GRZEGORZ DĄBROWSKI

ANGULAR

I FORMULARZE REAKTYWNE

Praktyczny przewodnik

Helion

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

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

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

Redaktor prowadzący: Małgorzata Kulik

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

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

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

ISBN: 978-83-289-0857-4

Copyright © Helion S.A. 2024

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Wstęp

Formularze są z nami od początku istnienia stron WWW. Pierwsza specyfikacja języka HTML opublikowana w 1993 roku zawierała m.in. specyfikację znaczników `<input>`[1]. Znaczniki te pozwalały użytkownikowi na interakcję ze stroną internetową poprzez wprowadzanie danych i wysyłanie ich w określone miejsce. Od tego czasu formularze przeszły długą drogę, rozwijając się od prostych elementów strony, po złożone struktury. Obecnie ciężko sobie wyobrazić nawet najprostszą witrynę, która nie zawierałaby ani jednego formularza. Niezależnie jednak od stopnia skomplikowania, przypadku użycia czy wyglądu ich podstawowa rola została niezmienna. Formularze dają możliwość komunikacji użytkownika z aplikacją webową. Bez nich strony internetowe byłyby jedynie nowoczesnymi prezentacjami.

Kolejne wersje języka HTML rozszerzały funkcjonalność formularzy, zachowując przy tym ich prostotę i uniwersalność. I to chyba w tym tkwi ich sekret. Kontrolki przetrwały próbę czasu, znakomicie współpracując z językiem JavaScript. Nie dały się wyprzeć apletom pisanym w Javie czy też aplikacjom tworzonym z użyciem technologii Adobe Flash. Rewolucja na rynku urządzeń mobilnych oraz ustandaryzowanie piątej wersji języka HTML sprawiły, że nikt nie myśli o tworzeniu swoich alternatyw dla tych rozwiązań. Aktualna specyfikacja stanowi stabilne i kompleksowe rozwiązanie, które może być wykorzystane w aplikacji webowej, jak również może zostać rozszerzone o customowe właściwości z użyciem niezliczonej ilości bibliotek i frameworków języka JavaScript.

Angular nie jest tutaj wyjątkiem. Posiada on rozbudowany system obsługi formularzy bazujący na natywnych kontrolkach i pozwalający w łatwy sposób zintegrować je ze swoim środowiskiem. W razie konieczności daje też możliwość rozszerzenia kontrolek zarówno pod względem funkcjonalnym, jak i wizualnym, oraz pozwala na tworzenie własnych rozwiązań przeznaczonych dla konkretnej aplikacji.

W niniejszej książce zostanie opisany sposób funkcjonowania mechanizmów obsługi formularzy reaktywnych w środowisku Angular. Zaczniemy od omówienia budowy API formularzy reaktywnych. Przeanalizujemy, w jaki sposób poszczególne elementy tego systemu są ze sobą powiązane. Omówimy proces przepływu danych oraz ich walidacji. Następnie przejdziemy do analizy praktycznych zastosowań poznanych mechanizmów, począwszy od tworzenia własnych reguł walidacji, po tworzenie mechanizmów, jak na przykład wysyłanie plików czy maskowanie wartości kontrolek. Na sam koniec omówimy metody tworzenia złożonych struktur formularzy oraz tworzenie niestandardowych kontrolek.

[1] *https://www.w3.org/MarkUp/HTMLPlus/htmlplus_41.html*.

Kup książkę Poleć książkę

Przykłady zaprezentowane w tej książce zostały zaprojektowane w ten sposób, aby maksymalnie uwypuklić omawiane zagadnienia. Oznacza to, że tam, gdzie było to konieczne, zostały użyte biblioteki zewnętrzne (np. Bootstrap). Reszta przykładów opiera się na natywnych rozwiązaniach. To samo dotyczy przykładów architektury. Zgodnie z zasadą: im prościej, tym czytelniej.

