator produktu,
  } ←—— który odpowiada argumentowi funkcji.
}
```

Ponieważ nie mamy prawdziwego serwera danych, obie metody odczytują cały plik *product.json*, a metoda `findById()` stosuje także metodę `find()` do tablicy produktów, aby znaleźć produkt z pasującym identyfikatorem.

Najlepsze praktyki

Zdefiniowaliśmy `Product` jako interfejs, a nie klasę. Ponieważ JavaScript nie obsługuje interfejsów, skompilowany kod nie będzie zawierał `Product`. Gdybyśmy zdefiniowali `Product` jako klasę, kompilator TypeScriptu przekształciłby ją w funkcję JavaScriptu albo w klasę i uwzględniłby ją w kodzie wykonywalnym. Definiowanie typów jako interfejsów TypeScriptu, a nie klas, redukuje rozmiar wykonywalnego kodu.

W następnym punkcie utworzymy moduł funkcyjny, który będzie zawierać komponent `HomeComponent`, będący pierwszym konsumentem usługi `ProductService`.

7.2.6. Tworzenie modułu strony głównej

Chcemy utworzyć każdy widok jako moduł funkcyjny. Umożliwi to leniwe ładowanie tych modułów, a kod każdego widoku zostanie skompilowany jako osobna paczka. Wygenerujemy moduł funkcyjny w następujący sposób:

```
ng generate module home
```

To polecenie spowoduje utworzenie katalogu *src/app/home* zawierającego plik *home.module.ts* z zawartością pokazaną w listingu 7.13.

Listing 7.13. Plik *home.module.ts*

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

@NgModule({
  imports: [
    CommonModule,
  ],
  declarations: []
})
export class HomeModule {}
```

Komponent strony głównej możemy wygenerować za pomocą następującego polecenia:

```
ng generate component home
```

Po uruchomieniu tego polecenia CLI Angulara wyświetli komunikat, że wygenerowane zostały cztery pliki (komponent strony głównej), a jeden plik został zaktualizowany (moduł strony głównej) — `HomeComponent` został dodany do sekcji `declarations` w dekoratorze `@NgModule`:

```
create src/app/home/home.component.scss (0 bytes)
create src/app/home/home.component.html (23 bytes)
create src/app/home/home.component.spec.ts (614 bytes)
create src/app/home/home.component.ts (262 bytes)
update src/app/home/home.module.ts (251 bytes)
```

W tym module użyjemy biblioteki Flex Layout, więc chcemy skonfigurować domyślną trasę, która będzie renderować HomeComponent. Ponadto będziemy wyświetlać produkty za pomocą komponentu `<mat-grid-list>` z biblioteki Angular Material. Dodaj wymagany kod do pliku *home.module.ts*, żeby był zgodny z listingiem 7.14.

Listing 7.14. Zmodyfikowany plik *home.module.ts*

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule } from '@angular/router';
import { FlexLayoutModule } from '@angular/flex-layout';
import { MatGridListModule } from '@angular/material/grid-list';
import { HomeComponent } from './home.component';

@NgModule({
  imports: [
    CommonModule,
    RouterModule.forChild([           ← Dodaje konfigurację trasy dla modułu funkcyjnego.
      { path: '', component: HomeComponent }
    ]),
    FlexLayoutModule, ← Dodaje bibliotekę Flex Layout.
    MatGridListModule ← Dodaje moduł Angular Material wymagany
                        przez <mat-grid-list>.
  ],
  declarations: [HomeComponent]
})
export class HomeModule {}
```

Następnym krokiem jest aktualizacja komponentu HomeComponent w wygenerowanym pliku *home.component.ts*. Wstrzykniemy do tego komponentu dwie usługi: Product Service i ObservableMedia. Wywołamy metodę `getAll()` na ProductService, aby uzyskać dane produktu. ObservableMedia będzie obserwować szerokość rzutni, aby odpowiednio zmieniać układ interfejsu użytkownika. Mówiąc ściślej, dane o produktach będą wyświetlane w siatce, a usługa ObservableMedia będzie zmieniać liczbę kolumn w siatce od jednej do pięciu na podstawie bieżącej szerokości okna. Kod komponentu HomeComponent został pokazany w listingu 7.15.

Listing 7.15. Plik *home.component.ts*

```
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';

import { Component } from '@angular/core';
import { ObservableMedia } from '@angular/flex-layout';
import { Product, ProductService } from '../shared/services';

@Component({
  selector: 'nga-home',
  styleUrls: [ './home.component.scss' ],
  templateUrl: './home.component.html'
})
export class HomeComponent {
  readonly columns$: Observable<number>; ← Strumień obserwowalny dostarczający
                                          liczbę kolumn w siatce.
```

```

readonly products$: Observable<Product[]>; ← Strumień obserwowalny dla produktów.

readonly breakpointsToColumnsNumber = new Map([ ← Mapuje alias zapytania o media
  [ 'xs', 1 ],                                na liczbę kolumn w siatce.
  [ 'sm', 2 ],
  [ 'md', 3 ],
  [ 'lg', 4 ],
  [ 'xl', 5 ],
]);

