
# ASP.NET Core, Angular i Bootstrap

## Kompletny przybornik front-end developera

Simone Chiaretta

Helion

Tytuł oryginału: Front-end Development with ASP.NET Core, Angular, and Bootstrap

Tłumaczenie: Robert Górczyński

ISBN: 978-83-283-5194-3



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/aspnca.zip*

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

Printed in Poland.

# Spis treści

# 2

# Zestaw narzędzi programisty front-endu

**W tym rozdziale:**

- Rodzaje narzędzi używanych w programowaniu front-endu.
- Najważniejsze narzędzia w poszczególnych grupach.

W poprzednim rozdziale przedstawiłem krótkie omówienie frameworków działających po stronie serwera, ASP.NET Core i ASP.NET Core MVC. W tym przejdę do podstaw programowania front-endu i pokażę, kiedy będziesz potrzebował dodatkowych narzędzi, aby znacznie efektywniej wykonywać pracę.

W rozdziale skoncentruję się na wymienionych tutaj kategoriach narzędzi:

- **Frameworki JavaScript**. Pomagają one w utworzeniu skomplikowanych interfejsów internetowych dzięki zaoferowaniu programistom front-endu możliwości stosowania najlepszych praktyk typowych dla systemów działających po stronie serwera, np. wzorców **MVC** (ang. *model-view-controller*) i **MVVM** (ang. *model-view-view-model*), wstrzykiwania zależności (ang. *dependency injection*), routingu oraz wielu innych.
- **Frameworki CSS**. Programiści zwykle nie radzą sobie z zapewnianiem ładnego i spójnego wyglądu aplikacji internetowych. Frameworki CSS oferują zestawy stylów i komponentów interfejsu użytkownika, które można wykorzystać do zbudowania aplikacji internetowych, by wyglądały, jakby zostały zaprojektowane przez prawdziwego artystę. Pomagają również w pokonaniu problemów związanych z projektem responsywnym

dostosowującym się do różnych rozdzielności i wielkości ekranu, a także umożliwiają stosowanie skomplikowanych animacji i przejść.

- **Menedżery pakietów**. Systemy stały się teraz połączeniem różnych komponentów, z których część ma swoje zależności od jeszcze innych komponentów. Samodzielne zarządzanie tymi wszystkimi zależnościami i ich prawidłowymi wersjami byłoby koszmarem, gdyby nie menedżery pakietów.
- **Systemy kompilacji**. Jeżeli masz doświadczenie w programowaniu na czystej platformie .NET, prawdopodobnie przywykłeś do stosowania pewnych systemów kompilacji takich jak NAnt i MSBuild. W trakcie programowania front-endu również można stosować systemy kompilacji, które zostały opracowane szczególnie pod kątem zarządzania kompilacją rozwiązań front-endu.
- **Języki programowania**. Wykraczamy poza C# i VB.NET. Większość narzędzi zaliczanych do wcześniej omówionych kategorii zostało zbudowanych i musi być używanych w połączeniu z językiem JavaScript lub innym językiem specjalizowanym (ang. *domain-specific language*).

Ten rozdział zawiera ogólne omówienie najpopularniejszych narzędzi w poszczególnych kategoriach. Zacznę od przedstawienia niezbędnych podstaw, czyli dodatkowych języków programowania używanych podczas tworzenia front-endu.

# DODATKOWE JĘZYKI PROGRAMOWANIA, KTÓRE NALEŻY ZNAĆ

C#, standardowo działający po stronie klienta JavaScript i CSS to nie są jedyne języki wymagane do tworzenia aplikacji internetowych przez programistę front-endu. Wiele narzędzi przedstawionych w książce opiera się na jeszcze innej wersji JavaScriptu, Node.js, a także kolejnych językach specjalizowanych takich jak Sass i Less oraz JSON.

## Node.js

Tak naprawdę Node.js nie jest językiem programowania. To raczej platforma przeznaczona do tworzenia bardzo szybkich i skalowanych aplikacji internetowych. Oparta na środowisku uruchomieniowym JavaScript V8 — to jest ten sam silnik, który się znalazł w przeglądarce WWW Google Chrome.

Node.js to asynchroniczny i bazujący na zdarzeniach framework wraz z nieblokującymi operacjami wejścia-wyjścia, co w praktyce oznacza, że aplikacja nie wykorzystuje cykli procesora podczas oczekiwania na zakończenie operacji wejścia-wyjścia (odczyt lub zapis strumienia danych, pojawienie się pliku na dysku, nawiązanie połączenia HTTP, standardowe wyjście lub cokolwiek innego, co „strumieniuje" dane) lub w trakcie oczekiwania na inne zdarzenie.

Jeżeli jeszcze nigdy nie spotkałeś się z Node.js, na listingu 2.1 przedstawiłem utworzony za jego pomocą przykładowy program typu *Witaj, świecie!*. Działanie tego kodu polega na wczytaniu modułu `http`, utworzeniu obiektu `server` przez określenie funkcji przeznaczonej do wywołania po otrzymaniu odpowiedzi i rozpoczęciu nasłuchiwania połączeń HTTP na porcie 8080.

**Listing 2.1. Przykład aplikacji typu Witaj, świecie! utworzonej w Node.js**

```
var http = require('http');

var server = http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Witaj, programisto ASP.NET Core!\n');
});

server.listen(8080);

console.log('Serwer nasłuchuje na porcie http://127.0.0.1:8080/');
```

Jednak operacje wejścia-wyjścia nie są powiązane jedynie z HTTP. Oznaczają również konieczność odczytywania i zapisywania plików z dysku bądź strumieni pamięci, więc Node.js to popularne rozwiązanie w zakresie tworzenia narzędzi działających w powłoce (systemy z rodziny UNIX) i wierszu poleceń (Windows). Wspominam o Node.js w książce poświęconej tworzeniu aplikacji internetowych za pomocą ASP.NET, ponieważ większość najpopularniejszych narzędzi w świecie front-endu powstała właśnie w Node.js.

Kolejnym powodem używania Node.js jest to, że ten framework jest dostarczany wraz z niezwykle użytecznym menedżerem pakietów, **npm** (ang. *node package manager*), który przedstawię w dalszej części rozdziału, a dokładniej omówię w rozdziale 5.

## JSON

Ściśle mówiąc, **JSON** (ang. *javascript object notation*) nie jest językiem programowania, a raczej formatem wymiany danych, który ma być w prosty sposób analizowany i generowany przez maszynę, pozostając jednocześnie łatwym do odczytania i do utworzenia przez człowieka. Jak jego nazwa wskazuje, to w zasadzie jest forma serializacji obiektu JavaScriptu. Na listingu 2.2 przedstawiłem przykładowe dane w formacie JSON. Mają one postać obiektu wraz z właściwościami jako pary klucz-wartość, w których klucz zawsze jest ciągiem tekstowym, natomiast wartość może być ciągiem tekstowym, liczbą, wartością boolowską, innym obiektem (ujętym w nawias klamrowy) lub tablicą wartości ujętą w nawias kwadratowy.

