Poznaj możliwości AngularJS!

AngularJS

Brad Green
Shyam Seshadri

Tytuł oryginału: AngularJS

Tłumaczenie: Robert Górczyński

ISBN: 978-83-246-9990-2

© 2014 Helion S.A.

Authorized Polish translation of the English edition AngularJS, ISBN 9781449344856
© 2013 Brad Green and Shyam Seshadri.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

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

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)

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

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

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści

Wprowadzenie **7**
Konwencje zastosowane w książce 8
Użycie przykładowych kodów 8
Podziękowania 9

Rozdział 1. Wprowadzenie do AngularJS **11**
Koncepcje 12
Przykład — koszyk na zakupy 18
Co dalej? 21

Rozdział 2. Anatomia aplikacji AngularJS **23**
Wywołanie AngularJS 23
Architektura MVC 24
Szablony i dołączanie danych 27
Organizacja zależności za pomocą modułów 51
Formatowanie danych za pomocą filtrów 55
Zmiana widoków za pomocą tras i usługi $location 57
Komunikacja z serwerem 61
Użycie dyrektyw do zmiany elementów drzewa DOM 63
Weryfikacja danych wejściowych użytkownika 65
Co dalej? 67

Rozdział 3. Programowanie w AngularJS **69**
Organizacja projektu 70
Narzędzia 73
Uruchamianie aplikacji 75
Testowanie w AngularJS 76
Testy jednostkowe 79

Kup ksi k Pole ksi k

Testy typu E2E/integracji 80
Kompilacja 82
Inne wspaniałe narzędzia 84
Narzędzie Yeoman — optymalizacja sposobu pracy 88
Integracja AngularJS i RequireJS 92

Rozdział 4. Analiza aplikacji AngularJS 101
Aplikacja 101
Relacje między modelem, kontrolerem i szablonem 102
Kontrolery, dyrektywy i usługi 105
Testy 122

Rozdział 5. Komunikacja z serwerami 129
Komunikacja za pomocą usługi $http 129
Testy jednostkowe 135
Praca z zasobami RESTful 137
Usługa $q i obietnica 143
Przechwycenie odpowiedzi 145
Kwestie bezpieczeństwa 146
XSRF 147

Rozdział 6. Dyrektywy 149
Dyrektywy i weryfikacja kodu HTML 149
Ogólny opis API 150
Co dalej? 170

Rozdział 7. Inne kwestie 171
Usługa $location 171
Metody modułu AngularJS 178
Komunikacja między zasięgami za pomocą $on, $emit i $broadcast 182
Ciasteczka 184
Internacjonalizacja i lokalizacja 185
Oczyszczanie kodu HTML i moduł Sanitize 188

Kup ksi k Pole ksi k

Rozdział 8. Ściąga i podpowiedzi 191
Opakowanie kontrolki jQuery datepicker 191
Lista klubów sportowych — filtrowanie i komunikacja 196
Przekazywanie plików w aplikacji AngularJS 201
Użycie biblioteki Socket.IO 204
Prosta usługa stronicowania 207
Praca z serwerami i logowaniem 210
Podsumowanie 214

Skorowidz 216

Kup ksi k Pole ksi k

Kup książkę Poleć książkę

ROZDZIAŁ 4.

Analiza aplikacji AngularJS

W rozdziale 2. przedstawiono pewne najczęściej używane funkcje frameworka AngularJS, natomiast w rozdziale 3. zajęliśmy się zagadnieniami związanymi ze sposobem prowadzenia prac programistycznych. Zamiast kontynuować wątek i podobnie szczegółowo omawiać poszczególne funkcje, w tym rozdziale przejdziemy do małej, rzeczywistej aplikacji. Na jej podstawie dowiesz się, jak połączyć ze sobą omówione dotąd fragmenty całości i utworzyć rzeczywistą, działającą aplikację.

Zamiast od razu przedstawiać całą aplikację, będziemy ją poznawać w małych częściach, omawiać interesujące zagadnienia związane z danym fragmentem i tym samym powoli budować kompletną aplikację, która będzie gotowa, zanim ukończysz lekturę rozdziału.

Aplikacja

GutHub to prosta aplikacja przeznaczona do zarządzania przepisami kulinarnymi. Została zaprojektowana w taki sposób, aby przechowywać przepisy kulinarne i jednocześnie pokazywać różne interesujące aspekty aplikacji AngularJS. Oto cechy charakteryzujące tworzoną przez nas aplikację:

- ma układ składający się z dwóch kolumn;
- pasek nawigacyjny znajduje się po lewej stronie;
- pozwala na dodawanie nowych przepisów kulinarnych;
- umożliwia przeglądanie istniejących przepisów kulinarnych.

Widok główny aplikacji znajduje się po prawej stronie. W zależności od adresu URL ulega ona zmianie i wyświetla listę przepisów kulinarnych,

Kup ksi k Pole ksi k

szczegóły dotyczące konkretnego przepisu lub edytowalny formularz pozwalający na dodanie nowego bądź na edycję istniejącego. Uruchomioną aplikację pokazano na rysunku 4.1.

Rysunek 4.1. GutHub, czyli prosta aplikacja przeznaczona do zarządzania przepisami kulinarnymi

Cała aplikacja jest dostępna w repozytorium GitHub na stronie: *https://github.com/shyamseshadri/angularjs-book/tree/master/chapter4*.

Relacje między modelem, kontrolerem i szablonem

Zanim przejdziemy do omawiania aplikacji, zatrzymajmy się na chwilę i zastanówmy, jak trzy fragmenty aplikacji współdziałają ze sobą oraz jak powinniśmy je traktować.

Model jest istotą aplikacji. Powtórz to zdanie kilkakrotnie. Działanie całej aplikacji opiera się na modelu, od którego zależą: wyświetlany widok, dane wyświetlane przez widok, zapisywane informacje i dosłownie wszystko. Warto więc poświęcić nieco czasu i dokładnie przemyśleć model — jakie właściwości powinien mieć obiekt modelu, jak będzie pobierany z serwera, jak będzie zapisywany i tak dalej. Ze względu na to, że uaktualnienie widoku następuje automatycznie dzięki użyciu wiązania danych, uwagę należy skoncentrować na modelu.

