Spis treści

Wprowadzenie

Zabawne jest to, że stale przeceniamy lub nie doceniamy wpływ określonych zdarzeń i projektów w naszym życiu. Naprawdę wierzyłem, że ostatni projekt, nad którym pracowałem w firmie Google, czyli Google Feedback, zakończy się całkowitą zmianą sposobu komunikowania się firmy z klientami. Uważałem też, że Angular (wtedy pod nazwą AngularJS) będzie tylko kolejnym słomianym ogniem, jeszcze jednym schematem, który nie przetrwa dłużej niż interfejs administratora w projekcie Feedback.

A z perspektywy czasu okazało się, że jest dokładnie na odwrót. Podczas gdy Feedback nadal istnieje i jest elementem wielu produktów Google'a, Angular rozwinął się z niewielkiego projektu używanego przez jeden wewnętrzny zespół, w narzędzie używane dziś przez tysiące deweloperów i firm na całym świecie. A w większości jest to zasługą Misko, Igora i całego zespołu i ich niezawodnego poświęcenia w celu poprawy sposobu, w jaki rozwijamy aplikacje sieciowe.

To, co zaczęło się jako dwuosobowy projekt, jest teraz jedną z największych otwartych społeczności w sieci, zaś sam schemat wywarł wpływ na tysiące projektów na całym świecie i stał się ich częścią. Dostępne są dziesiątki książek, setki materiałów do nauki oraz tysiące artykułów, które dotyczą Angular, a jego akceptacja i wsparcie rośnie z każdym dniem.

Niektóre najważniejsze koncepcje z pierwszej wersji Angular wyprzedzały swój czas (jak data binding*, SoC**, wstrzykiwanie zależności itp.), a teraz są powszechnie używanymi funkcjami nowych schematów.

Największą zmianą w ekosystemie AngularJS było wypuszczenie nowej wersji Angular (nazwanej na początku Angular 2.0, a obecnie nazywanej po prostu Angular). Była to drastyczna zmiana, bez kompatybilności wstecznej, która prawe podzieliła całą społeczność. Ale przy zaangażowaniu społeczności oraz otwartym połączonym zespole, to co mogło się stać rujnującym posunięciem, stało się bardzo potrzebną restrukturyzacją Angular, przynosząc nową epokę rozwoju sieci.

Szczerze mówiąc, tym, co sprawia, że Angular jest wspaniałą techniką i schematem, jest otaczająca go społeczność – ci, którzy współtworzą podstawowy schemat lub rozwijają wtyczki, a także ci, którzy używają go na co dzień.

---

* Technika, która składa się z relacji między źródłem danych a odbiorcą danych oraz mechanizmu synchronizacji danych między nimi (wg Wikipedii). Używany jest termin angielski. (Ten i wszystkie przypisy oznaczone gwiazdkami pochodzą od tłumacza).

** SoC – *Separation of Concerns* czyli podział zagadnień polegający na wyodrębnieniu w programie modułów jak najbardziej rozdzielnych funkcjonalnie.

Jako członek tej społeczności, jestem naprawdę poruszony, prezentując tę książkę i na swój sposób dokładając się do tego, co sprawia, że ta społeczność jest wspaniała.

Kto powinien przeczytać tę książkę

Jest to książka dla każdego, kto chce rozpocząć pracę z Angular (2.0 i późniejszym), jako projektem pobocznym w postaci dodatkowego narzędzia, lub jako swoim głównym narzędziem pracy. Oczekuje się, że czytelnicy rozpoczynający pracę z książką znają JavaScript oraz HTML, ale do nauki Angular wystarczy podstawowa znajomość JavaScript. Znajomość AngularJS 1.0 nie jest potrzebna ani oczekiwana.

Będziemy korzystać także z TypeScript, który jest zalecanym sposobem pracy w Angular, ale do czytania tej książki wystarczy jedynie początkowa wiedza w tym zakresie.

Będziemy działać krok po kroku, a więc teraz usiądźcie wygodnie i bawcie się dobrze, ucząc się ze mną.

Dlaczego napisałem tę książkę

Angular ogromnie rozrósł się jako schemat i ma obecnie duży zbiór funkcji i możliwości. Mając za sobą dużą społeczność, ma także napływ użytecznych treści. Ale zawartość pomocy, materiały do nauki i przewodniki są ukierunkowane albo na konkretne tematy, albo są fragmentaryczne i niespecjalnie przydatne dla osób początkujących.

Celem tej książki jest dostarczenie przewodnika, który krok po kroku pozwoli rozpocząć pracę z Angular. Każe pojęcie jest przedstawione w logiczny, zorganizowany sposób, z odwołaniem do pojęć poprzednich. Przy tylu ruchomych elementach i aktywnej społeczności, książka nie ma ambicji, aby przedstawić każdy aspekt, a zamiast tego w sposób szczegółowy skupia się na podstawowych elementach składowych, pozwalając użytkownikom odkrywać pozostałe elementy na własną rękę.

Po ukończeniu książki czytelnicy powinni znać większość schematu Angular i być zdolni do tworzenia za jego pomocą własnych aplikacji sieciowych, wykorzystując je w swoich własnych projektach.

Słowo o dzisiejszym tworzeniu aplikacji sieciowych

JavaScript przeszedł długą drogę do punktu, w którym jest dzisiaj, gdzie stanowi jeden z najszerzej używanych i przyjmowanych języków programowania. Dzisiaj deweloperzy aplikacji sieciowych rzadko muszą martwić się niespójnościami przeglądarek i podobnymi kwestiami, co było podstawową przyczyną istnienia schematów, takich jak jQuery.

Schematy (ang. *framework*), takie jak Angular i React, stanowią teraz powszechny wybór przy tworzeniu interfejsu użytkownika i rzadko kiedy ktoś decyduje się na tworzenie aplikacji front-end bez wykorzystania jednego z nich.

Zalety wykorzystywania schematów są wielorakie – od redukcji gotowego kodu, po zapewnienie spójnej struktury i układu do tworzenia aplikacji i nie tylko. Pierwszym celem jest zawsze zredukowanie czasu spędzanego na błędnych kodach i skupienie się na podstawowej funkcjonalności, którą chcemy dostarczyć. A jeśli działa ona na różnych przeglądarkach (oprócz komputerów osobistych, na takich platformach, jak Android i iOS), to daje jeszcze większe możliwości.