**Listing 2.2. Przykładowe dane w formacie JSON**

```
{
  "imię":"Simone",
  "wiek": 42,
  "adres": {
    "miasto":"Bruksela",
    "kraj":"Belgia"
  },
  "hobby": [
    "trójbój",
    "programowanie",
    "puzzle"
  ],
  "zatrudnienie": true
}
```

Przetwarzanie danych JSON za pomocą funkcji JavaScriptu `eval()` spowoduje pobranie serializowanej w pliku struktury danych i jej bezpośrednie umieszczenie w pamięci. Jednak takie rozwiązanie nie jest zalecane, ponieważ funkcja `eval()` wykonuje wszystkie polecenia, co może stanowić pewne niebezpieczeństwo. JavaScript oferuje natywną funkcję przetwarzania danych w formacie JSON, `JSON.parse(daneJson)`, która po sprawdzeniu danych pozbywa się kodu o działaniu złośliwym lub niebezpiecznym i zwraca „oczyszczoną" strukturę danych.

To jest operacja przeciwna do tworzenia danych JSON na podstawie obiektu JavaScript, która również jest natywnie obsługiwana w JavaScripcie za pomocą funkcji o nazwie `JSON.stringify(nazwaObiektu)`.

Biorąc pod uwagę istniejącą w JavaScripcie natywną obsługę tworzenia i przetwarzania danych w formacie JSON, ten format jest wykorzystywany również do przekazywania danych między klientem a serwerem w używających wywołań w technologii AJAX aplikacjach w postaci pojedynczej strony.

Z powodu ogromnej czytelności wraz z upływem czasu JSON zaczął być używany także jako format plików konfiguracyjnych. W projektach ASP.NET Core preferowanym formatem konfiguracji jest właśnie JSON. Ponadto pliki konfiguracyjne wszystkich menedżerów pakietów również są tworzone w tym formacie.

## Sass i Less

Jeżeli kiedykolwiek pracowałeś z kaskadowymi arkuszami stylów (CSS), prawdopodobnie zauważyłeś, że na początku ich składnia wydaje się bardzo prosta. Jednak w przypadku niezachowania właściwej organizacji poszczególnych stylów obsługa takiego pliku to horror. Jeżeli chcesz zmienić kolor czegokolwiek, być może będziesz to musiał zrobić w wielu różnych definicjach klas. Ponadto zdefiniowanie wielkości pudełka zwykle wymaga przeprowadzenia pewnych operacji matematycznych, aby prawidłowo obliczyć dopełnienie, margines i wielkość obramowania.

Aby przezwyciężyć te problemy ponad pięć lat temu społeczność języka Ruby opracowała dwa metajęzyki pozwalające na ich kompilację do standardowego formatu CSS: **Sass** (ang. *syntactically awesome stylesheets*) i **Less** (ang. *leaner style sheets*). Dzięki zastosowaniu tego podejścia możliwe się stało wprowadzenie koncepcji takich jak zmienna, funkcja, domieszka i zagnieżdżenie i mimo tego otrzymanie w wyniku standardowego pliku CSS. Początkowo Sass i Less zostały opracowane jako narzędzia języka Ruby, dopiero później powstały ich kompilatory dla innych języków programowania. Obecnie Sass i Less mogą być zintegrowane właściwie w każdym stylu pracy i środowisku IDE, m.in. w Visual Studio.

Pokażę teraz, jak wybrane funkcje podstawowe mogą być zaimplementowane w obu językach, a następnie skonwertowane na CSS.

Zacznijmy od podstawowej funkcji tych języków, czyli obsługi zmiennych.

W preprocesorze Sass zmienna jest definiowana za pomocą prefiksu `$`:

```
$dark-blue: #3bbfce;
#header {
  color: $dark-blue;
```

```
}
h2 {
  color: $dark-blue;
}
```

Z kolei Less do zdefiniowania zmiennej używa znaku `@`:

```
@dark-blue: #3bbfce;

#header {
  color: @dark-blue;
}
h2 {
  color: @dark-blue;
}
```

Kod w obu językach zostanie skompilowany na następujące style CSS:

```
#header {
  color: #3bbfce;
}
h2 {
  color: #3bbfce;
}
```

Następną podstawową funkcją jest domieszka, co oznacza możliwość umieszczenia wielu właściwości CSS w wielu definicjach klas. Domieszki akceptują parametry.

W języku Sass koncepcja dołączania jest w oczywisty sposób odzwierciedlona w składni. Domieszka jest definiowana za pomocą słowa kluczowego `@mixin`, natomiast do jej użycia służy słowo kluczowe `@include`:

```
@mixin menu-border($width: 1px) {
  border-top: dotted $width black;
  border-bottom: solid $width*2 black;
}

#menu {
  @include menu-border
}

#side-menu {
  @include menu-border(2px)
}
```

Z kolei Less nie wprowadza żadnej nowej składni, a po prostu ponownie wykorzystuje standardowe składnie klas CSS, w zasadzie pozwalając, aby klasa była częścią innej klasy:

```
.menu-border(@width: 1px) {
  border-top: dotted @width black;
  border-bottom: solid @width*2 black;
}

#menu {
  .menu-border
}

#side-menu {
  .menu-border(2px)
}
```

Przedstawiony kod w obu językach zostanie skompilowany na następujące style CSS:

```
#menu {
  border-top: dotted 1px black;
  border-bottom: solid 2px black;
}

#side-menu {
  border-top: dotted 2px black;
  border-bottom: solid 4px black;
}
```

Preprocesor Sass ma znacznie wyraźniejszą składnię, podczas gdy Less w maksymalnym stopniu ponownie wykorzystuje standardowy CSS. Ostatecznie oba języki są do siebie podobne i wybór konkretnego zależy od preferencji.

Zachęcam Cię do dokładniejszego zapoznania się z funkcjami oferowanymi przez Sass i Less (informacje na ten temat znajdziesz w ich witrynach domowych) i samodzielnego eksperymentowania, aby ustalić, który lepiej pasuje do Twojego sposobu pracy. Wybór właściwie sprowadza się do frameworka CSS, z którego zamierzasz korzystać. Z czterech przedstawionych w rozdziale dwa (Primer CSS i Material Design Lite) używają Sass, a dwa kolejne (Bootstrap CSS i Semantic UI) Less.

## Przyszłość języka JavaScript

JavaScript został oparty na ewoluującym standardzie, ECMAScript, który w 2015 r. osiągnął wersję 6 często określaną mianem ES6. Ta wersja przynosi nowe i interesujące funkcje, takie jak klasy (wraz z konstruktorami metodami setter i getter oraz z dziedziczeniem), moduły, procedury wczytywania modułów, składnię „strzałki" (wyrażenia lambda C#), standardowe struktury danych takie jak mapa, zbiór itd.

Wprawdzie najważniejsze przeglądarki WWW w ich najnowszych wydaniach implementują funkcje ES6, ale w starszych wersjach są one niedostępne. Dlatego też prawdopodobnie upłynie nieco czasu, zanim specyfikacja ES6 będzie powszechnie stosowana w programowaniu aplikacji internetowych. Podobnie jak Sass i Less pomogły w pokonaniu pewnych ograniczeń CSS, tak samo istnieją metajęzyki implementujące część nowej specyfikacji ES6. Jednym z nich jest TypeScript.

