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

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.

Spis treści

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. Statyczny 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
Dyrektywy kontenera	
`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 odstępy między elementami potomnymi.
Dyrektywy elementów potomnych	
`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 `@angle/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 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 `fxFlex` podrzędnych. Poniższy szablon używa dyrektywy poziomu podrzędnego `fxFlex`, aby przeznaczyć 30% dostępnej szerokości dla lewego elementu potomnego, a 70% dla prawego:

```
<div fxLayout="row" class="parent">
  <div fxFlex="30%" class="left">Lewy</div>
  <div fxFlex="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 `fxLayout="column"`:

```
<div fxLayout="column" class="parent">
  <div fxFlex="30%" class="left">Lewy</div>
  <div fxFlex="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

Załóżmy, 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.

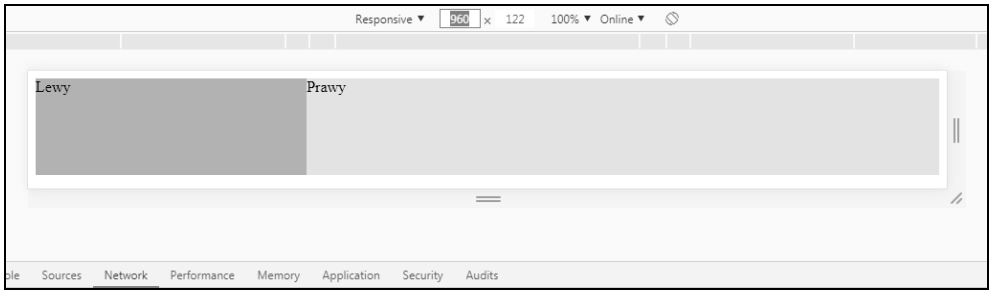
Listing 7.3. Dodanie przyrostka .sm

```
<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.

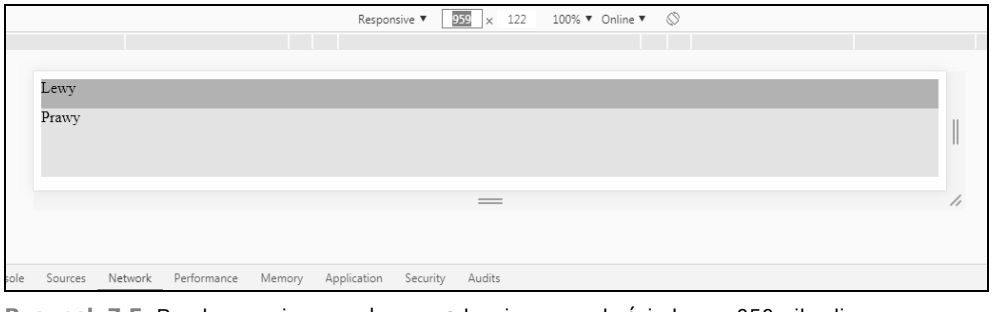
Na ekranach o małych rozmiarach elementy potomne są układane pionowo.

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.

showExtra$ 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 jakiejkolwiek 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 modułem 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

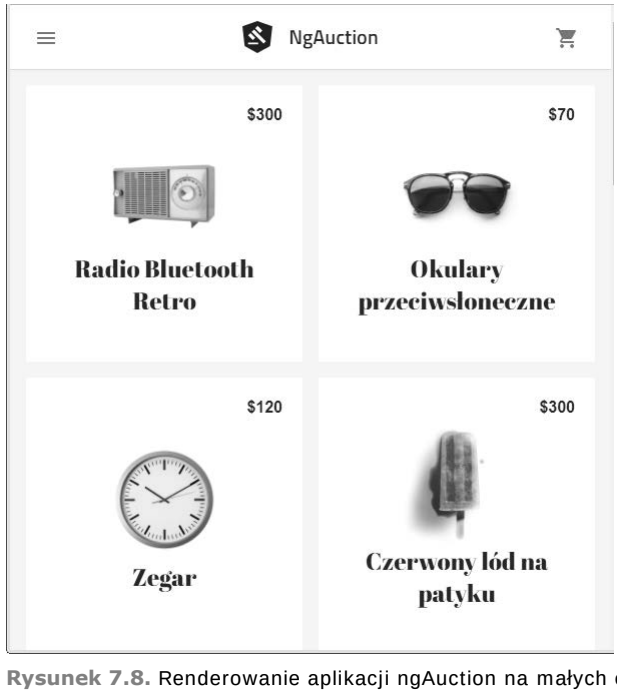
UWAGA Dane i obrazy zapożyczyliśmy z aplikacji Google ilustrującej bibliotekę Polymer (zobacz *http://mng.bz/Y5d9*).

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
```

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.

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

Instaluje bibliotekę Flex Layout.

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 zagnieździć 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 cząstkowe** (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 cząstkowego podkreślenia nie są wymagane, na przykład `@import './theme';`. Pliki cząstkowe 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 cząstkowy: *_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.