Biblioteka Bootstrap

Dołączenie biblioteki Bootstrap do projektu można przeprowadzić na wiele różnych sposobów. Najprostszym jest umieszczenie odpowiedniego znacznika w głównym pliku HTML, np.:

```
<link

href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css"
  rel="stylesheet"
  crossorigin="anonymous"
/>
```

Można też dodać odpowiedni pakiet do projektu, a następnie zaimportować plik SCSS. Niezależnie od wybranego sposobu przykłady będą działać poprawnie.

Kup książkę Poleć książkę

ROZDZIAŁ 1.

Budowa formularzy reaktywnych

Wstęp

Podczas codziennej pracy rzadko kiedy zdarza się pracować na pojedynczej kontrolce formularza. Przeważnie mamy do czynienia z grupą kontrolek zgromadzonych wewnątrz obiektu `FormGroup` lub też `FormArray`. Tworzą one spójną strukturę, w której poszczególne elementy oddziałują na siebie wzajemnie. Weźmy za przykład walidację. Jeśli któraś z kontrolek nie przejdzie walidacji, oznacza to, że cały formularz jej nie przechodzi. Muszą więc istnieć mechanizmy dbające o wymianę danych pomiędzy kontrolkami, grupami i tablicami na różnych poziomach zagnieżdżenia.

Mechanizmy te działają niejako bez wiedzy programisty. Są tak skonstruowane, aby aktualizacje wartości, stanów i walidacji były przeprowadzane automatycznie. Podejście to ułatwia pisanie kodu, który staje się czytelny i działa w przewidywalny sposób. Gdyby programista musiał sam dbać o aktualizację powiązanych obiektów, kod szybko stałby się nieczytelny i podatny na błędy. Wyobraźmy sobie przypadek, gdy wyłączenie przycisku *Wyślij* jest powiązane z walidacją formularza. Brak odpowiednich mechanizmów powodowałby konieczność aktualizacji głównego obiektu za każdym razem, kiedy któreś z jego dzieci otrzymałoby nową wartość bądź stan.

Zdarzają się jednak przypadki, gdy nie chcemy (bądź wręcz nie możemy) korzystać z tych automatycznych mechanizmów. Są to na przykład sytuacje, kiedy wgrywamy dane początkowe do formularza bądź też aktualizacja pól spowoduje nieskończoną pętlę aktualizacji. Framework Angular pozwala wpływać na aktualizację kontrolek na dwóch etapach. Pierwszym z nich jest moment tworzenia obiektu dziedziczącego po klasie `AbstractControl`, drugim dodanie opcjonalnych parametrów do metod modyfikujących wartości oraz stany.

W tej części książki skupimy się na omówieniu wspomnianych mechanizmów, prezentując różne metody przekazywania i odczytywania wartości formularzy.

Natywna obsługa formularzy

Zanim przejdziemy do omówienia mechanizmów formularzy reaktywnych we frameworku Angular, warto zastanowić się, czym tak naprawdę jest formularz. Zgodnie z definicją HTML-owy tag `<form>` używany do tworzenia formularza reprezentuje sekcję dokumentu będącą kontenerem dla interaktywnych kontrolek[1]. Jest to dość ogólna definicja tego znacznika, skupiająca się raczej na jego semantycznym aspekcie. Parząc na pierwotne

[1] *https://www.w3.org/TR/html401/interact/forms.html#edef-FORM*.

Kup książkę Poleć książkę

przeznaczenie (tj. wysyłanie na serwer danych wprowadzonych przez użytkownika), można pokusić się o inną definicję. A mianowicie taką, że znacznik `<form>` reprezentuje hiperłącze, którego ostateczna postać nie jest znana w chwili tworzenia strony, a jest tworzona w czasie rzeczywistym. Prawdziwość tej definicji potwierdzają domyślne wartości atrybutów definiujących formularz. I tak atrybut `method` odpowiada za rodzaj metody REST-owej, jaką dane zostaną wysłane — domyślnie `GET`, atrybut `enctype` określa, w jaki sposób dane zostaną przesłane na serwer. W ten sposób możliwe jest na przykład dołączenie do requestu dodatkowych danych, na przykład zawartości pliku (w przypadku wyboru metody `POST`). Wartość atrybutu `enctype` domyślnie to `application/x-www-form-urlencoded`. Wreszcie atrybut `target` definiuje miejsce, w którym nastąpi przekierowanie do zdefiniowanego adresu URL.