## TypeScript

W oczekiwaniu na wprowadzenie funkcji ES6 w różnych silnikach JavaScriptu Microsoft wydał TypeScript. Ten język wprowadził obsługę klas, modułów i składnię strzałki zgodnie z propozycjami zawartymi w ES6, a także inne koncepcje obecnie niedostępne w JavaScripcie, np. ścisłe typowanie, interfejsy, typy generyczne itd.

Trzeba wyraźnie podkreślić, że to nie jest przygotowana przez Microsoft wersja języka JavaScript. Podobnie jak w przypadku kodu Sass i Less kompilowanego na postać standardowego CSS, kod TypeScript jest kompilowany na standardowy JavaScript. Ponadto kompilator przeprowadza analizę statyczną i zgłasza potencjalne błędy.

Jak wcześniej wspomniałem, jedną z cech TypeScriptu jest ścisłe typowanie. W rzeczywistości to tylko rodzaj adnotacji, która jest sprawdzana w trakcie kompilacji.

```
function add(x: number, y: number): number {
  return x+y;
}

var sum = add(2,3);
var wrongSun = add("witaj","świecie");
```

Pierwsze wywołanie funkcji `add()` jest prawidłowe. Kompilacja odbywa się poprawnie, a po jej wykonaniu zwracany jest prawidłowy wynik.

Z kolei drugie wywołanie jest błędne. Wprawdzie zostanie wykonane poprawnie, ale nie będzie skompilowane, ponieważ w trakcie analizy statycznej okazuje się, że `witaj` to nie jest wartość liczbowa oczekiwana przez funkcję `add()`.

Na listingu 2.3 pokazałem przykładową klasę zdefiniowaną w TypeScripcie wraz z konstruktorem, metodami publicznymi, polami prywatnymi i akcesorami.

**Listing 2.3. Przykładowa klasa w języku TypeScript**

```
class Greeter {
  public greeting: string;
  private _name: string;

  constructor(message: string) {
    this.greeting = message;
  }

  greet() {
    return this.greeting + ", " + this._name+"!";
  }

  get name(): string {
    return this._name;
  }

  set name(newName: string) {
    this._name = newName;
  }
}

let greeter = new Greeter("Witaj");
greeter.name="świecie";

alert(greeter.greet()); // Dane wyjściowe to "Witaj, świecie!"
```

W tym fragmencie kodu po prostu zaprezentowałem kilka funkcji. Zachęcam Cię do znacznie dokładniejszego poznania TypeScriptu, ponieważ za pomocą tego języka są tworzone aplikacje JavaScriptu w Angularze, którego użycie pozwala na wykorzystanie pełni możliwości tego frameworka.

# FRAMEWORKI JAVASCRIPTU

Skoro nigdy nie tworzysz działającej po stronie serwera aplikacji przez samodzielną obsługę żądań HTTP i odpowiedzi na nie, tak samo nie definiujesz interakcji po stronie klienta przez bezpośrednią modyfikację modelu DOM i zarządzanie stanem aplikacji w prostych klasach JavaScriptu. Do tego celu są przeznaczone frameworki aplikacji JavaScript, takie jak Angular, React i in.

Jeżeli w branży jesteś od co najmniej kilku lat, być może zauważyłeś pojawianie się i znikanie różnych frameworków JavaScriptu, raz szybciej, innym razem wolniej. W paru kolejnych sekcjach przedstawię krótkie omówienie aktualnie najpopularniejszych frameworków JavaScriptu. Na podstawie sposobu ich współdziałania z innymi frameworkami i względnie długiej już dostępności można zaryzykować stwierdzenie, że pozostaną one z nami przynajmniej przez pewien czas.

## Angular

Framework Angular jest aktualnie rozwijany przez Google wraz ze społecznością programistów zarówno pojedynczych, jak i korporacyjnych. Angular to framework działający po stronie klienta i zbudowany na bazie idei rozszerzenia kodu HTML za pomocą nowych elementów nazywanych *komponentami* internetowymi, które definiują dodatkowe zachowanie. Komponentami mogą być np. atrybuty HTML. Mają powiązane z nimi szablony generujące dane komponentu za pomocą wyrażeń umieszczonych w podwójnym nawiasie klamrowym {{}}. Na listingu 2.4 przedstawiłem główne komponenty prostej aplikacji Angulara wykorzystującej dwukierunkowe łączenie danych. Zwróć uwagę na to, że aplikacja została podzielona na kilka plików.

**Listing 2.4. Przykładowa aplikacja utworzona za pomocą frameworka Angular**

*index.html: główny plik HTML.*

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Witaj, Angular!</title>
  <base href="/">
</head>
<body>
  <app-root>Wczytywanie danych...</app-root>
</body>
</html>
```

*main.ts: plik uruchamiający aplikację.*

```
import './polyfills.ts';

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

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

*app.module.ts: plik definiujący moduł aplikacji.*

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';

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

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

*app.component.ts: plik komponentu aplikacji.*

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

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

  public firstName: string = "Simone";
  public lastName: string = "Chiaretta";

 fullName() {
    return `${this.firstName} ${this.lastName}`;
  }
}
```

*app.component.html: szablon pliku komponentu aplikacji.*

```
<form>
  <div>
    <label for="firstName">Imię:</label>
    <input name="firstName" [(ngModel)]="firstName">
  </div>
  <div>
    <label for="lastName">Nazwisko:</label>
    <input name="lastName" [(ngModel)]="lastName">
  </div>
</form>
<hr/>
<h1>Witaj, <span>{{ fullName() }}</span>!</h1>
```

Wszystko rozpoczyna się od elementu `<app-root>` definiującego komponent główny, w którym zostanie uruchomiona aplikacja zdefiniowana w pliku *main.ts*. Następnie możesz zobaczyć podział aplikacji na trzy pliki komponentu `app`:

- *app.module.ts* — zadanie tego pliku polega m.in. na zdefiniowaniu wszystkich komponentów aplikacji;
- *app.component.ts* — w tym pliku jest zdefiniowany rzeczywisty komponent (wraz z elementem `<html>`, szablonem i stylami) i sposób jego działania;
- *app.component.html* — w tym pliku znajduje się szablon zawierający kod znaczników HTML wygenerowanych przez komponent.

Kolejną ważną dyrektywą jest `[(ngModel)]` — powoduje ona połączenie elementu formularza z właściwościami modelu komponentu.

Jak pewnie już zauważyłeś, kod JavaScriptu aplikacji Angulara jest tworzony za pomocą języka TypeScript.

W tym miejscu przedstawiłem jedynie absolutne podstawy, framework Angular oferuje znacznie więcej możliwości: wstrzykiwanie zależności, szablony, routing, moduły, testowanie oraz definiowanie własnych dyrektyw. Wszystkie te możliwości szczegółowo omówię w następnym rozdziale.

## Knockout

Knockout to framework JavaScriptu szczególnie popularny w świecie programistów korzystających z technologii opracowanych przez Microsoft. Pierwotnie utworzony przez programistę Microsoftu Steva Sandersona, ten framework implementuje wzorzec MVC. Pod pewnymi względami jego składnia jest bardzo podobna do stosowanej w Angularze, nawet mimo że oferuje mniej funkcji i wymaga większej ilości pracy do zdefiniowania właściwości zapewniających dwukierunkowe przekazywanie danych. Knockout obsługuje również szablony, co pozwala na używanie tych samych fragmentów kodu w całej aplikacji.

Na listingu 2.5 przedstawiłem przykładową aplikację typu *Witaj, świecie!* utworzoną w Knockoucie. To jest wersja aplikacji z poprzedniej sekcji.

**Listing 2.5. Przykładowa aplikacja utworzona za pomocą frameworka Knockout**