Kup ksi k Pole ksi k

Kontroler zawiera logikę biznesową i określa między innymi: jak będzie pobierany model, jakie będą rodzaje operacji przeprowadzanych na modelu, jakich informacji widok potrzebuje z modelu, a także jak przekształcić model, aby uzyskać potrzebne dane. Ponadto przeprowadzanie weryfikacji, wykonywanie wywołań do serwera, umieszczanie odpowiednich danych w widoku oraz właściwie wszystko inne powiązane z wymienionymi działaniami to również aktywność definiowana w kontrolerze.

I na koniec *szablon* określa sposób prezentacji modelu oraz interakcji użytkownika z aplikacją. Zadania wykonywane przez szablon powinny być ograniczone do wymienionych poniżej:

- wyświetlanie modelu;
- definiowanie sposobów, na jakie użytkownik może korzystać z aplikacji — kliknięcia, pola danych wejściowych i tak dalej;
- nadawanie stylu aplikacji oraz określanie, jak i kiedy pewne elementy mają być wyświetlane — pokazywanie lub ukrywanie i tak dalej;
- filtrowanie i formatowanie danych (zarówno wejściowych, jak i wyjściowych).

Trzeba pamiętać, że szablon w AngularJS niekoniecznie jest widokiem w architekturze MVC (model – widok – kontroler). Zamiast tego widok jest skompilowaną wersją wykonywanego szablonu, rodzajem połączenia szablonu i modelu.

W szablonie nie należy umieszczać żadnego rodzaju logiki biznesowej ani definiować zachowania — tego rodzaju dane powinny znajdować się w kontrolerze. Zachowanie prostoty szablonów pozwala na właściwą separację obowiązków, a ponadto na przetestowanie większości kodu za pomocą jedynie testów jednostkowych. Szablony powinny być testowane za pomocą testów scenariuszy.

W tym miejscu mógłbyś zapytać: gdzie należy umieszczać polecenia odpowiedzialne za modyfikacje obiektowego modelu dokumentu? Operacje na elementach drzewa DOM nie powinny być definiowane w kontrolerach lub szablonach. Najlepszym miejscem dla nich są dyrektywy AngularJS, choć czasami wspomniane operacje mogą być stosowane za pomocą usług, co pozwala na uniknięcie powielania kodu. Przykład takiego rozwiązania w aplikacji GutHub również zostanie zaprezentowany i omówiony.

Bez zbędnych ceregieli przechodzimy więc do modelu.

Kup ksi k Pole ksi k

Model

W omawianej aplikacji staramy się zachować maksymalną prostotę modelu — będą to po prostu przepisy kulinarne. Wspomniane przepisy to jedyny obiekt modelu w całej aplikacji. Wszystkie pozostałe komponenty zostaną zbudowane wokół modelu.

Każdy przepis składa się z następujących właściwości:

- identyfikator, jeśli przepis został zapisany na serwerze,
- nazwa,
- krótki opis,
- sposób przygotowania,
- informacje o ewentualnym wyróżnieniu przepisu,
- tablica składników podanych w postaci nazwy, ilości i jednostki miary.

I to tyle, model jest niezwykle prosty. Pozostałe komponenty aplikacji utworzymy na podstawie wymienionego modelu. Poniżej przedstawiono przykładowy przepis kulinarny w formacie JSON (przepis ten został pokazany na rysunku 4.1 we wcześniejszej części rozdziału):

```
{
  "id": "1",
  "title": "Ciasteczka",
  "description": "Wyborne, chrupiące na zewnątrz, ciągliwe" +
  " w środku i ociekające pyszną czekoladą " +
  "ciasteczka. Najlepsze w swoim rodzaju.",
  "ingredients": [
  {
    "amount": "1",
    "amountUnits": "opakowanie",
    "ingredientName": "Chips Ahoy"
  }
  ],
  "instructions": "1. Kup opakowanie Chips Ahoy\n" +
    "2. Podgrzej ciasteczka w piekarniku\n" +
    "3. Rozsmakuj się w ciepłych ciasteczkach\n" +
    "4. Z innego źródła naucz się, jak wypiekać wyborne ciasteczka"
}
```

W celu zachowania prostoty przykładu nie będziemy zajmować się serwerem, z którego przepisy kulinarne są pobierane lub w którym są zapisywane. Kod serwera znajduje się w repozytorium w serwisie GitHub, a do jego uruchomienia służy polecenie `node web-server.js`, które trzeba wydać z poziomu podstawowego katalogu aplikacji GutHub. Przechodzimy teraz do znacznie bardziej skomplikowanych funkcji interfejsu użytkownika, jakie można utworzyć na postawie naszego prostego modelu.

Kup ksi k Pole ksi k

Kontrolery, dyrektywy i usługi

Wreszcie możemy zająć się ciekawszymi aspektami tworzonej przez nas aplikacji. W pierwszej kolejności spojrzymy na kod dyrektyw i usług i powiemy sobie nieco o sposobie jego działania. Następnie przejdziemy do wielu kontrolerów niezbędnych do zapewnienia prawidłowego działania tworzonej aplikacji.

Usługi

Poniżej przedstawiono kod źródłowy usług.

```
// Plik: app/scripts/services/services.js.
var services = angular.module('guthub.services', ['ngResource']);

services.factory('Recipe', ['$resource', function($resource) {
  return $resource('/recipes/:id', {id: '@id'});
}]);

services.factory('MultiRecipeLoader', ['Recipe', '$q', function(Recipe, $q) {
  return function() {
    var delay = $q.defer();
    Recipe.query(function(recipes) {
      delay.resolve(recipes);
    }, function() {
      delay.reject('Nie można pobrać przepisów kulinarnych.');
    });
    return delay.promise;
  };
}]);

services.factory('RecipeLoader', ['Recipe', '$route', '$q', function(Recipe,
$route, $q) {
  return function() {
    var delay = $q.defer();
    Recipe.get({id: $route.current.params.recipeId}, function(recipe) {
      delay.resolve(recipe);
    }, function() {
      delay.reject('Nie można pobrać przepisu '  + $route.current.params.recipeId);
    });
    return delay.promise;
  };
}]);
```

Najpierw zajmiemy się usługami. Nie jest to nasze pierwsze spotkanie z usługami — zetknęliśmy się z nimi już w rozdziale 2. Tutaj zostaną omówione nieco dokładniej.

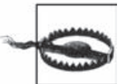
W przedstawionym pliku znajdują się trzy usługi AngularJS.

Kup ksi k Pole ksi k

Istnieje usługa przepisu kulinarnego, która zwraca tak zwany AngularJS Resource. Jest to zasób RESTful prowadzący do serwera RESTful. Wspomniany zasób hermetyzuje działającą na niskim poziomie usługę `$http`, a tym samym programista musi utworzyć jedynie kod odpowiedzialny za pracę z obiektami.

