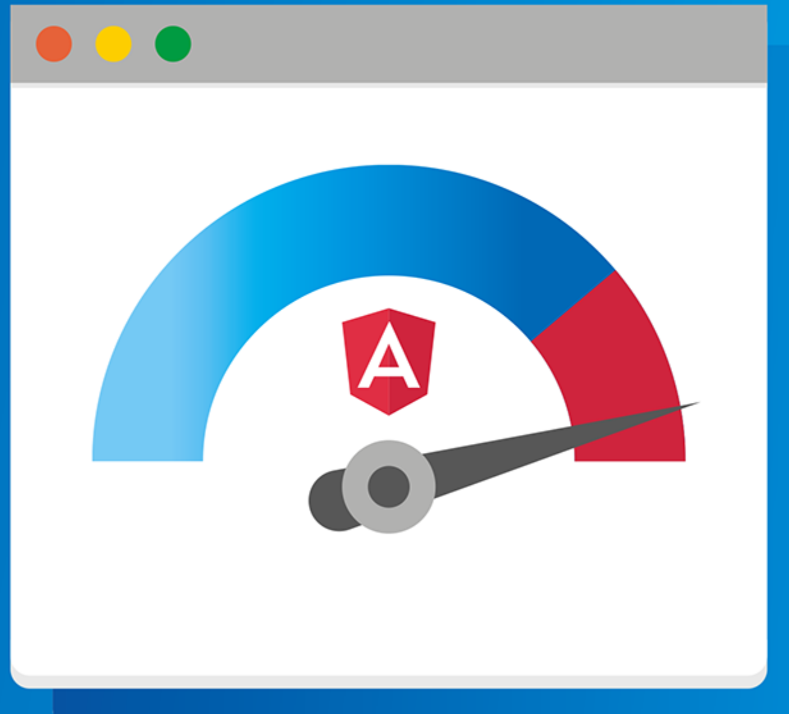


Jeremy Wilken

ANGULAR

W AKCJI

Tytuł oryginału: Angular in Action

Tłumaczenie: Lech Lachowski
Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

ISBN: 978-83-283-4798-4

Original edition copyright © 2018 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 jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

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

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

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

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

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

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści

Przedmowa 11

Podziękowania 13

O książce 15

O autorze 19

Rozdział 1. Angular — nowoczesna platforma internetowa 21

- **1.1. Dlaczego warto wybrać Angular?** 22
- **1.2. Czego się nauczysz z tej książki** 23
- **1.3. Podróż od AngularJS do Angulara** 24
- **1.4. Angular — platforma, a nie framework** 26
  - *1.4.1. CLI Angulara* 26
  - *1.4.2. Renderowanie serwerowe i kompilator* 27
  - *1.4.3. Możliwości mobilne i desktopowe* 29
  - *1.4.4. Biblioteki interfejsu użytkownika* 30
- **1.5. Architektura komponentowa** 32
  - *1.5.1. Najważniejsze cechy komponentów* 34
  - *1.5.2. Shadow DOM* 36
  - *1.5.3. Szablony* 37
  - *1.5.4. Moduły JavaScriptu* 39
- **1.6. Nowoczesny JavaScript i Angular** 40
  - *1.6.1. Strumienie obserwowalne* 43
- **1.7. TypeScript i Angular** 44
- **Podsumowanie** 46

Rozdział 2. Twoja pierwsza aplikacja Angular 47

- **2.1. Przegląd projektu z tego rozdziału** 48
- **2.2. Konfigurowanie projektu** 50
- **2.3. Rusztowanie bazowej aplikacji** 51
- **2.4. Jak Angular renderuje aplikację bazową** 53
  - *2.4.1. Komponent App* 53
  - *2.4.2. Moduł App* 54
  - *2.4.3. Początkowe ładowanie aplikacji* 55
- **2.5. Budowanie usług** 57
- **2.6. Tworzenie pierwszego komponentu** 60
- **2.7. Komponenty wykorzystujące komponenty i usługi** 66

Kup ksi k Pole ksi k

2.8. Komponenty z formularzami i ze zdarzeniami 69
2.9. Routing aplikacji 73
Podsumowanie 75

Rozdział 3. Najważniejsze elementy aplikacji *77*

3.1. Encje w Angularze 79
3.1.1. Moduły *79*
3.1.2. Komponenty *81*
3.1.3. Dyrektywy *82*
3.1.4. Potoki *85*
3.1.5. Usługi *86*
3.2. Jak Angular zaczyna renderować aplikację 87
3.3. Rodzaje kompilatorów 89
3.4. Wstrzykiwanie zależności 89
3.5. Wykrywanie zmian 90
3.6. Wyrażenia szablonów i wiązania 91
3.6.1. Interpolacja *93*
3.6.2. Wiązania właściwości *94*
3.6.3. Specjalne wiązania właściwości *95*
3.6.4. Wiązanie atrybutów *96*
3.6.5. Wiązanie zdarzeń *97*
Podsumowanie 98

Rozdział 4. Podstawy komponentów *101*

4.1. Konfigurowanie przykładu 102
4.1.1. Przygotowanie kodu *103*
4.2. Kompozycja i cykl życia komponentu 104
4.2.1. Cykl życia komponentu *106*
4.2.2. Zaczepy cyklu życia *107*
4.2.3. Zagnieżdżanie komponentów *109*
4.3. Rodzaje komponentów 110
4.4. Tworzenie komponentu Data 113
4.5. Używanie wejść z komponentami 116
4.5.1. Podstawy wejść *117*
4.5.2. Przechwytywanie wejść *120*
4.6. Rzutowanie zawartości 122
Podsumowanie 128

Rozdział 5. Zaawansowane aspekty komponentów *129*

5.1. Wykrywanie zmian i optymalizacje 130
5.2. Komunikacja między komponentami 134
5.2.1. Zdarzenia wyjściowe i zmienne szablonów *135*
5.2.2. Odwoływanie się do komponentów za pomocą ViewChild *137*
5.3. Stylizacja komponentów i tryby hermetyzacji 138
5.3.1. Dodawanie stylów do komponentu *139*
5.3.2. Tryby hermetyzacji *141*

Kup ksi k Pole ksi k

5.4. Dynamiczne renderowanie komponentów 145
5.4.1. Używanie okna modalnego ng-bootstrap dla komponentów dynamicznych 145
5.4.2. Dynamiczne tworzenie komponentu i jego renderowanie 149
Podsumowanie 153

Rozdział 6. Usługi 155

6.1. Konfigurowanie przykładu 156
6.1.1. Pobieranie kodu 158
6.1.2. Przykładowe dane 158
6.2. Tworzenie usług Angulara 159
6.3. Wstrzykiwanie zależności i drzewa wstrzykiwaczy 164
6.4. Usługi bez wstrzykiwania zależności 168
6.5. Korzystanie z usługi HttpClient 169
6.5.1. Interfejs HttpInterceptor 173
6.6. Usługi pomocnicze 176
6.7. Usługi udostępniania 180
6.8. Usługi dodatkowe 184
Podsumowanie 186

Rozdział 7. Routing 187

7.1. Konfigurowanie przykładu 188
7.2. Definiowanie tras i konfiguracja routera 190
7.3. Moduły funkcyjne i routing 193
7.4. Parametry trasy 194
7.4.1. Tworzenie linków w szablonach za pomocą dyrektywy routerLink 195
7.4.2. Uzyskiwanie dostępu do parametrów trasy w komponencie 196
7.5. Trasy podrzędne 198
7.6. Trasy drugorzędne 201
7.6.1. Definiowanie trasy drugorzędnej 202
7.6.2. Nawigacja między trasami drugorzędnymi 204
7.6.3. Zamykanie trasy drugorzędnej i routing programowy 205
7.7. Zabezpieczanie tras w celu ograniczenia dostępu 205
7.8. Leniwe ładowanie 211
7.9. Najlepsze praktyki dotyczące routingu 214
Podsumowanie 215

Rozdział 8. Budowanie niestandardowych dyrektyw i potoków 217

8.1. Konfigurowanie przykładu 219
8.2. Tworzenie niestandardowych dyrektyw 219
8.2.1. Tworzenie dyrektywy atrybutów 221
8.2.2. Modyfikowanie komponentu za pomocą dyrektywy ze zdarzeniami 223
8.2.3. Tworzenie dyrektywy strukturalnej 226

Kup ksi k Pole ksi k

8.3. Tworzenie niestandardowych potoków 229
8.3.1. Tworzenie czystego potoku 230
8.3.2. Tworzenie nieczystego potoku 232
Podsumowanie 236

Rozdział 9. Formularze 237

9.1. Konfigurowanie przykładu 238
9.1.1. Wstępny przegląd aplikacji 240
9.2. Formularze oparte na szablonach 241
9.2.1. Wiązanie danych modelu z wejściami za pomocą dyrektywy NgModel 241
9.2.2. Walidacja kontrolek formularza za pomocą dyrektywy NgModel 243
9.2.3. Niestandardowa walidacja za pomocą dyrektyw 246
9.2.4. Obsługa zdarzeń przesyłania lub zdarzeń anulowania 249
9.3. Formularze reaktywne 251
9.3.1. Definiowanie formularza 252
9.3.2. Implementowanie szablonu 254
9.3.3. Obserwowanie zmian 255
9.3.4. Niestandardowe walidatory z formularzami reaktywnymi 256
9.3.5. Obsługa zdarzeń przesyłania lub zdarzeń anulowania 260
9.3.6. Które podejście do formularzy jest lepsze? 264
9.4. Niestandardowe kontrolki formularzy 265
Podsumowanie 271

Rozdział 10. Testowanie aplikacji 273

10.1. Narzędzia testowe i konfiguracja przykładu 274
10.1.1. Narzędzia do testowania 275
10.2. Testy jednostkowe 276
10.2.1. Anatomia testów jednostkowych 276
10.2.2. Testowanie potoków 277
10.2.3. Testowanie usług, stuby i symulowanie żądań HTTP 279
10.2.4. Testowanie komponentów i korzystanie z modułów testujących 285
10.2.5. Testowanie dyrektyw 293
10.3. Testy e2e 297
10.4. Dodatkowe strategie testowania 302
10.4.1. Ile testów wystarczy? 303
10.4.2. Kiedy mam pisać testy? 304
10.4.3. Co mam napisać, e2e czy testy jednostkowe? 304
10.4.4. A co, jeśli nie mam czasu na pisanie testów? 305
10.4.5. A co z innymi rodzajami testów? 305
Podsumowanie 306

Rozdział 11. Angular w środowisku produkcyjnym 309

11.1. Kompilowanie Angulara dla środowiska produkcyjnego 310
11.1.1. Kompilacja produkcyjna 310
11.1.2. Optymalizacja dla przeglądarek docelowych 311
11.1.3. Progresywne aplikacje internetowe 312
11.1.4. Internacjonalizacja (i18n) 312

Kup ksi k Pole ksi k

11.1.5. Używanie alternatywnych narzędzi kompilacji 313
11.1.6. Renderowanie po stronie serwera lub renderowanie wstępne 314
11.1.7. Potoki kompilacji 314
11.2. Wybór architektury Angulara 315
11.2.1. Leniwe ładowanie tras 315
11.2.2. Ograniczanie zewnętrznych zależności 316
11.2.3. Bycie na bieżąco 319
11.3. Wdrożenie 319
Podsumowanie 321

Dodatek A Aktualizacja z AngularJS do Angulara 323

Dodatek B Komunikacja między komponentami Angular 329

Skorowidz 331

2 Twoja pierwsza aplikacja Angular

W tym rozdziale:

- komponenty frameworka Angular i sposób, w jaki tworzą bazę aplikacji;
- definiowanie różnych typów komponentów przy użyciu dekoratorów;
- wykorzystywanie usług do współdzielenia danych przez całą aplikację;
- konfigurowanie routingu do wyświetlania różnych stron.

W tym rozdziale zbudujesz całą aplikację Angular, zaczynając od zera, a w trakcie pracy będziesz poznawał podstawowe koncepcje tego frameworka. Zobaczysz w akcji kilka funkcjonalności TypeScriptu oraz poznasz nowe i nadchodzące funkcjonalności JavaScriptu.

Ten projekt będzie zwięzły i prosty, ale i tak będzie reprezentatywny dla wielu funkcjonalności, których będziesz używać w typowych aplikacjach. Aplikacja, którą utworzysz, będzie programem do śledzenia notowań akcji, z danymi pochodzącymi z Yahoo! Finance. Będzie w stanie pobierać aktualne ceny akcji, dodawać lub usuwać akcje z listy i dostosowywać ekran na podstawie zysków lub strat z bieżącego dnia.

W tym rozdziale zbudujemy tę aplikację kawałek po kawałku. Skoncentrujemy się na przejściu krok po kroku przez przykładową aplikację z wystarczającą liczbą szczegółów, żebyś mógł zrozumieć różne elementy i złożoność tego rozdziału.

Kup ksi k Pole ksi k

- **Początkowe ładowanie aplikacji.** Aby uruchomić aplikację, użyjemy funkcji **początkowego ładowania** (ang. *bootstrap*) do zainicjowania elementów po ich załadowaniu. Dzieje się to raz podczas cyklu życia aplikacji, a my załadujemy komponent App.
- **Tworzenie komponentów.** W Angularze wszystko kręci się wokół **komponentów**, a my utworzymy kilka komponentów do różnych celów. Dowiemy się, jak są budowane i zagnieżdżane, tworząc złożone aplikacje.
- **Tworzenie usług i korzystanie z HttpClient.** Aby zapewnić możliwość ponownego używania kodu, zhermetyzujemy trochę logiki, która pomoże nam utworzyć listę akcji. Logikę umieścimy w odpowiedniej **usłudze**. Dodatkowo korzystać będziemy z usługi HttpClient frameworka Angular w celu pobrania danych dotyczących notowań akcji.
- **Używanie potoków i dyrektyw w szablonach.** Za pomocą **potoków** (ang. *pipes*) możemy przekształcać dane z jednego formatu na inny podczas wyświetlania, formatując na przykład znacznik czasu na lokalny format daty. **Dyrektywy** (ang. *directives*) są przydatnymi narzędziami do modyfikowania zachowania elementów DOM wewnątrz szablonu, umożliwiającymi na przykład powtarzanie pewnych fragmentów lub warunkowe pokazywanie elementów.
- **Konfigurowanie routingu.** Większość aplikacji wymaga umożliwienia użytkownikom nawigacji po aplikacji, a wykorzystując router, możemy zobaczyć, jak routować między różnymi komponentami.