constructor(private media: ObservableMedia,
              private productService: ProductService) { ← Wstrzykuje ObservableMedia
  this.products$ = this.productService.getAll(); ← i ProductService.
  this.columns$ = this.media.asObservable() ← Pobiera dane o wszystkich
  .pipe(                                       produktach.
    map(mc => <number>this.breakpointsToColumnsNumber.get(mc.mqAlias)) ← Przekształca obiekt ObservableMedia
  );                                           w Observable.
}

```

Pobiera liczbę kolumn siatki na podstawie emitowanego aliasu zapytania o media; <number> oznacza konwertowanie obiektu na liczbę.

Metoda `getAll()` w usłudze `ProductService` inicjuje zmienną `product$` typu `Observable`. Nie widzimy tutaj wywołania metody `subscribe()`, ponieważ użyjemy potoku `async` w szablonie komponentu strony głównej.

Rola usługi `ObservableMedia` polega na wysyłaniu aliasów zapytań o media do komponentu i wskazywaniu bieżącej szerokości ekranu urządzenia użytkownika. Ta szerokość może się zmieniać, jeśli rzutnia jest oknem przeglądarki, a użytkownik zmienia jego rozmiar. Jeżeli użytkownik uruchomi tę aplikację na smartfonie, szerokość rzutni nie zmieni się, ale `HomeComponent` i tak musi to wiedzieć, aby wyrenderować siatkę produktów.

Teraz musisz zastąpić wygenerowany szablon w pliku `home.component.html` znacznikami służącymi do wyświetlania produktów w siatce wierszy i kolumn. Dla siatki użyjemy komponentu `<mat-grid-list>` z biblioteki Angular Material. Zawartość każdej komórki siatki będzie renderowana w komponencie `<mat-grid-tile>`.

W tym szablonie użyjemy potoku `async` dwukrotnie. Pierwszy potok `async` zasubskrybuje strumień obserwowalny, który emituje liczbę kolumn w siatce, a drugi potok zasubskrybuje strumień obserwowalny, który emituje dane o produktach. Kod pliku `home.component.html` został pokazany w listingu 7.16.

Listing 7.16. Plik `home.component.html`

```

<div class="grid-list-container">
  <mat-grid-list [cols]="columns$ | async" gutterSize="16"> ← Subskrybuje liczbę kolumn i wiąże ją z właściwością cols
    <mat-grid-tile class="tile" *ngFor="let product of products$ | async"> ← komponentu <mat-grid-list>.
      <a class="tile__content"
        fxLayout="column"
        fxLayoutAlign="center center"
      >
        <img alt="Product image" data-bbox="100 860 200 900" />
        <div>
          <h3>{{product.name}}</h3>
          <p>{{product.description}}</p>
          <button>Kup</button>
        </div>
      </a>
    </mat-grid-tile>
  </mat-grid-list>
</div>

```

Renderuje <mat-grid-tile> dla każdego produktu przy użyciu danych ze strumienia obserwowalnego products\$.

Opakowuje zawartość każdego kafelka w znacznik <a>, aby przekształcić kafelek w klikalną łączkę.

```

[routerLink]="['/products', product.id]">
<span class="tile__price-tag"
      ngClass.xs="tile__price-tag--xs">
  {{ product.price | currency:'USD': 'symbol': '.0' }}
</span>

<div class="tile__thumbnail"
      [ngStyle]="{'background-image': 'url(' + product.imageUrl + ')'}"></div>

<div class="tile__title"
      ngClass.xs="tile__title--xs"
      ngClass.sm="tile__title--sm">{{ product.title }}</div>
</a>
</mat-grid-tile>
</mat-grid-list>
</div>

```

Kliknięcie kafelka powoduje przejście do ścieżki `/products` i przekazanie jako parametru identyfikatora wybranego produktu.

Dla bardzo małych rzutni dodaje style zdefiniowane w `tile__price-tag--xs`.

UWAGA W tej wersji aplikacji *ngAuction* nawigacja do ekranu szczegółów produktu nie jest zaimplementowana. Kliknięcie kafelka produktu spowoduje wypisanie błędu w konsoli przeglądarki.

Chcielibyśmy szerzej wyjaśnić ostatnią adnotację z listingu 7.16. Ten element `<span>` ma styl zdefiniowany w `tile__price-tag`, ale w przypadku, gdy rozmiar rzutni stanie się bardzo mały (`xs`), dyrektywa `ngClass.xs` biblioteki Flex Layout doda style zdefiniowane w `tile__price-tag--xs`. Jeśli porównasz definicje stylów `tile__price-tag` i `tile__price-tag--xs` w listingu 7.17, zauważysz, że scalenie tych dwóch stylów oznaczałoby zmianę rozmiaru czcionki z 16 pikseli na 14.

WSKAZÓWKA W nazewnictwie niektórych stylów używamy symboli `__` i `--` zgodnie z zaleceniami metody „blok, element, modyfikator” (ang. *block, element, modifier* — BEM; zobacz <http://getbem.com>).

Aby dokończyć `HomeComponent`, musimy dodać kilka stylów w pliku *home.component.scss*, tak jak pokazano w listingu 7.17.

Listing 7.17. Plik *home.component.scss*

```

@import '../styles/theme';
:host {
  display: block;
}

.grid-list-container {
  margin: 16px;
}

.tile {
  background-color: mat-color($nga-background, card);
  &:hover {

```

Importuje nasz niestandardowy motyw.

Ustawia kolor tła na taki sam, jaki ma komponent `card` biblioteki Angular Material (biały).

```

    @include mat-elevation(4); ← Jeśli wskaźnik myszy zostanie przesunięty na kafelkę,
    transition: .3s;           spowoduje to uniesienie kafelka do poziomu 4. poprzez
  }                             dodanie efektu cienia (zwracanego przez domieszkę
                                mat-elevation).

.tile_content {
  display: block;
  height: 100%;
  width: 100%;
  padding: 16px;
  position: relative;
  text-align: center;
  text-decoration: none;
}

.tile_price-tag { ← Domyślny styl dla znacznika ceny produktu.
  color: mat-color($nga-foreground, text);
  font-size: 16px;
  font-weight: 700;
  position: absolute;
  right: 20px;
  top: 20px;
}

.tile_price-tag-xs { ← Styl dla znacznika ceny produktu dla bardzo małych rzutni.
  font-size: 14px;
}

.tile_thumbnail {
  background: no-repeat 50% 50%;
  background-size: contain;
  height: 50%;
  width: 50%;
}

.tile_title { ← Domyślny styl dla nazwy produktu.
  color: mat-color($nga-foreground, text);
  font-family: 'Abril Fatface', cursive;
  font-size: mat-font-size($nga-typography, display-1); ← Zgodnie ze specyfikacją
  line-height: mat-line-height($nga-typography, display-1); Material Design używa
                                                                display-1 dla stylów czcionki
                                                                zamiast określać rozmiar
                                                                na sztywno.
}