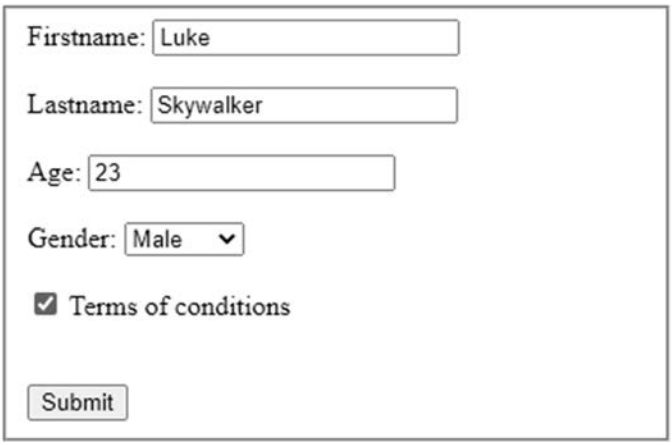
Sprawdźmy poprawność tej tezy. Stwórzmy przykładowy formularz, jak na listingu 1.1.

LISTING 1.1. Przykładowy formularz

```
<form method="GET" action="http://my-domain.com" target="_blank">
  <fieldset>
    <div>
      <label for="firstname">Firstname:</label>
      <input name="firstname" id="firstname" />
    </div>
    <br />
    <div>
      <label for="lastname">Lastname:</label>
      <input name="lastname" id="lastname" />
    </div>
    <br />
    <div>
      <label for="age">Age:</label>
      <input name="age" id="age" />
    </div>
    <br />
    <div>
      <label for="gender">Gender:</label>
      <select name="gender" id="gender">
        <option>Male</option>
        <option>Female</option>
        <option>Other</option>
      </select>
    </div>
    <br />
    <div>
      <input type="checkbox" id="conditions" name="conditions" />
      <label for="conditions">Terms of conditions</label>
    </div>
    <br />
    <br />
    <button type="submit">Submit</button>
  </fieldset>
</form>
```

Po uruchomieniu powyższego kodu w przeglądarce można przystąpić do testowania formularza. Wygląd i przykładowe dane zostały przedstawione na rysunku 1.1.

Kup książkę Poleć książkę



RYSUNEK 1.1. Formularz wypełniony przykładowymi danymi

Po wypełnieniu formularza i kliknięciu przycisku *Submit* przeglądarka otworzy nowe okno, próbując wczytać stworzony adres URL. W tym przypadku będzie to:
`http://my-domain.com/?firstname=Luke&lastname=Skywalker&age=23&gender=Male&`
`↪conditions=on`.

Wraz z rozwojem internetu pierwotna funkcja formularzy zaczęła być niewystarczająca. Szybko pojawiła się konieczność przesyłania większych porcji danych, a następnie możliwość wysłania plików. Kolejne wersje języka HTML wprowadzały coraz to nowe funkcje. Z czasem zaczęto też tworzyć nowe przeglądarki internetowe. Ich twórcy na swój sposób zaczęli implementować standardy języka HTML (najlepszym przykładem jest niechlubna sława przeglądarki Internet Explorer). Najlepszym przykładem różnic w natywnej obsłudze formularzy przez przeglądarki będzie wyświetlanie komunikatów walidacji. Rysunek 1.2 pokazuje implementację komunikatów w przeglądarkach Chrome oraz Firefox w wersjach desktopowych dla systemów z rodziny Windows.

RYSUNEK 1.2. Różnice pomiędzy przeglądarkami Chrome i Firefox

Kiedy taki sam przykład uruchomimy w innych systemach operacyjnych, jak na przykład macOS lub Linux, zobaczymy jeszcze więcej różnic. To samo dotyczy przeglądarek dla mobilnych systemów operacyjnych.