Angular (oraz inne schematy) zapewniają to przede wszystkim dzięki podstawom stanowiącym rdzeń schematu, w tym:

- rozbudowaną składnię szablonów opartą na programowaniu deklaratywnym;
- modularność i podział celów;
- *data binding* (wiązanie danych) i oparte na nim programowanie sterowane danymi;
- możliwość testowania oraz doskonałe jego wsparcie;
- routing i nawigację;
- oraz mnóstwo innych funkcji, od renderowania po stronie serwera, po zdolność do pisania natywnych aplikacji mobilnych plus wiele innych rzeczy!

Za pomocą Angular możemy skupić się na budowaniu wspaniałych doświadczeń, jednocześnie zarządzając złożonością i utrzymaniem w bezproblemowy sposób.

Przewodnik po książce

Celem tej książki jest przeprowadzenie czytelników przez każdy fragment Angular. Po każdym rozdziale, który wprowadza nowe pojęcie, następuje rozdział o tym, jak możemy je sprawdzić. Organizacja książki jest z grubsza następująca:

- *Rozdział 1, Wprowadzenie do Angular*, stanowi wstęp do Angular oraz do stojących za nim pojęć. Pokazuje także, co jest potrzebne, aby zacząć pisać aplikacje w Angular.
- *Rozdział 2, Witaj Angular*, prowadzi nas przez proces tworzenia prostej aplikacji w Angular oraz mówi, jak poszczególne elementy za sobą współdziałają. Wprowadza również interfejs wiersza poleceń (Angular CLI).
- *Rozdział 3, Użyteczne wbudowane dyrektywy Angular*, zagłębia się w temat podstawowych wbudowanych dyrektyw Angular (w tym `ngFor`, `ngIf` itp.) i mówi, kiedy i jak z nich korzystać.
- *Rozdział 4, Zrozumienie i używanie komponentów Angular*, bardziej szczegółowo omawia komponenty Angular, a także opisuje różne opcje dostępne podczas ich

tworzenia. Mówi także o podstawowych hookach cyklu życia dostępnych wraz z komponentami.

- *Rozdział 5, Testowanie komponentów Angular*, wprowadza nas do zagadnień jednostkowych testów komponentów Angular za pomocą Karma i Jasmine, wraz z całą strukturą testowania w Angular.
- *Rozdział 6, Praca z formularzami opartymi na szablonach*, opisuje tworzenie i działanie formularzy w Angular z naciskiem na formularze oparte na szablonach.
- *Rozdział 7, Praca z formularzami reaktywnymi*, opisuje inny sposób definiowania i działania formularzy, czyli tworzenie i stosowanie formularzy reaktywnych.
- *Rozdział 8, Usługi Angular*, opisuje usługi, w tym sposób używania wbudowanych usług Angular, a także jak i kiedy można definiować własne usługi w Angular.
- *Rozdział 9, Wywołania HTTP w Angular*, przechodzi do aspektów komunikacji z serwerem w Angular oraz zagłębia się w tworzenie wywołań HTTP, a także przedstawia niektóre zaawansowane tematy, jak kolektory i tym podobne.
- *Rozdział 10, Testowanie jednostkowe usług*, to powrót do zagadnień testowania jednostek, ale tym razem z naciskiem na usługi testowania jednostek. Obejmuje to testowanie prostych usług oraz nieco trudniejsze przypadki, jak przepływy asynchroniczne, a także usługi i komponenty, składające się na wywołania HTTP.
- *Rozdział 11, Routing w Angular*, zagłębia się w zagadnienia routingu w aplikacjach Angular i omawia większość jego cech.
- *Rozdział 12, Wprowadzanie aplikacji Angular do produkcji*, zbiera na koniec wszystkie pojęcia i omawia wprowadzanie gotowej aplikacji Angular do produkcji oraz różne związane z tym problemy i techniki.

Cały kod jest dostępny na GitHubie, więc jeśli nie chcecie przepisywać przykładów z tej książki albo chcecie mieć do czynienia z najnowszymi i najlepszymi przykładami, możecie odwiedzić repozytorium i pobrać zawartość.

We wszystkich przykładach w tej książce jest używany AngularJS w wersji 5.0.0.

Zasoby online

Jako lekturę początkową dla każdego dewelopera Angular można polecić następujące pozycje, które warto mieć zawsze pod ręką:

- The Official Angular API Documentation (*https://angular.io/api)*
- The Official Angular Quickstart Guide (*https://angular.io/guide/quickstart)*
- The Angular Heroes Tutorial App (*https://angular.io/tutorial)*

Konwencje stosowane w książce

W książce zastosowano wymienione poniżej konwencje typograficzne.

Kursywa
: Wskazuje nowe terminy, URL, adresy emailowe, nazwy plików i rozszerzenia plików.

`Czcionka stałopozycyjna`
: Używana jest do listingów programów, a także wewnątrz akapitów, które odnoszą się do elementów programu, jak zmienne lub nazwy funkcji, baz danych, typów danych, zmiennych środowiskowych, instrukcji oraz słów kluczowych.

`Czcionka stałopozycyjna pogrubiona`
: Pokazuje polecenie lub inne teksty, które powinny być wpisane bez zmian przez użytkownika.

`Czcionka stałopozycyjna pochylona`
: Pokazuje tekst, który powinien zostać zastąpiony wartościami podanymi przez użytkownika lub wartości określane przez kontekst, a także komentarze w kodzie.

Ten element oznacza wskazówkę lub sugestię.

Ten element oznacza uwagę ogólną.

Ten element oznacza ostrzeżenie lub zwrócenie uwagi.

Korzystanie z przykładowych kodów

Dodatkowe materiały (przykłady kodu, ćwiczenia itp.) są dostępne do pobrania pod adresem *https://github.com/shyamseshadri/angular-up-and-running*.