.tile_title--sm { ← Styl dla nazwy produktu dla małych rzutni.
  font-size: mat-font-size($nga-typography, headline);
  line-height: mat-line-height($nga-typography, headline);
}

.tile_title--xs { ← Styl dla nazwy produktu dla bardzo małych rzutni.
  font-size: mat-font-size($nga-typography, title);
  line-height: mat-line-height($nga-typography, title);
}

```

HomeComponent jest gotowy. Co trzeba zrobić, aby wyrenderować go pod paskiem narzędzi?

7.2.7. Konfigurowanie tras

Na początku tego ćwiczenia praktycznego stwierdziliśmy, że każdy widok w aplikacji *ngAuction* będzie osobnym modulem, i utworzyliśmy *HomeComponent* jako moduł. Teraz musimy skonfigurować trasę dla tego modułu. Utwórz plik *src/app/app.routing.ts* z następującą zawartością:

```
import {Route} from '@angular/router';

export const routes: Route[] = [
  {
    path: '',
    loadChildren: './home/home.module#HomeModule'
  }
];
```

Jak widzisz, użyliśmy składni dla modułów leniwie ładowanych, co wyjaśniliśmy w rozdziale 4., w podrozdziale 4.3. Ładujemy tę konfigurację w *app.module.ts*, wywołując *Router.forRoot()*:

```
...
import {RouterModule} from '@angular/router';
import {routes} from './app.routing';

@NgModule({
  ...
  imports: [
    ...
    RouterModule.forRoot(routes)
  ]
  ...
})
export class AppModule { }
```

Ostatnim krokiem jest odkomentowanie ostatniej linii w pliku *app.component.html*, która ma znacznik `<router-outlet>`, żeby szablon komponentu aplikacji był ułożony w następujący sposób:

```
<mat-toolbar>...</mat-toolbar>

<router-outlet></router-outlet>
```

Zakończyliśmy pracę związaną z kodowaniem strony głównej.

7.2.8. Uruchomienie aplikacji ngAuction

Pierwsza wersja nowej aplikacji *ngAuction* jest gotowa, więc utworzymy paczki programistyczne i zobaczymy, jak wygląda to w przeglądarce. Uruchomienie polecenia `ng serve` wygeneruje dane wyjściowe pokazane na rysunku 7.10.

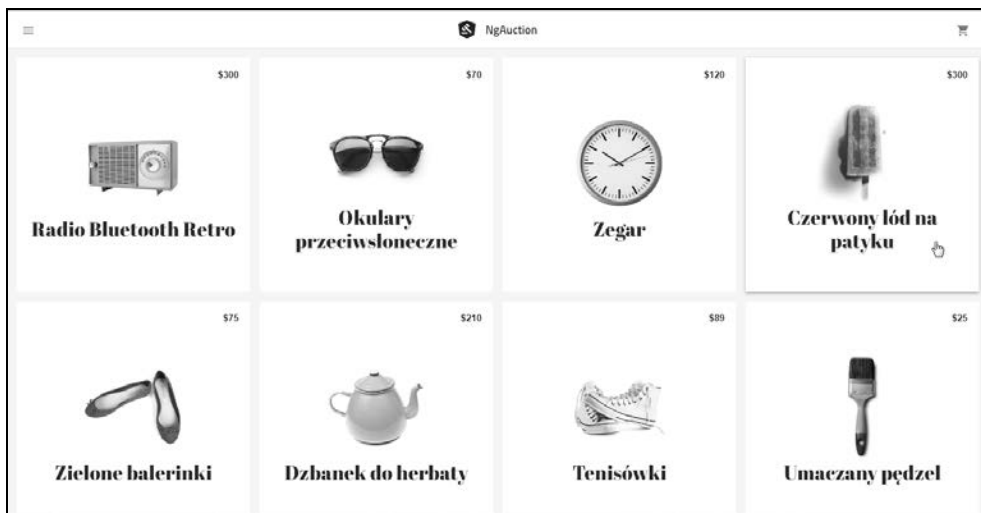
Zwróć uwagę na pierwszą linię: interfejs CLI Angulara umieścił moduł strony głównej w osobnej paczce. Stało się tak dlatego, że w konfiguracji tras użyliśmy składni dla modułów leniwie ładowanych, ale gdy otworzysz w przeglądarce stronę *http://localhost:4200*, zobaczysz, że moduł strony głównej został załadowany, tak jak pokazano na rysunku 7.11.