Za pomocą tylko pojedynczego wiersza kodu (`return $resource`) oraz oczywiście zależności w module `guthub.services` obiekt `Recipe` może być użyty jako argument w dowolnym kontrolerze — zostanie wówczas wstrzyknięty do wskazanego kontrolera. Co więcej, każdy obiekt `Recipe` ma wbudowane wymienione poniżej metody:

- `Recipe.get()`,
- `Recipe.save()`,
- `Recipe.query()`,
- `Recipe.remove()`,
- `Recipe.delete()`.



Jeżeli zamierzasz użyć metody `Recipe.delete()` i chcesz zapewnić działanie aplikacji w przeglądarce Internet Explorer, wtedy musisz użyć wywołania w postaci `Recipe['delete']()`. Wynika to z faktu, że `delete` jest słowem kluczowym w przeglądarce Internet Explorer.

Z wymienionych powyżej metod wszystkie poza `query()` działają z pojedynczym przepisem kulinarnym. Natomiast wartością zwrotną metody `query()` jest domyślnie tablica przepisów kulinarnych.

Wiersz kodu deklarujący zasób (`return $resource`) wykonuje również kilka innych użytecznych zadań.

1. Zwróć uwagę na `:id` w adresie URL wskazanym dla zasobu RESTful. Wspomniany identyfikator oznacza, że w trakcie wykonywania dowolnego zapytania (na przykład za pomocą `Recipe.get()`), jeśli przekażesz obiekt wraz z właściwością `id`, wówczas jej wartość zostanie umieszczona na końcu adresu URL.

   Oznacza to, że wywołanie `Recipe.get({id: 15})` faktycznie będzie wywołaniem do */recipe/15*.

2. Mógłbyś zapytać w tym miejscu: co z drugim obiektem, na przykład `{id: @id}`? Wiersz kodu jest wart tysiąca słów objaśnienia, więc przejdźmy od razu do odpowiedniego przykładu.

Kup ksi k Pole ksi k

Przyjmujemy założenie, że dostępny jest obiekt `Recipe` zawierający wszystkie niezbędne informacje, między innymi wartość `id`.

Wspomniany obiekt można zapisać za pomocą poniższego fragmentu kodu:

```
// Przyjęto założenie, że obiekt existingRecipeObj ma wszystkie niezbędne właściwości,
// w tym id (na przykład o wartości 13).
var recipe = new Recipe(existingRecipeObj);
recipe.$save();
```

Przedstawiony kod spowoduje wykonanie żądania `POST` do */recipe/13*.

Fragment `@id` powoduje pobranie wartości właściwości `id` obiektu i użycie jej jako parametru `id`. Takie rozwiązanie przyjęto dla wygody — pozwala ono zaoszczędzić kilka wierszy kodu.

W pliku *apps/scripts/services/services.js* istnieją jeszcze dwie inne usługi. Obie zaliczają się do komponentów wczytujących: pierwsza (`RecipeLoader`) wczytuje pojedynczy przepis, natomiast druga (`MultiRecipeLoader`) jest przeznaczona do wczytywania wszystkich przepisów kulinarnych. Wymienione usługi są używane podczas konfiguracji tras, a sposób działania tych usług jest bardzo podobny i został przedstawiony poniżej.

1. Utworzenie obiektu wstrzymanego `$q` (jest to rodzaj obietnicy frameworka AngularJS stosowanej w celu łączenia funkcji asynchronicznych).
2. Wykonanie wywołania do serwera.
3. Określenie obiektu wstrzymanego, gdy serwer zwraca wartość.
4. Zwrot obietnicy, która będzie używana przez mechanizm routingu frameworka AngularJS.

Obietnice frameworka AngularJS

Obietnica to interfejs przeznaczony do pracy z obiektami, które są zwracane lub będą wypełnione w przyszłości (w zasadzie są to akcje asynchroniczne). Ogólnie rzecz biorąc, na obietnicę składa się obiekt oraz funkcja `then()`.

Aby zobaczyć zalety obietnic, spójrzmy na przykład, w którym konieczne jest pobranie profilu użytkownika:

```
var currentProfile = null;
var username = 'dowolnaNazwa';
fetchServerConfig(function(serverConfig) {
  fetchUserProfiles(serverConfig.USER_PROFILES, username,
    function(profiles) {
    currentProfile = profiles.currentProfile;
  });
});
```


Kup ksi k Pole ksi k

Z powyższym podejściem związanych jest kilka problemów.

1. Kod wynikowy będzie koszmarnie powcinany, zwłaszcza jeśli zajdzie konieczność połączenia kilku wywołań.
2. Błędy zgłaszane między wywołaniami zwrotnymi i funkcjami mają tendencję do znikania, jeżeli nie zostaną ręcznie obsłużone na każdym etapie.
3. W wewnętrznym wywołaniu zwrotnym konieczna jest hermetyzacja logiki związanej z działaniami przeprowadzanymi za pomocą zmiennej `currentProfile` bezpośrednio lub za pomocą oddzielnej funkcji.

Obietnica rozwiązuje wymienione problemy. Zanim się przekonasz, w jaki sposób, najpierw spójrz na ten sam kod, ale zaimplementowany z użyciem obietnic:

```
var currentProfile = fetchServerConfig().then(function(serverConfig) {
  return fetchUserProfiles(serverConfig.USER_PROFILES, username);
}).then(function(profiles) {
  return profiles.currentProfile;
}, function(error) {
  // Obsługa błędów powstałych w fetchServerConfig()
  // lub w fetchUserProfiles().
});
```

Zwróć uwagę na zalety nowego rozwiązania.

1. Istnieje możliwość łączenia wywołań funkcji i nie spowoduje to koszmaru związanego ze stosowaniem wcięć w kodzie.
2. Masz gwarancję, że wywołanie poprzedniej funkcji zostanie zakończone, zanim nastąpi wywołanie kolejnej funkcji w łańcuchu.
3. Każde wywołanie `then()` pobiera dwa argumenty (oba to funkcje). Pierwszy to funkcja wywoływana w przypadku sukcesu operacji, natomiast drugi to procedura obsługi błędów.
4. W przypadku wystąpienia błędów w łańcuchu wspomniany błąd będzie propagowany przez pozostałe procedury obsługi błędów. Dlatego też błąd w dowolnym wywołaniu zwrotnym można obsłużyć na końcu.

