O'REILLY®

AngularJS

Szybkie wprowadzenie

BŁYSKAWICZNIE OPANUJ ANGULARJS!

Helion

Shyam Seshadri, Brad Green

Tytuł oryginału: AngularJS: Up and Running

Tłumaczenie: Łukasz Piwko

ISBN: 978-83-283-1619-5

Wydawnictwo HELION
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)

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

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

Printed in Poland.

Spis treści

ROZDZIAŁ 4.

Formularze, pobieranie danych i usługi

W poprzednich rozdziałach opisaliśmy podstawowe dyrektywy systemu AngularJS oraz objaśniliśmy techniki tworzenia kontrolerów i przekazywania danych z kontrolerów do interfejsu użytkownika. Później pokazaliśmy, jak pisać testy jednostkowe przy użyciu narzędzi Karma i Jasmine. W tym rozdziale kontynuujemy pracę rozpoczętą w rozdziale 2. Pokazujemy, jak pobierać dane od użytkownika za pośrednictwem formularzy do kontrolera, aby wysłać je potem do serwera, sprawdzić lub zrobić z nimi coś innego.

Następnie przejdziemy do usług AngularJS. Wyjaśnimy, jak wykorzystać niektóre z istniejących usług tego systemu oraz jak tworzyć własne usługi. Ponadto objaśnimy krótko, kiedy i dlaczego należy tworzyć usługi AngularJS.

Posługiwanie się dyrektywą ng-model

W poprzednim rozdziale przedstawiliśmy dyrektywę `ng-bind` i jej ekwiwalent w postaci notacji `{{ }}`. Dyrektywa ta służy do pobierania danych z kontrolerów i wyświetlania ich w interfejsie użytkownika. Jest to jednostronne wiązanie danych, które może być bardzo przydatne w wielu sytuacjach. Ale większość aplikacji wchodzi w interakcje z użytkownikiem i przyjmuje od niego dane. Formularze, od formularzy rejestracyjnych po interfejsy do zmiany danych profilowych, stanowią nieodłączną część aplikacji sieciowych. Dlatego w systemie AngularJS utworzono dyrektywę `ng-model`, służącą do obsługi odbieranych danych i dwustronnego wiązania danych:

```
<!-- Plik: r04/simple-ng-model.html -->
<html ng-app="notesApp">
<head>
  <meta charset="utf-8">
  <title>Notes App</title>
</head>
<body ng-controller="MainCtrl as ctrl">

  <input type="text" ng-model="ctrl.username"/>
  Wpisałeś {{ctrl.username}}

<script
  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.11/angular.js">
</script>
<script type="text/javascript">
  angular.module('notesApp', [])
```

```
    .controller('MainCtrl', [function() {
      this.username = 'nic';
    }]);
</script>
</body>
</html>
```

W przykładzie tym zdefiniowaliśmy kontroler ze zmienną egzemplarzową o nazwie `username`. Następnie za pomocą dyrektywy `ng-controller` i podwójnej klamry do wiązania jednostronnego przenosimy jej wartość do kodu HTML. Nowością w tym kodzie jest element `input`. Jest to zwykłe pole tekstowe, ale powiązaliśmy z nim dyrektywę `ng-model`. Wartością tej dyrektywy jest wartość zmiennej `username` z kontrolera `MainCtrl`. W ten sposób osiągnęliśmy następujące cele:

- Gdy tworzony jest egzemplarz kodu HTML i wiązany jest z nim kontroler, następuje przekazanie bieżącej wartości (w tym przypadku łańcucha `nic`) i wyświetlenie jej w interfejsie użytkownika.
- Gdy użytkownik wpisze, zaktualizuje lub zmieni wartość w polu tekstowym, następuje aktualizacja modelu w kontrolerze.
- Gdy wartość zmiennej zmienia się w kontrolerze (bo nadeszła nowa wartość z serwera albo nastąpiła wewnętrzna zmiana stanu), wartość ta zostaje automatycznie zaktualizowana w polu wejściowym.

Piękno tego rozwiązania jest podwójne:

- Jeżeli trzeba zaktualizować element formularza w interfejsie użytkownika, to wystarczy dokonać aktualizacji wartości w kontrolerze. Nie trzeba szukać pól wejściowych po identyfikatorach czy klasach. Trzeba tylko zaktualizować model.
- Jeśli trzeba pobrać najnowszą wartość wpisaną przez użytkownika w formularzu lub wprowadzoną w celu weryfikacji albo wysłania do serwera, to wystarczy wziąć ją z kontrolera. W kontrolerze zawsze dostępna jest najnowsza wartość.

Teraz rozbudujemy nasz przykład tak, aby rzeczywiście przetwarzać informacje z formularza. Spójrz na poniższy kod źródłowy:

```
<!-- Plik: r04/simple-ng-model-2.html -->
<html ng-app="notesApp">
<head>
  <meta charset="utf-8">
  <title>Notes App</title>
</head>
<body ng-controller="MainCtrl as ctrl">

  <input type="text" ng-model="ctrl.username">
  <input type="password" ng-model="ctrl.password">
  <button ng-click="ctrl.change()">Zmień wartości</button>
  <button ng-click="ctrl.submit()">Wyślij</button>

<script
  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.11/angular.js">
</script>
<script type="text/javascript">
  angular.module('notesApp', [])
    .controller('MainCtrl', [function() {
```

```
        var self = this;
        self.change = function() {
          self.username = 'zmieniono';
          self.password = 'hasło';
        };
        self.submit = function() {
          console.log('Użytkownik kliknął przycisk Zatwierdź z danymi ',
              self.username, self.password);
        };
      }]);
</script>
</body>
</html>
```

Dodaliśmy jedno pole wejściowe, które powiązaliśmy z polem o nazwie `password` w egzemplarzu kontrolera. Dodaliśmy też dwa przyciski:

- Pierwszy przycisk ma etykietę *Zmień wartości* i służy do symulowania wysyłania przez serwer danych, które mają zostać zaktualizowane w interfejsie użytkownika. Jego działanie polega na przypisywaniu najnowszych wartości do pól nazwy użytkownika i hasła w kontrolerze.
- Drugi przycisk ma etykietę *Wyślij* i służy do symulowania operacji wysłania formularza na serwer. Na razie przycisk ten tylko rejestruje wartość w konsoli.

Najważniejszą rzeczą w obu opisanych przypadkach jest to, że kontroler ani razu nie sięgał do interfejsu użytkownika. Nie zastosowano żadnego selektora jQuery, żadnej funkcji typu `findElementById` ani niczego podobnego. Gdy trzeba zaktualizować interfejs użytkownika, wystarczy zmienić pola modelu w kontrolerze. Jeżeli trzeba pobrać najnowszą wartość, należy udać się po nią do kontrolera. Tak właśnie robi się to w systemie AngularJS.

Teraz zobaczysz, jak wykorzystać tę wiedzę do pracy z formularzami w AngularJS.

Praca z formularzami

W pracy z formularzami w systemie AngularJS często stosuje się dyrektywę `ng-model`, służącą do przekazywania danych do i z formularza. Oprócz używania wiązania danych zalecane jest też posługiwanie się modelem i wiązaniami w taki sposób, aby zredukować ilość potrzebnej pracy oraz kodu źródłowego do napisania. Spójrz na poniższy przykład:

```
<!-- Plik: r04/simple-form.html -->
<html ng-app="notesApp">
<head>
  <meta charset="utf-8">
  <title>Notes App</title>
</head>
<body ng-controller="MainCtrl as ctrl">

  <form ng-submit="ctrl.submit()">
    <input type="text" ng-model="ctrl.user.username">
    <input type="password" ng-model="ctrl.user.password">
    <input type="submit" value="Wyślij">
  </form>

<script
```

```
  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.11/angular.js">
</script>
<script type="text/javascript">
  angular.module('notesApp', [])
    .controller('MainCtrl', [function() {
      var self = this;
      self.submit = function() {
        console.log('Użytkownik kliknął przycisk Zatwierdź z danymi ', self.user);
      };
    }]);
</script>
</body>
</html>
```

W tym przykładzie również używamy tych samych pól wejściowych co poprzednio, ale wprowadziliśmy pewne zmiany:

- Pola tekstowe i przycisk wstawiliśmy do formularza. Usunęliśmy też dyrektywę `ng-click` z przycisku i zamiast niej dodaliśmy dyrektywę `ng-submit` do elementu formularza. Dyrektywa `ng-submit` ma kilka zalet w porównaniu z dyrektywą `ng-click` dla przycisku. Zdarzenie zatwierdzenia formularza może zostać wyzwolone na kilka sposobów, np. przez kliknięcie przycisku albo naciśnięcie klawisza *Enter*, gdy aktywne jest pole tekstowe. Dyrektywa `ng-submit` włącza się dla wszystkich tych zdarzeń, a dyrektywa `ng-click` — tylko dla kliknięć przycisku.
- Zamiast do `ctrl.username` i `ctrl.password` wiążemy do `ctrl.user.username` i `ctrl.user.password`. Zwróć uwagę, że w kontrolerze nie zadeklarowaliśmy obiektu użytkownika (tzn. `self.user = {}`). Gdy w użyciu jest dyrektywa `ng-model`, AngularJS automatycznie tworzy obiekty i klucze potrzebne w procesie tworzenia wiązania danych. W tym przypadku, dopóki użytkownik nie wpisze czegoś w polu nazwy użytkownika lub hasła, nie istnieje żaden obiekt użytkownika. Pierwsza litera wpisana do któregokolwiek z tych dwóch pól powoduje utworzenie tego obiektu i przypisanie w nim wartości do odpowiedniego pola.

Wiązanie danych i modele

Przy projektowaniu formularzy i wybieraniu pól do związania z dyrektywą `ng-model` należy się zastanowić, jaki format danych będzie potrzebny. Spójrz na poniższy przykład:

```
<!-- Plik: r04/two-forms-databinding.html -->
<html ng-app="notesApp">
<head>
  <meta charset="utf-8">
  <title>Notes App</title>
</head>
<body ng-controller="MainCtrl as ctrl">

  <form ng-submit="ctrl.submit1()">
    <input type="text" ng-model="ctrl.username">
    <input type="password" ng-model="ctrl.password">
    <input type="submit" value="Wyślij">
  </form>

  <form ng-submit="ctrl.submit2()">
    <input type="text" ng-model="ctrl.user.username">
```

```
    <input type="password" ng-model="ctrl.user.password">
    <input type="submit" value="Wyślij">
  </form>

<script
  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.11/angular.js">
</script>
<script type="text/javascript">
  angular.module('notesApp', [])
    .controller('MainCtrl', [function() {
      var self = this;
      self.submit1 = function() {
        // utworzenie obiektu użytkownika do wysłania na serwer
        var user = {username: self.username, password: self.password};
        console.log('Pierwsze zatwierdzenie formularza z ', user);
      };
      self.submit2 = function() {
        console.log('Drugie zatwierdzenie formularza z ', self.user);
      };
    }]);
</script>
</body>
</html>
```

W kodzie tym znajdują się dwa formularze zawierające takie same pola. Pierwszy formularz jest związany bezpośrednio ze zmiennymi `username` i `password` z kontrolera, a drugi — z kluczami `username` i `password` w obiekcie `user` z kontrolera. Oba formularze w momencie zatwierdzenia wyzwalają funkcję za pomocą dyrektywy `ng-submit`. W przypadku pierwszego formularza pola przed wysłaniem do serwera muszą zostać pobrane z kontrolera i wstawione do obiektu lub czegoś podobnego. W drugim formularzu można bezpośrednio pobrać obiekt `user` z kontrolera i przekazać go w odpowiednie miejsce.

Drugie rozwiązanie jest bardziej sensowne, ponieważ wprost odwzorowuje sposób reprezentacji formularza jako obiektu w kontrolerze. Dzięki temu eliminujemy dodatkową pracę, której wykonanie byłoby konieczne, gdybyśmy pracowali z wartościami formularza.

Sprawdzanie danych z formularza i stany

Pokazaliśmy, jak tworzy się formularze i jak wykorzystać wiązanie danych w celu przekazywania danych do i z interfejsu użytkownika. Teraz chcemy przedstawić jeszcze inne zalety posługiwania się systemem AngularJS podczas pracy z formularzami, a w szczególności podczas weryfikowania poprawności danych i różnych stanów formularzy oraz ich pól:

```
<!-- Plik: r04/form-validation.html -->
<html ng-app="notesApp">
<head>
  <meta charset="utf-8">
  <title>Notes App</title>
</head>
<body ng-controller="MainCtrl as ctrl">

  <form ng-submit="ctrl.submit()" name="myForm">
    <input type="text"
```

```
            ng-model="ctrl.user.username"
            required
            ng-minlength="4">
      <input type="password"
            ng-model="ctrl.user.password"
            required>
      <input type="submit"
            value="Wyślij"
            ng-disabled="myForm.$invalid">
    </form>

  <script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.11/angular.js">
  </script>
  <script type="text/javascript">
    angular.module('notesApp', [])
      .controller('MainCtrl', [function() {
        var self = this;
        self.submit = function() {
          console.log('Użytkownik kliknął przycisk zatwierdzający z ', self.user);
        };
      }]);
  </script>
  </body>
  </html>
```

Jest to zmodyfikowana wersja poprzedniego przykładu, w której dodano mechanizm weryfikacji danych. Mówiąc konkretnie, wyłączamy przycisk zatwierdzania, dopóki użytkownik nie wypełni wszystkich wymaganych pól. Oto jak to zrobiliśmy:

1. Nadaliśmy formularzowi nazwę `myForm`, którą będziemy mogli się później posługiwać.
2. Użyliśmy znaczników z obsługą weryfikacji danych HTML5 i do każdego pola wejściowego dodaliśmy atrybut `required`.
3. Dodaliśmy weryfikator danych, `ng-minlength`, pilnujący, aby długość wartości w polu nazwy użytkownika wynosiła nie mniej niż cztery znaki.
4. Do przycisku zatwierdzania dodaliśmy dyrektywę `ng-disabled`, która wyłącza ten przycisk, jeśli jej warunek jest spełniony.
5. Jeżeli chodzi o wyłączenie możliwości zatwierdzenia formularza, wykorzystaliśmy formularz eksponujący kontroler z bieżącym stanem formularza. W tym przypadku nakazujemy przyciskowi pozostać w stanie nieaktywnym, dopóki formularz o nazwie `myForm` jest niepoprawny (`$invalid`).

Jeśli do pola wejściowego zostanie zastosowany weryfikator danych, to jego wartość w modelu zostanie ustawiona dopiero wtedy, gdy pole to będzie poprawnie wypełnione. Innymi słowy, w poprzednim przykładzie wartość pola w zmiennej `ng-model` (`ctrl.user.username`) zostanie ustawiona dopiero wtedy, gdy osiągnie minimalną długość. Do tego czasu pozostanie pusta.

Kiedy używane są formularze (z określonymi nazwami), system AngularJS tworzy obiekt `FormCon` ↪`troller`, zawierający bieżący stan formularza, jak również pewne metody pomocnicze. Dostęp do tego obiektu można uzyskać przez nazwę formularza, tak jak to zrobiliśmy w poprzednim przykładzie z formularzem `myForm`. W tabeli 4.1 znajduje się opis tego, co można znaleźć w ramach stanu i co jest aktualizowane poprzez wiązanie danych.