```

Date:
Hash:
Time:
ms
chunk {home-home-module} home-home-module.js, home-home-module.js.map (home-home-module) 254 kB [rendered]
chunk {main} main.js, main.js.map (main) 210 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 259 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 8.83 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 601 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 4.91 MB [initial] [rendered]
i [DWM] Compiled successfully.

```

Rysunek 7.10. Kompilowanie aplikacji ngAuction za pomocą polecenia ng serve



Rysunek 7.11. Uruchomienie aplikacji ngAuction

Moduł strony głównej został załadowany gorliwie, ponieważ został skonfigurowany jako trasa domyślna (zmapowany na pustą ścieżkę). Strona główna aplikacji ngAuction jest gotowa. Brakuje jedynie przycisku *Szukaj* na pasku narzędzi. Dodamy go w rozdziale 11., w podrozdziale 11.8.

WSKAZÓWKA Kliknięcie któregośkolwiek kafelka produktu spowoduje wyświetlenie w konsoli przeglądarki błędu, na przykład *Cannot match any routes. URL Segment: 'products/2'*. Ten błąd zniknie w wersji aplikacji *ngAuction*, którą utworzymy w rozdziale 9. po opracowaniu strony szczegółów produktu.

Podsumowanie

- Możesz utrzymywać jedną bazę kodu aplikacji internetowej, która będzie dostosowywała swój interfejs użytkownika na podstawie dostępnej szerokości ekranu urządzenia użytkownika.
- Biblioteka Flex Layout umożliwia subskrybowanie powiadomień dotyczących zmian szerokości rzutni i stosowanie odpowiedniego układu interfejsu użytkownika.
- Biblioteka Flex Layout zawiera klasę *ObservableMedia*, która może powiadamiać o aktualnej szerokości rzutni, oszczędzając Ci konieczności pisania kodu CSS dla tego zadania.

Skorowidz

A

- abstrakcje, 129
- ActivatedRoute, 78
- akcja, 400, 404, 413
 - init magazynu, 421
 - LoadSuccess, 441
 - search-success, 424
- akcje dla produktów, 435
- Angular, 24, 249
 - Elements, 444
 - Material, 139
- AOT, ahead-of-time, 33
- API
 - formularzy, 246
 - obserwatora, 526
- aplikacja ngAuction, 35
- aplikacje
 - jednostronicowe, 67
 - SPA, 107
 - z gniazdami routera, 107
 - zmodularyzowane, 134
- architektura
 - aplikacji, 27
 - Redux, 400
- asercja, 362
- async, 392
- await, 392

B

- Bazel, 445
- bąbelkowanie zdarzeń, 201, 409
- BDD, behavior-driven development, 361
- biblioteka
 - Angular Material, 139
 - Bootstrap, 63

- Flex Layout, 169, 170
- ngrx, 399, 403
- Protractor, 382
- Redux, 400
- RxJS, 354, 357, 517
- testowa frameworku Angular, 371
- Zone, 224

- bloki konstrukcyjne nawigacji, 71
- błąd 404, 77
- błędy
 - walidatorów, 267
 - warunkowe wyświetlanie, 267
- Bootstrap, 63

C

- CDK, Component Dev Kit, 445
- CLI, command-line interface, 29
- CLI Angulara, 28, 445
- cykl życia komponentu, 223, 227

D

- debugowanie, 227
 - strumieni obserwowalnych, 526
- definicje metod, 470
- deklarowanie
 - dostawców, 131
 - interfejsu, 495
- dekorator , 502
 - @Component(), 25
 - @ViewChildren, 214
 - UIComponent, 503
- dekoratory TypeScriptu, 42
- destrukuryzacja, 460
 - obiektów, 460
 - tablic, 462

DI, dependency injection, 115
 dodawanie paska narzędzi, 183
 domieszki, 180, 181
 dostawcy, 118, 132
 dla strażników, 103
 w modułach, 134, 136
 dynamiczne zmienianie walidatorów, 277
 dyrektywa, 43
 formControl, 255
 formControlName, 255
 formGroup, 254
 formGroupName, 255
 ngContent, 216
 NgForm, 247
 NgModel, 247, 248
 NgModelGroup, 248
 SsnValidatorDirective, 281
 dyrektywy
 Forms API, 259
 formularzy, 246, 249
 formularzy reaktywnych, 254
 kontenera, 171
 dziedziczenie, 465, 493

E

efekty, 410, 416
 nawigacja, 427
 dla produktów, 440
 w przepływie danych, 412
 elementy kontenera, 173
 emitowanie zdarzeń, 204
 emulacja błędów serwera, 321

F

fabryka, 131
 Flex Layout, 170
 dyrektywy biblioteki, 171
 układ stron, 169
 format JSON, 301
 Forms API, 146, 156
 formularze
 aktualizacja na podstawie danych, 261
 dynamiczne dodawanie kontrolek, 258
 dyrektywy, 246
 dyrektywy Forms API, 259
 funkcjonalności frameworku Angular, 249

oparte na szablonach, 246, 249
 niestandardowe walidatory, 281
 reaktywne, 246, 251
 API, 256
 dynamiczne zmienianie
 walidatorów, 277
 dyrektywy, 254
 walidatory niestandardowe, 270
 ukrywanie błędów, 267
 użycie FormArray, 258
 walidacja, 265
 wyświetlanie błędów, 267
 zagnieżdzone, 248
 framework
 Angular, 24, 249, 303
 Express, 298
 Jasmine, 361
 Node, 298, 509
 frameworki BDD, 361
 funkcja
 getStockPrice(), 458
 strzałkowa, 454
 super(), 468
 funkcje, 486
 czyste, 410
 generatora, 457

G

generowanie
 komponentów, 57
 paczek, 35
 projektu, 29
 głębokie linkowanie, 85
 gniazda
 routera, 67, 107
 WebSocket, 333, 336

H

hermetyzacja widoków, 218

I

implementowanie
 interfejsu, 496
 komponentu produktu, 238
 komunikacji komponentów, 195
 RWD, 177

- strażnika
 - CanActivate, 99
 - CanDeactivate, 101
 - Resolve, 103