```
<!doctype html>
<html>
  <head>
    <title>Witaj, Knockout!</title>
    <script src="knockout-min.js"></script>
  </head>
  <body>
  <div>
    <p>Imię: <input data-bind="value: firstName" /></p>
    <p>Nazwisko: <input data-bind="value: lastName" /></p>
    <hr>
    <h1>Witaj, <span data-bind="text: fullName"></span>!</h1>
    </div>
  </body>
```

```
    <script>
    function ViewModel() {
        this.firstName = ko.observable("Simone");
        this.lastName = ko.observable("Chiaretta");
        this.fullName = ko.computed(function() {
            return this.firstName() + " " + this.lastName();
        }, this);
    }

    ko.applyBindings(new ViewModel());
    </script>

</html>
```

Komponent główny tej prostej aplikacji znajduje się w funkcji `ViewModel()` definiującej właściwości modelu widoku za pomocą funkcji `ko.observable()` i `ko.computed()`. Pierwsza z nich wskazuje frameworkowi, że dana właściwość ma być „obserwowana" i powinna zostać uznana za mechanizm dwukierunkowego przekazywania danych. Natomiast druga definiuje właściwość zależną od innej i uaktualnia ją po każdej zmianie tej innej właściwości.

Do połączenia właściwości z interfejsem użytkownika jest używany atrybut `data-bind`:

- Wiązanie `value` zostało użyte w elementach formularza i powoduje powiązanie wartości elementu z właściwością w modelu widoku.
- Wiązanie `text` jest używane wszędzie tam, gdzie zachodzi potrzeba wyświetlenia wartości właściwości wyrażenia.

Łącznikiem między szablonem HTML a modelem widoku jest w omawianym przykładzie ostatni wiersz kodu, czyli polecenie `ko.applyBindings(new ViewModel());`.

Framework Knockout jest łatwiejszy do poznania niż Angular, choć jednocześnie oferuje mniejsze możliwości, a jego dalszy rozwój ostatnio odbywa się nieco wolniej. Dlatego też nie będę go dokładniej omawiał w książce. Jeżeli chcesz utworzyć prostszą aplikację niewymagającą potężnych możliwości (i poziomu skomplikowania) frameworka Angular, zachęcam Cię do zapoznania się z informacjami, które znajdziesz w oficjalnej witrynie internetowej Knockouta i zamieszczonym tam samouczkiem.

## React

Kolejnym frameworkiem JavaScriptu wartym wzmianki jest React. Został opracowany i jest rozwijany przez Facebooka. To biblioteka JavaScriptu przeznaczona do tworzenia interfejsu użytkownika bez konieczności zajmowania się pozostałą częścią aplikacji. React jest oparty na koncepcji oddzielnych komponentów generujących niezbędny kod HTML i opcjonalnie może zarządzać także ich wewnętrznym stanem. Przykład prostej aplikacji Reacta przedstawiłem na listingu 2.6. Zwróć uwagę na to, że aplikacja została podzielona na kilka plików.

**Listing 2.6. Przykładowa aplikacja typu Witaj, świecie! utworzona za pomocą biblioteki Reacta**

*index.html: główny plik HTML.*

```
<!doctype html>
<html lang="en">
  <head>
    <title>Witaj, React!</title>
  </head>
  <body>
    <div id="greet"></div>
  </body>
</html>
```

*index.js: plik uruchamiający aplikację.*

```
import React from 'react';
import ReactDOM from 'react-dom';
import Greeter from './Greeter';

ReactDOM.render(
  <Greeter firstName="Simone" />,
  document.getElementById('greet')
);
```

*greeter.js: plik komponentu aplikacji.*

```
import React, { Component } from 'react';

class Greeter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {firstName: props.firstName, lastName: props.lastName};

    this.handleFirstNameChange = this.handleFirstNameChange.bind(this);
    this.handleLastNameChange = this.handleLastNameChange.bind(this);
  }

  handleFirstNameChange(event) {
    this.setState({firstName: event.target.value});
  }

  handleLastNameChange(event) {
    this.setState({lastName: event.target.value});
  }

  render() {
    return (
      <div>
      <form>
        <div>
          <label>
            Imię:
            <input type="text" value={this.state.firstName}
            ↪onChange={this.handleFirstNameChange} />
          </label>
        </div>
```

```
          <div>
            <label>
              Nazwisko:
              <input type="text" value={this.state.lastName}
              ↪onChange={this.handleLastNameChange} />
            </label>
          </div>
        </form>
        <hr/>
        <h1>Witaj, <span>{this.state.firstName} {this.state.lastName}</span>!</h1>
        </div>
      );
    }
}