Tabela 4.1. Stany formularza w AngularJS

Stan formularza	Opis
`$invalid`	System AngularJS ustawia ten stan, gdy któryś z weryfikatorów (`required`, `ng-minlength` itd.) oznaczy którekolwiek z pól w formularzu jako niepoprawne
`$valid`	Odwrotność poprzedniego stanu, która oznacza, że wszystkie weryfikatory w formularzu aktualnie wskazują poprawność
`$pristine`	Od tego stanu zaczynają wszystkie formularze w systemie AngularJS. Umożliwia on sprawdzenie, czy użytkownik zaczął już pisać lub modyfikować zawartość któregoś pola. Możliwe zastosowanie: wyłączanie przycisku resetowania, jeśli formularz jest pusty
`$dirty`	Odwrotność stanu `$pristine`, która oznacza, że użytkownik już coś zmienił w formularzu (może to cofnąć, ale bit `$dirty` jest już ustawiony)
`$error`	Obiekt przechowujący referencje do elementów formularza, które nie przeszły pomyślnie weryfikacji danych. Szerzej na ten temat piszemy w następnym podrozdziale

Wszystkie opisane w tabeli stany (z wyjątkiem stanu `$error`) są typu logicznego, więc mogą być używane do warunkowego ukrywania, pokazywania, wyłączania i włączania elementów HTML w interfejsie użytkownika. Gdy użytkownik coś wpisze lub zmodyfikuje w formularzu, wartości są aktualizowane pod warunkiem, że używana jest dyrektywa `ng-model` z nazwą formularza.

Obsługa błędów w formularzu

Znasz już techniki weryfikacji danych na poziomie formularza, ale czasami trzeba zaimplementować sprawdzanie danych dla poszczególnych pól. W poprzednim przykładzie utworzyliśmy dwa wymagane pola, z których jedno musi mieć wartość nie krótszą niż cztery znaki. Co jeszcze możemy zrobić? W tabeli 4.2 znajduje się wykaz niektórych wbudowanych weryfikatorów dostępnych w systemie AngularJS.

Tabela 4.2. Wbudowane weryfikatory danych systemu AngularJS

Weryfikator	Opis
`required`	Jak napisaliśmy wcześniej, weryfikator ten oznacza pole jako wymagane i sprawia, że jest niepoprawne, dopóki nie pojawi się w nim wartość
`ng-required`	W odróżnieniu od `required`, oznaczającego pole jako zawsze wymagane, dyrektywa `ng-required` umożliwia warunkowe oznaczanie pola wejściowego jako wymaganego na podstawie logicznego warunku w kontrolerze
`ng-minlength`	Dyrektywa ta służy do ustawiania minimalnej długości danych w polu wejściowym
`ng-maxlength`	Dyrektywa ta służy do ustawiania maksymalnej długości danych w polu wejściowym
`ng-pattern`	Poprawność danych w polu wejściowym można zweryfikować przy użyciu wyrażenia regularnego zdefiniowanego w tej dyrektywie
`type="email"`	Pole tekstowe z wbudowanym mechanizmem sprawdzania poprawności adresu e-mail
`type="number"`	Pole tekstowe z wbudowanym mechanizmem sprawdzania poprawności wartości liczbowej. Dodatkowo można określić minimalną i maksymalną wartość liczby
`type="date"`	Jeśli przeglądarka obsługuje ten typ pola formularza, powoduje on wyświetlenie elementu do wyboru daty. W przeciwnym wypadku wyświetlane jest zwykłe pole tekstowe. Model `ng-model` powiązany z tym polem będzie obiektem `date`. Data powinna być zapisana w formacie rrrr-mm-dd (np. 2009-10-24). Składnik wprowadzony w AngularJS 1.3.0
`type="url"`	Pole tekstowe z wbudowanym mechanizmem sprawdzania poprawności adresu URL

Dodatkowo można tworzyć własne weryfikatory danych, o czym szerzej piszemy w rozdziale 13.

Wyświetlanie informacji o błędach

Do czego można wykorzystać te wszystkie weryfikatory? Służą one do sprawdzania poprawności danych wprowadzonych do formularza i wyłączania w razie potrzeby przycisków zatwierdzania. Ale ponadto powinniśmy poinformować użytkownika, co zrobił źle i jak to poprawić. W AngularJS dostępne są dwa narzędzia pozwalające rozwiązać ten problem:

- model umożliwiający wyświetlanie przyjaznych dla użytkownika powiadomień o błędach;
- automatycznie dodawane i usuwane klasy CSS pozwalające na zaznaczenie miejsc, w których znaleziono błędne informacje.

Najpierw przeanalizujemy technikę wyświetlania informacji o błędach na podstawie występującego problemu. Spójrz na poniższy przykład:

```
<!-- Plik: r04/form-error-messages.html -->
<html ng-app="notesApp">
<head>
  <meta charset="utf-8">
  <title>Notes App</title>
</head>
<body ng-controller="MainCtrl as ctrl">

  <form ng-submit="ctrl.submit()" name="myForm">
    <input type="text"
           name="uname"
           ng-model="ctrl.user.username"
           required
           ng-minlength="4">
    <span ng-show="myForm.uname.$error.required">
      To pole jest obowiązkowe
    </span>
    <span ng-show="myForm.uname.$error.minlength">
      Trzeba wpisać przynajmniej 4 znaki
    </span>
    <span ng-show="myForm.uname.$invalid">
      Pole wypełnione niepoprawnie
    </span>
    <input type="password"
           name="pwd"
           ng-model="ctrl.user.password"
           required>
    <span ng-show="myForm.pwd.$error.required">
      To pole jest obowiązkowe
    </span>
    <input type="submit"
           value="Wyślij"
           ng-disabled="myForm.$invalid">
  </form>

<script
  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.11/angular.js">
</script>
<script type="text/javascript">
  angular.module('notesApp', [])
    .controller('MainCtrl', [function () {
      var self = this;
```

```
        self.submit = function () {
          console.log('Użytkownik zatwierdził formularz z ', self.user);
        };
      }]);
  </script>
  </body>
  </html>
```

W kontrolerze nic się nie zmieniło. Zmiany zostały wprowadzone w formularzu HTML. Oto one:

1. Najpierw dodaliśmy atrybut `name` do obu pól wejściowych, które chcieliśmy poddać weryfikacji. Pole nazwy użytkownika nazwaliśmy `uname`, a pole hasła — `pwd`.
2. Następnie wykorzystaliśmy wiązania formularzy AngularJS, aby umożliwić sobie znalezienie błędów w poszczególnych polach. Gdy polu wejściowemu zostanie nadana nazwa, tworzony jest dla niego model ze stanem błędu.
3. W efekcie, jeśli pole nazwy użytkownika nie zostanie wypełnione, dostęp do niego można uzyskać przez `myForm.uname.$error.required`. Analogicznie w przypadku weryfikatora `ng-minlength` należałoby użyć `myForm.uname.$error.minlength`. Aby sprawdzić, czy w polu hasła została wpisana jakaś wartość, należy użyć `MyForm.pwd.$error.required`.
4. Ponadto pobraliśmy stan pola wejściowego za pomocą `myForm.uname.$invalid`. Tak samo dostępne są wszystkie pozostałe stany formularza (`$valid`, `$pristine` i `$dirty`).

W ten sposób stworzyliśmy mechanizm wyświetlania wiadomości o błędach tylko wtedy, gdy wystąpi określony rodzaj błędu. Każdy z weryfikatorów wymienionych w tabeli 4.2 udostępnia klucz w obiekcie `$error`, przy użyciu którego można go wybrać i wyświetlić wiadomość dla użytkownika dotyczącą konkretnego rodzaju błędu. Chcesz poinformować użytkownika, że wypełnienie pola jest wymagane? W takim razie gdy użytkownik rozpocznie wpisywanie tekstu, wyświetl informację o minimalnej długości oraz odpowiedni komunikat, kiedy zostanie przekroczona długość maksymalna. Wszystkie tego typu warunkowe powiadomienia można wyświetlać za pomocą weryfikatorów AngularJS.

Moduł ngMessages

W poprzednim podrozdziale pokazaliśmy, jak warunkowo wyświetlać powiadomienia o błędach w formularzu. Ale czasami potrzebne są bardziej złożone warunki. Możemy też chcieć utworzyć wspólne powiadomienia o błędach dla wszystkich stron i pól formularzy. Do tego celu w systemie AngularJS służy opcjonalny moduł o nazwie `ngMessages`. Nie należy on do rdzennego pliku systemu (*angular.js*), tylko znajduje się w osobnym pliku JavaScript, który trzeba oddzielnie dołączyć do strony. Moduł `ngMessages` dostarcza następujące funkcje:

- możliwość łatwego definiowania wiadomości dla każdego błędu zamiast używania skomplikowanych warunków `if` i `show`;
- możliwość porządkowania wiadomości według priorytetów;
- możliwość dziedziczenia i rozszerzania wiadomości o błędach w całej aplikacji.