Używając ograniczonej ilości kodu, będziesz w stanie utworzyć solidną aplikację, która wykonuje wiele skomplikowanych zadań. Kolejne rozdziały koncentrują się szczegółowo na poszczególnych funkcjach, abyś mógł uzyskać pełniejszy obraz wszystkiego, co Angular ma do zaoferowania.

Powinieneś zapoznać się z ES2015 i nowszymi możliwościami języka JavaScript. Nie będę się zajmował szczegółami dotyczącymi nowszych konstrukcji tego języka, takich jak importy lub klasy. Najlepiej, żebyś poświęcił trochę czasu na zapoznanie się ze szczegółami zamieszczonymi na stronie Mozilla Developer Network (*https://developer.mozilla.org/pl/docs/Web/JavaScript*) lub poszukał jakiejś książki na ten temat.

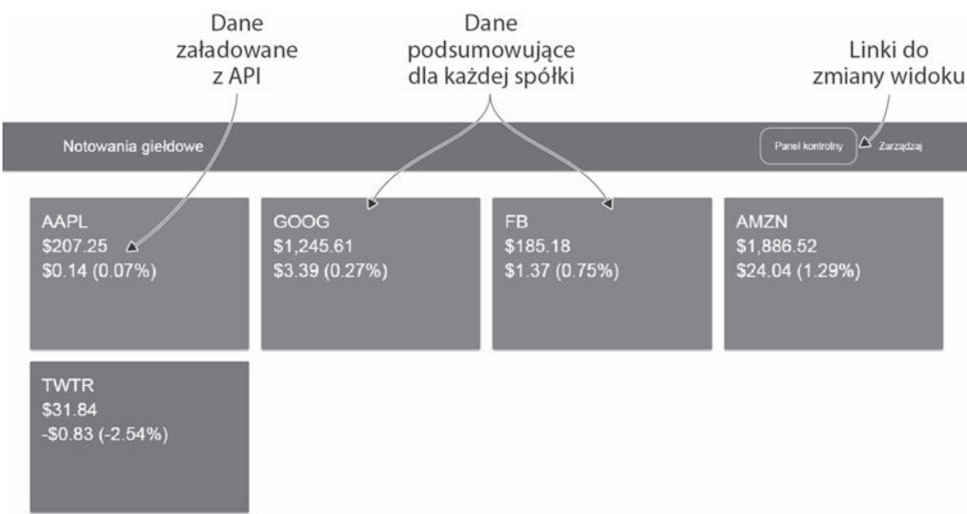
2.1. Przegląd projektu z tego rozdziału

Kiedy skończymy, aplikacja powinna wyglądać tak, jak to pokazano na rysunkach 2.1 i 2.2. Omówię pokrótce różne części, zanim je zbudujemy, więc zobaczysz, jak się ze sobą łączą.

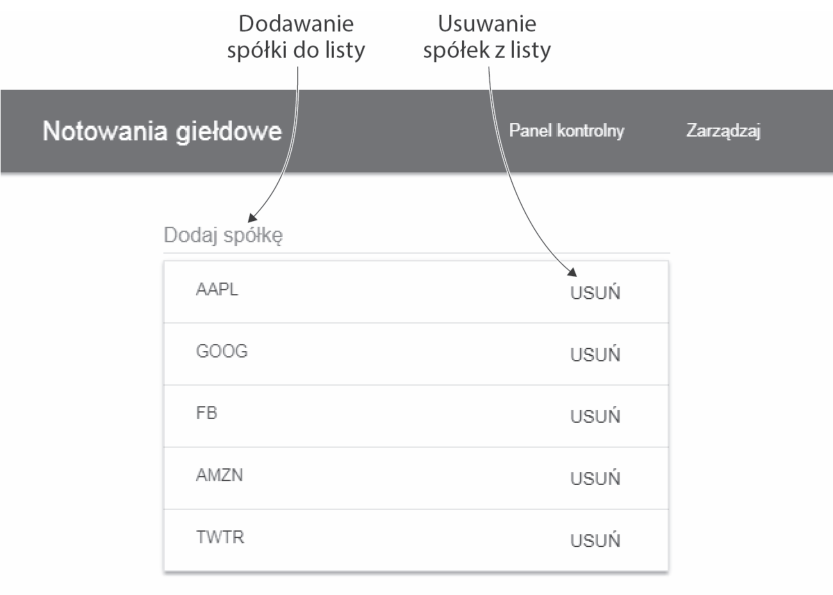
Przede wszystkim istnieje interfejs API, który ładuje aktualne dane o notowaniach akcji z Yahoo! Finance. Jest wdrożony na platformie Heroku i nie został omówiony w tym rozdziale, ale możesz przejrzeć kod tego API na stronie *https://github.com/angular-in-action/api*. Jest to standardowy interfejs REST API i nie wymaga uwierzytelnienia. Utworzymy usługę, która pomoże nam uzyskać dostęp do danych z tego API i je ładować.

Po załadowaniu aplikacji wyświetlana jest strona pulpitu kontrolnego (zobacz rysunek 2.1) z listą kart. Każda karta zawiera nazwę jednej spółki, aktualną cenę akcji

Kup ksi k Pole ksi k



Rysunek 2.1. Strona pulpitu kontrolnego aplikacji śledzenia notowań giełdowych z linkami i kartami sumarycznymi



Rysunek 2.2. Strona zarządzania aplikacji do śledzenia notowań z formularzem do zmiany listy wyświetlanych symboli spółek

i dzienną zmianę ceny (kwotową i procentową). Tło kart będzie czerwone dla spadku ceny, zielone dla wzrostu lub szare, gdy notowania się nie zmieniły. Każda z tych kart jest instancją komponentu, który pobierają dane o notowaniach i który określa sposób renderowania karty.

Górny pasek nawigacyjny ma dwa łącza, do widoków pulpitu kontrolnego i zarządzania, które umożliwiają ogólną nawigację między widokami. Użyjemy routera Angulara,

aby skonfigurować te trasy i zarządzać sposobem, w jaki przeglądarka określa, którą z nich wyświetlić.

Po kliknięciu na pasku nawigacyjnym linku *Zarządzaj* wyświetlona zostanie strona zarządzania (zobacz rysunek 2.2) z listą spółek. Tutaj można usunąć dowolną spółkę, klikając przycisk *Usuń*. Można także dodawać nowe spółki, wpisując symbol giełdowy w polu tekstowym i naciskając przycisk *Enter*.

Ta strona jest pojedynczym komponentem, ale zawiera formularz, który jest aktualizowany natychmiast po wprowadzeniu zmian przez użytkownika. Listę można rozszerzać, wpisując nowy symbol giełdowy w polu tekstowym i naciskając *Enter*. Można też skracać listę, klikając przycisk *Usuń* przy wybranej do usunięcia spółce. W obu przypadkach lista symboli zostanie natychmiast zmieniona, a jeśli wrócisz do pulpitu kontrolnego, wyświetli się zaktualizowana lista.

Ten projekt ma kilka ograniczeń, których powinieneś być świadomy. Aby przykład pozostał zwięzły i prosty, w aplikacji pominięto kilka szczegółów:

- **Brak persystencji.** Za każdym razem, gdy odświeżysz aplikację w przeglądarce, lista spółek zostanie zresetowana do domyślnej listy.
- **Brak sprawdzania błędów.** Niektóre sytuacje mogą rzucać błąd lub powodować dziwne zachowanie. Może się tak dziać na przykład przy próbie dodania spółki, która nie istnieje.
- **Brak testów jednostkowych.** W tym przykładzie skupiłem się na kodzie i celowo pominąłem testy jednostkowe, które zostaną omówione później.

Ten przykład ma na celu zaprezentowanie przeglądu sposobu, w jaki można budować aplikacje Angular, a nie dostarczenie solidnej aplikacji. Pod koniec rozdziału znajdziesz szereg interesujących wyzwań, które możesz podjąć; istnieje wiele możliwych funkcjonalności, które można sobie wyobrazić.

2.2. Konfigurowanie projektu

Zbudujemy ten przykład od zera, używając interfejsu CLI Angulara. Jeśli kiedykolwiek będziesz chciał przejrzeć kod tego projektu, możesz go znaleźć na stronie *https://github.com/angular-in-action/stocks*; każdy krok jest oznaczony, dzięki czemu można śledzić cały proces za pomocą Gita. Możesz też skopiować zawartość z rozdziału w takiej postaci, w jakiej się pojawia.

Jeśli nie skonfigurowałeś CLI Angulara, wróć do rozdziału 1. i zrób to. W tej książce używamy wersji 1.5 CLI, więc jeżeli korzystasz ze starszej wersji, powinieneś ją zaktualizować.

W terminalu zacznij od katalogu, w którym chcesz wygenerować folder nowego projektu. Następnie możesz wykonać poniższe polecenia, aby wygenerować nowy projekt i uruchomić serwer programistyczny:

```
ng new stocks
cd stocks
ng serve
```

Kup ksi k Pole ksi k

Zajmie to kilka chwil, ponieważ CLI instaluje wiele pakietów z `npm`, a zależy to w dużej mierze od szybkości Twojej sieci i stopnia zajętości rejestru. Po zakończeniu instalacji możesz użyć przeglądarki, aby wyświetlić aplikację pod adresem *http://localhost:4200*. Powinieneś zobaczyć prostą stronę z informacją, że jest to nowa aplikacja Angular, tak jak to pokazano na rysunku 2.3. Domyślna zawartość nowego projektu zmienia się od czasu do czasu, więc nie martw się, jeśli u Ciebie wygląda to trochę inaczej.

Rysunek 2.3. Interfejs CLI generuje pustą aplikację z pewną domyślną zawartością

Jeśli zobaczysz podobny ekran, wszystko powinno być skonfigurowane i gotowe do pracy. Nie jest to może najbardziej ekscytujący przykład, ale automatycznie konfiguruje za Ciebie kilka rzeczy. Teraz przyjrzymy się po kolei temu, co zostało wygenerowane, i temu, w jaki sposób wyświetlana jest ta prosta wiadomość.

2.3. Rusztowanie bazowej aplikacji

CLI wygenerował nowy projekt zawierający wiele plików. Przyjrzymy się najważniejszym z nich, a o reszcie przeczytasz więcej trochę później. Ważne jest, abyś zapamiętał, że CLI generuje pliki w specyficzny sposób i zmienianie lokalizacji lub nazw plików może spowodować awarię CLI. Na razie polecam pozostawienie plików tam, gdzie są, dopóki nie poczujesz się bardziej komfortowo lub nie zaplanujesz zbudowania później własnego narzędzia do kompilacji. Z czasem pliki i nazwy plików generowane przez CLI mogą ulec zmianie, więc jeśli masz problemy, przejrzyj dziennik zmian CLI i dokumentację.

Ten projekt zawiera kilka katalogów i plików. Główne pliki są wymienione w tabeli 2.1, wraz z ich ogólnymi rolami w aplikacji. Większość z nich to konfiguracje dla różnych aspektów programowania, takich jak reguły analizowania kodu, konfiguracja testów jednostkowych i konfiguracja CLI.

Kup ksi k Pole ksi k

Tabela 2.1. Zawartość głównego katalogu projektu wygenerowana przez interfejs CLI i role poszczególnych folderów i plików

Zasób	Rola
e2e	Folder testów *end-to-end*, zawiera podstawowy stub testu
node_modules	Standardowy katalog modułów NPM, nie należy tu umieszczać żadnego kodu
src	Katalog źródłowy dla aplikacji
.editorconfg	Ustawienia domyślne edytora
.angular-cli.json	Plik konfiguracyjny dla CLI dotyczący tego projektu
karma.conf.js	Plik konfiguracyjny narzędzia Karma do uruchamiania testów jednostkowych
package.json	Standardowy plik manifestu pakietu NPM
protractor.conf.js	Plik konfiguracyjny narzędzia Protractor do uruchamiania testów e2e
README.md	Standardowy plik *readme*, zawiera informacje startowe
tsconfg.json	Domyślny plik konfiguracyjny dla kompilatora TypeScriptu
tslint.json	Plik konfiguracyjny dla reguł analizy kodu TypeScriptu

W tym rozdziale będziesz modyfikować tylko te pliki, które znajdują się w katalogu *src*, zawierającym cały kod aplikacji. Tabela 2.2 przedstawia listę wszystkich zasobów wygenerowanych wewnątrz *src*. Może się wydawać, że to dużo plików, ale każdy z nich odgrywa pewną rolę, a jeśli nie jesteś pewien, co robi dany plik, na razie się nad tym nie zastanawiaj.