Książka ta nam ma pomóc w wykonaniu pracy. Ogólnie, jeśli w książce podany jest przykład kodu, można go użyć w swoich programach i dokumentacji. Nie trzeba kontaktować się z nami w celu otrzymania zgody, chyba że kopiowana jest znacząca ilość

kodu. Na przykład pisząc program, który wykorzystuje kilka kawałków kodu z książki, nie trzeba mieć zgody. Sprzedaż i rozpowszechnianie przykładowej płyty CD z książek wydawnictwa O'Reilly nie wymaga zgody. Wstawienie znaczącej ilości przykładowego kodu do dokumentacji produktu wymaga zgody.

Doceniamy powołanie się na nas, ale tego nie wymagamy. Powołanie się zawiera zwykle tytuł, autora, wydawcę oraz ISBN oryginalnego (pierwotnego) wydania. Na przykład „*Angular: Up and Running* by Shyam Seshadri (O'Reilly). Copyright 2018 Shyam Seshadri, 978-1-491-99983-7."

Jeśli ktoś sądzi, że wykorzystanie kodu wykracza poza uczciwe wykorzystanie lub wyrażoną powyżej zgodę, prosimy o kontakt pod adresem *permissions@oreilly.com*.

Kontakt

Komentarze i pytania dotyczące tej książki prosimy kierować do wydawcy oryginalnego wydania:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

Z książką powiązana jest witryna, na której podano erratę, przykłady i dodatkowe informacje. Można znaleźć tę stronę pod adresem *http://bit.ly/angularUR*.

Podziękowania

Książkę tę dedykuję mojej żonie, Sanchicie, oraz moim rodzicom i babci, którzy są moją ostoją, a także motywowali mnie do napisania tej książki w najlepszy możliwy sposób, gdy jednocześnie borykałem się z moim raczkującym startupem w jego najbardziej niepewnych chwilach (początek!).

Chciałbym też podziękować moim recenzentom, którymi byli Yakov Fain i Victor Mejia. Musieli oni przeczytać mój niezredagowany i chaotyczny tekst i sprawdzić, czy wyraziłem to, co było moim celem, w sposób najbardziej zwięzły i zrozumiały.

Książka ta nie powstałaby oczywiście bez wiary i wysiłku wspaniałego zespołu z O'Reilly, a zwłaszcza bez Angeli i Kirsten!

I wreszcie dziękuję wspaniałej społeczności Angular za cały jej wkład, opinie oraz wsparcie, a także za naukę, jak lepiej wykorzystywać i tworzyć w Angular.

Rozdział 1

Wprowadzenie do Angular

Nasze oczekiwania co do tego, co możemy zrobić w sieci (a przez *sieć* rozumiem tu zarówno sieć stacjonarną, jak i mobilną) dotarły do punktu, w którym to, co jeszcze niedawno było w pełni wyposażonymi, natywnymi aplikacjami komputerowymi, jest teraz uruchamiane w przeglądarce. Aplikacje Web przypominają teraz natywne aplikacje komputerowe co do zakresu i złożoności, co w wyniku powoduje dodatkową złożoność po naszej stronie jako deweloperów.

Co więcej, SPA (Single-Page Applications, aplikacje jednostronicowe) stały się bardzo typowym wyborem dla tworzeniu rozwiązań front-end, gdyż one pozwalają użytkownikowi na lepsze działanie w sensie szybkości i reakcji. Po pobraniu podstawowej aplikacji do przeglądarki klienta, kolejne działania muszą tylko pobierać dodatkowe dane, które są potrzebne, bez konieczności ponownego pobierania całej strony, jak to miało miejsce w przeszłości w przypadku stron renderowanych po stronie serwera.

AngularJS powstał, aby nadać tworzeniu aplikacji jednostronicowych strukturę i spójność, jednocześnie zapewniając możliwość szybkiego tworzenia skalowalnych i łatwych w utrzymaniu aplikacji sieciowych. W czasie, gdy powstawał, sieć i przeglądarki rozwijały się gwałtowanie, a niektóre problemy rozwiązywane przez AngularJS nie były aż tak istotne.

Angular był wtedy przepisaną całkiem na nowo strukturą, wbudowaną w sieć nowej ery. Wykorzystywał wiele nowinek, od modułów po komponenty sieci, poprawiając jednocześnie istniejące funkcje AngularJS, jak wstrzykiwanie zależności i szablony.

Od tego miejsca, jeśli piszę AngularJS, odnoszę się do oryginalnego schematu AngularJS czyli wersji 1.0. Gdy wspominam o samym Angular, odnosi się to do nowszego schematu, w wersji od 2.0 w górę. Wynika to z faktu, że Angular 2.0 i wersje następne nie opierają się wyłącznie na korzystaniu z JavaScript, lecz także obsługują aplikacje pisane w TypeScript.

Dlaczego Angular

Angular jako schemat ma kilka znaczących zalet, zapewniając jednocześnie strukturę dla deweloperów pracujących w zespole. Pozwala tworzyć duże aplikacje w sposób łatwy do utrzymania. W kolejnych rozdziałach zajmiemy się szczegółami niżej wymienionych zagadnień.

Komponenty niestandardowe
: Angular pozwala na tworzenie własnych deklaracyjnych komponentów, które łączą funkcje z ich kodem renderowania we fragmenty o wielkości bitowej możliwe do ponownego wykorzystania. Działa to także dobrze dla komponentów sieciowych.

Data binding
: Angular pozwala na bezproblemowe przenoszenie danych z podstawowego kodu JavaScript do widoku i reagować na zdarzenia widoku bez konieczności pisania samemu kodu łączącego.

Wstrzykiwanie zależności
: Angular pozwala na pisanie modułowych usług i na wstawianie ich w dowolne potrzebne miejsce. Poprawia to znacznie możliwość testowania i ponownego wykorzystania.

Testowanie
: Testy to podstawa, zaś Angular został utworzony od podstaw z myślą o możliwości testowania. Można (i należy!) testować każdą część swojej aplikacji.

Wszechstronność
: Angular to w pełni ukształtowany schemat, który zapewnia gotowe rozwiązania do komunikacji z serwerem, routing w obrębie naszej aplikacji i wiele więcej.