Abyś mógł używać modułu ngMessages w swoim projekcie, musisz wykonać następujące czynności:

1. Dołącz plik *angular-messages.js* (lub jego zminimalizowaną wersję) do pliku *index.html*.
2. Dodaj moduł ngMessages jako zależność do głównego modułu aplikacji.
3. Zastosuj dyrektywy ng-messages i ng-message.
4. Opcjonalnie zdefiniuj szablony dla najczęściej używanych wiadomości o błędach.

Poniżej przedstawiamy kompletny przykład ilustrujący sposób wykorzystania opisywanego modułu:

```
<!-- Plik: r04/ng-messages.html -->
<html ng-app="notesApp">
<head>
  <meta charset="utf-8">
  <title>Notes App</title>
</head>
<body ng-controller="MainCtrl as ctrl">

  <form ng-submit="ctrl.submit1()" name="simpleForm">
    <input type="email"
           name="uname"
           ng-model="ctrl.user1.username"
           required
           ng-minlength="6">
    <div ng-messages="simpleForm.uname.$error"
         ng-messages-include="error-messages"></div>
    <input type="password"
           name="pwd"
           ng-model="ctrl.user1.password"
           required>
    <div ng-messages="simpleForm.pwd.$error"
         ng-messages-include="error-messages"></div>
    <input type="submit"
           value="Wyślij"
           ng-disabled="simpleForm.$invalid">
  </form>

  <form ng-submit="ctrl.submit2()" name="overriddenForm">
    <input type="email"
           name="uname"
           ng-model="ctrl.user2.username"
           required
           ng-minlength="6">
    <div ng-messages="overriddenForm.uname.$error"
         ng-messages-include="error-messages">
      <div ng-message="required">Wpisz nazwę użytkownika</div>
    </div>
    <input type="password"
           name="pwd"
           ng-model="ctrl.user2.password"
           required>
    <div ng-messages="overriddenForm.pwd.$error">
      <div ng-message="required">Wpisz hasło</div>
    </div>
    <input type="submit"
           value="Wyślij"
           ng-disabled="overriddenForm.$invalid">
```

```
  </form>

<script
  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.11/angular.js">
</script>
<script src=
  "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.11/angular-messages.js">
</script>
<script type="text/ng-template" id="error-messages">
  <div ng-message="required">To pole jest obowiązkowe</div>
  <div ng-message="minlength">Wpisano za mało znaków</div>
  <div ng-message="email">Podaj prawidłowy adres e-mail</div>
</script>
<script type="text/javascript">
  angular.module('notesApp', ['ngMessages'])
    .controller('MainCtrl', [function() {
      var self = this;
      self.submit1 = function() {
        console.log('Użytkownik zatwierdził formularz z danymi ', self.user1);
      };
      self.submit2 = function() {
        console.log('Użytkownik zatwierdził formularz z danymi', self.user2);
      };
    }]);
</script>
</body>
</html>
```

W pierwszej chwili może się wydawać, że to długi i skomplikowany kod, ale bez trudu go zrozumiesz dzięki poniższemu opisowi fragment po fragmencie:

- Utworzyliśmy dwa formularze zawierające po dwa pola wejściowe: na nazwę użytkownika i hasło. Pole nazwy użytkownika w obu formularzach ma trzy weryfikatory: adresu e-mail (jest typu `email`), oznaczający, że wypełnienie tego pola jest obowiązkowe, oraz określający, że minimalna długość danych wynosi cztery znaki. Pole hasła ma tylko weryfikator, który oznacza, że jego wypełnienie jest obowiązkowe.
- Zanim przyjrzymy się dyrektywie `ng-messages`, przejdziemy na koniec dokumentu. Oprócz pliku *angular.js* dołączamy także plik *angular-messages.js* zawierający kod źródłowy modułu `ngMessages`. Ponadto dodaliśmy ten moduł jako zależność do modułu `notesApp`.
- Zdefiniowaliśmy element `<script>` typu `text/ng-template` i z identyfikatorem `error-messages`. Dzięki temu możemy definiować potrzebne nam fragmenty kodu HTML wewnątrz naszego głównego pliku *index.html*. Zawartość tego znacznika można by też było przenieść do osobnego pliku HTML.
- Kod HTML zdefiniowany w elemencie `<script>` zawiera domyślne powiadomienia o błędach i warunki ich wyświetlania. W tym przypadku zdefiniowaliśmy wiadomości o błędach dla weryfikatorów `required` i `minlength`.
- Teraz wracamy do formularzy, a konkretnie do formularza `simpleForm`. Po każdym polu tworzymy nowy element `<div>` z dyrektywą `ng-messages`. Do dyrektywy tej przekazujemy obiekt `$error`, na podstawie którego chcemy wyświetlać wiadomości o błędach. Dyrektywa `ng-messages` szuka kluczy w tym obiekcie i jeśli dopasuje konkretny warunek, powoduje wyświetlenie komunikatu w interfejsie użytkownika.

- Dyrektywa `ng-messages` umożliwia zdefiniowanie wiadomości o błędach bezpośrednio w dokumencie (jak w przypadku pola `password` w formularzu `overriddenForm`) lub dołączenie ich z zewnętrznego szablonu za pomocą atrybutu `ng-messages-include`.
- Atrybut `ng-messages-include` szuka osadzonego w dokumencie szablonu, czyli elementu `<script>` typu `text/ng-template` o określonym identyfikatorze bądź zewnętrznego pliku, który zostanie załadowany asynchronicznie przez system AngularJS.
- Kolejność definicji powiadomień w szablonie (albo jako dzieci dyrektywy `ng-messages`) określa porządek ich wyświetlania. W jednym polu formularza może jednocześnie występować kilka błędów. W naszym przypadku w polu nazwy użytkownika może zostać wpisany tekst zawierający mniej niż sześć znaków i niereprezentujący prawidłowego adresu e-mail. Jako że `minlength` zdefiniowaliśmy przed `email`, informacja o błędzie `minlength` zostanie wyświetlona pierwsza. A jeśli warunek dotyczący długości tekstu będzie spełniony, zostanie wyświetlona sama wiadomość dotycząca adresu e-mail.
- W razie potrzeby wiadomości o błędach znajdujące się w ogólnych szablonach można też przesłaniać. Dla pola `username` w formularzu `overriddenForm` przesłoniliśmy wiadomość dotyczącą błędu `required`, zamieniając ją na bardziej konkretną. Natomiast powiadomienia dla `minlength` i `email` pozostawiliśmy bez zmian. Moduł `ngMessages` rozpoznaje, które komunikaty należy zmienić, a które pozostawić.

Domyślnie moduł `ngMessages` wyświetla tylko pierwszą wiadomość o błędzie z listy warunków `ng-message`. Jeśli chcemy wyświetlić wszystkie informacje o błędach, które wystąpiły, należy dodatkowo użyć atrybutu `ng-messages-multiple`. Dzięki niemu zostaną pokazane wiadomości o wszystkich błędach, których warunki zostały spełnione.

Modułu `ngMessages` można też używać z własnymi wymaganiami i nie tylko w połączeniu z formularzami. Moduł ten sprawdza wartości kluczy danego obiektu i wyświetla komunikaty, jak instrukcja `switch`.

Podsumowując, moduł `ngMessages` pozwala znacznie uprościć mechanizm obsługi błędów w formularzach w AngularJS.

Stylizowanie formularzy i stanów

Wcześniej opisaliśmy różne stany formularzy (i ich pól wejściowych) — `$dirty`, `$valid` itd. Wiesz już, jak wyświetlać wybrane powiadomienia o błędach i wyłączać przyciski na podstawie tych warunków, ale nie umiesz jeszcze wyróżniać wybranych pól formularza lub całych formularzy za pomocą arkuszy stylów. Jedną możliwością jest użycie stanów formularzy i pól wejściowych w połączeniu z dyrektywą `ng-class` w celu dodania np. klasy `dirty`, gdy spełniony jest warunek `myForm.$dirty`. Jednak w systemie AngularJS istnieje prostsze rozwiązanie.