Tabela 2.2. Zawartość katalogu src i role poszczególnych folderów i plików

Zasób	Rola
app	Zawiera główny komponent i moduł aplikacji
assets	Pusty folder do przechowywania statycznych zasobów, takich jak obrazy
environments	Konfiguracja środowisk, umożliwiająca kompilowanie dla różnych celów, takich jak programowanie lub produkcja
favicon.ico	Obraz wyświetlany jako ikona ulubionej przeglądarki
index.html	Główny kod HTML dla aplikacji
main.ts	Punkt wejścia dla kodu aplikacji internetowej
polyfills.ts	Importuje niektóre typowe wypełnienia wymagane do prawidłowego działania Angulara w pewnych przeglądarkach
styles.css	Globalny arkusz stylów
test.ts	Punkt wejściowy testu jednostkowego, nie jest częścią aplikacji
tsconfg.app.json	Konfiguracja kompilatora TypeScriptu dla aplikacji
tsconfg.spec.json	Konfiguracja kompilatora TypeScriptu dla testów jednostkowych
typings.d.ts	Konfiguracja typowania

Teraz, gdy masz już ogólne pojęcie o tym, co zostało wygenerowane, zbadamy kilka kluczowych plików składających się na logikę aplikacji. W następnym podrozdziale przyjrzymy się bliżej sposobowi, w jaki Angular renderuje zawartość katalogu *app* do postaci danych wyjściowych, które są wyświetlane na ekranie.

Kup ksi k Pole ksi k

2.4. Jak Angular renderuje aplikację bazową

Zanim zaczniemy budować naszą aplikację, musisz zrozumieć, jak działa to podstawowe rusztowanie i co będziemy musieli dodać. Będzie to błyskawiczna wycieczka, abyś mógł jak najszybciej zacząć działać, więc większej liczby szczegółów i niuansów spodziewaj się w dalszej części książki. W rozdziale 3. poświęcimy tym tematom więcej czasu, abyś mógł lepiej zrozumieć, w jaki sposób to wszystko jest skonstruowane.

Angular wymaga co najmniej jednego komponentu i jednego modułu. **Komponent** jest podstawowym budulcem aplikacji Angular i działa podobnie jak każdy inny element HTML. **Moduł** to sposób, w jaki Angular organizuje różne części aplikacji w pojedynczą jednostkę zrozumiałą dla tego frameworka. Możesz potraktować komponenty jak klocki Lego, które mogą mieć wiele różnych kształtów, rozmiarów i kolorów, a moduły będą opakowaniem, w którym dostarczane są klocki. Komponenty dotyczą funkcjonalności i struktury, moduły przeznaczone są natomiast do pakowania i dystrybucji.

2.4.1. Komponent App

Zaczniemy od przyjrzenia się plikowi *src/app/app.component.ts*. Zawiera on tak zwany **komponent App**, który jest korzeniem aplikacji. W kategoriach Lego możesz wyobrazić sobie ten komponent jako wielką zieloną platformę, na której zaczynasz budowanie. Listing 2.1 przedstawia kod tego komponentu. I tym razem dokładny kod może się zmieniać z biegiem czasu, więc nie martw się, jeśli będzie się nieco różnił — będzie miał takie same podstawowe wymagania.

Listing 2.1. Wygenerowany komponent App (src/app/app.component.ts)

```
import { Component } from '@angular/core';       ← Importuje adnotację component.

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',          | Definiuje komponent
  styleUrls: ['./app.component.css']            | i jego właściwości.
})
export class AppComponent {                      | Tworzy kontroler komponentu
  title = 'app works!';                          | z pojedynczą właściwością.
}
```

Jeśli jesteś nowicjuszem w zakresie TypeScriptu, ten listing może zawierać trochę nieznanej Ci składni, więc przyjrzyjmy się bliżej każdej sekcji kodu. Najpierw importujemy adnotację `Component`. Służy ona do dekorowania komponentu App poprzez dodawanie szczegółów, które są związane z komponentem, ale nie są częścią logiki jego kontrolera, czyli klasy `AppComponent`. Angular sprawdza te adnotacje i używa ich z klasą kontrolera `AppComponent` do utworzenia komponentu w czasie wykonywania.

Adnotacja `@Component` deklaruje, że ta klasa jest komponentem, akceptując obiekt. Ma właściwość selektora, która deklaruje selektor HTML komponentu. Oznacza to, że komponent jest używany w szablonie przez dodanie znacznika HTML `<app-root></app-root>`.

Właściwość `templateUrl` deklaruje link do szablonu zawierającego szablon HTML. Podobnie właściwość `styleUrls` zawiera tablicę odnośników do wszelkich plików CSS,

które powinny zostać załadowane dla tego komponentu. Adnotacja `@Component` może mieć więcej właściwości i w tym rozdziale zobaczysz jeszcze kilka kolejnych w akcji.

Na koniec widać, że klasa `AppComponent` ma jedną właściwość o nazwie `title`. Ta wartość jest tym, co powinieneś zobaczyć wyrenderowane w przeglądarce, więc jest to źródło wartości, która ostatecznie się pojawia. Angular opiera się w dużej mierze na klasach ES2015 w celu tworzenia obiektów, a prawie wszystkie encje w Angularze są tworzone za pomocą klas i adnotacji.

Teraz spójrzmy na znaczniki związane z komponentem App, otwierając plik *src/app/app.component.html*, pokazany tutaj:

```
<h1>
Welcome to {{ title }}!</h1>
```

Jak widać, jest to po prostu zwykły znacznik nagłówka, ale istnieje właściwość `title` zdefiniowania pomiędzy podwójnymi nawiasami klamrowymi. Jest to powszechna konwencja dotycząca wiązania wartości do szablonu (być może znasz **szablony Moustache**) i oznacza, że Angular zastąpi `{{title}}` wartością właściwości `title` z komponentu. Jest to nazywane **interpolacją** i jest często używane do wyświetlania danych w szablonie.

Przyjrzeliśmy się komponentowi App, ale teraz musimy przyjrzeć się modułowi App, aby zobaczyć, jak rzeczy są łączone i renderowane za pomocą Angulara.

2.4.2. Moduł App

Moduł App to opakowanie, które pomaga informować Angular o tym, co jest dostępne do renderowania. Podobnie jak większość artykułów spożywczych ma opakowanie, które opisuje różne składniki znajdujące się w środku i inne ważne wartości, moduł opisuje różne zależności potrzebne do jego renderowania.

W aplikacji jest co najmniej jeden moduł, ale można utworzyć wiele modułów z różnych powodów (które zostaną omówione później). W tym przypadku są to wcześniejszy komponent App oraz dodatkowe możliwości, które są potrzebne w większości aplikacji (takie jak routing, formularze i HttpClient).

CLI wygenerował dla nas moduł, który możemy obejrzeć w lokalizacji *src/app/app.module.ts*, jak to pokazano w listingu 2.2. Po raz kolejny może się to zmienić z upływem czasu, ale struktura i cel pozostają te same.

Listing 2.2. Moduł App (src/app/app.module.ts)

```
import { BrowserModule } from '@angular/platform-browser';   | Importuje wymagane
import { NgModule } from '@angular/core';                    | zależności Angulara.

import { AppComponent } from './app.component'; <-- Importuje komponent App.

@NgModule({ <-- Używa adnotacji NgModule do zdefiniowania modułu przez przekazanie obiektu.
  declarations: [   | Deklaracje służą do wymienienia wszystkich komponentów
    AppComponent    | i dyrektyw użytych w aplikacji.
  ],
  imports: [        | Importy to pozostałe moduły, które są użyte w aplikacji.
    BrowserModule,
  ],
  providers: [], <-- Dostawcy to wszystkie usługi używane w aplikacji.
```

```
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Bootstrap deklaruje, którego komponentu użyć jako pierwszego do początkowego załadowania aplikacji.

Eksportuje pustą klasę, która jest adnotowana konfiguracją z NgModule.

Podobnie jak komponent, moduł jest obiektem z dekoratorem. Obiekt nazywa się tutaj `AppModule`, a `NgModule` jest dekoratorem. Pierwszy blok polega na zaimportowaniu wszelkich zależności Angulara, które są wspólne dla większości aplikacji, oraz komponentu App.

Dekorator `NgModule` przyjmuje obiekt z kilkoma różnymi właściwościami. Właściwość `declarations` ma dostarczyć listę wszystkich komponentów i dyrektyw, które mają być udostępnione całej aplikacji.

Właściwość `import` jest tablicą innych modułów, od których ten moduł zależy — w tym przypadku od modułu przeglądarki (zbioru wymaganych funkcji). Gdybyś kiedykolwiek dołączał inne moduły, takie jak moduły zewnętrzne lub te, które utworzyłeś, również muszą być one tutaj wymienione.

Następną właściwością jest `providers`, która jest domyślnie pusta. Wszelkie tworzone usługi mają być tu wymienione, a wkrótce zobaczysz, jak to zrobić.

Na koniec właściwość `bootstrap` określa, które komponenty mają zostać wstępnie załadowane podczas uruchamiania. Zazwyczaj będzie to ten sam komponent, a interfejs CLI już go skonfigurował. Właściwość `bootstrap` powinna pasować do komponentu, który inicjujesz w następnej sekcji.

Napisaliśmy kod, który tworzy konfigurację dla Angulara, aby na jej podstawie wiedział, jak renderować. Ostatnim krokiem jest przyjrzenie się kodowi, który zostanie wykonany podczas uruchamiania, co nazywa się **ładowaniem początkowym** (ang. *bootstrapping*).

2.4.3. Początkowe ładowanie aplikacji

Aby rozpocząć proces renderowania, aplikacja musi zostać zainicjowana w środowisku wykonawczym. Do tej pory deklarowaliśmy tylko kod, ale teraz zobaczymy, jak jest wykonywany. CLI zajmuje się podłączaniem narzędzi do kompilacji, które działa na podstawie webpacka.

Zacznijmy od przyjrzenia się plikowi *.angular-cli.json*. Zobaczymy tablicę aplikacji, a jedną z właściwości jest `main`. Domyślnie wskazuje ona na plik *src/ main.ts*. Oznacza to, że podczas kompilacji aplikacja automatycznie wywoła zawartość pliku *main.ts* jako pierwszy zestaw instrukcji.

Rolą pliku *main.ts* jest początkowe załadowanie aplikacji Angular. Zawartość *main.ts* została przedstawiona w listingu 2.3 i jest tam tylko kilka podstawowych instrukcji.

Listing 2.3. Plik main, który jest wywoływany podczas uruchamiania (src/main.ts)

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { enableProdMode } from '@angular/core';
import { environment } from './environments/environment';
import { AppModule } from './app/';
```

Importuje zależności.

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

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

Jeśli włączony jest tryb produkcyjny, wyłącza tryb programistyczny Angulara.

Początkowe ładowanie modułu App.

Pierwsza sekcja importuje niektóre zależności, w szczególności `platformBrowserDynamic` i `AppModule`. Nazwa jest trochę długa, ale obiekt `platformBrowserDynamic` jest używany do wskazania Angularowi, który moduł jest ładowany, a w tym przypadku jest to `AppModule` z wcześniejszego listingu. Renderowanie modułów zostanie omówione w dalszej części książki; na razie ważne jest zrozumienie, że właśnie w tym momencie zaczyna się wykonywanie kodu.

Ostatnim fragmentem, któremu należy się przyjrzeć, jest *index.html*. Jak być może pamiętasz z kodu komponentu App, istniał selektor `app-root`, który służy do identyfikacji komponentu w znacznikach. W pliku *src/index.html* powinieneś zobaczyć następujący fragment kodu:

```
<body>
  <app-root></app-root>
</body>
```

Po początkowym załadowaniu aplikacji (przez kod z listingu 2.3) Angular będzie szukać elementu `app-root` i zastąpi go uprzednio wyrenderowanym komponentem. Właśnie to zobaczysz na ekranie pokazanym na rysunku 2.1, ale podczas ładowania wyświetli się komunikat *Ładowanie…* Zanim komponent zostanie wyrenderowany, może upłynąć trochę czasu, bo wszystkie zasoby muszą zostać załadowane i zainicjowane. Nazywa się to kompilacją **JiT** (ang. *Just in Time*), co oznacza, że wszystko jest ładowane i renderowane w przeglądarce na żądanie. Kompilacja JiT jest przeznaczona tylko dla fazy rozwoju aplikacji i może zostać usunięta w przyszłych wersjach.

Chciałbym dodać kilka drobnych elementów, które pomogą nam nadać styl reszcie aplikacji, dopisując trochę podstawowego CSS i znaczniki. Najpierw do naszego pliku *src/index.html* musimy dodać dwa znaczniki linków:

```
<link rel="stylesheet" href="//storage.googleapis.com/code.getmdl.io/1.0.1/
     material.indigo-orange.min.css">
<link rel="stylesheet" href="//fonts.googleapis.com/
     icon?family=Material+Icons">
```

Spowoduje to załadowanie kilku ikon, czcionek i stylów globalnych dla aplikacji, które są oparte na projekcie Material Design Lite. Jest to jeden sposób ładowania zewnętrznych referencji do biblioteki stylów lub innych zasobów .

Chcielibyśmy nadać naszej aplikacji pewne globalne style. Dodaj poniższy fragment do pliku *src/styles.css* — nada to aplikacji jasnoszare tło:

```
body {
  background: #f3f3f3;
}
```

Na koniec chcemy ustawić kilka podstawowych znaczników, aby nadać strukturę naszej aplikacji. Zamieńmy zawartość *pliku src/app/app.component.html* znacznikami z listingu 2.4.

Kup ksi k Pole ksi k

Listing 2.4. Podstawowe rusztowanie znaczników (src/app/app.component.html)

```
<div class="mdl-layout mdl-js-layout mdl-layout--fixed-header">
  <header class="mdl-layout__header">
    <div class="mdl-layout__header-row">
      <span class="mdl-layout-title">Notowania giełdowe</span>
      <div class="mdl-layout-spacer"></div>
      <nav class="mdl-navigation mdl-layout--large-screen-only">
        <a class="mdl-navigation__link">Panel kontrolny</a>
        <a class="mdl-navigation__link">Zarządzaj</a>
      </nav>
    </div>
  </header>
  <main class="mdl-layout__content" style="padding: 20px;">

  </main>
</div>
```

Ten znaczniki są oparte na stylu projektowania Material Design Lite dla sposobu tworzenia podstawowego paska narzędzi i głównego ciała aplikacji. Pasek narzędzi ma tytuł i dwa łącza (które są obecnie nieaktywne) i powinien wyglądać tak, jak widać na rysunku 2.4.