W schemacie Angular przyjęto pewną semantykę ustalania wersji dla nowych wydań. Ponadto jego podstawowy zespół ma agresywnie określone cele, planując nowe duże wydanie co sześć miesięcy. Tak więc to, co zaczęło się jako Angular 2, jest teraz nazywane po prostu Angular, gdyż nie chcemy nazywać ich Angular 2, Angular 4, Angular 5 i tak dalej.

Wynika z tego, że w przeciwieństwie do przejścia od AngularJS do Angular, ulepszenia między wersjami (powiedzmy od 2 do 4) są przyrostowe i bardzo często dość trywialne. Dlatego nie musimy się martwić o to, że duża aktualizacja co kilka miesięcy przyniesie drastyczne zmiany w kodzie.

Czego nie znajdziecie w tej książce

Sam Angular jako schemat jest dość duży, ale społeczność wokół niego jest jeszcze większa. Wiele świetnych funkcji i opcji używanych w Angular wyrasta z tej społeczności. To sprawia, że mnie jako autorowi jest jeszcze trudniej wymyślić, jak pisać książkę, która przygotowuje czytelników do roli deweloperów Angular, ograniczając jednocześnie jej zakres do tego, co uważam za niezbędne.

W tym zakresie, choć Angular można rozszerzać na bardzo wiele sposobów, od pisania podstawowych apek mobilnych za pomocą Angular (patrz NativeScript), renderowanie swoich aplikacji Angular na serwerze (patrz Angular Universal), używanie Redux jako pierwszej klasy opcji w Angular (wiele opcji; patrz ngrx) i o wiele innych możliwości, wstępna wersja książki skupia się na podstawowej platformie Angular i wszystkich zapewnianych przez niego możliwościach. Będziemy też dążyć do skupiania się na bardziej popularnych przypadkach, zamiast opisywania każdej funkcji i możliwości Angular, gdyż wtedy książka miałaby tysiąc stron.

Chodzi o to, aby skupić się na elementach, które będą niezbędne i użyteczne dla wszystkich deweloperów Angular, zamiast zajmować się fragmentami i częściami użytecznymi tylko dla niektórych.

Rozpoczynamy od naszego własnego środowiska pracy

Angular oczekuje od nas sporej porcji podstawowej pracy, aby osiągnąć umiejętność bezproblemowego tworzenia aplikacji na swoim komputerze. Określone elementy trzeba na wstępie zainstalować, co omawiamy w tym punkcie.

Node.js

Wprawdzie nigdy nie będziemy kodować w Node.js, jednak Angular używa go jako podstawy większej części swojego środowiska. Dlatego, aby móc rozpocząć pracę w Angular, musimy mieć w naszym środowisku zainstalowany Node.js. Istnieje wiele sposobów instalowania Node.js, odsyłam więc do witryny Node.js Download Page[1], gdzie można znaleźć dokładne instrukcje.

W systemie macOS instalacja Node.js przez Homebrew powoduje czasami pewne problemy. W takim przypadku należy spróbować zainstalować go bezpośrednio.

1 *https://nodejs.org/en/download/*

Musimy zainstalować Node.js w wersji 6.9.0 lub wyższej oraz wersję 3.0.0 lub wyższą narzędzia npm. Swoją wersję można zweryfikować po instalacji, uruchamiając następujące polecenia:

```
node --version
npm --v
```

TypeScript

TypeScript dodaje do kodu JavaScript zbiór typów, pozwalając nam na pisanie JavaScript, który jest łatwiejszy do zrozumienia, analizy i śledzenia. Zapewnia też dostępność najnowszych proponowanych funkcji ECMAScript. W końcu cały nasz kod w TypeScript skraca się do JavaScript, który łatwo uruchamia się w każdym środowisku.

TypeScript nie jest konieczny przy tworzeniu aplikacji Angular, ale jest bardzo zalecany, gdyż oferuje nieco miłej składni, a także sprawia, że podstawowy kod jest łatwiejszy do zrozumienia i utrzymania. W tej książce TypeScript jest używany do tworzenia aplikacji w Angular.

TypeScript jest instalowany w pakiecie NPM, więc można go łatwo zainstalować za pomocą polecenia:

```
npm install -g TypeScript
```

Pamiętajcie, aby zainstalować wersję 2.4.0 lub wyższą.

Wprawdzie będziemy opisywać większość podstawowych funkcji/pojęć używanych w TypeScript, jednak dobrze jest też dowiedzieć się więcej z oficjalnej dokumentacji TypeScript[2].

Interfejs wiersza poleceń (CLI) w Angular

W przeciwieństwie do AngularJS, gdzie łatwo było mieć jeden plik źródłowy jako zależny i wykonywany, Angular ma nieco bardziej skomplikowaną konfigurację. W tym zakresie zespół Angular utworzył narzędzie interfejsu wiersza poleceń (CLI), aby ułatwić ładowanie i tworzenie aplikacji w Angular.

Jest to znacząca pomoc, która ułatwia proces tworzenia, dlatego zalecam CLI przynajmniej dla początkowych projektów, dopóki nie opanujemy wszystkiego i nie poczujemy się pewni, robiąc to samodzielnie. W tej książce omawiamy polecenia CLI, a także działania wykonywane przezeń w tle, aby czytelnicy dobrze zrozumieli wszystkie potrzebne zmiany.

Instalacja najnowszej wersji (1.7.3 w czasie pisania tej książki) jest tak prosta jak uruchomienie poniższego polecenia:

```
npm install -g @angular/cli
```

2 *https://www.TypeScriptlang.org/docs/home.html*

Jeśli ktoś drapie się w głowę, widząc tę nowomodną konwencję nazewnictwa w pakietach Angular, to trzeba wiedzieć, że ta nowa składnia jest funkcją NPM, która nosi nazwę pakiety objętościowe (*scoped packages*). Pozwala ona na grupowanie pakietów w obrębie NPM w jednym folderze. Więcej można o tym przeczytać na witrynie: *https://docs.npmjs.com/misc/scope*.