Mógłbyś zapytać: co z wywołaniami `resolve()` i `reject()`? Wywołanie `deferred()` to we frameworku AngularJS sposób tworzenia obietnic. Z kolei wywołanie `resolve()` powoduje spełnienie obietnicy (i wywołanie procedury obsługi w przypadku sukcesu operacji), podczas gdy wywołanie `reject` powoduje wywołanie procedury obsługi błędów w obietnicy.

Kup ksi k Pole ksi k

Do tego zagadnienia powrócimy jeszcze podczas konfiguracji tras.

Dyrektywy

Przechodzimy teraz do dwóch dyrektyw, które będą używane w tworzonej tutaj aplikacji.

`butterbar`

Ta dyrektywa będzie pokazana lub ukryta w trakcie wczytywania informacji przez stronę, a także po zmianie trasy. Jest połączona z mechanizmem zmiany trasy i automatycznie zostaje ukryta lub umieszczona w znaczniku na podstawie stanu strony.

`focus`

Ta dyrektywa jest używana w celu zagwarantowania, że pewne pola tekstowe (lub elementy) formularza sieciowego są aktywne.

Spójrz na przykładowy fragment kodu:

```
// Plik: app/scripts/directives/directives.js.
var directives = angular.module('guthub.directives', []);

directives.directive('butterbar', ['$rootScope', function($rootScope) {
  return {
    link: function(scope, element, attrs) {
      element.addClass('hide');

      $rootScope.$on('$routeChangeStart', function() {
        element.removeClass('hide');
      });

      $rootScope.$on('$routeChangeSuccess', function() {
        element.addClass('hide');
      });
    }
  };
}]);

directives.directive('focus', function() {
  return {
    link: function(scope, element, attrs) {
      element[0].focus();
    }
  };
});
```

Przedstawiona dyrektywa zwraca obiekt wraz z pojedynczą właściwością `link`. Dokładne omówienie tematu tworzenia własnych dyrektyw znajdziesz w rozdziale 6., teraz musisz jedynie wiedzieć o dwóch rzeczach.

Kup ksi k Pole ksi k

1. Dyrektywy przechodzą przez proces składający się z dwóch etapów. Na pierwszym etapie (faza kompilacji) następuje wyszukanie wszystkich dyrektyw dołączonych do elementu drzewa DOM, a następnie ich przetworzenie. Wszelkie operacje na elementach drzewa DOM są przeprowadzane na etapie kompilacji. Na końcu fazy otrzymujesz funkcję łączącą.
2. Na drugim etapie (faza łączenia — tę fazę wcześniej wykorzystaliśmy) wygenerowany szablon elementów drzewa DOM jest dołączany do zasięgu (`scope`). Ponadto dodawane są wszelkie komponenty monitorujące lub nasłuchujące, co oznacza powstanie funkcjonującego wiązania między zasięgiem `scope` i elementem. Wszystko, co jest powiązane z zasięgiem `scope`, zachodzi na etapie łączenia.

Co się dzieje w naszej dyrektywie? Zajrzyjmy do niej i przekonajmy się.

Dyrektywa `butterbar` może być używana w następujący sposób:

```
<div butterbar>Komunikat informujący o wczytywaniu...</div>
```

Działanie dyrektywy polega na ukryciu elementu oraz dodaniu dwóch komponentów monitorujących zasięg główny (`scope`). Za każdym razem, gdy rozpoczyna się zmiana trasy, następuje pokazanie elementu (przez zmianę jego klasy), a po zakończonej powodzeniem zmianie trasy mamy ponowne ukrycie dyrektywy `butterbar`.

Interesującą cechą, na którą warto tutaj zwrócić uwagę, jest sposób wstrzyknięcia `$rootScope` do dyrektywy. Wszystkie dyrektywy mają bezpośrednie powiązanie z systemem wstrzykiwania zależności w AngularJS, co pozwala na wstrzykiwanie do nich usług oraz innych niezbędnych komponentów.

Ostatnia kwestia warta uwagi to API przeznaczone do pracy z elementem. Programiści przyzwyczajeni do biblioteki jQuery będą szczęśliwi, wiedząc, że zastosowanie ma doskonale znana im składnia (`addClass`, `removeClass`). Framework AngularJS implementuje pewien podzbiór wywołań jQuery, a więc biblioteka jQuery stanowi opcjonalną zależność dla każdego projektu AngularJS. Jeżeli w projekcie chcesz wykorzystać pełnię możliwości oferowanych przez jQuery, wtedy powinieneś wiedzieć, że AngularJS używa jej zamiast wbudowanej implementacji.

Druga dyrektywa (`focus`) jest znacznie prostsza. Jej działanie polega na wywołaniu metody `focus()` dla bieżącego elementu. Można ją wywołać przez dodanie atrybutu `focus` do dowolnego elementu danych wejściowych, na przykład:

```
<input type="text" focus></input>
```

Podczas wczytywania strony element automatycznie jest aktywny.

Kup ksi k Pole ksi k

Kontrolery

Po zaprezentowaniu dyrektyw i usług możesz wreszcie przejść do kontrolerów, których w naszej aplikacji mamy pięć. Wszystkie zostały zdefiniowane w pojedynczym pliku (*app/scripts/controllers/controllers.js*), ale omówimy je tutaj pojedynczo. Przechodzimy więc do pierwszego kontrolera (`ListCtrl`), odpowiedzialnego za wyświetlenie listy wszystkich przepisów kulinarnych przechowywanych w systemie.

```
app.controller('ListCtrl', ['$scope', 'recipes', function($scope, recipes) {
  $scope.recipes = recipes;
}]);
```

Zwróć uwagę na jedną bardzo ważną kwestię w przypadku omawianego kontrolera: w konstruktorze nie zawiera on żadnego kodu dotyczącego nawiązania połączenia z serwerem i pobrania przepisów kulinarnych. Zamiast tego kod zajmuje się obsługą wcześniej pobranych przepisów. Być może zastanawiasz się, jak to zostało zrobione. Cóż, dokładną odpowiedź poznasz w sekcji poświęconej routingowi, ale już teraz możemy powiedzieć, że wiąże się to z usługą `MultiRecipeLoader`. Po prostu o tym pamiętaj.