Dla każdego z opisanych wcześniej stanów AngularJS dodaje i usuwa klasy CSS, opisane w tabeli 4.3, do formularzy i elementów wejściowych.

Analogicznie dla każdego weryfikatora dodanego do pól wejściowych również otrzymujemy klasę CSS o podobnej nazwie, jak widać w tabeli 4.4.

Tabela 4.3. Klasy CSS dla stanów formularzy

Stan	Klasa CSS
`$invalid`	`ng-invalid`
`$valid`	`ng-valid`
`$pristine`	`ng-pristine`
`$dirty`	`ng-dirty`

Tabela 4.4. Klasy CSS dla stanów pól wejściowych

Stan pola wejściowego	Klasa CSS
`$invalid`	`ng-invalid`
`$valid`	`ng-valid`
`$pristine`	`ng-pristine`
`$dirty`	`ng-dirty`
`required`	`ng-valid-required` lub `ng-invalid-required`
`min`	`ng-valid-min` lub `ng-invalid-min`
`max`	`ng-valid-max` lub `ng-invalid-max`
`minlength`	`ng-valid-minlength` lub `ng-invalid-minlength`
`maxlength`	`ng-valid-maxlength` lub `ng-invalid-maxlength`
`pattern`	`ng-valid-pattern` lub `ng-invalid-pattern`
`url`	`ng-valid-url` lub `ng-invalid-url`
`email`	`ng-valid-email` lub `ng-invalid-email`
`date`	`ng-valid-date` lub `ng-invalid-date`
`number`	`ng-valid-number` lub `ng-invalid-number`

Oprócz stanu pola wejściowego AngularJS pobiera nazwę weryfikatora (`number`, `maxlength`, `pattern` itd.) i w zależności od tego, czy warunek tego weryfikatora został spełniony, czy nie, dodaje klasę `ng-valid-`*`nazwa_weryfikatora`* lub `ng-invalid-`*`nazwa_weryfikatora`*.

Spójrzmy na przykład wykorzystania tego do wyróżniania pól wejściowych na różne sposoby:

```
<!-- Plik: r04/form-styling.html -->
<html ng-app="notesApp">
<head>
  <title>Notes App</title>
  <meta charset="utf-8">
  <style>
    .username.ng-valid {
      background-color: green;
    }
    .username.ng-dirty.ng-invalid-required {
      background-color: red;
    }
    .username.ng-dirty.ng-invalid-minlength {
      background-color: lightpink;
    }
```

```
  </style>
</head>
<body ng-controller="MainCtrl as ctrl">

  <form ng-submit="ctrl.submit()" name="myForm">
    <input type="text"
           class="username"
           name="uname"
           ng-model="ctrl.user.username"
           required
           ng-minlength="4">
    <input type="submit"
           value="Wyślij"
           ng-disabled="myForm.$invalid">
  </form>

<script
  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.11/angular.js">
</script>
<script type="text/javascript">
  angular.module('notesApp', [])
    .controller('MainCtrl', [function() {
      var self = this;
      self.submit = function() {
        console.log('Użytkownik zatwierdził formularz z danymi ', self.user);
      };
    }]);
</script>
</body>
</html>
```

W przykładzie tym zachowaliśmy istniejącą funkcjonalność weryfikatorów, ale usunęliśmy wiadomości o błędach. Zamiast je wyświetlać, oznaczymy wymagane pola klasami CSS. Oto szczegółowy opis sposobu działania powyższego kodu:

- Gdy użytkownik poprawnie wypełni pole formularza, kolor pola zmienia się na zielony. Efekt ten osiągnięto poprzez ustawienie koloru tła w klasie CSS `ng-valid`, która jest dodawana do pola wejściowego.
- Kolor tła na czerwony zmieniamy tylko wtedy, gdy użytkownik zacznie coś wpisywać w polu tekstowym, a potem to cofnie. Innymi słowy, ustawiamy czerwone tło, które oznacza, że pole jest wymagane, dopiero wówczas, kiedy użytkownik zmodyfikuje jego zawartość. A zatem kolor tła zmienia się na czerwony, gdy zostaną zastosowane klasy CSS `ng-dirty` (oznacza modyfikację zawartości pola) i `ng-invalid-minlength` (oznacza, że użytkownik wpisał za mało znaków).

W podobny sposób można by było dodać klasę CSS wyświetlającą czerwony znak *, gdy pole jest wymagane, ale nie zostało zmienione. Przy użyciu kombinacji tych klas (oraz stanów formularza i pól wejściowych) można bez problemu sformatować i wyświetlić wszystkie informacje potrzebne użytkownikowi do poprawnego wypełnienia formularza.

ngModelOptions

Jak pewnie zauważyłeś, używając dyrektywy `ng-model`, każde naciśnięcie klawisza sprawia, że model AngularJS natychmiast aktualizuje i odświeża interfejs użytkownika. Powoduje to aktualizację całego interfejsu (jest to tzw. cykl *digest cycle*, który szczegółowo opisano w rozdziale 13.). Ale nie zawsze o to chodzi programiście. Czasami lepiej jest aktualizować model, gdy użytkownik na chwilę przestanie pisać albo przejdzie do następnego elementu wejściowego, co pozwala zredukować liczbę cyklów aktualizacji.

W AngularJS 1.3 wprowadzono możliwość dokładnego określania sposobu działania dyrektywy `ng-model` i aktualizowania interfejsu użytkownika. Za pomocą dyrektywy `ng-model-options` można definiować następujące opcje dyrektywy `ng-model`:

`updateOn`
: Jest to łańcuch określający, których zdarzeń pola wejściowego ma nasłuchiwać dyrektywa `ng-model` i dla których ma dokonywać aktualizacji interfejsu. Można użyć wartości `default`, oznaczającej nasłuchiwanie domyślnych zdarzeń kontrolki (dla pól tekstowych jest nim naciśnięcie klawisza, dla pól wyboru jest to kliknięcie itd.), lub wpisać konkretną nazwę zdarzenia, np. `blur`. W razie potrzeby można też podać listę zdarzeń rozdzielanych spacjami.

`debounce`
: Może to być liczba całkowita bądź obiekt. Opcja ta określa, ile milisekund system AngularJS ma odczekać z aktualizacją zmiennej modelu po tym, jak użytkownik przerwie wpisywanie tekstu. Jeśli przekazany zostanie obiekt, to można określić wartość `debounce` dla każdego zdarzenia określonego w opcji `updateOn`, np. `{"default": 500, "blur": 0}`. To ustawienie powoduje aktualizację pola tekstowego, gdy użytkownik przestanie pisać przez pół sekundy lub skończy edycję tego pola. Wartość `0` oznacza natychmiastową aktualizację.

`allowInvalid`
: Domyślnie opcja ta ma wartość `false`, sprawiającą, że AngularJS nie zapisze wartości w zmiennej modelu `ng-model`, jeśli wartość ta jest niepoprawna według obowiązujących weryfikatorów. Jeżeli wartość ma być ustawiana bez względu na stan weryfikatora, należy ustawić tę opcję na `true`.

`getterSetter`
: Wartość logiczna umożliwiająca traktowanie wyrażenia `ng-model` jako metody pobierającej i ustawiającej zamiast zmiennej.

Poniżej znajduje się przykład użycia dyrektywy `ng-model-options` w praktyce:

```
<!-- Plik: r04/ng-model-options.html -->
<html ng-app="modelApp">
<head>
  <meta charset="utf-8">
  <title>Ng Model Options</title>
</head>
<body ng-controller="MainCtrl as ctrl">

  <div>
    <input type="text"
           ng-model="ctrl.withoutModelOptions"
           ng-minlength="5"/>
```

```
    Wpisałeś {{ctrl.withoutModelOptions}}
  </div>
  <div>
    <input type="text"
           ng-model="ctrl.withModelOptions"
           ng-model-options="ctrl.modelOptions"
           ng-minlength="5"/>
    Wpisałeś {{ctrl.withModelOptions()}}
  </div>

<script
  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.11/angular.js">
</script>
<script type="text/javascript">
  angular.module('modelApp', [])
    .controller('MainCtrl', [function() {
      this.withoutModelOptions = '';
      var _localVar = '';
      this.modelOptions = {
        updateOn: 'default blur',
        debounce: {
          default: 1000,
          blur: 0
        },
        getterSetter: true,
        allowInvalid: true
      };
      this.withModelOptions = function(txt) {
        if (angular.isDefined(txt)) {
          _localVar = txt;
        } else {
          return _localVar;
        }
      };
    }]);
</script>
</body>
</html>
```

W przykładzie tym zostały wykorzystane dwa pola wejściowe. Pierwsze jest związane za pomocą dyrektywy `ng-model` ze zmienną `withoutModelOptions`. Zastosowano do niego weryfikator określający minimalną długość tekstu na pięć znaków. Wartość tego pola jest pokazywana w interfejsie użytkownika natychmiast po osiągnięciu minimalnej długości.