- wzorca Mediator, 203
- importy, 181
- interfejs użytkownika wielokomponentowy, 207
- inferencja typów, 198, 485
- informacje
 - o pogodzie, 162
 - o trasach, 72
- instalowanie Karmy, 366
- instancja FormGroup, 253
- integracja gniazd Websocket, 341
- interfejs, 494
 - CanDeactivate, 102
 - CLI Angulara, 29, 445
 - Forms API, 156, 248
 - History API, 70
 - Route Angulara, 74
 - użytkownika
 - komponenty, 28
 - ValidatorFn, 266
 - wiersza poleceń, CLI, 29
- interfejsy
 - API formularzy, 246
 - strażników, 98
- interpolacja łańcuchów znaków, 450
- iterowanie, 463

J

- Jasmine, 361
 - skrypty testowe, 363
 - uruchamianie skryptów, 365
 - zestaw testowy, 362
- język TypeScript, 298, 481
- JIT, just-in-time, 33

K

- Karma, 365
 - instalowanie, 366
 - plik konfiguracyjny, 369
- karuzela, 61, 148
- klasa, 465, 489
 - ActivatedRoute, 166
 - BidServer, 349

- DataResolver, 104
- FormArray, 253
- FormBuilder, 263
- FormControl, 252
- FormGroup, 253
- InjectionToken, 133
- klient, 325
 - do wysyłania żądań post, 316
- kolejka przepływu sterowania, 392
- kompilacja
 - AOT, 33
 - JIT, 33
- kompilacje
 - produkcyjne, 31
 - programistyczne, 31
- kompilator Closure, 445
- komponent, 25, 39, 138
 - Angular Material, 142
 - CategoriesComponent, 312
 - HomeComponent, 70, 89, 149
 - NavbarComponent, 143
 - ProductComponent, 352
 - ProductDetailComponent, 73, 94, 101, 352
 - ProductDetailComponentParam, 78
 - ProductItemComponent, 88, 148
 - SearchComponent, 146, 414
 - StarsComponent, 91
- komponenty
 - cykl życia, 223, 227
 - formularza wyszukiwania, 283
 - generowanie, 57
 - implementowanie komunikacji, 195
 - interfejsu użytkownika, 28
 - karuzeli, 61
 - kategorii, 290
 - komunikacja, 195
 - kontenerowe, 409
 - modyfikacja wyglądu, 143
 - nadrzędne, 203
 - najwyższego poziomu, 183
 - paska nawigacyjnego, 59
 - potomne, 213
 - prezentacyjne, 409
 - produktu, 238
 - stopki, 61
 - strony głównej, 64
 - szczegółów produktu, 241
 - wyszukiwania, 60, 287

- komponenty
 - właściwości wejściowe, 196
 - właściwości wyjściowe, 196
- komunikacja
 - komponentów, 195
 - master-detail, 166
 - usługi z serwerem, 336
- komunikaty od klienta, 350
- konfiguracja
 - modułu testowego, 371
 - projektów, 52
 - routera, 97
 - serwera proxy, 308
 - tras, 64, 72, 94, 108, 193
- konstruktory, 467
- konsumowanie danych JSON, 305
- kontener, 195
- kontrolki formularza, 253
 - dotknięte, 276
 - niedotknięte, 276
 - sprawdzanie prawidłowości, 276
 - status, 276
- kreator akcji, 414
- kryteria wyszukiwania, 394
- krzywa uczenia się, 430

L

- leniwe ładowanie, 109
- literały szablonów, 450
- lokalizacja, 69
- luźne powiązania, 117

Ł

- ładowarki wstępne, 113
- łańcuchy znaków wieloliniowe, 451
- łączenie
 - obietnic, 474
 - operatorów, 523

M

- magazyn, 400, 404
 - ngrx, 420
- matcher, 362
- mechanizm renderujący Ivy, 445
- Mediator, 203, 208

- menedżer pakietów
 - npm, 509, 514
 - Yarn, 514
- metadane, 42
- metoda, 492
 - constructor, 349
 - forEach(), 463
 - navigate(), 76
 - ngAfterContentChecked(), 229
 - ngAfterContentInit(), 229
 - ngAfterViewChecked(), 229
 - ngAfterViewInit(), 229
 - ngDoCheck(), 229
 - ngOnChanges(), 229
 - ngOnDestroy(), 229
 - ngOnInit(), 229, 230
- metody
 - pobierające, 470
 - ustawiające, 470
- model formularza, 252
- moduł, 26, 46, 479
 - FlexLayoutModule, 172
 - HttpClientModule, 126
 - produktu, 236
 - strony głównej, 188

- moduły
 - ES6, 478
 - funkcyjne, 46
 - leniwie ładowane, 109
- modyfikacja
 - HttpRequest, 320
 - interfejsu użytkownika, 146
 - wyglądu komponentu, 143
- modyfikator readonly, 501
- modyfikatory dostępu, 490
- monitorowanie stanu routera, 424
- motyw Angular Material, 180
- mutacja stanu, 429

N

- nadpisywanie stylów, 63
- narzędzie
 - TSLint, 507
 - WebDriverJS, 392
- nawigacja
 - bloki konstrukcyjne, 71
 - metoda navigate(), 76

- oparta
 - na interfejsie, 70
 - na znaku kratki, 70
 - w efektach, 427
- ngAuction
 - dodanie
 - nawigacji, 86
 - paska narzędzi, 183
 - widoku produktu, 236
 - formularz wyszukiwania, 282
 - generowanie
 - komponentów, 57
 - nowej aplikacji, 180
 - kod klienta, 351
 - komponent, 68
 - aplikacji, 57
 - karuzeli, 61
 - paska nawigacyjnego, 59
 - stopki, 61
 - strony głównej, 64
 - wyszukiwania, 60
 - konfigurowanie tras, 193
 - moduł strony głównej, 188
 - niestandardowy motyw, 180
 - obsługa stanu routera, 433
 - przegląd kodu serwera, 344
 - przepisanie kodu aplikacji, 177
 - refaktoryzacja
 - kodu, 288
 - modułu strony głównej, 289
 - stosowanie ngrx, 431
 - testy e2e, 392
 - tryb programistyczny, 344
 - tworzenie aplikacji, 55
 - uruchamianie aplikacji, 193, 344
 - usługa ProductService, 87, 186
 - wstępna konfiguracja, 55
- ngrx, 403, 420
 - a usługi Angulara, 428
 - monitorowanie stanu, 420, 424
 - przepływ danych, 411
 - refaktoryzacja aplikacji, 412