Po zapoznaniu się z kontrolerem `ListCtrl` zobaczysz, że pozostałe są całkiem podobne do omówionego. Mimo wszystko zaprezentujemy je po kolei, wskazując przy tym interesujące aspekty:

```
app.controller('ViewCtrl', ['$scope', '$location', 'recipe',
    function($scope, $location, recipe) {
  $scope.recipe = recipe;

  $scope.edit = function() {
    $location.path('/edit/' + recipe.id);
  };
}]);
```

Interesującym aspektem kontrolera `ViewCtrl` jest funkcja edycji udostępniana obiektowi `scope`. Zamiast pokazywać i ukrywać pola lub stosować podobne rozwiązanie, kontroler wykorzystuje framework AngularJS i zleca mu wykonanie najtrudniejszych zadań (powinieneś stosować takie samo podejście!). Funkcja `edit()` po prostu zmienia adres URL na odpowiednik przepisu kulinarnego, a AngularJS zajmuje się resztą. Ponadto framework wykrywa zmianę adresu URL i wczytuje odpowiedni widok (w trybie edycji będzie to po prostu dany przepis kulinarny). Wspaniale!

Przechodzimy teraz do kontrolera `EditCtrl`:

```
app.controller('EditCtrl', ['$scope', '$location', 'recipe',
    function($scope, $location, recipe) {
  $scope.recipe = recipe;
```


Kup książkę   Poleć książkę

```
  $scope.save = function() {
    $scope.recipe.$save(function(recipe) {
      $location.path('/view/' + recipe.id);
    });
  };

  $scope.remove = function() {
    delete $scope.recipe;
    $location.path('/');
  };
}]);
```

W tym kontrolerze nowością są metody `save()` i `remove()`, które `EditCtrl` udostępnia obiektowi `scope`.

Metoda `save()` obiektu `scope` działa zgodnie z oczekiwaniami. Zapisuje bieżący przepis kulinarny, a po zakończeniu operacji zapisu przekierowuje użytkownika do widoku wyświetlającego ten sam przepis. Funkcja wywołania zwrotnego jest użyteczna, ponieważ pozwala na przeprowadzenie pewnych operacji po zapisie.

Istnieją dwa sposoby zapisania przepisu. Jeden z nich został przedstawiony w kodzie i polega na wywołaniu funkcji `$scope.recipe.$save()`. Takie rozwiązanie jest możliwe tylko dlatego, że `recipe` jest obiektem zasobu zwróconego przez `RecipeLoader`.

Natomiast drugi sposób zapisu to wywołanie:

```
Recipe.save(recipe);
```

Metoda `remove()` również należy do prostych, a jej działanie polega na usunięciu przepisu z obiektu `scope` oraz przekierowaniu użytkownika na stronę główną. Zwróć uwagę, że nie powoduje to rzeczywistego usunięcia przepisu kulinarnego z serwera. Wykonanie dodatkowego wywołania nie powinno być zbyt trudne.

Kolejny kontroler nosi nazwę `NewCtrl`:

```
app.controller('NewCtrl', ['$scope', '$location', 'Recipe',
    function($scope, $location, Recipe) {
  $scope.recipe = new Recipe({
    ingredients: [ {} ]
  });

  $scope.save = function() {
    $scope.recipe.$save(function(recipe) {
      $location.path('/view/' + recipe.id);
    });
  };
}]);
```


Kup ksi k Pole ksi k

Ten kontroler jest niemal dokładnie taki sam jak `EditCtrl` (jako ćwiczenie mógłbyś oba wymienione kontrolery połączyć w jeden). Jedyna różnica polega na tym, że pierwszym krokiem w działaniu kontrolera `NewCtrl` jest utworzenie nowego przepisu kulinarnego (wspomniany przepis to zasób, a więc kontroler ma funkcję `save()`). Cała pozostała funkcjonalność nie ulega zmianie.

Ostatni kontroler to `IngredientsCtrl`. Jest to kontroler specjalny, ale zanim przejdziemy do jego omówienia, spójrz na tworzący go kod:

```
app.controller('IngredientsCtrl', ['$scope', function($scope) {
  $scope.addIngredient = function() {
    var ingredients = $scope.recipe.ingredients;
    ingredients[ingredients.length] = {};
  };
  $scope.removeIngredient = function(index) {
    $scope.recipe.ingredients.splice(index, 1);
  };
}]);
```

Wszystkie przedstawione dotąd kontrolery są połączone z określonymi widokami w interfejsie użytkownika. Pod tym względem kontroler `IngredientsCtrl` działa nieco inaczej. To po prostu kontroler potomny używany do edycji stron i hermetyzacji pewnych funkcji niepotrzebnych na ogólnym poziomie. Warto w tym miejscu wspomnieć o pewnej interesującej kwestii. Skoro to kontroler potomny, dziedziczy obiekt `scope` po kontrolerze nadrzędnym (w omawianym przykładzie jest to kontroler `EditCtrl` lub `NewCtrl`). Dlatego też uzyskanie dostępu do obiektu `$scope.recipe` odbywa się z poziomu kontrolera nadrzędnego.

Sam kod kontrolera nie zawiera nic szczególnie interesującego lub unikalnego. Dodaje kilka nowych składników do tablicy składników przepisu kulinarnego lub też usuwa określony składnik z listy.

W ten sposób omówiliśmy wszystkie kontrolery. Jedyny fragment kodu JavaScript, jaki pozostał do przeanalizowania, dotyczy konfiguracji routingu:

```
// Plik: app/scripts/controllers/controllers.js.
var app = angular.module('guthub',
  ['guthub.directives', 'guthub.services']);

app.config(['$routeProvider', function($routeProvider) {
  $routeProvider.
    when('/', {
      controller: 'ListCtrl',
      resolve: {
        recipes: function(MultiRecipeLoader) {
          return MultiRecipeLoader();
        }
      },
```


Kup książkę Poleć książkę

```
        templateUrl:'/views/list.html'
      }).when('/edit/:recipeId', {
        controller: 'EditCtrl',
        resolve: {
          recipe: function(RecipeLoader) {
            return RecipeLoader();
          }
        },
        templateUrl:'/views/recipeForm.html'
      }).when('/view/:recipeId', {
        controller: 'ViewCtrl',
        resolve: {
          recipe: function(RecipeLoader) {
            return RecipeLoader();
          }
        },
        templateUrl:'/views/viewRecipe.html'
      }).when('/new', {
        controller: 'NewCtrl',
        templateUrl:'/views/recipeForm.html'
      }).otherwise({redirectTo:'/'});
}]);
```

Zgodnie z wcześniejszą obietnicą docieramy do miejsca, w którym używana jest funkcja `resolve()`. W poprzednim fragmencie kodu skonfigurowano moduł `guthub` AngularJS, a także trasy i szablony wykorzystywane w aplikacji.