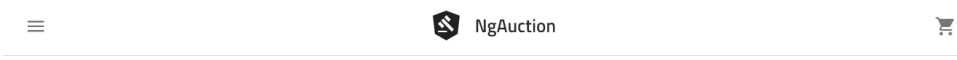
```
  font-family: mat-font-family($nga-typography);
  line-height: mat-line-height($nga-typography, body-1);
  font-size: mat-font-size($nga-typography, body-1);
  height: 100%;
  margin: 0;
}
```

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`. Zacznijmy 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 korzystamy 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.

Zaktualizujmy wygenerowany przez interfejs CLI plik *app.module.ts*, aby uwzględniał 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="/"
       fxLayout>
      <img class="toolbar__logo"
           src="/assets/logo.svg"
           alt="NgAuction Logo">
    </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. Zacznijmy 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, który może zwracać wszystkie obiekty Product.
    return this.http.get<Product[]>('/data/products.json');
  }

  getById(productId: number): Observable<Product> {   <-- Ta funkcja deklaruje obiekt Observable, który może zwracać produkty według identyfikatora.
    return this.http.get<Product[]>('/data/products.json')
      .pipe(
        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 `getById()` 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. Wygenerujmy 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[]>;

  readonly breakpointsToColumnsNumber = new Map([
    [ 'xs', 1 ],
    [ 'sm', 2 ],
    [ 'md', 3 ],
    [ 'lg', 4 ],
    [ 'xl', 5 ],
  ]);

  constructor(private media: ObservableMedia,
              private productService: ProductService) {
    this.products$ = this.productService.getAll();

    this.columns$ = this.media.asObservable()
      .pipe(
        map(mc => <number>this.breakpointsToColumnsNumber.get(mc.mqAlias))
      );
  }
}
```

Strumień obserwowalny dla produktów.

Mapuje alias zapytania o media na liczbę kolumn w siatce.

Wstrzykuje ObservableMedia i ProductService.

Pobiera dane o wszystkich produktach.

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">
    <mat-grid-tile class="tile" *ngFor="let product of products$ | async">

      <a class="tile__content"

         fxLayout="column"
         fxLayoutAlign="center center"
```

Subskrybuje liczbę kolumn i wiąże ją z właściwością cols komponentu <mat-grid-list>.

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 klikalne łącze.

```
        [routerLink]="['/products', product.id]">
                                                          Kliknięcie kafelka powoduje przejście
                                                          do ścieżki /products i przekazanie
                                                          jako parametru identyfikatora
                                                          wybranego produktu.
        <span class="tile__price-tag"
              ngClass.xs="tile__price-tag--xs">
          {{ product.price | currency:'USD':'symbol':'.0' }}
        </span>
                                                          Dla bardzo małych rzutni dodaje
                                                          style zdefiniowane
                                                          w tile__price-tag--xs.

        <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>
```

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';   <-- Importuje nasz niestandardowy motyw.

:host {
  display: block;
}

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

.tile {
  background-color: mat-color($nga-background, card);   <-- Ustawia kolor tła na taki sam, jaki ma komponent card biblioteki Angular Material (biały).

  &:hover {
```

```
    @include mat-elevation(4);
    transition: .3s;
  }
}

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

.tile__price-tag {
  color: mat-color($nga-foreground, text);
  font-size: 16px;
  font-weight: 700;
  position: absolute;
  right: 20px;
  top: 20px;
}

.tile__price-tag--xs {
  font-size: 14px;
}

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

.tile__title {
  color: mat-color($nga-foreground, text);
  font-family: 'Abril Fatface', cursive;
  font-size: mat-font-size($nga-typography, display-1);
  line-height: mat-line-height($nga-typography, display-1);
}

.tile__title--sm {
  font-size: mat-font-size($nga-typography, headline);
  line-height: mat-line-height($nga-typography, headline);
}

.tile__title--xs {
  font-size: mat-font-size($nga-typography, title);
  line-height: mat-line-height($nga-typography, title);
}
```

Jeśli wskaźnik myszy zostanie przesunięty na kafelek, spowoduje to uniesienie kafelka do poziomu 4. poprzez dodanie efektu cienia (zwracanego przez domieszkę mat-elevation).

Domyślny styl dla znacznika ceny produktu.

Styl dla znacznika ceny produktu dla bardzo małych rzutni.

Domyślny styl dla nazwy produktu.

Zgodnie ze specyfikacją Material Design używa display-1 dla stylów czcionki zamiast określać rozmiar na sztywno.

Styl dla nazwy produktu dla małych rzutni.

Styl dla nazwy produktu dla bardzo małych rzutni.

`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 modułem, 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 utwórzmy paczki programistyczne i zobaczmy, 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 ｢wdm｣: 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 któregokolwiek 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

E

F

G

H

I

L

Ł

M

N

O

P

R

S

W

Y

Z

Ż

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

ISBN 978-83-283-5666-5

9 788328 356665

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 89,00 zł