Drugi model jest związany z funkcją pobierającą i ustawiającą (`withModelOptions`). Normalnie funkcja ta nie działałaby z dyrektywą `ng-model`, ale dlatego właśnie dodatkowo zdefiniowaliśmy `ng-model-options`. Wartość przekazana do `ng-model-options` jest obiektem, który może być zdefiniowany bezpośrednio w kodzie HTML lub, jak jest w tym przypadku, może odnosić się do zmiennej z kontrolera. W dyrektywie `ng-model-options` zdefiniowaliśmy następujące opcje:

- powiązanie z domyślnymi typami zdarzeń i zdarzeniem `blur`;
- ustawienie czasu odbicia (`debounce`) na 1 sekundę dla wpisywania i natychmiastowej aktualizacji dla zdarzenia `blur`;

- powiązanie z funkcją pobierającą i ustawiającą zdefiniowaną w kontrolerze, a nie w zmiennej;
- umożliwienie ustawiania niepoprawnych wartości, dzięki czemu nawet jeśli pole formularza zostanie niepoprawnie wypełnione, nadal będziemy mieć dostęp do tej nieprawidłowej wartości w kontrolerze.

Ewentualnie można też dodać dyrektywę `ngModelOptions` do elementu wyższego poziomu (np. formularza albo elementu zawierającego całą aplikację AngularJS), aby zastosować ją domyślnie do wszystkich dyrektyw `ngModel` w aplikacji naraz, zamiast dodawać ją do każdego elementu z osobna.

Zagnieżdżanie formularzy i dyrektywa ng-form

Umiesz już tworzyć formularze oraz pobierać dane do i z kontrolerów (przez wykorzystanie wiązania danych z modelem). Wiesz już też, jak sprawdzać poprawność informacji oraz jak formatować i wyświetlać warunkowe powiadomienia o błędach.

W tym podrozdziale przedstawiamy techniki pracy z bardziej skomplikowanymi formularzami i grupami elementów. Czasami trzeba sprawdzić poprawność wypełnienia grupy pól formularza, a nie pojedynczych elementów. Sam element `<form>` języka HTML nie zapewnia odpowiednich możliwości w tym zakresie, ponieważ formularzy nie powinno się zagnieżdżać.

W systemie AngularJS dostępna jest dyrektywa `ng-form`, która działa podobnie jak element `<form>`, ale umożliwia zagnieżdżanie, dzięki czemu można grupować powiązane ze sobą pola formularza w sekcje:

```
<!-- Plik: r04/nested-forms.html -->
<html ng-app>
<head>
  <meta charset="utf-8">
  <title>Notes App</title>
</head>

<body>
  <form novalidate name="myForm">
    <div>
      <input type="text"
             class="username"
             name="uname"
             ng-model="ctrl.user.username"
             required=""
             placeholder="Nazwa użytkownika"
             ng-minlength="4" />
      <input type="password"
             class="password"
             name="pwd"
             ng-model="ctrl.user.password"
             placeholder="Hasło"
             required="" />
    </div>

    <ng-form name="profile">
      <input type="text"
```

```
            name="firstName"
            ng-model="ctrl.user.profile.firstName"
            placeholder="Imię"
            required>
      <input type="text"
            name="middleName"
            placeholder="Drugie imię"
            ng-model="ctrl.user.profile.middleName">
      <input type="text"
            name="lastName"
            placeholder="Nazwisko"
            ng-model="ctrl.user.profile.lastName"
            required>
      <input type="date"
            name="dob"
            placeholder="Data urodzenia"
            ng-model="ctrl.user.profile.dob">
    </ng-form>

    <span ng-show="myForm.profile.$invalid">
      Wypełnij pola formularza
    </span>

    <input type="submit"
           value="Wyślij"
           ng-disabled="myForm.$invalid"/>
  </form>

<script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.11/angular.js">
</script>
</body>

</html>
```

W przykładzie tym zagnieździliśmy formularz w formularzu głównym, ale jako że formularzy HTML nie można zagnieżdżać, wykorzystaliśmy dyrektywę `ng-form`. Teraz w naszym formularzu mamy stan podrzędny, możemy szybko sprawdzać poprawność wypełnienia każdej sekcji oraz mamy dostęp do tych samych stanów wiązania i formularza, których używaliśmy do tej pory. Oto krótki opis najważniejszych cech tego kodu:

- Formularz podrzędny został utworzony za pomocą dyrektywy `ng-form`. Można mu nadać nazwę, aby móc odwoływać się do jego stanu.
- Stan formularza podrzędnego można sprawdzić bezpośrednio (`profile.$invalid`) lub przez formularz nadrzędny (`myForm.profile.$invalid`).
- Poszczególnych elementów formularza można używać normalnie (`profile.firstName.$error.` ↪`required`).
- Formularze podrzędne i zagnieżdżone mają wpływ na formularz zewnętrzny (wartością `myForm.$invalid` jest `true` ze względu na zastosowanie wymaganych znaczników).

Istnieje możliwość utworzenia formularzy podrzędnych i grup z własnymi metodami określania poprawności danych, a dyrektywa `ng-form` umożliwia modelowanie tych grup w kodzie HTML.

Inne kontrolki formularzy

Pokazaliśmy, jak posługiwać się formularzami, dyrektywą `ng-model` i wiązaniami, ale tylko w odniesieniu do zwykłych pól tekstowych. W tym podrozdziale przedstawiamy także sposoby pracy z innymi elementami formularza w AngularJS.

Obszary tekstowe

Obszary tekstowe w systemie AngularJS obsługuje się dokładnie tak samo jak pola tekstowe. Można tworzyć dla nich dwustronne powiązania danych oraz oznaczać je jako wymagane, np.:

```
<textarea ng-model="ctrl.user.address" required></textarea>
```

Techniki wiązania danych, dostępu do stanów błędów i klasy CSS są takie same jak w przypadku zwykłych pól tekstowych.

Pola wyboru

Pola wyboru pod pewnymi względami są nawet łatwiejsze w obsłudze, ponieważ obsługują tylko dwie wartości: prawdę i fałsz. W efekcie dwustronne wiązanie danych `ng-model` do takiego pola przyjmuje wartość logiczną i określa stan zaznaczenia na podstawie tej wartości. Później wszelkie zmiany w tym polu powodują przełączenie stanu modelu:

```
<input type="checkbox" ng-model="ctrl.user.agree">
```

A gdybyśmy mieli coś innego niż wartości logiczne? Co, gdybyśmy chcieli przypisać łańcuch `TAK` lub `NIE` do modelu bądź sprawić, aby pole wyboru było zaznaczane dla wartości `TAK`? W AngularJS dostępne są dwa atrybuty dla pól wyboru, które umożliwiają określenie własnych wartości do oznaczania prawdy i fałszu. Poniżej znajduje się przykład ich użycia:

```
<input type="checkbox"
           ng-model="ctrl.user.agree"
           ng-true-value="'TAK'"
           ng-false-value="'NIE'">
```

To spowoduje ustawienie wartości pola `agree` na `TAK`, jeśli użytkownik zaznaczy pole wyboru, i na `NIE` w przeciwnym wypadku.