Po zainstalowaniu możemy potwierdzić, czy instalacja się powiodła, za pomocą następującego polecenia:

```
ng --version
```

Pobieranie Codebase

Wszystkie przykłady z tej książki, wraz z ćwiczeniami i ostatecznym rozwiązaniem, są umieszczone w repozytorium Git. Wprawdzie pobranie go nie jest konieczne, ale można to zrobić, jeśli chcemy odwołać się lub pobawić się z przykładami z tej książki. Można zrobić to także, klonując repozytorium Git poprzez uruchomienie następującego polecenia:

```
git clone https://github.com/shyamseshadri/angular-up-and-running.git
```

Utworzy to w bieżącym katalogu roboczym folder o nazwie *angular-up-and-running* zawierający wszystkie potrzebne przykłady. W tym katalogu znajdziecie podkatalogi zawierające przykłady uporządkowane według rozdziałów.

Podsumowanie

W tym momencie mamy przygotowane nasze środowisko pracy i możemy zacząć tworzyć aplikacje w Angular. Zainstalowaliśmy Node.js, TypeScript, a także Angular CLI i rozumiemy ich potrzebę i sposób użytkowania.

W następnym rozdziale wreszcie zaczniemy budować naszą pierwszą aplikację w Angular i zaczniemy poznawać niektóre podstawowe pojęcia i terminy tego środowiska.

Rozdział 2

Witaj Angular

W poprzednim rozdziale mieliśmy bardzo szybki przegląd Angular i jego funkcji, a także przewodnik, jak ustawić środowisko do tworzenia aplikacji w Angular. W tym rozdziale zajmiemy się różnymi częściami aplikacji Angular, tworząc proste aplikacje od zera. Dzięki korzystaniu z tych aplikacji omówimy podstawową terminologię i pojęcia, takie jak moduły, komponenty, data binding (łączenie danych) i event binding (łączenie zdarzeń) oraz przekazywanie danych do i z komponentów.

Zaczniemy od bardzo prostej aplikacji giełdowej, która pozwala nam zobaczyć listę akcji, z których każda zawiera nazwę, kod i cenę. Podczas pracy w tym rozdziale zobaczymy, jak zapakować renderowanie akcji w pojedynczym komponencie, możliwy do ponownego wykorzystania, oraz jak korzystać ze zdarzeń Angular oraz z data binding.

Rozpoczynamy pierwszy projekt w Angular

Jak wspomniano w poprzednim rozdziale, będziemy w dużym stopniu polegać na Angular CLI jako pomocy w ładowaniu i rozwoju naszej aplikacji. Zakładam, że wykonaliście początkowe instrukcje instalacyjne z poprzedniego rozdziału i macie zainstalowane w swoim środowisku Node.js, TypeScript oraz Angular CLI.

Tworzenie nowej aplikacji jest tak proste, jak poniższe polecenie:

```
ng new stock-market
```

Po uruchomieniu tego polecenia wygeneruje ono automatycznie szkielet aplikacji w folderze o nazwie *stock-market** z kilkoma plikami oraz zainstaluje wszystkie potrzebne

* Określenie „stock-market” czyli giełda, jest tematem wszystkich przykładów. W tłumaczeniu zmienna zawsze nazywa się „stock”, w opisie używane jest określenie „akcje”.

zależności potrzebne do pracy Angular. Może to zająć trochę czasu, ale w rezultacie powinniśmy zobaczyć na terminalu następujący wiersz:

```
Project 'stock-market' successfully created.
```

Gratulacje! Właśnie utworzyliście pierwszą aplikację w Angular!

Wprawdzie utworzyliśmy naszą pierwszą aplikację za pomocą zwykłego polecenia Angular CLI, polecenie `ng new` używa kilku argumentów, które pozwalają nam dostosować wygenerowaną aplikację do naszych potrzeb. Obejmują one:

- Czy chcemy używać zwykłego CSS lub SCSS albo jakiegoś innego schematu CSS (na przykład, `ng new --style=scss`)
- Czy chcemy wygenerować moduł routingu (na przykład `ng new --routing`); to zagadnienie omawiamy w rozdziale 11.
- Czy chcemy używać wstawianych styli/szablonów.
- Czy chcemy użyć we wszystkich komponentach wspólnego prefiksu (na przykład ngnew `--prefix=acme`)

Jest ich wiele więcej. Warto zapoznać się z tymi opcjami za pomocą polecenia `ng help`, gdy już nieco lepiej poznamy schemat Angular, aby zdecydować, czy mamy jakieś określone preferencje.

Zrozumienie Angular CLI

Utworzyliśmy właśnie naszą pierwszą aplikację w Angular, ale Angular CLI wykonuje coś poza utworzeniem początkowego jej szkieletu. W istocie jest to z wielu powodów użyteczne w całym procesie tworzenia aplikacji, a w tym do:

- ładowania naszej aplikacji,
- obsługi aplikacji,
- uruchamiania testów (zarówno jednostkowych, jak i całościowych),
- tworzenia wersji dystrybucyjnej,
- generowania nowych komponentów, usług, tras i innych elementów dla naszej aplikacji.

Każdy z tych elementów wiąże się z jednym lub więcej poleceniem Angular CLI i wszystkie je omawiamy, gdy staną się one potrzebne lub się na nie natkniemy, zamiast próbować na wstępie opisać wszystkie polecenia i ich zastosowania. Każde polecenia zapewnia kolejny stopień elastyczności, wykorzystując wiele argumentów i opcji, co sprawia, że Angular CLI jest prawdziwie zróżnicowanym i sprawnym narzędziem dla wielu zastosowań.

Wykonywanie aplikacji

Po wygenerowaniu naszej aplikacji nadchodzi pora na jej uruchomienie, aby zobaczyć na żywo w przeglądarce działającą aplikację. Możemy to zrobić na dwa sposoby.

- Uruchamiając ją w trybie programowania, w którym Angular CLI kompiluje zmiany w miarę ich wprowadzania i odświeża interfejs użytkownika (UI).
- Uruchamiając ją w trybie produkcyjnym z zastosowaniem zoptymalizowanej kompilacji, obsługiwanej przez pliki statyczne.