Kod łączy dyrektywy z utworzonymi przez nas usługami, a następnie wskazuje różne trasy, które będą stosowane w aplikacji.

Dla każdej trasy definiowany jest adres URL, kontroler odpowiedzialny za obsługę danego adresu, wczytywany szablon, a także (wreszcie) obiekt `resolve`.

Obiekt `resolve` nakazuje frameworkowi AngularJS spełnienie wymagań każdego klucza, zanim trasa będzie mogła zostać użyta do wyświetlenia odpowiedniego widoku użytkownikowi. Zadanie aplikacji polega na wczytaniu wszystkich przepisów kulinarnych (lub tylko wskazanego), a serwer ma udzielić odpowiedzi przed wyświetleniem strony użytkownikowi. Dostawcę tras informujemy więc o posiadaniu przepisów kulinarnych (lub przepisu), a następnie podajemy mu sposób, w jaki mają być pobrane dane.

W trakcie wykonywania operacji pobierania danych wykorzystywane są dwie usługi (`MultiRecipeLoader` i `RecipeLoader`) zdefiniowane na początku tworzenia aplikacji. Framework AngularJS został dość sprytnie zaprojektowany — jeżeli wartością zwrotną funkcji `resolve()` będzie obietnica AngularJS, wtedy framework poczeka na spełnienie wspomnianej obietnicy przed przejściem dalej. Oznacza to konieczność zaczekania, aż serwer udzieli odpowiedzi.

Kup ksi k Pole ksi k

Wynik jest w postaci argumentów (o nazwach parametrów będących polami obiektu) przekazywany konstruktorowi.

Na końcu funkcja `otherwise()` wskazuje domyślny adres URL dla przekierowania, jeśli nie nastąpi dopasowanie żadnej trasy.

Być może zauważyłeś, że kontrolery `EditCtrl` i `NewCtrl` korzystają z tego samego szablonu, czyli *views/recipeForm.html*. Co się tutaj dzieje? Po prostu ponownie wykorzystaliśmy szablon przeznaczony do edycji przepisu kulinarnego. Szablon wyświetla różne elementy w zależności od wywołanego kontrolera.

Po zakończeniu omawiania kontrolerów możemy przejść do szablonów. Zobaczysz, w jaki sposób wymienione kontrolery zostały powiązane z szablonami, a także dowiesz się, jak zarządzać danymi, które są wyświetlane użytkownikowi.

Szablony

Rozpoczynamy od zapoznania się z najbardziej zewnętrznym, głównym szablonem zdefiniowanym w pliku *index.html*. Stanowi on podstawę dla naszej aplikacji składającej się z pojedynczej strony, a wszystkie pozostałe widoki są wczytywane w kontekście omawianego tutaj szablonu:

```
<!DOCTYPE html>
<html lang="pl" ng-app="guthub">
<head>
  <title>GutHub - tworzenie przepisów kulinarnych i dzielenie się nimi</title>
  <script src="scripts/vendor/angular.min.js"></script>
  <script src="scripts/vendor/angular-resource.min.js"></script>
  <script src="scripts/directives/directives.js"></script>
  <script src="scripts/services/services.js"></script>
  <script src="scripts/controllers/controllers.js"></script>
  <link href="styles/bootstrap.css" rel="stylesheet">
  <link href="styles/guthub.css" rel="stylesheet">
</head>
<body>
  <header>
    <h1>GutHub</h1>
  </header>

  <div butterbar>Wczytywanie...</div>

  <div class="container-fluid">
    <div class="row-fluid">
      <div class="span2">
        <!-- Pasek boczny. -->
        <div id="focus"><a href="#/new">Nowy przepis</a></div>
```

```
      <div><a href="#/">Lista przepisów</a></div>
    </div>
    <div class="span10">
      <div ng-view></div>
    </div>
  </div>
</div>
</body>
</html>
```

W przedstawionym szablonie istnieje pięć elementów, na które warto zwrócić uwagę. Większość z nich miałeś okazję poznać w rozdziale 2. Wspomniane elementy omówimy po kolei.

`ng-app`

Ustawienie modułu dla aplikacji GutHub. Jest to dokładnie ten sam moduł, który wykorzystaliśmy we funkcji `angular.module()`. W ten sposób framework AngularJS wie, jak wszystko ma zostać ze sobą połączone.

`script` *znacznik*

W tym miejscu następuje wczytanie AngularJS w aplikacji. Framework trzeba wczytać przed wszystkimi plikami JavaScript, które go używają. W idealnej sytuacji znaczniki odpowiedzialne za wczytywanie skryptów JavaScript powinny znajdować się na końcu pliku szablonu.

`butterbar`

Aha! To pierwsze użycie naszej własnej dyrektywy. Ta dyrektywa została zdefiniowana wcześniej i chcemy ją wykorzystać wraz z elementem wyświetlanym podczas zmiany trasy. Po zakończeniu powodzeniem operacji zmiany trasy element powiązany z dyrektywą `butterbar` powinien zostać ukryty. Dyrektywa powoduje wyświetlenie tekstu (w omawianym przypadku jest to nudny komunikat `Wczytywanie...`), gdy zachodzi potrzeba.

Łącza `href` *wartości*

To łącza `href` do różnych stron naszej aplikacji składającej się z pojedynczej strony. Zwróć uwagę na użycie znaku `#` gwarantującego, że strona nie zostanie ponownie wczytana. Adresy są podawane względem strony bieżącej. Framework AngularJS monitoruje wspomniane adresy URL (dopóki strona nie zostanie ponownie wczytana) i wykonuje całą pracę związaną z ich obsługą (w rzeczywistości jest to bardzo nudne zarządzanie trasami zdefiniowane przez nas wcześniej wraz z trasami), gdy zachodzi potrzeba.

Kup ksi k Pole ksi k

ng-view

W tym miejscu wykonywana jest pozostała część pracy. Wcześniej we fragmencie rozdziału poświęconym kontrolerom zdefiniowaliśmy trasy. Częścią definicji jest adres URL trasy, powiązany z nią kontroler i szablon. Kiedy framework AngularJS wykryje zmianę trasy, wtedy następuje wczytanie szablonu, dołączenie do niego kontrolera oraz zastąpienie elementu ng-view zawartością szablonu.