Zwróć uwagę na pojedyncze cudzysłowy, w które ujęto wartości `TAK` i `NIE`. Od wersji AngularJS 1.3 atrybuty `ng-true-value` i `ng-false-value` przyjmują jako argumenty wyrażenia stałe. Wcześniej można było wpisywać wartości reprezentujące prawdę i fałsz bezpośrednio, np. `ng-true-value="TAK"`. Ale w AngularJS 1.3 dla spójności z resztą dyrektyw wprowadzono zasadę, że atrybuty `ng-true-value` i `ng-false-value` jako argumenty przyjmują tylko wyrażenia stałe.

Należy podkreślić, że obecnie nie ma możliwości przekazania do atrybutów `ng-true-value` i `ng-false-value` referencji do zmiennej. Przyjmują one wyłącznie stałe łańcuchy. Nie można odnieść się do zmiennej kontrolera dla atrybutu `ng-true-value` lub `ng-false-value`.

A co zrobić, jeśli nie chcemy korzystać z dwustronnego wiązania danych, tylko użyć pola wyboru do wyświetlenia bieżącej wartości logicznej? Jest to jednostronne wiązanie, w którym stan zaznaczenia pola wyboru zmienia się wraz ze stojącą za nim wartością, ale nie w reakcji na zaznaczenie czy usunięcie zaznaczenia tego pola.

W takim przypadku można skorzystać z dyrektywy `ng-checked`, wiążącej się z wyrażeniami AngularJS. Gdy wartość jest prawdziwa, system AngularJS ustawia elementowi wejściowemu własność oznaczającą, że jest zaznaczony, oraz usuwa ją, gdy wartość ta jest fałszywa. Poniższej znajduje się przykład zastosowania opisywanych technik:

```
<!-- Plik: r04/checkbox-example.html -->
<html ng-app="notesApp">
<head>
  <meta charset="utf-8">
  <title>Notes App</title>
</head>
<body ng-controller="MainCtrl as ctrl">
  <div>
    <h2>Jaki jest Twój ulubiony sport?</h2>
    <div ng-repeat="sport in ctrl.sports">
      <label ng-bind="sport.label"></label>
      <div>
        Z wiązaniem:
        <input type="checkbox"
                    ng-model="sport.selected"
                    ng-true-value="'TAK'"
                    ng-false-value="'NIE'">
      </div>
      <div>
        Z użyciem ng-checked:
        <input type="checkbox"
               ng-checked="sport.selected === 'TAK'">
      </div>
      <div>
        Aktualny stan: {{sport.selected}}
      </div>
    </div>
  </div>

<script
  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.11/angular.js">
</script>
<script type="text/javascript">
  angular.module('notesApp', [])
    .controller('MainCtrl', [function() {
      var self = this;
      self.sports = [
        {label: 'Koszykówka', selected: 'TAK'},
        {label: 'Krykiet', selected: 'NIE'},
        {label: 'Piłka nożna', selected: 'NIE'},
        {label: 'Pływanie', selected: 'TAK'}
      ];
    }]);
</script>
</body>
</html>
```

W przykładzie zdefiniowaliśmy blok ng-repeat, zawierający pola wyboru z dyrektywami ng-model i ng-checked oraz element , w którym wyświetlono bieżący stan. Pierwsze pole wyboru wykorzystuje tradycyjne dwustronne wiązanie danych przy użyciu dyrektywy ng-model. Natomiast drugie wykorzystuje dyrektywę ng-checked. Oznacza to, że:

- Jeśli użytkownik zaznaczy pierwsze pole wyboru, wartość selected zmieni się na TAK, ponieważ dyrektywa ng-true-value definiuje właśnie taką wartość oznaczającą prawdę. To powoduje uruchomienie dyrektywy ng-checked i zaznaczenie drugiego pola (lub usunięcie jego zaznaczenia).
- Gdy użytkownik usunie zaznaczenie pierwszego pola wyboru, wartość selected zostaje ustawiona na NIE dzięki dyrektywie ng-false-value.
- Drugie pole wyboru w każdym elemencie pętli wyświetla stan ng-model przy użyciu dyrektywy ng-checked. Powoduje ono aktualizację stanu pola wyboru za każdym razem, gdy zmieni się model ng-model. Zaznaczenie i usunięcie zaznaczenia tego drugiego pola wyboru nie ma wpływu na wartość modelu.

Jeśli więc potrzebujesz dwustronnego wiązania danych, użyj dyrektywy ng-model. A jeżeli potrzebujesz jednostronnego wiązania z polami wyboru, skorzystaj z dyrektywy ng-checked.

Przyciski radiowe

Przyciski radiowe są podobne do pól wyboru, ale różnią się od nich pod kilkoma względami. Można utworzyć wiele przycisków radiowych (normalnie tak się robi), z których każdy w razie zaznaczenia przypisuje inną wartość do modelu. Wartość tę można określić przy użyciu zwykłego atrybutu value elementu <input>. Spójrz na poniższy przykład:

```
<div ng-init="user = {gender: 'female'}">
    <input type="radio"
           name="gender"
           ng-model="user.gender"
           value="male">
    <input type="radio"
           name="gender"
           ng-model="user.gender"
           value="female">
</div>
```

W kodzie tym zdefiniowane są dwa przyciski radiowe o takiej samej nazwie, dzięki czemu zaznaczenie jednego spowoduje usunięcie zaznaczenia drugiego. Oba te przyciski są związane z tym samym modelem (user.gender). Każdy ma określoną wartość, która zostanie zapisana w user.gender (male w przypadku pierwszego przycisku i female w przypadku drugiego). Ponadto całość znajduje się w bloku ng-init, domyślnie ustawiającym wartość user.gender na female. Dzięki temu bezpośrednio po wczytaniu tego kodu do przeglądarki domyślnie zostanie zaznaczone drugie pole.

Ale co by było, gdyby wartości były dynamiczne? Wartość do przypisania mogłaby być wybierana w kontrolerze lub innym miejscu. Wówczas należałoby użyć atrybutu AngularJS ng-value, którym można posługiwać się w połączeniu z przyciskami radiowymi. Atrybut ten pobiera wyrażenie AngularJS i przypisuje do modelu wartość zwrotną tego wyrażenia:

```
<div ng-init="otherGender = 'inna'">
    <input type="radio"
           name="gender"
           ng-model="user.gender"
           value="male">Mężczyzna
    <input type="radio"
           name="gender"
           ng-model="user.gender"
           value="female">Kobieta
    <input type="radio"
           name="gender"
           ng-model="user.gender"
           ng-value="otherGender">{{otherGender}}
</div>
```

W kodzie tym trzecia opcja przyjmuje wartość dynamiczną. Przypisujemy ją w bloku inicjacyjnym (`ng-init`), ale w prawdziwej aplikacji inicjacji dokonywano by w kontrolerze, a nie bezpośrednio w kodzie HTML. Wyrażenie `ng-value="otherGender"` nie powoduje przypisania do `user.gender` wartości `otherGender` jako łańcucha, tylko wartości zmiennej `otherGender`, czyli *inna*.

Pola kombi i listy rozwijane

Ostatni element formularzy HTML (którego można też używać poza formularzami) to pole `<select>` czy lista rozwijana zwana również polem kombi. Spójrzmy na prosty przykład zastosowania takiej listy w AngularJS:

```
<div ng-init="location = 'Indie'">
    <select ng-model="location">
      <option value="USA">USA</option>
      <option value="India">Indie</option>
      <option value="Other">Żadne z powyższych</option>
    </select>
</div>
```

Jest to definicja prostej listy rozwijanej powiązanej danymi ze zmienną `location`. Na początku zainicjowaliśmy tę zmienną wartością `Indie`, dzięki czemu po wczytaniu strony automatycznie zostanie zaznaczona opcja *Indie*. Gdy użytkownik wybierze inną opcję, do modelu `ng-model` zostanie przypisana nowa wartość atrybutu `value`. Do pola tego mają zastosowanie standardowe weryfikatory i stany (`required` itd.).

Obowiązują jednak pewne ograniczenia:

- Trzeba z góry znać wartości listy.
- Wartości muszą być wpisane na sztywno.
- Wartości muszą być łańcuchami.