 - testy jednostkowe reduktorów, 442
 - używanie, 427
- notacja kropkowa, 132
- numeracja wersji, 513

O

- obiekt
 - HttpClient, 126
 - RouterStateSnapshot, 425
 - Subject RxJS, 528
 - ValidationErrors, 269
- obietnice ES6, 472
- obraz, 148
- obserwator, 518, 519
- obsługa
 - błędów, 536
 - 404, 77
 - serwera WebSocket, 350
 - formularzy reaktywnych, 252
 - komunikatów od klienta, 350
 - protokołu WebSocket, 341
 - stanu routera, 433
 - tras, 72
 - zamykanych połączeń, 350
 - zdarzenia, 152
 - connection, 349
 - keyup, 152
 - zdarzeń obserwowalnych, 156
 - zmian parametrów, 352
 - zadań post, 315
- odczytywanie pliku JSON, 295
- określanie zależności projektu, 510
- opakowywanie strumienia obserwowalnego, 334
- opcje polecenia ng new, 32
- operator, 519
 - catchError, 536
 - flatMap, 530
 - reszty, 455
 - rozwijania, 456
 - switchMap, 158, 534
- operatory
 - potokowe, 523
 - RxJS, 521

P

- paczki, 34
- pakiet
 - CDK, 177, 445
 - npm, 509

- panel narzędzi Redux DevTools, 425
- parametry
 - domyślne, 487
 - opcjonalne, 451, 488
- pasek
 - narzędzi, 141, 143, 183
 - nawigacyjny, 59
- pełny duplex, 328
- pętla
 - for-in, 464
 - for-of, 464
- plik
 - _theme.scss, 182
 - actions.ts, 413
 - app.component.asyncpipe.ts, 310
 - app.component.html, 184, 288
 - app.component.scss, 185
 - app.component.spec.ts, 374, 381
 - app.component.ts, 74, 77, 105, 201, 204, 205, 215, 217, 221, 231, 296, 318, 323, 325, 380, 406, 419, 426
 - app.e2e-spec.ts, 387
 - app.module.ts, 75, 110, 184, 405, 419, 433
 - app.po.ts, 386
 - app.routing.ts, 74, 237
 - app-routing.module.ts, 94
 - asyncpipe/app.component.ts, 162
 - asyncpipe-products/app.component.ts, 164
 - asyncpipe-products/product.service.ts, 163
 - async-validator/app.component.ts, 280
 - bid.service.ts, 356
 - categories.component.html, 290
 - categories.component.ts, 290, 437
 - child.component.ts, 214, 217, 221, 232, 407
 - child.component-docheck.ts, 233
 - client/package.json, 314
 - console.logging.service.ts, 321
 - counter.spec.ts, 364
 - counter.ts, 363
 - data.resolver.ts, 104
 - data/product.json, 377
 - data/products.json, 295
 - db-auction.ts, 347
 - ebay.component.ts, 212, 418
 - effects.ts, 416, 427
 - environmient.ts, 343
 - environment.prod.ts, 343
 - flex-layout/app.component.ts, 172
 - formcontrol/app.component.ts, 156
 - fromevent/app.component.ts, 155
 - home.component.css, 149
 - home.component.html, 90, 190
 - home.component.scss, 191
 - home.component.ts, 90, 136, 189
 - home.module.ts, 189, 434
 - home.po.ts, 388
 - home/store/actions/products.ts, 436
 - home/store/effects/products.ts, 441
 - home/store/reducers/index.ts, 439
 - home/store/reducers/products.spec.ts, 443
 - home/store/reducers/products.ts, 437
 - index.ts, 439
 - input/app.component.ts, 197
 - iprice.quote.ts, 200
 - istock.ts, 204
 - karma.conf.js, 369
 - logging.interceptor.service.ts, 322
 - logging.service.ts, 321
 - login.component.ts, 100, 388
 - login.e2e-spec.ts, 389
 - login.guard.ts, 99
 - login.po.ts, 389
 - luxury.component.ts, 110, 135
 - luxury.module.ts, 110, 135
 - luxury.service.ts, 135
 - master-detail/app.component.ts, 166
 - master-detail/product.detail
 - ↪component.ts, 167
 - my-express-server.ts, 300
 - observable.service.ts, 334
 - observable-media/app.component.ts, 175
 - observableservice/app.component.ts, 335
 - order.component.ts, 197, 206
 - package.json, 299, 510, 512
 - price.quoter.component.ts, 200, 205
 - product.component.html, 240
 - product.component.ts, 123, 238
 - product.description.component.ts, 84
 - product.detail.component.ts, 83
 - product.factory.ts, 131
 - product.module.ts, 237, 238, 242, 243
 - product.service.spec.ts, 376, 378
 - product.service.ts, 87, 187, 416
 - product-detail.componenten.css, 96
 - product-detail.component.html, 95, 241
 - product-detail.component.ts, 95, 241, 353

- product-item.component.css, 93
 - product-item.component.html, 89, 148
 - product-item.component.ts, 89
 - product-service.ts, 124
 - product-suggestion.component.html, 242
 - protractor.conf.js, 384
 - proxy-conf.json, 308
 - reducer.ts, 405
 - reducers.ts, 415
 - rest-auction.ts, 346
 - restclient/app.component.ts, 306
 - rest-server.ts, 302, 304
 - rest-server-angular.ts, 304
 - rest-server-angular-post.ts, 316
 - search.component.css, 147
 - search.component.html, 146
 - search.component.ts, 212, 414
 - search.e2e-spec.ts, 397
 - search.po.ts, 396
 - search-form.component.ts, 285
 - search-form.component.html, 284
 - search-results.component.html, 288
 - search-results.component.ts, 287
 - selectors.ts, 417
 - seller.info.component.ts, 84
 - serializer.ts, 426
 - server/two-way-websocket-server.ts, 339
 - plik shipping.component.ts, 137
 - plik shipping.module.ts, 136
 - plik simple-websocket-client.html, 332
 - plik simple-websocket-server.ts, 331
 - plik src/data/products.json, 186