Na razie uruchomimy aplikację w trybie programowania, co jest proste i wymaga jedynie wpisania

```
ng serve
```

z głównego folderu wygenerowanego projektu, który w tym przypadku jest folderem z *stock-market*. Po chwili przetwarzania i kompilacji powinniśmy zobaczyć na terminalu coś podobnego do tego:

```
** NG Live Development Server is listening on localhost:4200,
   open your browser on http://localhost:4200/ **
Date: 2018-03-26T10:09:18.869Z
Hash: 0b730a52f97909e2d43a
Time: 11086ms
chunk {inline} inline.bundle.js (inline) 3.85 kB [entry] [rendered]
chunk {main} main.bundle.js (main) 17.9 kB [initial] [rendered]
chunk {polyfills} polyfills.bundle.js (polyfills) 549 kB [initial] [rendered]
chunk {styles} styles.bundle.js (styles) 41.5 kB [initial] [rendered]
chunk {vendor} vendor.bundle.js (vendor) 7.42 MB [initial] [rendered]

webpack: Compiled successfully.
```

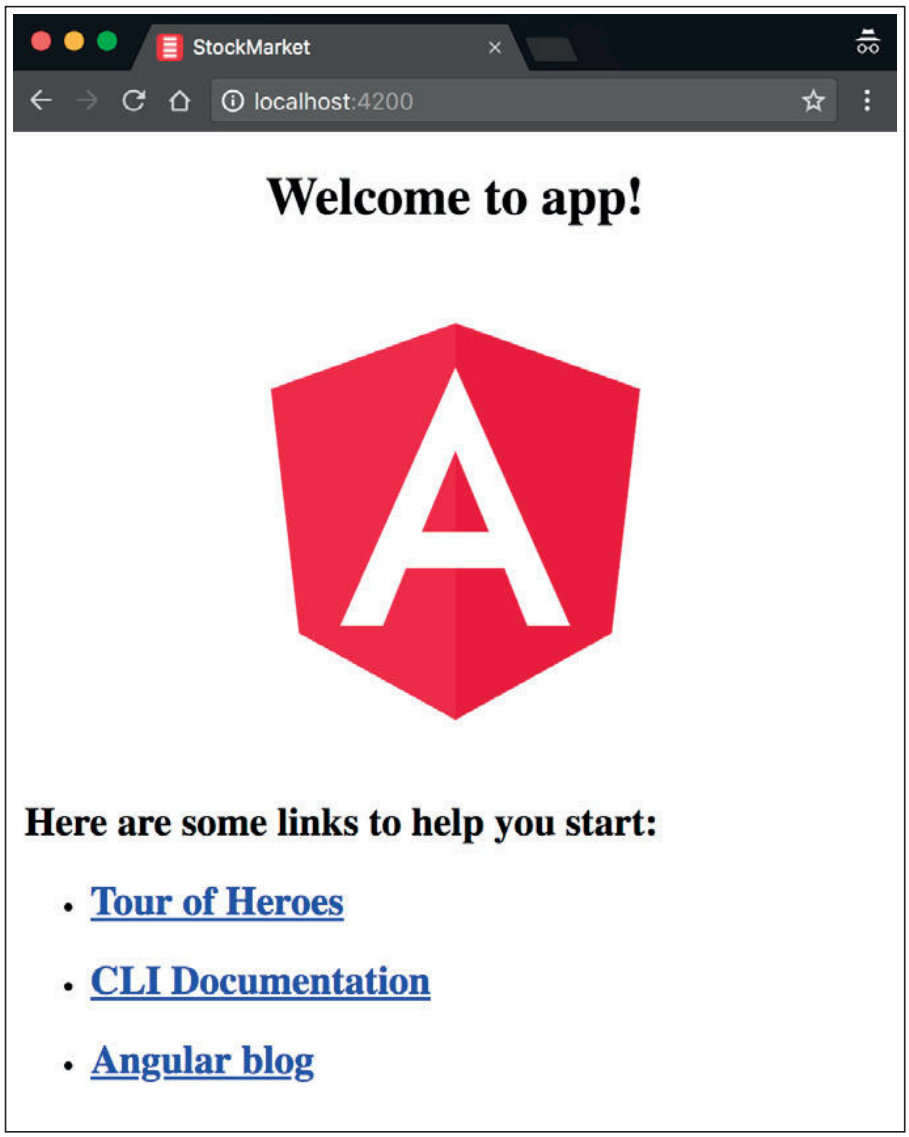
Powyższe wyniki stanowią zdjęcie wszystkich plików, które generuje Angular CLI, aby z powodzeniem obsłużyć naszą aplikację. Obejmuje to plik *main.bundle.js*, który zawiera kod po transpilacji, właściwy dla naszej aplikacji oraz plik *vendor.bundle.js*, który obejmuje wszystkie biblioteki i schematy innych producentów, na których się opieramy (w tym Angular). Plik *styles.bundle.js* stanowi kompilację wszystkich styli CSS, które są potrzebne naszej aplikacji, zaś plik *polyfills.bundle.js* obejmuje wszystkie polyfille* potrzebne do obsługi niektórych funkcji w starszych przeglądarkach (jak zaawansowane funkcje ECMA Script, które nie są jeszcze dostępne we wszystkich przeglądarkach). Wreszcie plik *inline.bundle.js* to niewielki plik zawierający narzędzia modułu webpack oraz ładowarki potrzebne do ładowania aplikacji.

`ng serve` uruchamia lokalny serwer programowania na porcie 4200, aby otrzymać trafienia z przeglądarki. Otwarcie w naszej przeglądarce *http://localhost:4200* powinno

* Polyfill to kod implementujący w przeglądarce funkcję przez nią nieobsługiwaną.

dać w wyniku widok działającej aplikacji Angular, który będzie wyglądać podobnie jak na rysunku 2.1.

Możemy pozostawić na terminalu działające polecenie `ng serve` i kontynuować wprowadzanie zmian. Jeśli mamy w przeglądarce otwartą aplikację, będzie ona się automatycznie odświeżać po każdym wprowadzeniu zmian. To sprawia, że programowanie jest szybkie i iteracyjne.