export default Greeter;
```

Jak możesz zobaczyć w przykładzie przedstawionym na listingu 2.6, kod jest bardziej skomplikowany niż w wersji opartej na Knockoucie i nieco podobny do wersji używającej Angulara. Wymaga konwersji kodu na komponent, także w przypadku niewielkich formularzy, takiego jak w omawianym przykładzie. Nawet przy pomocy „dziwnej" składni łączącej JavaScript z XML i nazywanej JSX (spójrz na metodę `render()`), użytej do uproszczenia wygenerowania elementów HTML, React nadal przeprowadza bezpośrednie operacje na modelu DOM (w rzeczywistości to wirtualny model DOM).

React ma taką składnię, ponieważ ta biblioteka została zaprojektowana do obsługi treści dynamicznej Facebooka, gdzie dane wejściowe użytkownika to niewielki fragment interakcji, a setki elementów jest dynamicznie dodawanych do stron. Aby to zrobić w maksymalnie najszybszy sposób, konieczne okazało się bezpośrednie manipulowanie obiektami modelu DOM bez przetwarzania szablonów i stosowania dwukierunkowego wiązania danych. Jeżeli masz podobną sytuację, ale wraz ze standardową aplikacją stosującą REST, wówczas prawdopodobnie lepszym rozwiązaniem będzie użycie Angulara.

> **UWAGA** Na listingu 2.6 możesz zauważyć „dziwną" składnię na początku dwóch plików JavaScriptu: `import * from *`. To jest funkcja specyfikacji ES6 używana do definiowania i importowania modułów. W narzędziach kompilacji kodu React ta składnia zostaje za pomocą kompilatora Babel skonwertowana na „standardowy" kod JavaScriptu.

## jQuery

Ostatnim wartym wzmianki frameworkiem JavaScriptu jest jQuery, czyli prawdopodobnie najbardziej znana i najczęściej używana biblioteka JavaScriptu. Ułatwia ona pobieranie elementów HTML, przeprowadzanie operacji na modelu DOM, obsługę zdarzeń, animacji i wywołań w technologii AJAX. Do tego celu wykorzystuje API zapewniające abstrakcję różnic implementacji między poszczególnymi przeglądarkami WWW. Biblioteka jQuery została zaprezentowana w 2006 r., gdy powszechną praktyką było wielokrotne tworzenie tych samych funkcji, aby poradzić sobie z różnicami między przeglądarkami WWW.

jQuery nie jest pełnym frameworkiem aplikacji takim jak Angular, Knockout i React. To raczej biblioteka narzędziowa pomagająca w przygotowaniu interaktywnych interfejsów w HTML. Dlatego też w książce nie przedstawię dokładniejszego omówienia jQuery.

> **UWAGA** Oto bardzo spójne i efektywne wyjaśnienie różnicy między frameworkiem i biblioteką: framework wywołuje utworzony przez Ciebie kod, podczas gdy Twój kod wywołuje bibliotekę.

# FRAMEWORKI CSS

Programiści front-endu i projektanci nie lubią wyważać otwartych drzwi w każdym projekcie, nad którym pracują. Dlatego też jeszcze kilka lat temu projektanci internetowi we wszystkich projektach korzystali z tego samego małego i samodzielnie opracowanego zbioru klas CSS i fragmentów kodu HTML. W 2010 r. zespół pracujący nad serwisem Twitter zdecydował się na publiczne udostępnienie opracowanej na własne potrzeby biblioteki CSS. Później inne firmy i zespoły zdecydowały się na taki sam krok, choć tylko niewiele z nich mogło rywalizować z Bootstrapem.

W kilku kolejnych sekcjach przedstawię krótkie wprowadzenie do Bootstrapa, a także zaprezentuję opracowany przez GitHub framework CSS o nazwie Primer, względnie nowy, choć niezwykle obiecujący Material Design Lite opracowany przez Google, a także Semantic UI, czyli framework charakteryzujący się zorientowanym na komponenty podejściem do CSS.

## Bootstrap

Najpopularniejszym frameworkiem CSS pozostaje ten wydany pierwotnie jako Twitter Blueprint. Nazwa wzięła się stąd, że został opracowany po to, aby zapewnić spójność różnych witryn internetowych należących do Twittera. Później jego nazwa została zmieniona na Bootstrap i framework wydano latem 2011 r. jako projekt typu open source. Okazał się najpopularniejszą biblioteką w serwisie GitHub, która zebrała ponad 100 tys. gwiazdek.

Bootstrap zawiera zbiór szablonów opartych na CSS i HTML przeznaczonych do nadawania stylu formularzom, elementom, przyciskom, nawigacji, typografii i wielu innym komponentom interfejsu użytkownika. Jest dostarczany wraz z opcjonalnymi wtyczkami JavaScriptu pozwalającymi na dodanie interaktywności komponentom.

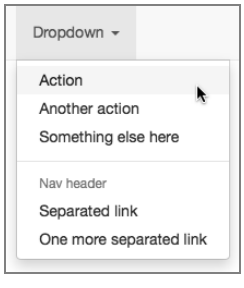
Bootstrap to framework typu „najpierw urządzenia mobilne”, oparty na dwunastokolumnowym układzie siatki pozwalającym rozmieszczać komponenty na ekranie. Spójrz na przedstawiony tutaj przykład siatki automatycznie dopasowującej się do wielkości ekranu.

```
<div class="row">
  <div class="col-xs-12 col-sm-6 col-md-8">.col-xs-12 .col-sm-6 .col-md-8</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>
```

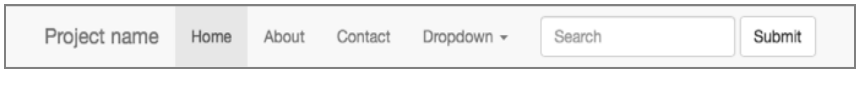
To jest przykład siatki responsywnej wyświetlanej odmiennie na ekranach o różnej wielkości:

- Na ekranie tradycyjnego komputera te dwie komórki zostaną wyświetlone obok siebie. Pierwsza składa się z ośmiu kolumn, natomiast druga z czterech. (Zachowanie siatki dla ekranów o zwykłej wielkości jest definiowane przez klasy o nazwach rozpoczynających się prefiksem `col-md-`).
- Na malutkim ekranie smartfona (siatka zdefiniowana przez klasy rozpoczynające się prefiksem `col-xs-`) pierwsza komórka będzie zajmowała pełną szerokość, natomiast druga znajdzie się w nowym wierszu i zajmie połowę szerokości ekranu.
- Na ekranie tableta (siatka zdefiniowana przez klasy rozpoczynające się prefiksem `col-sm-`) pierwsza komórka użyje tylko sześciu kolumn, natomiast druga dziedziczy definicję wielkości dla małego ekranu. Dlatego też każda z komórek będzie zajmowała połowę szerokości ekranu.

Jeżeli zastanawiasz się, jak te komponenty wyglądają, spójrz na serwis Twitter. Wygląd i sposób działania aplikacji w pełni opracowanej z użyciem frameworka Bootstrap jest podobny do tego sławnego serwisu społecznościowego. Spójrz na rysunki 2.1 i 2.2.

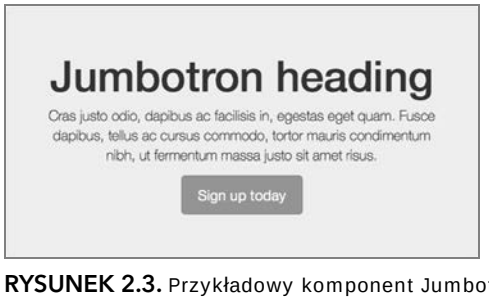



**RYSUNEK 2.1.** Przykład rozwijanego menu utworzonego za pomocą frameworka Bootstrap




**RYSUNEK 2.2.** Pasek nawigacyjny utworzony z użyciem Bootstrapa

Pomijając standardową nawigację i menu, jednym z interesujących komponentów jest Jumbotron pokazany na rysunku 2.3. Jest przeznaczony do użycia w charakterze najważniejszego nagłówka, który ma przyciągnąć uwagę odwiedzającego stronę.




**RYSUNEK 2.3.** Przykładowy komponent Jumbotron w Bootstrapie

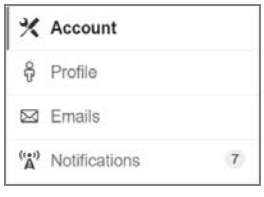
Oczywiście można zmienić styl i utworzyć własny motyw wraz z wybranymi kolorami. To się odbywa przez zmianę pewnych zmiennych w plikach Bootstrapa, a następnie ponowną kompilację pliku CSS lub skorzystanie ze strony *Customize Bootstrap* w oficjalnej witrynie internetowej Bootstrapa.

Przedstawiłem tutaj zaledwie krótkie wprowadzenie do tego oferującego potężne możliwości frameworka CSS. Jego dokładniejsze omówienie znajdziesz w rozdziale 4.

Bootstrap to nie jedyny framework CSS, pojawiło się (i zniknęło) także wiele innych, najczęściej opartych na siatce. Warto wspomnieć o jeszcze trzech frameworkach CSS, a pierwszym z nich jest Primer CSS opracowany przez GitHub.

## Primer CSS

GitHub swoje wewnętrzne zalecenia projektowe wydał w postaci projektu typu open source o nazwie Primer CSS. Ten framework nie jest tak w pełni wyposażony jak Bootstrap. Dlatego też pomimo jego oparcia na systemie siatki nie zapewnia responsywności. Jeżeli lubisz stosowane przez GitHub podejście do projektowania interfejsu użytkownika, ten framework możesz uznać za łatwy w użyciu i elegancko wykonany. Znajdziesz w nim również sławne ikony, *octicons*, jak pokazałem na rysunku 2.4.




**RYSUNEK 2.4.** Przykład paska nawigacyjnego utworzonego za pomocą Primer CSS

Interesujący komponent nosi nazwę *blank slate*, powinien być używany wtedy, gdy element treści nie zawiera niczego do wyświetlenia. Przykład tego komponentu pokazałem na rysunku 2.5.


**RYSUNEK 2.5.** Komponent blank slate

Ponieważ styl jest kwestią osobistego wyboru, więc jeśli nie jesteś fanem stylu serwisów Twitter i GitHub, być może polubisz opracowany przez Google framework Material Design Lite.

## Material Design Lite

Material Design Lite to opracowany przez Google framework CSS, którego celem miało być przyniesienie filozofii Material Design do internetu. W przeciwieństwie do frameworków Bootstrap i Primer CSS Material Design Lite to połączenie CSS i JavaScriptu, w którym to

połączeniu styl elementów i klasy frameworka są w trakcie działania wzbogacane przez bibliotekę JavaScriptu dodającą zachowanie do komponentów. Jak możesz zobaczyć na przykładowych komponentach pokazanych na rysunkach 2.6 i 2.7, ten projekt jest bardzo podobny z wyglądu i sposobu działania do aplikacji na platformie Android.


**RYSUNEK 2.6.** Przykład nawigacji utworzonej za pomocą Material Design Lite


**RYSUNEK 2.7.** Przyciski utworzone za pomocą Material Design Lite

W oficjalnej witrynie internetowej znajdziesz przykłady stron utworzonych za pomocą frameworka Material Design Lite, które to przykłady można wykorzystać jako punkt wyjścia do własnego projektu.

## Semantic UI

Ostatnim frameworkiem CSS wartym wzmianki jest Semantic UI. Jak jego nazwa wskazuje, nadaje klasom CSS nazwy łatwiejsze do zrozumienia niż stosowane w innych frameworkach. Dlatego też aby przyciskowi nadać styl wraz z kolorem podstawowym, należy użyć znacznika `<button class="ui primary button">`.

Semantic UI oferuje układ responsywny zbudowany na podstawie siatki składającej się z szesnastu kolumn.