Jedyną rzeczą rzucającą się w oczy jest brak znacznika ng-controller. Większość aplikacji zawiera pewnego rodzaju kontroler MainController powiązany z szablonem głównym. Najczęstszym miejscem jego podania jest znacznik <body>. W omawianej aplikacji nie używamy wspomnianego znacznika, ponieważ cały szablon główny nie zawiera treści AngularJS wymagającej odwołania do obiektu scope.

Spójrzmy teraz na szablony powiązane z poszczególnymi kontrolerami. Na początek przyglądamy się szablonowi wyświetlającemu listę przepisów kulinarnych:

```
<!-- Plik: chapter4/guthub/app/views/list.html. -->
<h3>Lista przepisów</h3>
<ul class="recipes">
  <li ng-repeat="recipe in recipes">
    <div><a ng-href="#/view/{{recipe.id}}">{{recipe.title}}</a></div>
  </li>
</ul>
```

To naprawdę bardzo nudny szablon. Znajdują się tutaj jedynie dwa interesujące punkty. Pierwszy to standardowy sposób użycia znacznika ng-repeat. Zadanie wymienionego znacznika polega na pobraniu wszystkich przepisów z obiektu scope, a następnie ich wyświetleniu.

Drugi interesujący punkt to użycie znacznika ng-href zamiast href. Ma to na celu uniknięcie wygenerowania nieprawidłowego łącza podczas wczytywania frameworka AngularJS. Znacznik ng-href gwarantuje, że w żadnej chwili użytkownikowi nie zostanie wyświetlony nieprawidłowy znacznik. Wymienionego znacznika powinieneś używać zawsze, gdy adresy URL są dynamiczne, a nie statyczne.

Być może zadajesz sobie pytanie: gdzie podział się kontroler? Nie mamy zdefiniowanego znacznika ng-controller i tak naprawdę nie ma zdefiniowanego kontrolera głównego. W tym miejscu do gry wchodzi mapowanie tras. Może pamiętasz (mówiliśmy o tym kilka stron wcześniej), że trasa / powoduje przekierowanie do wyświetlającego listę przepisów kulinarnych szablonu, któremu przypisano kontroler ListCtrl. Dlatego też wszelkie odniesienia do zmiennych pozostają w zasięgu wymienionego kontrolera.

Kup ksi k Pole ksi k

Teraz przechodzimy do znacznie ciekawszego szablonu, czyli odpowiedzialnego za wyświetlenie przepisu.

```
<!-- Plik: chapter4/guthub/app/views/viewRecipe.html. -->
<h2>{{recipe.title}}</h2>

<div>{{recipe.description}}</div>

<h3>Składniki</h3>

<span ng-show="recipe.ingredients.length == 0">Brak składników</span>
<ul class="unstyled" ng-hide="recipe.ingredients.length == 0">
  <li ng-repeat="ingredient in recipe.ingredients">
    <span>{{ingredient.amount}}</span>
    <span>{{ingredient.amountUnits}}</span>
    <span>{{ingredient.ingredientName}}</span>
  </li>
</ul>

<h3>Sposób przygotowania</h3>
<div>{{recipe.instructions}}</div>

<form ng-submit="edit()" class="form-horizontal">
  <div class="form-actions">
    <button class="btn btn-primary">Edycja</button>
  </div>
</form>
```

To kolejny mały, przydatny szablon. Warto zwrócić uwagę na dwa punkty powyższego szablonu, choć niekoniecznie w kolejności ich wymienienia.

Pierwszy to całkiem standardowy sposób użycia dyrektywy `ng-repeat`. Przepisy kulinarne znajdują się w zasięgu kontrolera `ViewCtrl` wczytanego przez funkcję `resolve()` przed wyświetleniem strony użytkownikowi. Dzięki temu gwarantujemy prawidłowe działanie strony, gdy zostaje wyświetlona.

Drugi punkt to użycie dyrektywy `ng-submit` w formularzu. Wymieniona dyrektywa oznacza, że wysłanie formularza spowoduje wywołanie funkcji `edit()` obiektu `scope`. Wysłanie formularza następuje, gdy kliknięty będzie przycisk niepowiązany z żadną funkcją (w omawianym przypadku to przycisk *Edycja*). I znów działanie frameworka AngularJS zostało zaprojektowane bardzo sprytnie — potrafi on prawidłowo ustalić zasięg, do którego ma się odwoływać (na przykład: modułu, trasy lub kontrolera), i wywołać odpowiednią metodę we właściwym czasie.

Teraz możemy przejść do ostatniego (i prawdopodobnie najbardziej skomplikowanego) szablonu, czyli formularza pozwalającego na dodanie lub edycję przepisu kulinarnego.

Kup ksi k Pole ksi k

```
<!-- Plik: chapter4/guthub/app/views/recipeForm.html. -->
<h2>Edycja przepisu</h2>
<form name="recipeForm" ng-submit="save()" class="form-horizontal">
  <div class="control-group">
    <label class="control-label" for="title">Nazwa:</label>
    <div class="controls">
      <input ng-model="recipe.title" class="input-xlarge" id="title" focus required>
    </div>
  </div>

  <div class="control-group">
    <label class="control-label" for="description">Opis:</label>
    <div class="controls">
      <textarea ng-model="recipe.description" class="input-xlarge"
        id="description"></textarea>
    </div>
  </div>

  <div class="control-group">
    <label class="control-label" for="ingredients">Składniki:</label>
    <div class="controls">
      <ul id="ingredients" class="unstyled" ng-controller="IngredientsCtrl">
        <li ng-repeat="ingredient in recipe.ingredients">
          <input ng-model="ingredient.amount" class="input-mini">
          <input ng-model="ingredient.amountUnits" class="input-small">
          <input ng-model="ingredient.ingredientName">
          <button type="button" class="btn btn-mini"
            ng-click="removeIngredient($index)"><i class="icon-minus-sign"></i>
          Usuń</button>
        </li>
        <button type="button" class="btn btn-mini" ng-click="addIngredient()">
        <i class="icon-plus-sign"></i>Dodaj</button>
      </ul>
    </div>
  </div>

  <div class="control-group">
    <label class="control-label" for="instructions">Sposób
przygotowania:</label>
    <div class="controls">
      <textarea ng-model="recipe.instructions" class="input-xxlarge"
      id="instructions"></textarea>
    </div>
  </div>

  <div class="form-actions">
    <button class="btn btn-primary" ng-disabled="recipeForm.$invalid">Zapisz
    </button>
    <button type="button" ng-click="remove()" ng-show="!recipe.id" class="btn">
    Usuń</button>
  </div>
</form>
```


Kup książkę Poleć książkę

Nie panikuj! Wygląda na to, że szablon zawiera całkiem sporą ilość kodu, i faktycznie tak jest. Jednak po rzeczywistym zagłębieniu się weń można się przekonać, że kod nie jest skomplikowany. Tak naprawdę to prosta, powtarzająca się struktura, pokazująca, jak edytowalne pola tekstowe zostały zastosowane w formularzu przeznaczonym do edycji przepisów kulinarnych.