 - plik stars.component.css, 93
 - plik stars.component.html, 93
 - stars.component.ts, 92
 - state.service.ts, 211
 - styles.scss, 182
 - tsconfig.json, 300, 484
 - union.ts, 505
 - weather/app.component.ts, 159
 - ws-auction.ts, 348
 - wsservice/app.component.ts, 338
 - wsservice/websocket.service.ts, 337
 - pliki
 - częstkowe, 181
 - definicji typów, 505
 - JSON, 295
 - pola
 - czyste i brudne, 277
 - oczekujące, 277
 - pole
 - ssn, 249
 - username, 249
 - polecenie
 - ng generate, 40
 - ng new, 32
 - ng serve, 111
 - ng update, 446
 - port, 332
 - potok AsyncPipe, 161, 309
 - potoki, 45
 - półdupleks, 328
 - prefiksy selektora, 41
 - profilowanie wykrywania zmian, 227
 - programowanie
 - asynchroniczne, 517
 - reaktywne, 151
 - synchroniczne, 517
 - protokół
 - HTTP, 293
 - WebSocket, 327, 341, 357
 - Protractor, 382, 392
 - przechowywanie stanu aplikacji, 402
 - przechwytywacze HTTP, 319
 - przechwytywanie
 - błędów, 537
 - zmian, 230, 233
 - przekazywanie parametrów do trasy, 77-80
 - przekształcanie zdarzeń DOM, 154
 - przełączanie wstrzykiwaczy, 127
 - przepływ pracy, 36
 - logowania, 98
 - przesyłanie danych na serwer, 315
 - przetwarzanie asynchroniczne, 471
 - przyrostek .sm, 173
- ## R
- reduktory, 402, 404, 415
 - dla produktów, 437
 - ngrx, 442
 - Redux, 400
 - przepływ danych, 401
 - refaktoryzacja, 412

- rejestracja użytkownika, 274
 - style komponentu, 274
 - szablon, 274
- renderowanie, 92, 174
- responsywne projektowanie stron, RWD, 169
- router, 67
 - obserwowalne właściwości, 165
 - zaawansowana konfiguracja, 97
- routing, 68
- rozgłaszanie, 529
- rozszerzenie DevTools, 420
- RWD, responsive web design, 169
- RxJS, 517
 - obiekt Subject, 528
 - Obserwator, 519
 - operator, 519, 521
 - catchError, 536
 - flatMap, 530
 - switchMap, 534
 - operatory potokowe, 523
 - Strumień obserwowalny, 519
 - Subskrybent, 519
- rzutowanie
 - na obszary szablonu, 220
 - szablonów, 216

S

- Schematics, 446
- SCSS, 181
- selektor, 41, 410, 417
- semantyczna numeracja wersji, 513
- serwer
 - do obsługi żądań post, 315
 - emulacja błędów, 321
 - HTTP, 346
 - Node, 303, 330
 - obsługa protokołu WebSocket, 341
 - wysyłanie danych, 330
 - proxy, 308
 - przesyłanie danych, 315
 - WebSocket, 341, 348
 - obsługa błędów, 350
 - wdrażanie aplikacji, 312
 - zasoby statyczne, 303
- serwowanie danych, 301
- składnia
 - alternatywna DI, 125
 - SCSS, 181

- składowe
 - instancji, 492
 - statyczne, 492
- skrypt obsługi danych, 347
- skrypty npm, 312
- słowo kluczowe
 - async, 476
 - await, 476
 - export, 479
 - implements, 495
 - import, 479
 - let, 450
 - super, 468
 - that, 452
 - this, 448, 452
 - var, 448
- SPA, single-page application, 67
- specyfikacja ECMAScript, 447
- stan
 - aplikacji, 399, 443
 - bieżący, 400
 - routera, 424, 433
- stopka, 61
- strategie
 - lokalizacji, 69
 - wykrywania zmian, 225
- strażnik
 - CanActivate, 99
 - CanDeactivate, 101
 - Resolve, 103
- strona główna, 64
- strony responsywne, 169
- struktura
 - komponentu, 27
 - projektu, 38
- strumień obserwowalny, 154, 165, 352, 518, 526
 - zagnieżdżone, 531
- strzeżenie tras, 98
- styl kodowania, 507
 - imperatywny, 518
 - reaktywny, 518
- style pola tekstowego, 272
- subskrybent, 519
- subskrybowanie
 - strumieni, 309
 - żądania HTTP, 532
- sygnatura reduktora, 402

systemy typów, 499

szablon

HTML, 26

komponentu

AppComponent, 58, 100

karuzeli, 62

paska nawigacyjnego, 59

stopki, 61

strony głównej, 64

wyszukiwania, 60

Ś

śledzenie zdarzeń, 324

T

tester Karma, 365

testowalność, 119

testowanie

aplikacji, 359

interfejsu REST-owego, 317

komponentów, 372

komponentów routingu, 379

przepływu pracy, 392

strony logowania, 387

usług, 376

w wielu przeglądarkach, 370

testy

end-to-end, 360, 382

generowane przez CLI Angulara, 386

jednostkowe, 359, 360

jednostkowe reduktorów ngrx, 442

token, 118

transkompilacja, 481

trasy, 64

podrzedne, 81

ponowne ładowanie, 106

z klasą DataResolver, 104

ze strażnikiem, 100

tryb

produkcyjny, 235

programistyczny, 344

tryby hermetyzacji widoków, 218

TSLint, 507

tworzenie

aplikacji ngAuction, 55

klienta, 316

modułu strony, 188

niestandardowego motywu, 180

paczek, 34

paska narzędzi, 141

serwera WWW, 298

strumieni obserwowalnych, 520

układu stron, 169

usługi produktowej, 186

zakresu bloku, 450

typ unii, 504

TypeScript, 481

dekoratory, 502

dziedziczenie, 493

funkcje, 486

instalacja, 482

interfejsy, 494

klasy, 489

metody, 492

modyfikator readonly, 501

modyfikatory dostępu, 490

nadzbior JavaScriptu, 484

parametry domyślne, 487

parametry opcjonalne, 488

tworzenie serwera WWW, 298

typy

niestandardowe, 494

opcjonalne, 485

sparametryzowane, 497, 500

U

układ

kolumnowy elementów, 173

stron, 169

unie, 504

uruchamianie

aplikacji, 31

bez AOT, 35

z AOT, 34

serwerów