```
<div class="ui grid">
  <div class="four wide column"></div>
  <div class="four wide column"></div>
  <div class="four wide column"></div>
  <div class="four wide column"></div>
  <div class="two wide column"></div>
  <div class="eight wide column"></div>
  <div class="six wide column"></div>
</div>
```

Nazwy języka naturalnego to jedynie wierzchołek głębokich i niemal filozoficznych powodów prowadzących do powstania Semantic UI. Celem tego frameworka było zmniejszenie barier technicznych między koncepcjami programowania a odpowiadającymi im koncepcjami w otaczającym nas świecie.

Wprawdzie Semantic UI jest dostarczany wraz z motywem domyślnym, ale istnieją również dodatkowe motywy pozwalające na nadanie wyglądu i sposobu działania frameworków Bootstrap, Primer CSS i Material Design.

Na tym kończę przedstawienie Semantic UI w książce. Jeżeli zainteresowało Cię podejście zastosowane w tym frameworku, więcej informacji na jego temat znajdziesz w witrynie *http://learnsemantic.com/*.

# MENEDŻERY PAKIETÓW

Skoro do opracowania nowoczesnej aplikacji internetowej za pomocą ASP.NET Core MVC potrzebnych jest wiele komponentów, bibliotek i narzędzi, konieczne jest jeszcze użycie pewnego rozwiązania pozwalającego na elegancką organizację całości, automatyzację instalowania i aktualizacji poszczególnych komponentów i ich zależności. Do tego celu są używane menedżery pakietów. Działanie menedżera polega na pobraniu komponentów i narzędzi z oficjalnych repozytoriów, zarządzanie wszystkimi zależnościami i łatwe przygotowanie lokalnej kopii środowiska programistycznego projektu przez po prostu pobranie danych z repozytorium źródłowego.

W książce przedstawię wymienione tutaj menedżery pakietów:

- NuGet przeznaczony do zarządzania bibliotekami i komponentami .NET;
- Bower przeznaczony do frameworków JavaScript i CSS;
- npm przeznaczony do narzędzi i bibliotek JavaScriptu działających po stronie serwera.

Bower jest przeznaczony do obsługi zależności po stronie klienta, natomiast NuGet i npm mogą być używane do wszystkiego po trochu i właśnie w taki sposób korzystam z nich w książce.

## NuGet

NuGet to domyślny menedżer pakietów w świecie programowania na platformie .NET i został zintegrowany wraz z wieloma wydaniami Visual Studio. W przypadku ASP.NET Core wszystkie odwołania do projektu są zapisywane jako `PackageReference` w pliku definicji projektu (*.csproj*), jak pokazałem na listingu 2.7. Wszystko, łącznie z odwołaniami do podstawowych bibliotek *Base Class Library* (wszystkie zgrupowane w metapakiecie `Microsoft.AspNet.Core.All`), jest pobierane za pomocą pakietów NuGeta.

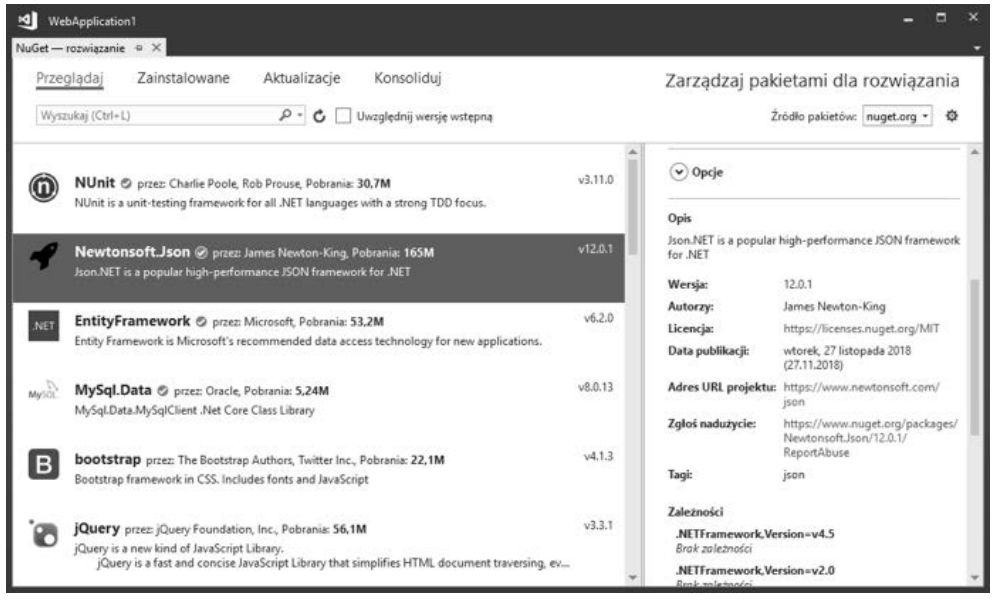
**Listing 2.7. Przykład pliku WebApplication.csproj**

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.0" />
    <PackageReference Include="Newtonsoft.Json" Version="10.0.3" />
  </ItemGroup>
```

```
  <ItemGroup>
    <DotNetCliToolReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Tools"
    ↪Version="2.0.0" />
  </ItemGroup>
</Project>
```