Rysunek 2.4. Zmodyfikowane podstawowe rusztowanie zawierające znaczniki Material Design Lite

W porządku, utworzyliśmy podstawowe rusztowanie aplikacji za pomocą interfejsu CLI, zobaczyliśmy komponent App, moduł App i logikę początkowego ładowania oraz znaleźliśmy znaczniki renderujące komponent. Gratulacje, napisałeś swoją pierwszą aplikację Angular! OK, wiem, że nie jest to zbyt imponujące (jeszcze), ale to jest fundamentalna część każdej aplikacji Angular. W pozostałej części rozdziału na podstawie tej bazowej aplikacji utworzymy pełny przykład śledzenia notowań giełdowych. Na początek nauczysz się tworzyć usługę Angular, która ładuje dane z API.

2.5. Budowanie usług

Usługi są obiektami stanowiącymi abstrakcję jakiejś wspólnej logiki, która ma być wielokrotnie używana w wielu miejscach. Mogą zrobić wszystko, czego potrzebujesz, ponieważ są obiektami. Gdy korzystamy z modułów ES2015, klasy te są eksportowane, a więc w razie potrzeby każdy komponent może je zaimportować. Mogą również posiadać funkcje lub nawet wartości statyczne, takie jak łańcuch znaków lub liczba, jako sposób udostępniania danych różnym częściom aplikacji.

Usługi można też potraktować jako współdzielone obiekty, które w razie potrzeby może zaimportować dowolna część aplikacji. Są w stanie wyodrębniać jakąś logikę lub

dane (na przykład logikę niezbędną do załadowania pewnych danych ze źródła), dzięki czemu można ich łatwo użyć w dowolnym komponencie.

Chociaż usługi często pomagają w zarządzaniu danymi, nie są ograniczone do żadnych konkretnych zadań. Intencją usługi jest umożliwienie ponownego użycia kodu. Usługa może być zestawem wspólnych metod, które należy udostępnić. Możesz mieć różne „metody pomocnicze", których nie chcesz pisać za każdym razem od nowa; mogą to być narzędzia do parsownia formatów danych lub logika uwierzytelniania, która musi być uruchamiana w wielu miejscach.

W tej aplikacji będziesz potrzebował listy spółek, która będzie używana przez strony pulpitu kontrolnego i zarządzania. Jest to idealny scenariusz, w którym należy skorzystać z usługi, aby ułatwić zarządzanie danymi i udostępniać je różnym komponentom.

CLI daje nam dobry sposób tworzenia usługi, która ma potrzebne na początek rusztowanie. Wygeneruje również prostą usługę i stub testowy dla tej usługi. Aby wygenerować usługę, wykonaj następujące polecenie:

```
ng generate service services/stocks
```

CLI wygeneruje te pliki w katalogu *src/app/services*. Zawiera on najbardziej podstawową usługę, która nie robi nic. Pójdźmy dalej i wpiszmy kod dla całej usługi, a potem sprawdźmy, jak to działa. Ostatecznie uzupełnisz to, co zostało wygenerowane, kodem z listingu 2.5. Usługa Stocks będzie miała tablicę zawierającą listę symboli spółek giełdowych i będzie udostępniać zestaw metod do pobierania lub modyfikowania tej listy.

Listing 2.5. Usługa Stocks (src/app/services/stocks.service.ts)

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

let stocks: Array<string> = ['AAPL', 'GOOG', 'FB', 'AMZN', 'TWTR'];
let service: string = 'https://angular2-in-action-api.herokuapp.com';

export interface StockInterface {
  symbol: string;
  lastTradePriceOnly: number;
  change: number;
  changeInPercent: number;
}

@Injectable()
export class StocksService {

  constructor(private http: HttpClient) {}

  get() {
    return stocks.slice();
  }

  add(stock) {
    stocks.push(stock);
    return this.get();
  }
```

Importuje zależności. (importy)

Deklaruje tablicę spółek i zmienne API. (stocks, service)

Definiuje i eksportuje interfejs TypeScriptu dla obiektu stock. (StockInterface)

Adnotuje za pomocą Injectable, aby podłączyć wstrzykiwanie zależności. (@Injectable())

Definiuje klasę i eksportuje ją. (export class StocksService)

Metoda constructor do wstrzykiwania usługi HttpClient do właściwości http klasy. (constructor)

Metoda do pobierania spółek. (get())

Metoda do dodawania do listy nowej spółki. (add(stock))

```
  remove(stock) {
    stocks.splice(stocks.indexOf(stock), 1);
    return this.get();
  }

  load(symbols) {
    if (symbols) {
      return this.http.get<Array<StockInterface>>(service + '/stocks/
     snapshot?symbols=' + symbols.join());
    }
  }
}
```

Metoda do usuwania spółki z listy.

Metoda do wywoływania usługi HttpClient do ładowania wartości spółek z API.

Usługa musi najpierw zaimportować swoje zależności: jedną jest dekorator dla usługi, a drugą jest usługa HttpClient. Następnie deklaruje dwie zmienne. Jedna służy do śledzenia listy symboli spółek giełdowych, a druga to adres URL punktu końcowego interfejsu API.

Następnie interfejs `StockInterface` jest definiowany i eksportowany, aby mogły go używać inne komponenty. Zapewnia to typeScriptową definicję tego, co powinien zawierać obiekt `stock`, który jest używany przez TypeScript, aby upewnić się, że korzystanie z danych pozostaje spójne. Użyjemy tego później, aby mieć pewność, że prawidłowo typujemy nasze obiekty `stock`, gdy są stosowane.

Klasa `StocksService` jest eksportowana i dekorowana przez dekorator Injectable. Dekorator służy do ustawienia odpowiedniego podłączenia, aby Angular wiedział, jak używać jej w innym miejscu, więc jeśli zapomnisz dołączyć dekorator, klasa może nie zostać wstrzyknięta do reszty Twojej aplikacji.

W metodzie konstruktora usługa HttpClient jest wstrzykiwana za pomocą pochodzącej z TypeScriptu techniki deklarowania zmiennej prywatnej zwanej `http`, a następnie nadawany jest jej typ HttpClient. Angular może sprawdzić definicję typu i określić, w jaki sposób wstrzyknąć żądany obiekt do klasy. Jeśli jesteś nowicjuszem w zakresie TypeScriptu, pamiętaj, że zawsze, gdy po deklaracji zmiennej pojawia się dwukropek, definiujesz typ obiektu, który powinien być przypisany do tej zmiennej.

Ta usługa zawiera cztery metody. `get()` jest prostą metodą, która zwraca bieżącą wartość tablicy `stocks`, ale zawsze zwraca kopię, a nie bezpośrednią wartość. Ma to na celu hermetyzowanie wartości spółek i zapobieganie ich bezpośredniej modyfikacji. Metoda `add()` dodaje nowy element do tablicy `stocks` i zwraca nowo zmodyfikowaną wartość. Metoda `remove()` usuwa element z tablicy `stocks`.

Na koniec metoda `load()` wywołuje usługę HttpClient, aby załadować dane dla bieżących wartości cen akcji. Usługa HttpClient jest wywoływana i zwraca strumień obserwowalny, który jest konstrukcją do obsługi zdarzeń asynchronicznych, takich jak dane z wywołania API. Strumienie obserwowalne zostały pokrótce omówione w rozdziale 1. i będziemy je widywać częściej w innych rozdziałach, ale jest to pierwszy raz, kiedy możesz zobaczyć je w akcji.

Istnieje też niewielka funkcjonalność HttpClient, która pojawia się jako część metody `get()` i jest umieszczona między dwoma nawiasami trójkątnymi:

```
this.http.get<Array<StockInterface>>(...
```

Kup ksi k Pole ksi k

Jest to znane jako **zmienna typowa** i jest funkcjonalnością TypeScriptu, która pozwala wskazać metodzie `http.get()`, jakiego typu obiektu powinna się spodziewać; w tym przypadku będzie oczekiwać tablicy obiektów zgodnych z `StockInterface` (naszych obiektów `stock`). Jest to opcjonalne, ale powiadamianie kompilatora, jeśli spróbujesz uzyskać dostęp do właściwości, która nie istnieje, jest bardzo przydatne.

Jest jeszcze jeden krok, który musimy wykonać, ponieważ interfejs CLI nie rejestruje automatycznie tej usługi w module App i musimy również zarejestrować HttpClient w aplikacji. Otwórz plik *src/app/app.module.ts* i w górnej części dodaj te dwa importy:

```
import { HttpClientModule } from '@angular/common/http';
import { StocksService } from './services/stocks.service';
```

Spowoduje to zaimportowanie do pliku usługi Stocks i modułu `HttpClientModule`, ale musimy zarejestrować `HttpClientModule` w aplikacji. Znajdź sekcję importu zdefiniowaną w `@NgModule` i zaktualizuj ją tak, jak pokazano poniżej, aby uwzględniała `Http`
↪`ClientModule`:

```
imports: [
  BrowserModule,
  HttpClientModule
],
```

Teraz musimy zarejestrować nową usługę `StocksService` z właściwością `providers`, aby poinformować Angular, że powinna zostać udostępniona do używania przez moduł:

```
providers: [StocksService],
```

Twoja usługa jest podłączona i gotowa do wykorzystywania, ale nie użyliśmy jej jeszcze w żadnym miejscu naszej aplikacji. W następnym podrozdziale zostanie omówiony sposób jej konsumowania.

Ta usługa nie jest zbyt skomplikowana. Jest głównie przeznaczona do abstrahowania modyfikacji tablicy, aby nie była modyfikowana bezpośrednio, i ładowania danych z interfejsu API. Gdy aplikacja działa, tablica `stocks` może być modyfikowana, a zmiany są uwzględniane zarówno w komponencie panelu kontrolnego, jak i komponencie zarządzania, o czym wkrótce się przekonasz. Ponieważ jest wyeksportowana, można ją łatwo zaimportować, gdy będzie potrzebna.

Teraz utworzymy komponent wykorzystujący pewne domyślne dyrektywy i pozwolimy, aby konfigurowalne właściwości modyfikowały wyświetlanie komponentu.

2.6. Tworzenie pierwszego komponentu

Widziałeś już podstawowy komponent (App). Teraz zbudujemy bardziej złożony komponent, który używa niektórych dyrektyw i potoków i ma właściwość. Zamierzamy utworzyć komponent, który wyświetla podstawową kartę podsumowującą informacje o cenie akcji spółki.

Ten komponent będzie tylko otrzymywać dane do wyświetlenia ze swojego komponentu nadrzędnego i modyfikować własne wyświetlanie na podstawie tej wartości wejściowej. Komponent nadrzędny przekaże na przykład bieżące dane dla konkretnej

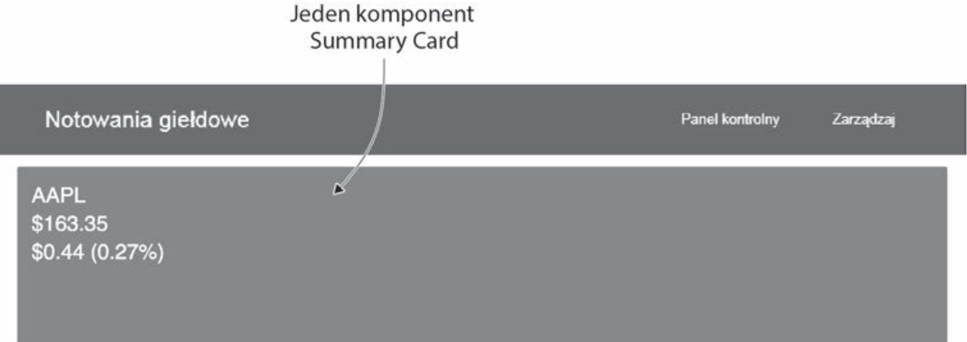
Kup ksi k Pole ksi k

spółki, a komponent Summary użyje dziennej zmiany notowań do ustalenia, czy tło powinno być w kolorze zielonym, czy czerwonym, na podstawie tego, czy cena akcji poszła w górę, czy w dół.

Kluczowymi celami tego komponentu są:

- przyjmowanie danych giełdowych i wyświetlanie ich;
- zmiana koloru tła w zależności od dziennej aktywności (zielony dla wzrostu, czerwony dla spadku);
- formatowanie wartości w celu prawidłowego wyświetlania (na przykład odpowiednia waluta lub odpowiednie wartości procentowe).

Rysunek 2.5 pokazuje ten komponent i nawet go podłączymy, aby ładował dane z API. Ostatecznie utworzymy wiele kopii tego komponentu do wyświetlenia karty dla każdej ze spółek.



Rysunek 2.5. Pojedynczy komponent Summary wyświetlający dane giełdowe

Oczywiście po uruchomieniu tego komponentu wartości akcji będą się zmieniać na podstawie najnowszych danych, ale możesz zobaczyć kartę wyświetlającą aktualne dane. Zbadamy szczegóły budowania tej karty, a następnie przejdziemy przez poszczególne części i zobaczymy, w jaki sposób powstaje ten wynik.

Wróć do terminala i uruchom następujące polecenie:

```
ng generate component components/summary
```

CLI wygeneruje nowy komponent wewnątrz katalogu *src/app/components/summary*. Musieliśmy utworzyć najpierw katalog *src/app/components*, ponieważ CLI nie tworzy automatycznie nowych folderów, jeśli ich brakuje. Pomaga to zorganizować komponenty w jednym katalogu, ale możesz je wygenerować gdzie indziej.

Teraz zawartość komponentu jest podobna do pierwotnej zawartości komponentu App. Zawiera pusty plik CSS, podstawowy szablon HTML, stub testowy i pustą klasę, już zainicjalizowaną za pomocą adnotacji `Component`.

Zaczniemy od skonfigurowania szablonu dla naszego komponentu, a następnie utworzymy kontroler do zarządzania nim. Otwórz plik *src/app/components/summary/summary.component.html* i zamień zawartość na to, co widzisz w listingu 2.6.

Listing 2.6. Szablon komponentu Summary

```
<div class="mdl-card stock-card mdl-shadow--2dp" [ngClass]="{increase:
     isPositive(), decrease: isNegative()}" style="width: 100%;">   ← Używa dyrektywy NgClass, aby pomóc przełączać kolor tła.
  <span>
    <div class="mdl-card__title">
      <h4 style="color: #fff; margin: 0">
        {{stock?.symbol?.toUpperCase()}}<br />   ← Wyświetla wartość symbolu i konwertuje na wielkie litery.
        {{stock?.lastTradePriceOnly | currency:'USD':'symbol':'.2'}}<br />   ← Wyświetla ostatnią cenę i formatuje ją zgodnie z walutą.