Naszym celem jako programistów jest stworzenie aplikacji, która będzie tak samo wyglądać i działać niezależnie od tego, w jakiej przeglądarce jest uruchomiona. Są oczywiście przypadki, kiedy nie zależy nam na ujednolicaniu wyglądu komunikatów. Jednak poleganie na domyślnych komunikatach jest już bardzo ryzykowne.

Kup książkę Poleć książkę

Weźmy za przykład właśnie walidację. Natywne API formularzy umożliwia za pomocą metody `setCustomValidity` zmianę treści komunikatu. W połączeniu z obsługą emitowanego zdarzenia `invalid` można stworzyć prosty mechanizm reagujący na błędnie wypełnione pola. Listing 1.2 przedstawia przykład takiego rozwiązania.

LISTING 1.2. Customizacja komunikatów walidacji

```
<input
  name="phone"
  type="text"
  required
  oninvalid="this.setCustomValidity('Wprowadź numer telefonu')"
  oninput="setCustomValidity('')"
/>
```

Sytuacja nieco się skomplikuje, kiedy będziemy chcieli wyświetlić różne komunikaty dla różnych typów walidacji, na przykład gdy numer telefonu jest wymagany i musi mieć odpowiedni format. Konieczne jest zbudowanie mechanizmu odpowiedzialnego za wyświetlanie komunikatu zależnego od rodzaju błędu walidacji wykorzystującego interfejs `ValidityState` kontrolki. Przykład ten pokazano na listingu 1.3.

LISTING 1.3. Dostosowanie komunikatu o błędzie zależne od typu walidacji

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Walidacja w HTML</title>
  </head>
  <body>
    <form>
      <input id="phone" name="phone" type="text" required pattern="[0-9]{10}" />
      <br />
      <br />
      <button type="submit">Submit</button>
    </form>

    <script>
      document.getElementById('phone').addEventListener('invalid', function () {
        switch (true) {
          case this.validity.valueMissing:
            this.setCustomValidity('Pole nie może być puste');
            break;

          case this.validity.patternMismatch:
            this.setCustomValidity('Błędny format numeru');
            break;
        }
      });