Pomijając ręczne dodawanie pakietów do pliku *.csproj*, jak to się odbywało w poprzednich wersjach, pakiety mogą być instalowane za pomocą opracowanego zupełnie od nowa interfejsu użytkownika menedżera pakietów (zob. rysunek 2.8) lub konsoli menedżera pakietów:

```
PM> Install-Package Newtonsoft.Json
```




**RYSUNEK 2.8.** Interfejs menedżera pakietów w Visual Studio 2017

Jeżeli używałeś menedżera pakietów NuGet w poprzednich wersjach ASP.NET, to w ASP.NET Core zauważysz ogromną różnicę koncepcyjną pod tym względem, ponieważ tylko zależności po stronie serwera powinny być wskazywane i pobierane za pomocą NuGeta. Natomiast dla zależności po stronie klienta Microsoft zdecydował się na użycie innego bardzo popularnego menedżera pakietów — Bowera, który został opracowany właśnie w tym celu.

## Bower

Bower to bardzo proste w użyciu narzędzie. Podobnie jak w przypadku NuGeta, do działania potrzebuje pliku w formacie JSON, *bower.json*, wymieniającego listę pakietów używanych w projekcie. Przykład takiego pliku konfiguracyjnego przedstawiłem na listingu 2.8.

**Listing 2.8. Plik konfiguracyjny bower.json**

```
{
  "name": "asp.net",
  "private": true,
  "dependencies": {
    "bootstrap": "3.3.6",
    "jquery": "2.2.0",
    "jquery-validation": "1.14.0",
    "jquery-validation-unobtrusive": "3.2.6",
    "jquery-file-upload":"https://github.com/blueimp/jQuery-File-Upload/"
  }
}
```

Jak możesz zobaczyć na listingu 2.8, pakiety są podawane wraz z informacjami dodatkowymi:

- Przede wszystkim konieczne jest podanie nazwy pakietu w postaci, w jakiej została ona zarejestrowana w witrynie *https://bower.io/*. Odwołując się do pakietu w taki właśnie sposób, Bower pobiera całe repozytorium git podane podczas rejestracji pakietu.
- Inną możliwością jest bezpośrednie podanie repozytorium git lub svn, z którego ma zostać pobrany pakiet.
- Istnieje również możliwość podania standardowego adresu URL. W takim przypadku pakiet zostanie pobrany z tego adresu URL (i rozpakowany, jeśli adres prowadził do archiwum).

Po wydaniu w konsoli polecenia `bower install` wszystkie pakiety zostaną pobrane bezpośrednio z podanych lokalizacji i umieszczone w katalogu o nazwie *bower_components*. Na tym kończy się działanie menedżera Bower. Sposób użycia pobranych pakietów zależy całkowicie od programisty. Do tych plików można się odwoływać bezpośrednio lub — co jest zalecane, aby zachować porządek — skopiować niezbędne pliki do struktury projektu. (Pamiętaj, że narzędzie Bower mogło pobrać całe repozytoria git).

## npm

Menedżer pakietów Node.js (ang. *node package manager*, **npm**) został pierwotnie opracowany do zarządzania pakietami Node.js działającymi po stronie serwera. Później zyskał popularność w zakresie rozprowadzania narzędzi powłoki (wiersza poleceń) opracowanych za pomocą Node.js. Podobnie jak w przypadku innych menedżerów, npm pobiera pakiety wymienione w pliku manifestu o nazwie *package.json*, a następnie instaluje je w podkatalogu projektu o nazwie *node_modules*.

Wprawdzie pakiety powinny być wymieniane we węźle `dependencies`, ale w kontekście projektu ASP.NET Core ten menedżer jest używany przede wszystkim do instalacji menedżera zadań (ang. *task runner*) i jego wtyczek, więc w tym przypadku pakiety są wymieniane w węźle `devDependencies`, jak pokazałem na listingu 2.9.

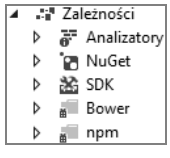
**Listing 2.9. Przykładowy plik package.json**

```
{
  "name": "app",
  "version": "1.0.0",
```

```
  "private": true,
  "devDependencies": {
    "del": "^2.2.2",
    "gulp": "^3.9.1",
    "gulp-concat": "^2.6.1",
    "gulp-cssmin": "^0.1.7",
    "gulp-htmlmin": "^3.0.0",
    "gulp-uglify": "^2.0.0",
    "merge-stream": "^1.0.1"
  }
}
```

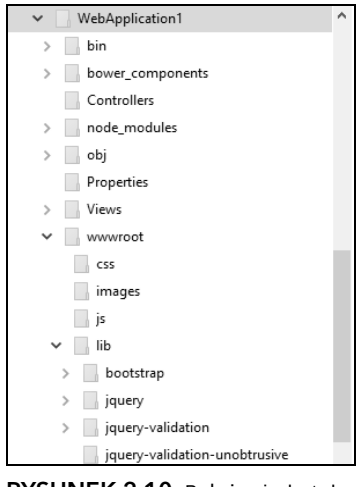
## Struktura katalogów

Aby zakończyć sekcję poświęconą menedżerom pakietów, chciałbym pokazać, jak te wszystkie odwołania i pliki manifestu przedstawiają się w oknie *Eksplorator rozwiązań* i w systemie plików. W zasadzie każdy projekt może mieć trzy rodzaje zależności zdefiniowane w odpowiednich plikach manifestu: *.csproj* dla odwołań pakietów NuGet działających po stronie serwera, *bower.json* dla komponentów Bowera działających po stronie klienta i *package.json* dla narzędzi kompilacji. Na rysunku 2.9 pokazałem okno *Eksplorator rozwiązań* wraz z wszystkimi zależnościami w drzewie projektu.