        {{stock?.change | currency:'USD':'symbol':'.2'}} ({{stock?.
    changeInPercent | percent:'.2'}})   ← Wyświetla dzienną zmianę w walucie i procentowo.
      </h4>
    </div>
  </span>
</div>
```

Szablon zawiera kilka znaczników, aby nadać karcie strukturę, taką jak struktura karty Material Design. Jeśli spojrzymy na pierwszą linię, zobaczymy ten fragment jako atrybut w elemencie `div`:

```
[ngClass]="{increase: isPositive(), decrease: isNegative()}"
```

Jest to szczególny rodzaj atrybutu, zwany dyrektywą. **Dyrektywy** umożliwiają modyfikowanie zachowania i wyświetlania elementów DOM w szablonie. Potraktuj je jako atrybuty w elementach HTML, które powodują, że element zmienia swoje zachowanie, na przykład atrybut `disabled` wyłączający element wejściowy HTML. Dyrektywy umożliwiają dodawanie logiki warunkowej lub w inny sposób modyfikują sposób zachowania bądź renderowania szablonu.

Dyrektywa NgClass może dodawać klasy CSS do elementu lub usuwać je z niego. Ma przypisaną wartość będącą obiektem zawierającym właściwości, które są nazwami klas CSS, a te właściwości są mapowane na metodę na kontrolerze (zostanie on jeszcze przez nas napisany). Jeśli metoda zwraca `true`, doda klasę. Jeżeli zwraca `false`, klasa zostanie usunięta. W tym fragmencie karta otrzyma klasę CSS `increase`, gdy dzienna zmiana ceny jest dodatnia, lub klasę CSS `decrease`, kiedy zmiana jest ujemna.

Angular ma wbudowane pewne dyrektywy, a w tym rozdziale zobaczysz jeszcze kilka innych. Dyrektywy zazwyczaj przyjmują wyrażenie (jak nasz obiekt w tym przykładzie), które jest ewaluowane przez Angular i przekazywane do dyrektywy. Wyrażenie może być ewaluowane do wartości logicznej lub innej wartości prostej, albo rozwiązywać się na wywołanie funkcji, która zostanie uruchomiona w celu zwrócenia wartości przed uruchomieniem dyrektywy. Na podstawie wartości wyrażenia dyrektywa może robić różne rzeczy, na przykład pokazywać lub ukrywać jakiś element, gdy wyrażenie jest prawdziwe lub fałszywe.

Przykład interpolacji widzieliśmy już wcześniej, ale teraz mamy przykład bardziej złożony, który wyświetla symbol spółki. Kontroler powinien mieć właściwość o nazwie `stock`, która jest obiektem o różnych wartościach:

```
{{stock?.symbol?.toUpperCase()}}
```

Składnia podwójnych nawiasów klamrowych jest sposobem wyświetlania wartości na stronie. Jak pewnie pamiętasz, nazywa się to **interpolacją**, choć ta jest dość skomplikowana. Zawartość między nawiasami nazywana jest **wyrażeniem Angulara** i jest porównywana z kontrolerem (podobnie jak dyrektywa), co oznacza, że spróbuje znaleźć w kontrolerze właściwość do wyświetlenia. Jeśli się to nie uda, zwykle rzucany jest błąd, ale operator bezpiecznej nawigacji (`?.`) wyciszy błąd i nie wyświetli niczego, jeśli właściwości nie będzie.

Ten blok wyświetli symbol giełdowy, ale pisany wielkimi literami. Większość wyrażeń JavaScriptu jest poprawnymi wyrażeniami Angulara, chociaż niektóre rzeczy się różnią, na przykład operator bezpiecznej nawigacji. Możliwość wywołania metod prototypowych, takich jak `toUpperCase()`, pozostaje bez zmian i dlatego można renderować tekst wielkimi literami.

Następna interpolacja pokazuje ostatnią cenę transakcyjną i dodaje kolejną funkcjonalność, zwaną **potokami**, które są dodawane bezpośrednio do wyrażenia w celu sformatowania danych wyjściowych. Wyrażenie interpolacji jest rozszerzone o symbol potoku (`|`), a następnie potok jest nazywany i opcjonalnie konfigurowany z wartościami oddzielonymi dwukropkiem (`:`). Wartość ceny wraca jak normalny typ zmiennoprzecinkowy (na przykład `111.8`), w odróżnieniu od formatu waluty, który powinien wyglądać jak `$111.80`:

```
{{stock?.lastTradePriceOnly | currency:'USD':'symbol':'.2'}}
```

Potoki jedynie modyfikują dane przed ich wyświetleniem i nie zmieniają wartości w kontrolerze. W tym kodzie podwójne nawiasy klamrowe wskazują na to, że chcesz powiązać dane przechowywane we właściwości `stock.lastTradePriceOnly`, aby je wyświetlić. Te dane są przesyłane strumieniowo za pośrednictwem potoku `currency` (waluta), który konwertuje wartość na cenę w określonej walucie na podstawie konfiguracji `USD` i zaokrągla do dwóch miejsc po przecinku. Teraz spójrzmy na następną linię kodu:

```
{{stock?.change | currency:'USD':'symbol':'.2'}} ({{stock?.changeInPercent |
     percent:'.2'}})
```

Ten kod ma dwa różne wiązania interpolacji z potokami `currency` i `percent` (procenty). Pierwsze z nich zostanie przeliczone na ten sam format waluty, ale drugie przyjmie dziesiętną wartość procentowa, taką jak `0.06`, i zamieni ją na 6%. Dokumentacja Angulara szczegółowo opisuje wszystkie dostępne opcje i sposób ich użycia dla każdego potoku.

Ten szablon nie działa w izolacji; wymaga kontrolera do podłączenia danych i metod. Otwórz plik *src/app/components/summary/summary.component.ts* i zamień kod na ten pokazany w listingu 2.7.

Listing 2.7. Kontroler komponentu Summary

```
import { Component, Input } from '@angular/core';   ← Importuje zależności.

@Component({
  selector: 'summary',
  styleUrls: ['./summary.component.css'],          Deklaruje metadane komponentu.
  templateUrl: './summary.component.html'
})
```

Kup książkę   Poleć książkę

```
export class SummaryComponent {      ← Eksportuje klasę komponentu Summary.

  @Input() stock: any;     ← Deklaruje właściwość, która jest wartością wejściową.

  isNegative() {
    return (this.stock && this.stock.change < 0);   | Metoda do sprawdzania,
  }                                                 | czy zmiana ceny jest ujemna.

  isPositive() {
    return (this.stock && this.stock.change > 0);   | Metoda do sprawdzania,
  }                                                 | czy zmiana ceny jest dodatnia.
}
```

Ten kontroler importuje zależności, co prawie zawsze jest pierwszym blokiem każdego pliku napisanego w TypeScripcie. Metadane komponentu opisują selektor, podłączone style i podłączone pliki szablonów, które składają się na komponent. Za moment dodamy trochę CSS do stylów.

Klasa kontrolera komponentu Summary rozpoczyna się od właściwości o nazwie `stock`, która jest poprzedzona adnotacją `Input`. Oznacza to, że ta właściwość ma być dostarczana do komponentu przez komponent nadrzędny, przekazujący ją do podsumowania. Właściwości są wiązane do elementu za pomocą atrybutu, tak jak widać tutaj — w tym przykładzie ustawimy wartość `stockData` komponentu nadrzędnego we właściwości `stock` komponentu Summary:

```
<summary [stock]="stockData"></summary>
```

Ponieważ dane wejściowe są przekazywane przez atrybut wiązania, dokona on ewaluacji wyrażenia i przekaże je do tej właściwości, aby komponent Summary mógł go zużyć. Wyrażenia Angular zachowują się tak samo zawsze, gdy istnieje wiązanie. Próbują znaleźć odpowiednią wartość w kontrolerze w celu powiązania z właściwością.

Na koniec mamy dwie metody do sprawdzania, czy wartość `stock` jest dodatnia, czy ujemna. Może również być neutralna, więc jest to stan domyślny i tylko wtedy, gdy wartość `stock` się zmieni, jedna z metod zwróci `true`. Te metody są używane przez dyrektywę NgClass, aby określić, czy powinna dodać określoną klasę CSS, tak jak to opisano wcześniej w szablonie.

Ostatnim fragmentem, który chcemy dodać, są same klasy CSS. Angular ma kilka interesujących sposobów hermetyzacji stylów CSS, aby miały zastosowanie tylko do jednego komponentu. Szczegółami zajmiemy się później, a teraz otwórz plik *src/app/components/summary/summary.component.css* i dodaj style, tak jak to pokazano w listingu 2.8.

Listing 2.8. Style CSS komponentu Summary

```
:host .stock-card {          ← Używa selektora CSS :host do zwiększenia szczegółowości selektora
  background: #333333;         i ustawienia domyślnego tła.
}
:host .stock-card.increase {    ← Klasa increase definiuje zielone tło.
  background: #558B2F;
  color: #fff;
}
```

Kup ksi k   Pole ksi k

```
:host .stock-card.decrease {   ← Klasa decrease definiuje czerwone tło.
  background: #C62828;
  color: #fff;
}
```

Jest to typowy CSS, chociaż być może nie widziałeś lub nie używałeś dotąd selektora `:host`. Ponieważ komponenty muszą być jak najbardziej autonomiczne, opierają się na koncepcji Shadow DOM, omówionej w rozdziale 1. Kiedy Angular renderuje ten komponent, modyfikuje dane wyjściowe, aby się upewnić, że selektor CSS jest unikatowy i nie wpływa przypadkowo na inne elementy na stronie. To zachowanie jest konfigurowalne, ale zostanie omówione później.

Selektor `host` jest sposobem określenia, że chcesz zastosować style do elementu, który hostuje ten element, więc w tym przypadku sprawdzony zostanie sam komponent Summary, a nie zawartość tego elementu. Głównym celem CSS jest tutaj ustalenie koloru tła komponentu Summary.

Przeszliśmy przez proces generowania komponentu Summary i zbudowaliśmy funkcjonalny komponent. Użyjmy go szybko, aby zobaczyć, jak się zachowuje.

Spójrz na zawartość pliku *src/app/app.module.ts*, a zobaczysz, że CLI już zmodyfikował ten moduł, aby uwzględnić komponent Summary w module App. Nie ma tu nic do zrobienia, ale chciałem to zaznaczyć.

Teraz spójrz na zawartość pliku *src/app/app.component.ts* i zaktualizuj ją zgodnie z listingiem 2.9. Załączy to usługę Stocks i użyje jej do przechowywania danych o notowaniach we własności. Następnie użyjemy tego, aby wyświetlić kartę podsumowania.

Listing 2.9. Kontroler komponentu App

```
import { Component } from '@angular/core';
import { StocksService, StockInterface } from './services/stocks.service';   ← Importuje StockInterface.

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  stocks: Array<StockInterface>;   ← Deklaruje właściwość tablicy spółek.

  constructor(service: StocksService) {
    service.load(['AAPL']).subscribe(stocks => {
      this.stocks = stocks;   ← Ładowane dane będą przechowywane we właściwości stocks.
    });
  }
}
```

Tutaj przechowujemy załadowane dane notowań we właściwości o nazwie `stocks`. Zapewniamy również pewne informacje o typowaniu, które są importowane z naszej usługi Stocks, aby TypeScript wiedział, jakiego rodzaju wartości oczekiwać. Wreszcie, zamiast wypisywać dane do konsoli, zapisujemy je we właściwości `stocks`.

Teraz musimy zaktualizować plik *src/app/app.component.html*, aby użyć komponentu Summary. Oto fragment, który musisz zaktualizować w szablonie:

```
<main class="mdl-layout__content" style="padding: 20px;" *ngIf="stocks">
  <summary [stock]="stocks[0]"></summary>
</main>
```

Pierwsza dodana linia, `*ngIf="stocks"`, jest dyrektywą, która zrenderuje tylko zawartość wewnątrz elementu, gdy wyrażenie będzie prawdziwe. W takim przypadku nie będzie renderować komponentu Summary do momentu załadowania danych o notowaniach.

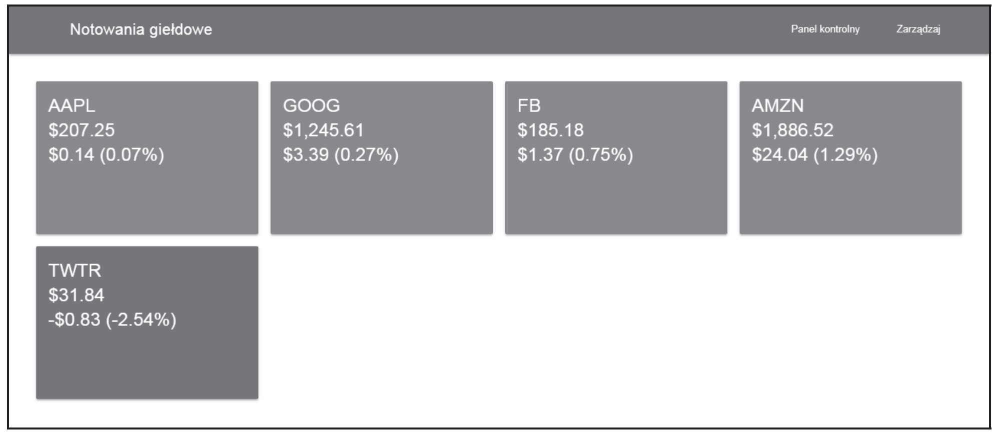
Środkowa linia pokazuje instancję pojedynczego komponentu Summary, a pierwsza wartość tablicy `stocks` jest dowiązana do właściwości `stock`. Dane zwracane są jako tablica, więc mamy bezpośredni dostęp do pierwszej wartości. Przypomnij sobie wartość wejściową, którą zadeklarowaliśmy w komponencie Summary, a która również jest nazwana `stock`.

Gdy to zapiszesz i uruchomisz aplikację, w efekcie powinna wyświetlić się pojedyncza karta podsumowującą z aktualnymi danymi notowań dla akcji spółki Apple. Zrobiliśmy nasz pierwszy komponent i wyświetliliśmy go w naszej aplikacji!

Teraz utworzymy następny komponent i użyjemy go razem z komponentem Summary, aby utworzyć pulpit kontrolny wyświetlający listę spółek i ich bieżące statusy.

2.7. Komponenty wykorzystujące komponenty i usługi

Jesteśmy gotowi do połączenia wcześniej utworzonego komponentu Summary i usługi Stocks w działający komponent Dashboard. Ten komponent będzie obejmował całą jedną stronę aplikacji, tak jak to pokazano na rysunku 2.6. Będzie zarządzał ładowaniem danych za pomocą usługi Stocks, a następnie będzie wyświetlał każdą spółkę za pomocą kopii komponentu Summary.



Rysunek 2.6. Komponent Dashboard połączony z ładowaniem danych i wyświetlający pięć instancji komponentu Summary

Zobaczymy, jak prawidłowo zaaranżować pełny widok zamiast naszych dotychczasowych odizolowanych przykładów. Na początek możemy ponownie użyć interfejsu CLI do wygenerowania kolejnego komponentu:

```
ng generate component components/dashboard
```

Kup ksi k Pole ksi k

Spowoduje to wygenerowanie w katalogu *src/app/components/dashboard* nowych plików dla HTML, CSS, kontrolera i testu jednostkowego. Doda także komponent do modułu App, aby mógł być natychmiast konsumowany. Zresetujmy nasz projekt roboczy, aby wyświetlić ten nowy komponent, modyfikując plik *src/app/app.component.html* poniższą zawartością:

```
<main class="mdl-layout__content" style="padding: 20px;">
  <dashboard></dashboard>
</main>
```

Powinno to spowodować wyświetlenie w aplikacji domyślnej wiadomości komponentu, ponieważ jest to domyślny kod wygenerowany przez CLI. Musimy również usunąć pewną logikę z kontrolera komponentu App; powinna teraz wyglądać tak, jak widzisz tutaj. Usuwa to importy i ładowanie danych notowań z samego komponentu App, a zamiast tego za chwilę wrzucimy to do komponentu Dashboard. Zastąp zawartość pliku *src/app/app.component.ts* następującym kodem:

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {}
```

Doskonale! Oczyściliśmy komponent App i jesteśmy gotowi do rozpoczęcia budowania panelu kontrolnego. Naszym pierwszym zadaniem jest skonfigurowanie kontrolera komponentu Dashboard. Jego rola polega na używaniu usługi Stocks do ładowania danych i udostępnianiu ich komponentowi do konsumowania.

Otwórz kontroler w pliku *src/app/components/dashboard/dashboard.component.ts* i zastąp go kodem z listingu 2.10.

Listing 2.10. Kontroler komponentu Dashboard

```
import { Component, OnInit } from '@angular/core';
import { StocksService, StockInterface } from '../../services/stocks.
     service';

@Component({
  selector: 'dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent implements OnInit {
  stocks: Array<StockInterface>;
  symbols: Array<string>;

  constructor(private service: StocksService) {
    this.symbols = service.get();
  }
```

Importuje zależności.

Eksportuje klasę komponentu, ale implementuje również interfejs OnInit.

Deklaruje właściwość do przechowywania tablicy notowań.

Deklaruje właściwość do przechowywania tablicy symboli giełdowych.

Pobiera symbole giełdowe z usługi, kiedy komponent jest konstruowany.

```
  ngOnInit() {
    this.service.load(this.symbols)
      .subscribe(stocks => this.stocks = stocks);
  }
}
```

Implementuje metodę ngOnInit i wywołuje usługę, aby załadować dane giełdowe przez Http.

Kontroler rozpoczyna się od zaimportowania adnotacji `Component` i interfejsu `OnInit`. Jeśli nie implementowałeś wcześniej interfejsu, wyjaśnię, że interfejs jest środkiem do egzekwowania tego, by klasa zawierała wymaganą metodę — w tym przypadku metodę o nazwie `ngOnInit`. Wykorzystanie możliwości TypeScriptu do wymuszania typowania kodu i interfejsów jest pomocne, gdy projekty stają się coraz większe.

Klasa `DashboardComponent` jest kontrolerem komponentu i deklaruje, że musi zaimplementować wymagania `OnInit`. Jeśli tak się nie stanie, TypeScript nie skompiluje kodu i rzuci błąd. Ma ona dwie właściwości: tablicę notowań i tablicę łańcuchów znaków reprezentujących symbole spółek do wyświetlenia. Początkowo są one pustymi tablicami, więc musimy je załadować danymi, aby komponent mógł być renderowany.

Metoda `constructor` jest uruchamiana natychmiast po utworzeniu komponentu. Będzie importować usługę Stocks do właściwości `service`, a następnie zażąda od niej aktualnej listy symboli spółek. To działa, ponieważ jest to akcja synchroniczna, które ładuje wartość bezpośrednio z pamięci.

Ale nie pobieramy danych z usługi w konstruktorze z wielu powodów. W złożoności zagłębimy się w dalszej części książki, teraz powinieneś wiedzieć, że główny powód jest związany ze sposobem renderowania komponentów. Konstruktor jest uruchamiany na wczesnym etapie renderowania komponentu, co oznacza, że często wartości nie są jeszcze gotowe do konsumowania. Komponenty udostępniają wiele zaczepów cyklu życia, które pozwalają wykonywać polecenia na różnych etapach renderowania, dając większą kontrolę nad tym, kiedy coś się dzieje.

W naszym kodzie używamy zaczepu cyklu życia `ngOnInit`, aby wywołać usługę ładowania danych giełdowych. Używa ona listy symboli spółek, która została załadowana w konstruktorze. Następnie ustanawiamy subskrypcję, aby czekać na zwrócenie wyników i zapisać je we właściwości `stocks`. Jest to wykorzystanie metody strumieni obserwowalnych do obsługi asynchronicznych żądań. Strumieniom obserwowalnym bardziej uważnie przyjrzymy się później. Tutaj używamy ich, ponieważ HttpClient zwraca nam strumień obserwowalny, abyśmy odbierali odpowiedź. Jest to udostępnione jako strumień danych, chociaż jest to pojedyncze zdarzenie.

Teraz musimy uzupełnić komponent, dodając szablon. Otwórz plik *src/app/components/dashboard/dashboard.component.html* i zastąp jego zawartość kodem z listingu 2.11.

Listing 2.11. Szablon komponentu Dashboard

```
<div class="mdl-grid">
  <div class="mdl-cell mdl-cell--12-col" *ngIf="!stocks" style="text-align:
     center;">
    Ładowanie
  </div>
  <div class="mdl-cell mdl-cell--3-col" *ngFor="let stock of stocks">
```

Używa NgIf, aby wyświetlić komunikat o ładowaniu, dopóki notowania nie zostaną załadowane.

Używa NgFor, aby wykonać pętlę przez wszystkie spółki.

```
    <summary [stock]="stock"></summary>   ← Tworzy instancję komponentu Summary
  </div>                                    dla każdej spółki i przekazuje dane giełdowe.
</div>
```

Ten szablon ma kilka klas, aby użyć frameworka interfejsu użytkownika Material Design Lite dla struktury siatki. Szablon zawiera kolejny atrybut NgIf, który wyświetla komunikat ładowania podczas ładowania danych, podobnie jak wcześniej. Po zwróceniu danych giełdowych z API wiadomość o ładowaniu zostanie ukryta.

Następnie widzimy kolejny element, który ma nową dyrektywę, NgFor. Podobnie jak NgIf, zaczyna się ona od znaku *, a wyrażenie jest podobne do tego, czego użyłbyś w tradycyjnej pętli `for` JavaScriptu. To wyrażenie zawiera `let stock of stocks`, co oznacza, że będzie wykonywać pętle przez wszystkie elementy w tablicy `stocks` i udostępniać lokalną zmienną o nazwie `stock`. Ponownie jest to ten sam rodzaj zachowania, który można zobaczyć w pętli `for` JavaScriptu, ale zastosowany w kontekście elementów HTML.

NgFor utworzy instancję komponentu Summary dla każdego z elementów `stock`. Wiąże to dane notowań z komponentem. Każda kopia komponentu Summary jest odrębna od pozostałych i nie współdzielą one danych bezpośrednio.

Ukończyłeś widok panelu kontrolnego, który korzysta z usługi i innego komponentu do renderowania doświadczenia użytkownika. Gdy uruchomisz teraz aplikację, powinieneś zobaczyć pięć domyślnych spółek; pojawiają się one jako oddzielne karty na stronie. Układ siatki powinien rozmieszczać je w czterech kolumnach.

Za chwilę zbudujesz nowy komponent z formularzem, zarządzający listą symboli spółek giełdowych, które mają być używane podczas wyświetlania notowań.

2.8. Komponenty z formularzami i ze zdarzeniami

Chcemy zarządzać wyświetlanymi spółkami, dlatego musimy dodać kolejny komponent, który ma formularz do edycji listy spółek (zobacz rysunek 2.7). Ten formularz pozwoli użytkownikom wprowadzać nowe symbole spółek giełdowych, aby dodać je do listy, i będzie miał listę aktualnych spółek z przyciskami, za pomocą których możne je usuwać listy. Ta lista spółek jest współdzielona przez całą aplikację, więc wszelkie zmiany będą replikowane we wszystkich innych miejscach.

Formularze są niezbędne w aplikacjach, a Angular ma wbudowaną obsługę tworzenia złożonych formularzy z wieloma funkcjonalnościami. Formularze we frameworku Angular składają się z dowolnej liczby **kontrolek**, które są różnymi rodzajami pól wprowadzania danych, jakie może zawierać formularz (takimi jak pole tekstowe, pole wyboru lub jakiś niestandardowy element).

Zacznijmy od wygenerowania nowego komponentu dla widoku zarządzania. Korzystając z interfejsu CLI, uruchom podane poniżej polecenie; pamiętaj, że automatycznie zarejestruje ono również komponent w module App, dzięki czemu będzie gotowy do użycia.

```
ng generate component components/manage
```

Rysunek 2.7. Komponent Manage z formularzem dodawania nowych elementów i przyciskami do usuwania elementów istniejących

Teraz zaktualizuj plik *src/app/app.component.html* i zmień zawartość elementu `main` na taką, jaką widać w poniższym kodzie, żeby komponent Manage wyświetlił się w aplikacji. Po uruchomieniu aplikacji wyświetli się domyślny komunikat charakterystyczny dla każdego nowego komponentu:

```
<main class="mdl-layout__content" style="padding: 20px;">
  <manage></manage>
</main>
```

Musimy również dodać do naszej aplikacji `FormsModule`, ponieważ będziemy używać funkcjonalności formularzy, które nie są automatycznie dołączane przez Angular. Otwórz plik *src/app/app.module.ts* i dodaj nowy import:

```
import { FormsModule } from '@angular/forms';
```

Następnie zaktualizuj definicję importów modułu, aby zadeklarować `FormsModule`, tak jak to pokazano poniżej:

```
imports: [
  BrowserModule,
  HttpClientModule,
  FormsModule
],
```

Zacznijmy tworzenie naszego komponentu Manage od zaktualizowania kontrolera za pomocą pewnej logiki. Na rysunku 2.7 zobaczysz, że musimy załadować listę symboli przechowywanych w pamięci. Potrzebne będą również dwie metody: jedna do obsługi usuwania spółki z listy, a druga do dodawania nowych symboli spółki do listy.

Otwórz plik *src/app/components/manage/manage.component.ts* i zaktualizuj go zawartością listingu 2.12. Obejmuje to dodatkowe metody i wymagane ustawienia dla tego widoku.