HTTP, 345

WebSocket, 345

skryptów Jasmine, 365

urządzenia mobilne, 28

usługa, 26, 42

HttpClient, 125, 294, 310, 325

ObservableMedia, 170, 175

ProductService, 87, 118, 311

- usługa
 - produktowa, 186
 - WebSocketService, 337
- użycie
 - FormArray, 258
 - gniazd WebSocket, 333
 - komponentów, 138
 - ngrx, 427
 - obietnicy, 473

W

- walidacja
 - formularzy, 265
 - grupy kontrolek, 273
 - rozpoczynanie, 269
- walidator, 253
 - required, 266
 - z opcją blur, 270
- walidatory
 - asynchroniczne, 279
 - niestandardowe, 270, 281
 - wbudowane, 266, 268
- wartości
 - domyślne, 451
 - mutowalne, 231
- wdrażanie aplikacji, 312
- Webpack, 31, 314
- WebSocket, 327, 333, 336, 341, 345, 350
- wiązanie
 - danych, 50
 - dwukierunkowe, 51
 - jednokierunkowe, 51
 - obiektu NgForm, 247
 - właściwości i zdarzeń, 50
 - z innerHTML, 222
- widok produktu, 236
- wielokrotne wykorzystywanie, 117
- właściwości
 - wejściowe, 197
 - wyjściowe, 199
- właściwość
 - innerHTML, 222
 - provideIn, 126
 - title, 375
 - touched, 276
 - useFactory, 131
 - useValue, 131
- wstępne ładowanie, 113
- wstrzykiwacze, 119
 - przełączanie, 127
- wstrzykiwalne usługi, 208, 210
- wstrzykiwanie
 - obiektu
 - HttpClient, 125, 126, 310
 - Router, 81
 - usługi produktowej, 122
 - zależności, DI, 115
 - podczas testowania, 120
 - w aplikacji zmodularyzowanej, 134
- wykrywanie zmian, 223, 235
 - profilowanie, 227
 - strategie, 225
- wymiana komunikatów, 341
- wynoszenie deklaracji zmiennych, 448
- wyrażenia
 - funkcji strzałkowych, 452
 - lambda, 452
- wysyłanie
 - błędów, 335
 - zdarzeń, 335
 - zadania
 - GET, 324
 - POST, 316
- wyszukiwanie, 394
 - produktów, 392
- wywołania zwrotne, 229, 471
- wywołanie getPrice(), 459
- wzorzec projektowy
 - Mediator, 203, 204
 - Obiekt Strony, 386
 - Wstrzykiwanie Zależności, 116

Y

Yarn, 514

Z

- zaczep
 - ngDoCheck, 233
 - ngOnChanges, 230
- zaczepy cyklu życia, 235
- zagnieżdżanie, 180, 181

zakładka

 Action, 422

 Diff, 423

 State, 422

zakres globalny, 479

zanieczyszczenie reduktora, 410

zapytania o media, 175

zarządzanie stanem, 434

zasoby statyczne, 303

zastępowanie karuzeli, 148

zdarzenia, 152

 DOM, 152, 154

 niestandardowe, 199

 obserwowalne, 156

 postępu, 323

 routera, 85

zdarzenie

 connection, 349

 keyup, 152

zmienne, 180, 181

 statyczne, 469

Ż

żądanie HTTP, 158

 GET, 294, 298, 324

 POST, 315

Notatki

PROGRAM PARTNERSKI

— GRUPY HELION —



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion 

Angular jest znakomitym frameworkiem wybieranym przez programistów, którym zależy na szybkiej, wydajnej i satysfakcjonującej pracy. Umożliwia sprawne tworzenie zarówno lekkich klientów internetowych, jak i w pełni funkcjonalnych aplikacji. Angular pozwala na wykorzystywanie TypeScriptu, który w porównaniu z JavaScriptem o wiele lepiej spisuje się jako język programowania profesjonalnych aplikacji internetowych. Ten framework zapewnia również możliwość korzystania z wielu nowoczesnych bibliotek, dzięki którym w łatwy sposób można tworzyć i rozwijać zaawansowane, atrakcyjne aplikacje.

Ta książka jest drugim, przejrzanym i uzupełnionym wydaniem

praktycznego podręcznika, znakomicie ułatwiającego naukę tworzenia aplikacji. Wyjaśniono tu tak istotne zagadnienia, jak zarządzanie stanem, kontrola wprowadzanych danych, budowa formularzy i komunikacja z serwerem. Prezentowane treści uzupełniono praktycznymi przykładami kodu, składającego się na w pełni funkcjonalną aplikację internetową. Pokazano, w jaki sposób wstrzykiwać zależności oraz projektować reaktywne interfejsy użytkownika i komunikację między komponentami aplikacji. Nie zabrakło opisu przydatnych bibliotek, takich jak: RxJS, NgRx czy Flex Layout.

W tej książce między innymi:

- wprowadzenie do architektury Angulara i sposób pracy z frameworkiem
- praca z obserwowalnymi strumieniami danych
- podstawowe i zaawansowane funkcje routera Angulara
- formularze reaktywne i walidacja danych
- testowanie aplikacji, w tym testy jednostkowe i testy przepływu pracy

YAKOV FAIN jest konsultantem i programistą, mistrzem programowania w Javie. Udziela porad dotyczących projektowania za pomocą frameworka Angular. Autor wielu książek na temat rozwoju oprogramowania.

ANTON MOISEEV od ponad dziesięciu lat tworzy aplikacje z wykorzystaniem technologii Java i .NET. Specjalizuje się we wdrażaniu najlepszych praktyk płynnej współpracy front-endu z back-endem. Szkoli developerów pracujących z frameworkiem Angular i AngularJS.

Angular: szybki, wydajny, bezpieczny!

Helion  helion.pl HELION SA ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	<i>Sprawdź nasze szkolenia!</i> SZKOLENIA AKADEMIA IT & BUSINESS WWW.SZKOLENIA.HELION.PL	KOD KORZYŚCI Sięgnij po więcej! ►  ISBN 978-83-283-5666-5  9 788328 356665 INFORMATYKA W NAJLEPSZYM WYDANIU Cena: 89,00 zł
---	--	---