      document
        .getElementById('phone')
        .addEventListener('input', function (event) {
          this.setCustomValidity('');
        });
    </script>
  </body>
</html>
```

Kup książkę Poleć książkę

ValidityState

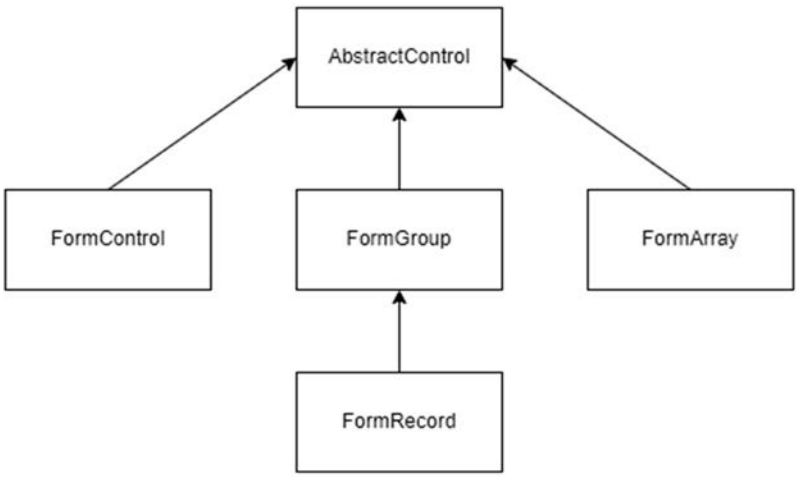
Interfejs ValidityState jest częścią składową natywnego API walidacji wchodzącego w tzw. WebAPI języka HTML. Szczegółowe omówienie tego zagadnienia wykracza poza zakres tej książki. Szczegóły dotyczące implementacji tego mechanizmu dostępne są w oficjalnej dokumentacji zamieszczonej na portalu MDN Web Docs: *https://developer.mozilla.org/en-US/docs/Web/API/ValidityState*.

Ilość kodu, jaki musimy napisać, rośnie wraz ze złożonością formularza. W związku z tym twórcy frameworków webowych tworzą własne rozwiązania zapewniające jednolitą funkcjonalność i użyteczność (ang. *user experience*) niezależnie od przeglądarki i platformy, na jakiej uruchomiona jest aplikacja webowa. Framework Angular nie jest tutaj wyjątkiem. Dostarcza on kompleksowe API dające możliwość rozszerzenia o własne rozwiązania dedykowane pod konkretne wymagania tworzonych aplikacji.

Struktura formularzy w środowisku Angular

Formularze reaktywne w Angularze opierają się na klasie `AbstractControl`. Reprezentuje ona podstawową funkcjonalność każdej kontrolki formularza, taką jak system śledzenia zmian, walidacja danych, interakcja z użytkownikiem i zarządzanie stanem. `AbstractControl` jest klasą abstrakcyjną, co oznacza, że nie może być bezpośrednio używana w kodzie. W zamian Angular dostarcza kilka klas dziedziczących po niej, które na swój sposób implementują wspomniane funkcjonalności.

Drzewo dziedziczenia po klasie `AbstractControl` przedstawia rysunek 1.3.



RYSUNEK 1.3. Struktura bazowych klas formularzy

Każda z przedstawionych klas ma swoje przeznaczenie w strukturze obiektu formularza. I tak:

- `FormControl` — reprezentuje pojedynczą kontrolkę. Obiekty tej klasy stanowią podstawowy element w drzewie formularza. Nie mogą zawierać obiektów potomnych, przechowując jedynie swoją własną wartość.
- `FormGroup` — reprezentuje grupę powiązanych ze sobą kontrolek formularza (najczęściej są to obiekty `FormControl`). Kontrolki są zgrupowane w formie obiektu, gdzie nazwy pól odpowiadają nazwom kontrolek. Obiekty `FormGroup` mogą zawierać w sobie inne obiekty `FormGroup`, `FormRecord` lub `FormArray`, tworząc tym samym formularze zagnieżdżone. Wartość stanu obiektu `FormGroup` stanowi sumę stanów wszystkich obiektów potomnych. Jeśli więc któryś obiekt `FormControl` nie przejdzie walidacji, jego rodzic otrzymuje stan `INVALID` (więcej o stanach i walidatorach można znaleźć w rozdziale „Walidacja"). Wartością obiektu `FormGroup` jest natomiast obiekt reprezentujący wartości wszystkich kontrolek potomnych.
- `FormRecord` — jest klasą dziedziczącą po `FormGroup`. Jej zasada działania jest taka sama. Różnica pomiędzy nią a `FormGroup` polega na tym, że `FormRecord` może zawierać elementy potomne, których wartość jest tego samego typu, a ich ilość oraz nazwy nie muszą być z góry zdefiniowane.
- `FormArray` — reprezentuje dynamiczną kolekcję kontrolek formularza. Kolekcja ma postać tablicy, której elementy mogą być dowolnie dodawane i usuwane. Podobnie jak `FormGroup`, może tworzyć formularze zagnieżdżone poprzez dołączanie do niej obiektów takich jak `FormGroup` czy też `FormArray`.

Struktura formularzy umożliwia poruszanie się po nich jak po drzewie. Klasa `AbstractControl` udostępnia getter `parent` pozwalający pobrać referencję do obiektu będącego wyżej w hierarchii formularza. Listing 1.4 przedstawia kod źródłowy wspomnianego gettera.

LISTING 1.4. getter parent[2]