Rysunek 2.1 *Aplikacja Witaj Angular w przeglądarce*

W kolejnym punkcie wchodzimy w niektóre szczegóły związane z działaniami, które miały miejsce pod powierzchnią, aby zobaczyć, jak działa wygenerowana aplikacja Angular i czym są jej poszczególne fragmenty.

Podstawy aplikacji w Angular

W zasadzie każda aplikacja Angular jest aplikacją SPA (Single-Page Application)*, więc jej ładowanie jest uruchamiane przez podstawowe żądanie do serwera. Gdy otwieramy w naszej przeglądarce jakikolwiek URL, pierwsze żądanie jest kierowane do serwera (który w tym przypadku działa w ramach `ng serve`). To wstępne żądanie jest spełniane przez stronę HTML, która następnie ładuje niezbędne pliki JavaScript, aby załadować kod i szablony zarówno Angular, jak i naszej aplikacji.

Warto odnotować, że choć tworzymy naszą aplikację Angular w TypeScript, aplikacja sieciowa pracuje po transpiracji w JavaScript. Polecenie `ng serve` jest odpowiedzialne za przetłumaczenie naszego kodu w TypeScript na JavaScript, aby przeglądarka to załadowała.

Jeśli popatrzymy na strukturę wygenerowaną przez Angular CLI, zobaczymy coś w tym rodzaju:

```
stock-market
+----e2e
+----src
     +----app
          +----app.component.css
          +----app.component.html
          +----app.component.spec.ts
          +----app.component.ts    ❶
          +----app.module.ts    ❷
     +----assets
     +----environments
     +----index.html    ❸
     +----main.ts    ❹
+----.angular-cli.json    ❺
```

❶ Podstawowy komponent

❷ Moduł główny

❸ Podstawowy HTML

❹ Punkt wejścia

❺ Angular CLI config (konfiguracja)

Wymieniono tu jeszcze kilka innych plików poza tymi, które mamy w folderze *stock-market*, ale te są najważniejsze i na nich skupimy się w tym rozdziale. Ponadto są tu testy jednostkowe, testy całościowe (end-to-end, e2e), aktywa, które obsługują naszą aplikację,

* SPA (Single Page Application) to aplikacja lub strona internetowa, która w całości wczytuje się za jednym razem (Wikipedia).

elementy związane z konfiguracją w różnych środowiskach (dev, prod, itp.) oraz resztę ogólnej konfiguracji, którą zajmiemy się w rozdziałach 5, 10 i 12.

Podstawowy HTML – index.html

Jeśli spojrzymy na plik *index.html* file, który znajduje się w folderze *src*, zauważymy, że wygląda on niezwykle czysto i dziewiczo, bez żadnych odwołań do skryptów lub zależności:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>StockMarket</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>    ❶
</body>
</html>
```

❶ Podstawowy komponent naszej aplikacji Angular

Jedyne, co warto odnotować w powyższym kodzie, to element `<app-root>` w HTML, który jest znacznikiem do pobierania kodu naszej aplikacji.

A co z fragmentem, który pobiera podstawowe skrypty Angular i kod naszej aplikacji? Jest on wstawiany dynamicznie podczas działania przez polecenie `ng serve`, które łączy ze sobą wszystkie biblioteki dostawców, kod naszej aplikacji, style, wstawione szablony, w oddzielne paczki, a następnie wstawia je do *index.html,* aby zostały załadowane, gdy tylko w przeglądarce rozpocznie się renderowanie naszej strony.

Punkt wejścia – main.ts

Drugą ważną częścią naszego fragmentu ładującego jest plik *main.ts* file. Plik *index.html* jest odpowiedzialny za decyzję, które pliki należy pobrać. Natomiast plik *main.ts* identyfikuje, który moduł Angular (o czym mówimy trochę więcej w kolejnych punktach) zostanie pobrany po rozpoczęciu aplikacji. Może też zmienić konfigurację na poziomie aplikacji (jak wyłączenie zapewnień na poziomie struktury i weryfikacji za pomocą flagi `enableProdMode()`), co omawiamy w rozdziale 12:

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

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

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

platformBrowserDynamic().bootstrapModule(AppModule)     ❶
  .catch(err => console.log(err));
```

❶ Ładowanie głównego modułu `AppModule`

Większość kodu w pliku *main.ts* jest ogólna i rzadko kiedy będziemy musieli dotykać lub zmieniać ten plik punktu wejścia. Jego głównym celem jest wskazanie schematu Angular w podstawowym module naszej aplikacji i pozwolenie, aby uruchomił resztę kodu źródłowego aplikacji od tego punktu.

Moduł główny – app.module.ts

W tym miejscu zaczyna się właściwy kod naszej aplikacji. Plik modułu aplikacji można traktować jako podstawę konfiguracji naszej aplikacji, od załadowania wszystkich odpowiednich i potrzebnych zależności przy zadeklarowaniu, które komponenty będą używane w naszej aplikacji, po zaznaczenie, który komponent naszej aplikacji jest głównym punktem wejścia