W prawdziwie dynamicznej aplikacji niektóre z tych warunków mogą nie być spełnione. Wówczas można dynamicznie generować opcje elementu `<select>` i posługiwać się obiektami zamiast łańcuchami. W tym celu należy wykorzystać dyrektywę `ng-options`. Zobaczmy, jak to się robi:

```
<!-- Plik: r04/select-example.html -->
<html ng-app="notesApp">
<head>
  <meta charset="utf-8">
  <title>Notes App</title>
</head>
<body ng-controller="MainCtrl as ctrl">

<div>
  <select ng-model="ctrl.selectedCountryId"
          ng-options="c.id as c.label for c in ctrl.countries">
  </select>
  Identyfikator wybranego kraju: {{ctrl.selectedCountryId}}
</div>

<div>
  <select ng-model="ctrl.selectedCountry"
          ng-options="c.label for c in ctrl.countries">
  </select>

  Wybrany kraj: {{ctrl.selectedCountry}}
</div>

<script
  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.11/angular.js">
</script>
<script type="text/javascript">
  angular.module('notesApp', [])
    .controller('MainCtrl', [function() {
      this.countries = [
        {label: 'USA', id: 1},
        {label: 'Indie', id: 2},
        {label: 'Inny', id: 3}
      ];
      this.selectedCountryId = 2;
      this.selectedCountry = this.countries[1];
    }]);
</script>
</body>
</html>
```

W przykładzie tym zdefiniowano dwie listy rozwijane, z których każda jest związana z innym modelem w kontrolerze. Pierwszy element `<select>` jest związany z modelem `ctrl.selectedCountryId`, a drugi — z modelem `ctrl.selectedCountry`. Zauważ, że pierwszy jest liczbą, a drugi obiektem. Jak to możliwe?

- W każdym elemencie `<select>` zdefiniowaliśmy atrybut `ng-options`, umożliwiający przeglądanie tablicy (albo obiektu, podobnie jak przy użyciu dyrektywy `ng-repeat` opisanej w podrozdziale „Praca z tablicami i wyświetlanie ich zawartości”) i wyświetlanie dynamicznych opcji.
- Także składnia jest podobna do składni dyrektywy `ng-repeat`, z dodatkiem możliwości wybierania tego, co jest wyświetlone jako etykieta i jest związane z modelem.
- W pierwszej liście rozwijanej zdefiniowano atrybut `ng-options="c.id as c.label for c in ctrl.countries"`. Nakazuje on systemowi AngularJS utworzenie po jednej opcji dla każdego kraju znajdującego się w tablicy `countries`. Składnia tego jest następująca: *`wartośćModelu`* `as`

wartośćEtykiety `for` *element* `in` *tablica*. W tym przypadku informujemy system AngularJS, że naszą wartością modelu jest identyfikator każdego elementu, a wartością etykiety jest klucz `label` każdego elementu tablicy.

- W drugiej liście rozwijanej zdefiniowano atrybut `ng-options="c.label for c in ctrl.countries"`. W tym przypadku opuszczono *wartośćModelu*, więc system AngularJS przyjmuje założenie, że każdy element w pętli jest rzeczywistą wartością modelu, i gdy użytkownik wybierze opcję z drugiej listy, następuje przypisanie obiektu kraju (`c`) tej opcji do zmiennej `ctrl.selectedCountry`.
- Jako że każda z tych list używa innego modelu, zmiana w jednej nie ma wpływu na wybór wartości w drugiej.
- Ewentualnie można dodać klauzulę grupowania w postaci `ng-options="`*wartośćModelu* `as` *wartośćEtykiety* `group by` *wartośćGrupowania* `for` *element* `in` *tablica*`"`. Podobnie jak określiliśmy wartości modelu i etykiety, *wartośćGrupowania* możemy ustawić na inny klucz w obiekcie (np. `continent`).
- Dla obiektów klauzula zmienia się następująco: *wartośćModelu* `as` *wartośćEtykiety* `group by` *wartośćGrupowania* `for (`*klucz*`,` *wartość*`) in` *obiekt*.

System AngularJS porównuje poszczególne wartości `ng-options` z `ng-model` przez referencję. Dzięki temu nawet jeśli oba porównywane elementy są obiektami zawierającymi takie same klucze i wartości, AngularJS nie wyświetli elementu jako zaznaczonego na liście wyboru, jeżeli oba porównywane elementy nie będą tym samym obiektem. W przedstawionym przykładzie poradziliśmy sobie poprzez użycie elementu z tablicy `countries` do przypisania wartości początkowej modelu.

Ale jest lepszy sposób, polegający na wykorzystaniu w dyrektywie `ng-options` opcji `track by`. Dyrektywę tę mogliśmy zapisać tak:

```
ng-options="c.label for c in ctrl.countries track by c.id"
```

Dzięki temu obiekt `c` będzie porównywany pod względem wartości pola `id`, a nie domyślnie przez referencję.

Podsumowanie

Zaczęliśmy od najczęściej spotykanych wymagań, czyli pobierania danych do i z formularzy w interfejsie użytkownika. Wykorzystywaliśmy dyrektywę `ng-model`, służącą do dwustronnego wiązania danych, aby pozbyć się dużej części szablonowego kodu, który normalnie trzeba by było napisać do obsługi formularzy. Później pokazaliśmy, jak zastosować techniki weryfikacji wprowadzonych do formularza danych oraz jak wyświetlać i formatować powiadomienia o błędach. Na koniec przedstawiliśmy metody pracy z innymi typami elementów formularza w systemie AngularJS.

W następnym rozdziale poznasz usługi systemu AngularJS i nauczysz się komunikować z serwerem za pomocą usługi `$http`.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

T

U

W

X

Z

Ż

Notatki

O'REILLY®

Praktyczne wprowadzenie do świata AngularJS!

AngularJS to szkielet pozwalający tworzyć zaawansowane aplikacje działające w przeglądarce internetowej. Dzięki niemu udało się przenieść do języka JavaScript najlepsze wzorce znane z tradycyjnych języków programowania, takich jak Java czy C#. To posunięcie pozwoliło też programistom na szybsze testowanie kodu, tworzenie przejrzystej architektury oraz wydajniejszą pracę.

Jeżeli chcesz skorzystać z tych wszystkich udogodnień, musisz poznać budowę oraz najlepsze praktyki tworzenia aplikacji z wykorzystaniem AngularJS. Lektura tej książki ułatwi Ci to zadanie! Sięgnij po nią i poznaj wzorzec MVC (ang. Model-View-Controller), skonfiguruj swoje środowisko pracy oraz stwórz pierwszą aplikację. W trakcie lektury kolejnych rozdziałów będziesz zdobywać fundamentalną wiedzę na temat dyrektyw, testów jednostkowych i pracy z formularzami. Następnie nauczysz się komunikować z serwerem, korzystając z usługi $http, oraz przekonasz się, do czego służy moduł ngRoute. Na sam koniec dowiesz się, jak tworzyć, testować i korzystać z własnych dyrektyw, a także zapoznasz się z najlepszymi praktykami, które ułatwią Ci codzienne życie. Dzięki tej książce błyskawicznie poznasz i wykorzystasz możliwości AngularJS!

Dzięki tej książce:

- Poznasz wzorzec MVC i zastosujesz go w praktyce
- Wykorzystasz podstawowe dyrektywy oraz stworzysz własne
- Nawiążesz połączenie i pobierzesz dane z serwera
- Opanujesz zagadnienia związane z testowaniem
- Zaznajomisz się z najlepszymi praktykami
- Wybierzesz AngularJS do swojego kolejnego projektu

Shyam Seshadri – twórca popularnego szkieletu Slim Framework. Pomysłodawca inicjatywy PHP The Right Way, promującej najlepsze praktyki kodowania w języku PHP. Programista w firmie New Media Campaigns z siedzibą w Karolinie Północnej.

Brad Green – kierownik projektów inżynieryjnych w Google. Zaangażowany w projekt AngularJS. Wcześniej zdobywał doświadczenie w branży mobilnej oraz współpracował ze Steve'em Jobsem w firmie NeXT.

Helion

37700 numer katalogowy

księgarnia internetowa

http://helion.pl

zamówienia telefoniczne

0 801 339900

0 601 339900

Informatyka w najlepszym wydaniu

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-283-1619-5

9 788328 316195

cena: 49,00 zł