```
get parent(): FormGroup|FormArray|null {
  return this._parent;
}
```

Analizując typ zwracanej wartości, możemy sformułować następujące wnioski:

1. **Kontrolka może nie mieć rodzica** — jak na przykład główny obiekt formularza, będący na samej górze w hierarchii, lub samodzielna kontrolka, nieprzypisana do żadnego formularza. Wtedy `parent` będzie miał wartość `null`.
2. **Rodzicem może być `FormGroup` lub `FormArray`** — oznacza to, że `FormControl` nie może pełnić roli rodzica. Jest to zgodne z jej definicją jako najmniejszej, niepodzielnej części formularza. `FormRecord` dziedziczy po `FormGroup`, więc również może być rodzicem.
3. **Kontrolki można zagnieżdżać** — to chyba oczywiste. Skoro obiekt klasy `FormGroup` (dziedziczący po `AbstractControl`) może mieć za rodzica inny obiekt `FormGrup`.

[2] Źródło: *https://github.com/angular/angular/blob/95712872999c54337595acb3014cb↪2c67d1af683/packages/forms/src/model/abstract_model.ts#L463*.

Kup książkę Poleć książkę

Chcąc odwołać się do elementów znajdujących się w dół drzewa, należy skorzystać z pola `controls`. Pole to przechowuje referencje do obiektów potomnych. W zależności od swojego typu obiekty te mogą być zgrupowane w formie obiektu (dla `FormGroup` i `FormRecord`) lub też w formie tablicy (dla `FormArray`). Pole `controls` jest niedostępne dla kontrolek typu `FormControl`, ponieważ z założenia nie posiadają one obiektów potomnych.

Istnieje też możliwość odwołania się bezpośrednio do głównego obiektu formularza z dowolnego miejsca w jego strukturze. W tym celu został stworzony getter `root`. Wykorzystuje on pętlę, za pomocą której odnajdywany jest w drzewie element nieposiadający rodzica. Kod źródłowy gettera `root` przedstawia listing 1.5.

LISTING 1.5. getter parent[3]

```
get root(): AbstractControl {
  let x: AbstractControl = this;

  while (x._parent) {
    x = x._parent;
  }

  return x;
}
```

[3] Źródło: *https://github.com/angular/angular/blob/main/packages/forms/src/model/abstract_model.ts#L1270*.

Kup książkę Poleć książkę

Kup książkę Poleć książkę


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

BEZ NICH STRONY INTERNETOWE BYŁYBY JEDYNIE NOWOCZESNYMI PREZENTACJAMI

Ewolucja formularzy internetowych zaczęła się od prostych znaczników języka HTML 2.0 opublikowanego w 1993 roku. Z czasem HTML oferował bardziej zaawansowane funkcje obsługi formularzy. Późniejsze wersje, HTML4, a następnie HTML5, wprowadziły nowe typy pól, takie jak pola daty, koloru czy też adresu e-mail. Rozszerzyły również natywną walidację danych poprzez wprowadzenie nowych atrybutów dla znaczników. Jednak prawdziwa rewolucja w projektowaniu formularzy internetowych nastąpiła z chwilą pojawienia się bibliotek opartych na języku JavaScript. Umożliwiły one tworzenie dynamicznych formularzy z walidacją na żywo i interaktywnymi elementami, a to pozwoliło przekształcić witryny internetowe w pełnoprawne aplikacje.

Obecnie frameworki takie jak Angular przenoszą formularze internetowe na nowy poziom. Oferują potężne narzędzia do tworzenia skomplikowanych formularzy z zaawansowaną walidacją, wiązaniem danych i logiką biznesową. Dzięki temu twórcy aplikacji mogą się skupić na tworzeniu atrakcyjniejszych interfejsów użytkownika i zapewniać lepsze doświadczenia dla użytkowników.

Ta książka przybliża sposób funkcjonowania mechanizmów obsługi formularzy w ujęciu reaktywnym w środowisku Angular. Omawia budowę API formularzy i objaśnia sposób, w jaki poszczególne elementy tego systemu są ze sobą powiązane. Prezentuje proces przepływu danych, a także ich walidacji. Analizuje praktyczne zastosowania poznanych mechanizmów — od tworzenia własnych reguł walidacji po kreowanie mechanizmów, jak wysyłanie plików czy maskowanie wartości kontrolek. Przedstawia też metody budowania złożonych struktur formularzy i niestandardowych kontrolek.

Helion

helion.pl

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

KOD KORZYŚCI
Sięgnij po więcej! ▶

ISBN 978-83-289-0857-4

9 788328 908574

Cena: 69,00 zł