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

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

@NgModule({    ❶
  declarations: [
    AppComponent    ❷
  ],
  imports: [    ❸
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]    ❹
})
export class AppModule { }
```

❶ Adnotacja `NgModule` języka TypeScript w celu oznaczenia tej klasy definicji w module Angular.

❷ Deklaracje pokazujące, których komponentów i dyrektyw można używać w obrębie naszej aplikacji.

❸ Importowanie innych modułów, które zapewniają funkcjonalność potrzebną w aplikacji.

❹ Komponent punktu wejścia do rozpoczęcia aplikacji.

Tu po raz pierwszy mamy do czynienia z funkcją specyficzną dla TypeScript, którą są dekoratory (można traktować je jako adnotacje). Dekoratory pozwalają nam dekorować klasy za pomocą przypisów i własności, a także jako meta funkcjonalność.

Angular w znacznym stopniu wykorzystuje tę funkcję TypeScript w informacjach, korzystając z dekoratorów w modułach, komponentach i wielu innych przypadkach.

Więcej na temat dekoratorów w TypeScript można przeczytać w oficjalnej dokumentacji[1].

Szczegóły poszczególnych elementów są omawiane w kolejnych rozdziałach, ale poniżej pokazano ich podstawy:

`declarations`
: Blok `declarations` definiuje wszystkie komponenty, które są dopuszczalne w danym module w zakresie HTML. Każdy utworzony komponent przed użyciem musi zostać zadeklarowany.

`imports`
: Nie będziemy tworzyć każdej funkcjonalności używanej w aplikacji, a tablica `imports` pozwala na zaimportowanie innej aplikacji Angular oraz modułów bibliotecznych i skorzystanie z komponentów, usług i innych możliwości, które już zostały utworzone w tych modułach.

`bootstrap`
: Tablica ładowania `bootstrap` definiuje komponent, który działa jak punkt wejścia do naszej aplikacji. Jeśli w tym miejscu nie dodamy głównego komponentu, nasza aplikacja nie rozpocznie działania, gdyż Angular nie będzie wiedział, jakich elementów ma szukać w naszym *index.html*.

  Zwykle ostatecznie musimy (jeśli nie korzystamy z jakiegoś powodu z CLI!) zmodyfikować ten plik, wtedy i tylko wtedy, gdy dodajemy nowe komponenty, usługi lub dodajemy/integrujemy się z nowymi bibliotekami i modułami.

1 *http://bit.ly/2IDQd1U*

Komponent podstawowy – AppComponent

Wreszcie dochodzimy do właściwego kodu Angular, który kieruje funkcjami aplikacji, a w tym przypadku jest głównym (i jedynym) komponentem, jaki mamy: `AppComponent`. Jego kod wygląda jakoś tak:

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

@Component({
  selector: 'app-root',    ❶
  templateUrl: './app.component.html',    ❷
  styleUrls: ['./app.component.css']    ❸
})
export class AppComponent {
  title = 'app';    ❹
}
```

❶ Selektor DOM, który jest tłumaczony na instancję tego komponentu

❷ Szablon HTML obsługujący ten komponent – w tym przypadku URL prowadzący do niego

❸ Każde modelowanie specyficzne dla komponentu, w tym przypadku także wskazujące na oddzielny plik

❹ Klasa komponentu ze swoimi członkami i funkcjami

W Angular *komponent* jest niczym innym jak klasą TypeScript, ozdobioną pewnymi atrybutami i metadanymi. Klasa zawiera w sobie wszystkie dane i funkcjonalności komponentu, zaś dekorator określa, jak przekłada się to na HTML.

Selektor aplikacji (*app-selector) to selector CSS, który określa, jak Angular znajduje dany komponent na dowolnej stronie HTML. Mimo że w zasadzie używamy selektorów elementów (app-root w poprzednim przykładzie, który oznacza szukanie elementów* `<app-root>` w HTML), może to być dowolny selektor CSS, od klasy CSS po atrybut.

Element `templateUrl` to ścieżka do HTML używana do renderowania tego komponentu. Zamiast podawania `templateUrl` , jak to ma miejsce w tym przykładzie, możemy także używać wstawianych szablonów. W tym konkretnym przypadku szablon, do którego się odwołujemy, to *app.component.html*.

Element `styleUrls` to odpowiednik szablonu, który ustawia styl zawierający w sobie wszystkie style danego komponentu. Angular sprawia, że style są hermetyzowane, więc nie musimy się martwić, że Klasy CSS z jednego komponentu będą miały wpływ na inny. W przeciwieństwie do `templateUrl`, `styleUrls` jest tablicą.

Sama klasa komponentu zawiera w sobie wszystkie funkcjonalności obsługujące nasz komponent. Ułatwia myślenie o dwojakiej odpowiedzialności klasy komponentów:

- Pobieranie i utrzymywanie wszystkich danych potrzebnych do renderowania komponentu.

- Obsłużenie i przetworzenie wszystkich zdarzeń, które mogą zaistnieć dzięki dowolnemu elementowi komponentu

Dane w klasie będą sterować tym, co może zostać wyświetlone jako część komponentu. Przyjrzyjmy się więc, jak wygląda szablon tego komponentu:

```
<h1>
  {{title}}    ❶
</h1>
```

❶ Tytuł związany z danymi komponentu

Nasz HTML jest najprostszy z możliwych dla komponentu. Ma on tylko jeden element, który jest związany danymi z polem w klasie naszego komponentu. Składnia z podwójnymi nawiasami klamrowymi ({{ }}) wskazuje, że Angular ma zastąpić wartość między nawiasami wartością zmiennej z odpowiedniej klasy.

W tym przypadku, gdy aplikacja zostanie załadowana i komponent zostanie renderowany, `{{title}}` zostanie zastąpiony tekstem `app works!`. Temat data binding jest omawiany bardziej szczegółowo w punkcie „Zrozumienie data binding” na stronie 19.

Tworzenie komponentu

Jak dotąd zajmowaliśmy się podstawowym szkieletem kodu, który Angular CLI dla nas wygenerował. Popatrzmy teraz na dodawanie nowych komponentów i temu, co to za sobą pociąga. Wykorzystamy Angular CLI do wygenerowania nowego komponentu, ale zajrzymy pod spód, aby zobaczyć, jakich kroków to wymaga. Potem wykonamy kilka bardzo podstawowych zadań, które spróbujemy osiągnąć za pomocą tych komponentów.

Kroki przy tworzeniu nowych komponentów

Korzystanie z Angular CLI przy tworzeniu nowego komponentu to po prostu wykonanie prostego polecenia. Spróbujemy najpierw utworzyć widżet giełdowy, który wyświetla nazwę akcji, kod akcji oraz bieżącą cenę, a także, czy zmieniła się ona na plus lub na minus.

Możemy po prostu utworzyć nowy `stock-item` (element giełdowy), wykonując z głównego folderu aplikacji podane niżej polecenie:

```
ng generate component stock/stock-item
```

Warto tu zauważyć kilka interesujących elementów:

- Angular CLI zawiera polecenie o nazwie `generate`, które można wykorzystać do wygenerowania komponentów (jak to zrobiliśmy w powyższym przykładzie), a także do generowania innych elementów Angular, jak interfejsy, usługi, moduły i inne.