**RYSUNEK 2.9.** Zależności wyświetlone w oknie Eksplorator rozwiązań

W systemie plików pakiety są przechowywane w różnych podkatalogach: komponenty Bowera w *wwwroot/libs* (Visual Studio przechowuje je w innym położeniu niż domyślne), zaś pakiety npm w *node_modules*. Spójrz na rysunek 2.10.




**RYSUNEK 2.10.** Położenie katalogów zależności w systemie plików

# MENEDŻERY ZADAŃ

Menedżery zadań automatyzują ostatni krok w trakcie pracy nad projektem, jakim jest kompilacja i wydanie aplikacji. To nic nowego w świecie programowania po stronie serwera. Prawdopodobnie przez lata stosowałeś automatyzację kompilacji za pomocą skryptów MSBuild lub zadań NAnt, natomiast w świecie front-endu koncepcja menedżera zadań jest całkiem nowa.

W tym momencie mógłbyś zadawać sobie pytanie, jakie są powody używania tej nowości w świecie front-endu. To całkiem sensowne pytanie. Największą zaletą menedżera zadań jest to, że dzięki niemu front-end pozostaje całkowicie niezależny od języka programowania po stronie serwera. Dlatego też te menedżery mogą być używane przez każdego i stoją za nimi większe społeczności, które opracowują gotowe do użycia zadania. Jednak na ostatnim etapie pracy nad projektem możesz wykorzystać swoje wcześniejsze doświadczenie. Jak już wspomniałem w rozdziale 1., definicja projektu ASP.NET Core wciąż jest tworzona za pomocą MSBuild i nadal może być używana do kompilowania aplikacji.

> **UWAGA** Ostatnio część społeczności programistów front-endu, głównie używająca systemów operacyjnych z rodziny Linux lub macOS, całkowicie przestała używać menedżerów zadań i zamiast nich wykorzystuje funkcję npm o nazwie *skryptów npm*. Taki skrypt to po prostu rozwiązanie pozwalające na wywoływanie poleceń systemu operacyjnego, aplikacji Node.js utworzonych specjalnie na potrzeby budowanej aplikacji lub narzędzi programistycznych dostarczanych wraz z używanymi przez nie frameworkami takimi jak Angular i React.

Najważniejszym graczem w tej kategorii jest Gulp. Jego działanie opiera się na podstawie kodu aplikacji i wykorzystuje naprawdę niewielkie powiązane ze sobą wtyczki zamiast oddzielnych zadań wykonywanych kolejno. Jeżeli to brzmi niezrozumiale, spójrz na listing 2.10 przedstawiający przykładowy plik konfiguracyjny Gulp.

**Listing 2.10. Przykładowy plik konfiguracyjny menedżera zadań Gulp**

```
var gulp = require('gulp');
var jshint = require('gulp-jshint');
var concat = require('gulp-concat');
var minifyCss = require('gulp-minify-css');

gulp.task('default', function(done){
  gulp.src('src/**/*.js')
    .pipe(jshint())
    .pipe(concat('bundle.js'))
    .pipe(gulp.dest('dist '))
    .on('end', done);
});

gulp.watch('src/**/*.js', ['default']);
```

To po prostu jest kod. Zadanie rozpoczyna się od wskazania za pomocą `gulp.src()` pliku kodu źródłowego, który powinien zostać przetworzony. Następnie poszczególne operacje są łączone ze sobą za pomocą funkcji `pipe()`, a wynik końcowy jest za pomocą wywołania `gulp.dest()` zapisywany w pliku.

Tutaj przedstawiłem jedynie krótkie wprowadzenie do menedżera zadań Gulp. Więcej informacji na ten temat znajdziesz w rozdziale 6.

## PODSUMOWANIE

Nowoczesne programowanie internetowe z użyciem technologii opracowanych przez Microsoft nie sprowadza się jedynie do C# i ASP.NET. Wykorzystywane są różne narzędzia i frameworki, z których każdy został zbudowany za pomocą języka najlepszego do danego celu. Pojawianie się coraz większej liczby komponentów dodatkowych jeszcze bardziej utrudnia wybór narzędzi do użycia. Krótki cykl życia niektórych z nich nie pomaga programiście. W dwóch następnych rozdziałach znacznie dokładniej przedstawię najpopularniejsze frameworki front-endu, czyli Angular i Bootstrap.

# Skorowidz

## N



## O



## P



## R



## S



## Ś



## T



## U



## V

## W



## Y



## Z

# Znakomite frameworki, specjalne narzędzia — poznaj je wszystkie!

Dobry webdeveloper to wszechstronny webdeveloper. Nie może poprzestawać na znajomości jednego języka i umiejętności korzystania z jednej, konkretnej technologii. Co więcej, rozpowszechnianie się jednostronicowych aplikacji internetowych (ang. single page application, SPA) zaciera wyraźną do niedawna różnicę między pracą programisty back-endu a obowiązkami programisty front-endu. Programiści back-endu muszą dziś poznawać narzędzia do niedawna uznawane za typowe w przyborniku programisty front-endu, takie jak wybrane frameworki JavaScriptu. Powinni też nieźle sobie radzić z technologią CSS. Do tego muszą możliwie szybko zorientować się, które języki i frameworki najlepiej sprostają potrzebom konkretnego projektu.

Ta książka jest przeznaczona dla projektantów, którzy chcą poznać narzędzia do programowania front-endu i nauczyć się ich efektywnego użytkowania w połączeniu z ASP.NET Core MVC. Zawiera najlepsze praktyki tworzenia front-endu i praktyczną wiedzę dotyczącą programowania za pomocą ASP.NET Core MVC. Znalazła się tu również prezentacja najpopularniejszych frameworków i narzędzi służących do tworzenia front-endu, takich jak Angular, Bootstrap, NuGet, Bower, Webpack, Gulp i Azure, ponadto omówiono wprowadzone w Visual Studio 2017 nowe funkcje przeznaczone do tego celu. Przedstawiono także rozwiązania, które umożliwiają wykorzystywanie .NET Core na platformie macOS. Poszczególne koncepcje zostały zilustrowane przejrzystymi fragmentami przykładowego kodu.

W tej książce między innymi:

- zwięzłe wprowadzenie do ASP.NET Core MVC
- praca z Angularem w Visual Studio
- Bootstrap i responsywność stron internetowych
- narzędzia i technologie przydatne do programowania front-endu
- zintegrowane podejście do etapów testowania, kompilowania i wdrażania aplikacji

Simone Chiaretta od ponad dwóch dekad jest webdeveloperem, architektem sieci, programistą i autorem książek. Kilkukrotnie otrzymał tytuł Microsoft MVP. Bierze udział w organizacji wielu prestiżowych konferencji, na przykład wysoko cenionej Web.NET European Conference. Lubi eksperymentować z Arduino, dronami i robotami podwodnymi. Na co dzień kieruje zespołem programistów odpowiedzialnych za obsługę publicznej witryny internetowej Rady Unii Europejskiej.

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



ISBN 978-83-283-5194-3



Cena: 49,00 zł