Kup ksi k Pole ksi k

Listing 2.12. Kontroler komponentu Manage

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

import { StocksService } from '../../services/stocks.service';

@Component({
  selector: 'manage',
  templateUrl: './manage.component.html',
  styleUrls: ['./manage.component.css']
})
export class ManageComponent {
  symbols: Array<string>;
  stock: string;

  constructor(private service: StocksService) {
    this.symbols = service.get();
  }

  add() {
    this.symbols = this.service.add(this.stock.toUpperCase());
    this.stock = '';
  }

  remove(symbol) {
    this.symbols = this.service.remove(symbol);
  }
}
```

Importuje zależności.

Deklaruje metadane komponentu.

Definiuje klasę i dwie właściwości dla przechowywania tablicy symboli i łańcucha znaków dla danych wejściowych.

Pobiera aktualną listę symboli podczas tworzenia instancji klasy.

Metoda dodawania nowej spółki do listy.

Metoda usuwania symbolu spółki z listy.

Jak zwykle zaczynamy od zaimportowania zależności dla komponentu. Następnie metadane komponentu deklarowane są za pomocą adnotacji `@Component`. Potem deklarowany jest obiekt `class`, który zawiera dwie właściwości: pierwszą jest tablica symboli pobierana z usługi Stocks, a drugą właściwość do przechowywania wartości danych wejściowych. W szablonie zobaczymy, jak właściwość `stock` jest połączona z polem danych wejściowych, ale właśnie tu jest ona najpierw definiowana.

Konstruktor korzysta z usługi, aby uzyskać tablicę symboli giełdowych i zapisać ją we właściwości `symbols`. Nie wymaga to zaczepu cyklu życia `OnInit`, ponieważ jest to synchroniczne żądanie uzyskania danych, które istnieją w pamięci.

Następnie mamy dwie metody: dodawania lub usuwania symboli z listy. Ta usługa zawsze zwraca kopię tablicy symboli `stocks`, więc do zarządzania listą (która jest zhermetyzowana wewnątrz usługi i nie jest bezpośrednio modyfikowalna) musimy wykorzystać metody usługi. Metoda `add` dodaje nową pozycję do listy symboli, a następnie zapisuje zmodyfikowaną listę na liście symboli. Analogicznie metoda `remove` usuwa pozycję z tablicy i odświeża listę symboli w kontrolerze.

Ten kontroler zaspokaja nasze potrzeby w zakresie obsługi działań formularza, ale teraz chcemy utworzyć szablon do wyświetlenia formularza i jego zawartości. Otwórz plik *src/app/components/manage/manage.component.html* i dodaj zawartość listingu 2.13.

W tym szablonie jest przyzwoita liczba znaczników tylko dla układu siatki. Każda klasa, która zaczyna się od `mdl-`, jest częścią stylów dostarczonych przez siatkę Material Design Lite i bibliotekę interfejsu użytkownika.

Listing 2.13. Szablon komponentu Manage

```
<div class="mdl-grid">
  <div class="mdl-cell mdl-cell--4-col"></div>
  <div class="mdl-cell mdl-cell--4-col">
    <form style="margin-bottom: 5px;" (submit)="add()">  ← Wiąże zdarzenie submit z wywołaniem metody add.
      <input name="stock" [(ngModel)]="stock" class="mdl-textfield__input"
     type="text" placeholder="Dodaj spółkę" />  ← Dwukierunkowe wiązanie dla pola wprowadzania symboli.
    </form>
    <table class="mdl-data-table mdl-data-table--selectable mdl-shadow--2dp"
     style="width: 100%;">
      <tbody>
        <tr *ngFor="let symbol of symbols">  ← Wykonuje pętlę przez listę symboli za pomocą NgFor.
          <td class="mdl-data-table__cell--non-numeric">{{symbol}}</td>  ← Wyświetla symbol.
          <td style="padding-top: 6px;">
            <button class="mdl-button" (click)="remove(symbol)">Usuń</button>  ← Wiąże zdarzenie kliknięcia przycisku z wywołaniem metody remove.
          </td>
        </tr>
      </tbody>
    </table>
  </div>
  <div class="mdl-cell mdl-cell--4-col"></div>
</div>
```

Pierwsza interesująca sekcja to formularz, która ma nowy typ atrybutu, taki, jakiego jeszcze nie widzieliśmy. Atrybut `(submit)="add()"` to sposób na dodanie nasłuchiwacza zdarzeń, znanego jako **wiązanie zdarzeń** (ang. *event binding*). Po przesłaniu formularza (co odbywa się za pomocą naciśnięcia przycisku *Enter*) nasłuchiwacz wywoła metodę `add`. Każdy atrybut umieszczony w nawiasach jest wiązaniem zdarzeń, a nazwa zdarzenia powinna odpowiadać nazwie zdarzenia bez przedrostka **on** (dla `onsubmit` będzie to `submit`).

Formularz zawiera pojedynczy element danych wejściowych, który ma kolejny nowy typ atrybutu. Atrybut `[(ngModel)]="stock"` jest wiązaniem dwukierunkowym, które zsynchronizuje wartość danych wejściowych i wartość właściwości w kontrolerze przy każdej zmianie w którejkolwiek z nich. W ten sposób, gdy użytkownik wpisze tekst w pole wprowadzania, wartość będzie natychmiast dostępna dla kontrolera do konsumpcji. Gdy użytkownik wciśnie *Enter*, zdarzenie `submit` zostanie wysłane i użyje wartości właściwości `stock` podczas dodawania nowego symbolu. Koncepcje formularzy zostaną omówione szczegółowo później, a teraz jest to Twój pierwszy kontakt ze sposobem konstrukcji prostego formularza.

Następna sekcja wykonuje pętlę przez listę symboli, używając NgFor. Sposób działania tego omówiłem wcześniej, więc nie będę tu wchodził w szczegóły. Dla każdego symbolu utworzone zostaną: zmienna lokalna o nazwie `symbol`, nowy wiersz tabeli, który wiąże wartość, i przycisk przeznaczony do usuwania elementu.

Przycisk `remove` zawiera kolejne wiązanie zdarzeń, które służy do obsługi zdarzenia kliknięcia. Atrybut `(click)="remove (symbol)"` dodaje nasłuchiwacz zdarzeń do zdarzenia kliknięcia i wywołuje metodę `remove` w kontrolerze, przekazując symbol. Ponieważ istnieje wiele instancji przycisku, każda przekazuje lokalną zmienną, aby wiadomo było, który symbol usunąć.

Ostatnim zadaniem jest dodanie do aplikacji routingu w celu aktywowania tras dla dwóch widoków, aby działały jak dwie różne strony.

2.9. Routing aplikacji

Ostatnim elementem aplikacji jest **routing**, który konfiguruje różne strony, jakie aplikacja może renderować. Większość aplikacji wymaga pewnej formy routingu, aby mogła wyświetlać poprawną część w oczekiwanym czasie. Angular ma router, który dobrze się sprawdza w architekturze tego frameworka, mapując komponenty na trasy.

Router działa poprzez zadeklarowanie w szablonie outletu, który jest miejscem wyświetlania ostatecznie zrenderowanego komponentu. Potraktuj outlet jako domyślny element zastępczy dla treści, a dopóki nie jest ona gotowa do wyświetlenia, pozostaje on pusty.

Aby skonfigurować nasze trasy, połączymy komponenty Manage i Dashboard z dwoma trasami. Sami poradzimy sobie z konfiguracją, ponieważ w tej konkretnej wersji interfejs CLI nie obsługuje konfigurowania tras. Na początek utwórz nowy plik *src/app/app.routes.ts* i umieść w nim kod z listingu 2.14.

Listing 2.14. Konfigurowanie routingu aplikacji

```
import { Routes, RouterModule } from '@angular/router';    ← Importuje zależności routera.

import { DashboardComponent } from './components/dashboard/dashboard.component';
import { ManageComponent } from './components/manage/manage.component';
                                     Importuje komponenty aplikacji,
                                     które są powiązane z trasami.
const routes: Routes = [
  {
    path: '',
    component: DashboardComponent
  },                                 Definiuje tablicę konfiguracji tras.
  {
    path: 'manage',
    component: ManageComponent
  }
];
                                                          Eksportuje trasy,
export const AppRoutes = RouterModule.forRoot(routes);  ← aby można było ich używać.
```

Głównym celem tego pliku jest skonfigurowanie tras dla aplikacji, a my zaczynamy od zaimportowania `RouterModule` i definicji typu Route. `RouterModule` jest używany do aktywowania routera i przyjmuje konfigurację tras po jej zainicjowaniu. Importujemy również dwa routowalne komponenty, Dashboard i Manage, abyśmy mogli odpowiednio odwoływać się do nich w konfiguracji tras.

Trasy są definiowane jako tablica obiektów, które mają co najmniej jedną właściwość; w tym przypadku są dwie: dla ścieżki adresu URL i komponentu. Dla pierwszej trasy nie ma ścieżki, więc działa ona jako indeks aplikacji (którym będzie *http://localhost:4200*) i linkuje do komponentu Dashboard. Druga trasa zapewnia ścieżkę URL zarządzania (którą będzie *http://localhost:4200/manage*) i linkuje do komponentu Manage. To jest

najbardziej prawdopodobny typ routingu, jaki będziesz wykonywał za pomocą Angulara, chociaż istnieje wiele sposobów na konfigurowanie i zagnieżdżanie tras.

Na koniec tworzymy nową wartość, `AppRoutes`, która jest przypisana do wyniku `Router↳Module.forRoot(routes)`. Dokładniej zachowaniem metody `forRoot` zajmiemy się później, teraz tylko wspomnę, że jest to sposób na przekazanie konfiguracji do modułu. W tym przypadku przekazujemy tablicę tras. Przeprowadzamy eksportowanie, abyśmy mogli zaimportować to do naszego modułu App i zarejestrować.

Otwórz plik *src/app/app.module.ts* i na końcu importów dodaj nową linię, która importuje obiekt `AppRoutes` utworzony w poprzednim pliku:

```
import { AppRoutes } from './app.routes';
```

Teraz zaktualizuj właściwość `imports` modułu, aby uwzględnić obiekt `AppRoutes`. To zarejestruje moduł routera i naszą konfigurację w aplikacji:

```
imports: [
  BrowserModule,
  HttpClientModule,
  FormsModule,
  AppRoutes
],
```

Ostatnim krokiem jest zadeklarowanie miejsca do renderowania przez router i zaktualizowanie linków używanych przez router do nawigacji. Otwórz po raz ostatni plik *src/app/app.component.html* i wprowadź kilka modyfikacji. Najpierw zmień zawartość elementu `main`, aby miała inny element — outlet routera:

```
<main class="mdl-layout__content" style="padding: 20px;">
  <router-outlet></router-outlet>
</main>
```

To deklaruje konkretną lokalizację w aplikacji, w którą router powinien wyrenderować komponent. Jest to to samo miejsce, w którym umieściliśmy nasze komponenty podczas ich budowania, wiec najwyraźniej jest to najlepsze miejsce.

Następnie musimy zaktualizować linki, aby używały nowej dyrektywy, która skonfiguruje nawigację między trasami. Dyrektywa RouterLink wiąże się z tablicą ścieżek, które są używane do budowania URL:

```
<nav class="mdl-navigation mdl-layout--large-screen-only">
  <a class="mdl-navigation__link" [routerLink]="['/']">Panel kontrolny</a>
  <a class="mdl-navigation__link" [routerLink]="['/manage']">Zarządzaj</a>
</nav>
```

Ta dyrektywa parsuje tablicę i próbuje znaleźć dopasowanie się do znanej trasy. Kiedy dopasuje trasę, dodaje `href` do znacznika kotwicy, który poprawnie łączy z tą trasą.

Router umożliwia bardziej zaawansowane konfiguracje, takie jak zagnieżdżone trasy, przyjmowanie parametrów i posiadanie wielu outletów. Szczegółowo omówię router w rozdziale 7.

Teraz Twój projekt jest kompletny i możesz przeładować aplikację w przeglądarce, aby zobaczyć, jak działa routing. Gratulacje! Masz działającą aplikację Angular, a teraz możesz spróbować dodać do niej kolejne funkcjonalności.

Kup ksi k Pole ksi k

Podsumowanie

Gratulacje — przebrnąłeś przez cały proces tworzenia funkcjonalnej aplikacji Angular! Przeszliśmy szybko przez wiele funkcjonalności frameworka Angular i powinieneś już rozumieć, jak różne części są składane w aplikację. Oto krótkie podsumowanie głównych kwestii do zapamiętania:

- Aplikacje Angular to komponenty zawierające drzewo komponentów. Główna aplikacja ładowana jest początkowo podczas ładowania strony w celu zainicjowania tej aplikacji.
- Komponent to klasa ES6 z adnotacją `@Component`, która dodaje metadane do klasy, aby Angular mógł ją poprawnie renderować.
- Usługi są również modułami ES6 i powinny być zaprojektowane pod kątem przenośności. Użyta może być każda klasa ES6, nawet jeśli nie jest specjalnie przeznaczona dla Angulara.
- Dyrektywy to atrybuty modyfikujące szablon w pewien sposób, na przykład dyrektywa NgIf warunkowo pokazuje lub ukrywa element DOM na podstawie wartości wyrażenia.
- Angular posiada wbudowaną obsługę formularzy, która obejmuje funkcję automatycznego sprawdzania poprawności, grupowania i wiązania danych z dowolną kontrolką formularza, a także korzystanie ze zdarzeń.
- Routing w Angularze opiera się na mapowaniu ścieżek na komponent. Trasy będą renderować pojedynczy komponent, a ten komponent będzie również mógł renderować dowolne dodatkowe komponenty, których potrzebuje.

Kup ksi k Pole ksi k

Kup ksi k Pole ksi k

Skorowidz

A

adnotacja @Component, 75
adresy URL, 214
aktualizacja, 323
    stopniowa, 326
alias, 166
anatomia testów jednostkowych, 276
Angular Material, 30
AngularJS, 24
animacje, 105
AoT, Ahead-of-Time, 89
API komponentu, 118
aplikacje, 46
    bazowe, 87
    jednostronicowe, 187
architektura komponentowa, 32
atrapy danych, 280
automatyzacja procesów, 305
awaryjne ładowanie pliku, 320

B

BEM, Block Element Modifier, 37
bezpieczeństwo, 316
biblioteka
    Clarity, 31
    Covalent, 31
    Fuel-UI, 31
    Ionic, 31
    Kendo UI, 31
    Ng-bootstrap, 31, 149
    PrimeNG, 31
    Wijmo, 31
biblioteki interfejsu użytkownika, 30

C

cechy komponentów, 34
ciągła integracja i dostarczanie, 314
CLI, Command Line Interface, 26
CORS, 320
CSS, 139
cykl życia komponentu, 106, 107
czyste
    funkcje, 230
    potoki, 230, 236

D

dane, 128
definiowanie
    formularza, 252
    tras, 190
dekorator, 43
    @Component, 105, 128
    @Input, 222
    metadanych komponentu, 105
    NgModule, 55
    ViewChild, 137
    zmian, 91
DI, dependency injection, 89, 99, 164
dodawanie stylów, 139
DOM, Document Object Model, 36
dostawcy, providers, 90
dostęp
    do kontrolera komponentu, 137
    do parametrów trasy, 196
drzewo
    komponentów, 75, 103
    wstrzykiwaczy, 164
dynamiczne
    renderowanie komponentów, 145
    tworzenie komponentu, 149
dyrektywa, 48, 62, 75, 82, 92, 217
    CardHover, 224
    CardType, 222
    Delay, 227
    NgForm, 271
    NgModel, 241, 243, 244
    routerLink, 195, 216

Kup ksi k   Pole ksi k

dyrektywy
- atrybutów, 219, 221, 236
- strukturalne, 219, 226, 236
- walidacji, 247
- walidacji godzin, 269
- ze zdarzeniami, 223

E

encje, 79, 80
ES6, 40, 46, 75

F

fabryka, 166
formularz logowania, 82
formularze, 237–271
- definiowanie, 252
- implementowanie szablonu, 254
- kontrolki, 254, 271
- niestandardowa walidacja, 246
- niestandardowe kontrolki, 265
- niestandardowe walidatory, 256
- obserwowanie zmian, 255
- obsługa zdarzeń, 249, 260
- oparte na szablonach, 241
- reaktywne, 237, 251, 271
- szablony, 237, 242, 250
- tworzenie kontrolki, 271
- ustawianie stanu, 253
- walidacja kontrolek, 243
- właściwości walidacji kontrolki, 245

framework, 26
- Electron, 29
- Ionic, 29
- NativeScript, 29
- Progressive Web Apps, 29
- React Native, 29
- Windows Universal, 29

funkcja HoursValidator, 258

G

granica cienia, 37

H

hermetyzacja
- stylizacji, styling encapsulation, 141
- znaczników i stylów, 144

HTML, 35
HttpClient, 48, 186
HTTPInterceptor, 174

I

implementowanie szablonu, 254
importowanie niezbędnych elementów, 318
interfejs
- CLI, 51
- HttpInterceptor, 173

internacjonalizacja, 312
interpolacja, 54, 63, 93

J

JavaScript
- moduły, 39

JiT, Just in Time, 56, 89

K

katalog
- główny, 52
- src, 52

klasa, 42, 166, 186
- DebugElement, 287
- ES6, 75
- StocksService, 59

kompatybilność, 318
kompilacja produkcyjna, 310
kompilator, 27
- AoT, 89
- JiT, 56, 89
- typ komponentu, 102

kompilowanie Angulara, 310
komponent, 32, 48, 53, 75, 79, 81, 101–153, 307
- App, 53, 65, 110, 136, 161
- Chat, 204
- Dashboard, 66, 125
- Data, 110–113
- Display, 110, 111
- Forum, 197
- Investments, 163
- InvoiceForm, 261
- Login, 209
- Manage, 70
- Metric, 132

Nodes Row, 131
Route, 110, 113
Summary, 61, 62
Thread, 201
Threads, 200
komponenty
animacje, 105
API, 118
cechy, 34
cykl życia, 104, 106
dekorator metadanych, 105
dodawanie stylów, 139
dostawcy, 105
drzewo, 103
dynamiczne renderowanie, 145
dynamiczne tworzenie, 149
hermetyzacja, 105
hosty, 105
kompozycja, 104
komunikacja, 134, 329
metryczne, 121
przechwytywanie wejść, 120
renderowanie, 149
rodzaje, 110
rzutowanie zawartości, 122, 128
stylizacja, 105, 138
szablon, 105
tryb
braku hermetyzacji, 142
emulowanej hermetyzacji, 143
natywnej hermetyzacji, 144
tryby hermetyzacji, 138, 141
używanie wejść, 116
wejścia, 105, 117
wyjścia, 105
wykorzystujące komponenty, 66
wykorzystujące usługi, 66
z animacjami, 290
z formularzami, 69
z routerem, 292
z wejściami, 290
ze zdarzeniami, 69
zaczepy cyklu życia, 105
zagnieżdżanie, 109
kompresja GZIP, 320
komunikacja między komponentami, 134, 329
komunikaty walidacji, 258
konfiguracja
nginx, 320
routera, 190
konfigurowanie routingu, 48
aplikacji, 73
kontroler komponentu, 105
Alert, 150
App, 65, 137, 150, 171
Dashboard, 67, 114
HoursControl, 266
Investments, 163
InvoiceForm, 261
Manage, 71
Metric, 117
Nodes Detail, 146
Nodes Row, 125
Summary, 63
kontrolki
formularza, 271
niestandardowe, 265

L

leniwe ładowanie, 211
tras, 315
linki w szablonach, 195
lokalizator stron, 299

Ł

ładowanie
asynchroniczne, 211
obiektów, 41
początkowe, 55

M

metadane, 105
moduł App, 54, 80
moduły, 53, 79
JavaScriptu, 39
routera, 212
testujące, 285
monitorowanie
aktywności, 306
błędów, 306
możliwości
desktopowe, 29
mobilne, 29
multidostawca, 175

N

nagłówek aplikacji, 235
narzędzia
 alternatywne kompilacji, 313
 do kompilacji, 106
 testowe, 274
narzędzie source-map-explorer, 317
nasłuchiwacz zdarzeń, 97
nawigacja między trasami, 204
nieczyste potoki, 234, 236

O

obiekt
 DashboardPage, 299
 ManagePage, 300
 strony, 299
obietnice, 43
obsługa
 formularzy, 75
 zadań asynchronicznych, 295
 zdarzeń, 249
odwrotny serwer proxy, 320
ograniczanie zewnętrznych zależności, 316
okno ng-bootstrap, 145
opcje uaktualnienia, 324
optymalizacje, 130
 dla przeglądarek, 311
organizacja W3C, 34

P

parametry trasy, 194
pasek nawigacyjny, 104
persystencja, 50
pierwsza aplikacja, 47
platforma, 26
plik
 index.html, 320
 main, 55, 169
początkowe ładowanie aplikacji, 48, 55
potok, 48, 79, 85, 92, 306
 Change, 231
 ChangeDetector, 233
 News, 235
potoki
 czyste, 229, 230, 236
 kompilacji, 314
 nieczyste, 229, 232, 236
 niestandardowe, 229
potomek
 widoku, view child, 109
 zawartości, content child, 109
programowanie
 oparte na testach, TDD, 304
 reaktywne, 43
progresywne aplikacje internetowe, 312
przechwytywanie wejść, 120
przekierowania, 285
przyrostowa aktualizacja, 328
pulpit kontrolny, 49, 102

R

reguły CSS, 141
renderowanie, 87, 88, 138
 serwerowe, 27, 28, 314
 wstępne, 314
rodzaje
 kompilatorów, 89
 komponentów, 110
RouterModule, 215
routing, 48, 75, 187–216
 aplikacji, 73
 definiowanie tras, 190
 dostęp do parametrów trasy, 196
 konfiguracja routera, 190
 moduł dla blogów, 213
 moduły funkcyjne, 193, 216
 najlepsze praktyki, 214
 parametry trasy, 194
 programowy, 205
 trasy drugorzędne, 201
 trasy podrzędne, 198
 właściwości trasy, 191
 zabezpieczanie tras, 205
 zamykanie trasy drugorzędnej, 205
rzutowanie
 informacji metrycznych, 127
 zawartości, 122, 126, 128

S

selektor app-root, 56
Shadow DOM, 36, 144
składnia JavaScriptu, 42

Kup ksi k Pole ksi k

słowo kluczowe
export, 42
this, 42
SMACSS, Scalable Modular Architecture for CSS, 37
strategie testowania, 302
strażnicy, 206, 216
strumienie obserwowalne, 43
stub usługi Stocks, 284
stuby, 279
dyrektyw, 287
style CSS, 64
stylizacja HoursControl, 269
symulowanie
usługi, 283
żądań HTTP, 279, 282
szablon komponentu, 37, 99, 105
CustomerForm, 242, 244
Dashboard, 68
HoursControl, 268
InvoiceForm, 254, 263
Manage, 72
Metric, 118, 126
Nodes, 123
Detail, 146
Row, 124, 147
Summary, 62
szablony Moustache, 54

Ś

środowisko produkcyjne, 309

T

TDD, test-driven development, 304
test
dyrektywy
CardHover, 294
Delay, 296
jednostkowy
kontrolera, 287
widoku, 287
komponentu
Dashboard, 288
Manage, 285
Summary, 290
potoku Change, 277
usługi Stocks, 281
testowanie
aplikacji, 273
dyrektyw, 293, 307
komponentów, 285
z animacjami, 290
z routerem, 292
z wejściami, 290
potoków, 277
usług, 279
testy
e2e, 274, 297, 300–307
integracyjne, 274
jednostkowe, 273, 276, 304, 306
oparte
na rezultatach, 306
na użytkownikach, 306
trasy, 215
drugorzędne, 201, 216
modułu Forums, 199
podrzędne, 198, 216
właściwości, 191
tryb OnPush, 131, 132
tryby hermetyzacji, 141
tworzenie
czystego potoku, 230
dyrektywy
atrybutów, 221
strukturalnej, 226
komponentów, 48
komponentu Data, 113
linków, 195
nieczystego potoku, 232
niestandardowych
dyrektyw, 219
potoków, 229
pierwszego komponentu, 60
usług, 48, 159
typ dostawcy, 166
TypeScript, 44–46
typy encji, 79, 98

U

usługa, 48, 57, 75, 79, 86, 155–186, 307
Account, 160, 178
Account z alertami, 183
Alert, 181
AuthGuard, 206
Config, 168

Kup ksi k Pole ksi k

usługa
  FormBuilder, 253
  HttpClient, 169, 186
  Stock, 86
  Stocks, 58, 170, 171
usługi
  bez wstrzykiwania zależności, 168
  danych, 156
  dodatkowe, 184
  niewstrzykiwalne, 156
  pomocnicze, 156, 176
  tworzenie, 159
  udostępniania, 180
  wstrzykiwalne, 156
używanie potoków, 48

V

ViewChild, 137

W

W3C, World Wide Web Consortium, 34
walidacja niestandardowa, 256, 271
walidator
  godzin, 257
  kontrolek, 243
  numeru telefonu, 247
wartość, 166
wbudowany CSS, 139
wdrożenie, 319
wejścia, inputs, 94, 105
wiązania, 91
  atrybutów, 92, 96
  danych, 241
  specjalne właściwości, 95
  właściwości, 92, 94
  zdarzeń, 72, 92, 97
widok osadzony, 227
właściwości
  trasy, 191
  walidacji kontrolki, 245
właściwość
  styles, 140
  templateUrl, 53
wsparcie techniczne, 318
wstrzykiwacz, injector, 90
wstrzykiwanie zależności, DI, 89, 99, 164
wybór architektury Angulara, 315
wydajność w przeglądarce, 28
wyjścia, outputs, 105
wykrywanie zmian, 90, 99, 130
wyrażenia
  Angulara, 63
  szablonów, 91

Z

zabezpieczanie tras, 205
zaczepy cyklu życia, 105, 107, 153
zagnieżdżanie komponentów, 109
zatwierdzenie formularza, 97
zdarzenia
  anulowania, 249, 260
  bez przedrostka on, 72, 97
  przesyłania, 249, 260
  wyjściowe, 135
zmienne
  szablonów, 135
  typowe, 60
znaczniki, 56
  Material Design Lite, 57

Kup ksi k Pole ksi k

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

Framework Angular od wielu lat jest jednym z najważniejszych narzędzi do rozwijania średnich i dużych aplikacji internetowych. Co istotne, aplikacje te bez problemu działają w każdej nowoczesnej przeglądarce, a także na platformach mobilnych. Obecnie Angular jest dojrzałą, stabilną i wciąż ewoluującą technologią, a w kolejnych wydaniach frameworka pojawiają się nowe, ułatwiające pracę funkcje. Aplikacje napisane w Angularze opierają się na hierarchicznym systemie komunikujących się komponentów i na dobrze zbudowanych interfejsach API. Przejrzystość i zrozumiałe zasady rządzące tym systemem sprawiają, że nauka posługiwania się Angularem przychodzi szybko i jest bardzo satysfakcjonująca.

Ta książka jest przeznaczona dla programistów, którzy chcą możliwie szybko zacząć budować poprawne aplikacje w Angularze i bezproblemowo uruchamiać je w środowisku produkcyjnym. Podręcznik został napisany w sposób, który pozwala na natychmiastowe rozpoczęcie kodowania i zrozumienie — niemal mimochodem — tak zaawansowanych technik jak testowanie, wstrzykiwanie zależności czy regulowanie wydajności. W książce położono nacisk na korzystanie z TypeScriptu i ES2015 oraz na tworzenie poprawnego kodu zgodnie z najlepszymi praktykami. Nie zabrakło licznych wskazówek i opisu nieoczywistych, ale bardzo pomocnych technik pracy. Dzięki temu szybko wykorzystasz potencjał Angulara do pisania wydajnych, odpornych i bezpiecznych aplikacji!

Najważniejsze zagadnienia:

- **Przegląd architektury Angulara i jego funkcji**
- **Zasady tworzenia aplikacji w Angularze**
- **Komponenty i interakcje między nimi**
- **Usługi, wstrzykiwanie zależności i wzorce nawigacyjne**
- **Testowanie, debugowanie i wdrażanie aplikacji**

Jeremy Wilken jest ekspertem programu Google Developers Experts w zakresie technologii internetowych i Angulara. Otrzymał też prestiżowy tytuł Asystenta Google. Na co dzień pisze aplikacje w Angularze, a także szkoli, prowadzi warsztaty i bierze udział w konferencjach technologicznych. Pracował dla takich firm jak eBay, Teradata i VMware, a od wielu lat jest konsultantem.

Oto Angular: ambitne narzędzie dla profesjonalistów!

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-4798-4
9 788328 347984

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 59,00 zł