- W pierwszym polu tekstowym (`title`) została umieszczona dyrektywa `focus`. Dzięki temu po przejściu na tę stronę wskazane pole zostanie wybrane, a użytkownik będzie mógł natychmiast rozpocząć wprowadzanie danych wejściowych.
- Dyrektywa `ng-submit` jest użyta w bardzo podobny sposób jak w poprzednim przykładzie, a więc nie będziemy jej tutaj dokładnie omawiać. Warto wiedzieć, że powoduje zapisanie stanu przepisu kulinarnego i wskazuje koniec procesu edycji. Ponadto jest powiązana z funkcją `save()` zdefiniowaną w kontrolerze `EditCtrl`.
- Dyrektywa `ng-model` służy do połączenia różnych pól tekstowych formularza sieciowego z polami modelu.
- Jednym z najbardziej interesujących aspektów omawianej strony jest umieszczona w części poświęconej liście składników dyrektywa `ng-controller`, której naprawdę warto poświęcić nieco uwagi i spróbować w pełni zrozumieć sposób jej działania. Zobaczmy więc, co się tutaj dzieje.

  Lista składników jest wyświetlana, a zawierający ją znacznik jest powiązany z dyrektywą `ng-controller`. Oznacza to, że cały znacznik `<ul>` znajduje się w zasięgu kontrolera `IngredientsCtrl`. Mógłbyś w tym miejscu zapytać: co z rzeczywistym kontrolerem `EditCtrl` powiązanym z szablonem? Jak się okazuje, `IngredientsCtrl` jest tworzony jako kontroler potomny `EditCtrl` i tym samym dziedziczy po nim. Dlatego też dostęp do obiektu `recipe` następuje z poziomu kontrolera `EditCtrl`.

  Ponadto kontroler `IngredientsCtrl` dodaje metodę `addIngredient()` używaną przez dyrektywę `ng-click` i dostępną jedynie w zasięgu znacznika `<ul>`. Dlaczego zdecydowaliśmy się na takie rozwiązanie? To najlepszy sposób na rozdzielenie obowiązków. Po co umieszczać metodę `addIngredient()` w kontrolerze `EditCtrl`, skoro 99% szablonu jej nie potrzebuje? Kontrolery potomne i zagnieżdżone doskonale sprawdzają się w tego rodzaju sytuacjach i pozwalają na oddzielenie logiki biznesowej przez umieszczenie jej w łatwiejszych do zarządzania elementach.

Kup ksi k Pole ksi k

- Pozostałe dyrektywy, które chcemy tutaj omówić, są kontrolkami przeznaczonymi do weryfikacji formularza sieciowego. We frameworku AngularJS można bardzo łatwo określić, że dane pole formularza jest wymagane. W tym celu wystarczy dodać do tego pola dyrektywę `required` (jak to zrobiono w omawianym fragmencie kodu). Rodzi się jednak pytanie: co dalej?

  Przechodzimy do przycisku *Zapisz*. Zwróć uwagę na użycie dyrektywy `ng-disabled`, która ma wartość `recipeForm.$invalid`. Człon pierwszy (`recipeForm`) to nazwa formularza zawierającego deklarację dyrektywy. Framework AngularJS dodaje do niego pewne zmienne specjalne (zaliczamy do nich `$valid` i `$invalid`) pozwalające na kontrolowanie elementów formularza sieciowego. AngularJS wyszukuje wszystkie wymagane elementy, a następnie odpowiednio uaktualnia wspomniane zmienne specjalne. Jeżeli pole służące do podania nazwy przepisu kulinarnego pozostanie niewypełnione, wartością `recipeForm.$invalid` będzie `true` (a wartością `$valid` będzie `false`) i przycisk *Zapisz* zostanie zablokowany.

Istnieje również możliwość określenia minimalnej i maksymalnej długości pola tekstowego, a także wzorzec wyrażenia regularnego przeznaczonego do przeprowadzenia weryfikacji danego pola. Co więcej, pewne funkcje zaawansowane można wykorzystać do wyświetlania komunikatów błędów po wystąpieniu pewnych określonych warunków. Spójrzmy na prosty przykład:

```
<form name="myForm">
  Nazwa użytkownika:<input type="text"
    name="userName"
    ng-model="user.name"
    ng-minlength="3">
  <span class="error"
    ng-show="myForm.userName.$error.minlength">Zbyt krótka!</span>
</form>
```

Za pomocą użycia dyrektywy `ng-minlength` w powyższym fragmencie kodu zdefiniowano, że nazwa użytkownika musi składać się z przynajmniej trzech znaków. Teraz formularz zostaje wypełniony danymi pochodzącymi z obiektu `scope` — w omawianym przykładzie to jedynie `userName`. Wszystkie pola tekstowe mają obiekt `$error` (zawiera informacje o rodzaju ewentualnego błędu: `required`, `minlength`, `maxlength` lub `pattern`) oraz właściwość `$valid` wskazującą poprawność bądź też niepoprawność danych wejściowych.

Kup ksi k Pole ksi k

Takie rozwiązanie pozwala na selektywne wyświetlanie użytkownikowi komunikatu błędu w zależności od jego rodzaju, jak to pokazano w powyższym fragmencie kodu.

Do drugiego przycisku dołączona jest dyrektywa `ng-click` używana podczas usuwania przepisu kulinarnego. Zwróć uwagę, że przycisk jest wyświetlany tylko wtedy, gdy przepis nie został jeszcze zapisany. Wprawdzie znacznie sensowniejsze wydaje się użycie `ng-hide="recipe.id"`, ale czasami bardziej semantyczne rozwiązanie to `ng-show="!recipe.id"`. Oznacza to wyświetlenie przycisku, gdy przepis kulinarny nie zawiera identyfikatora, zamiast ukrywania przycisku, jeśli przepis ma zdefiniowany identyfikator.

Testy

Wstrzymywaliśmy się z przedstawieniem testów wraz z kontrolerami, ale musiałeś się spodziewać, że kiedyś wreszcie do nich przejdziemy. W tym podrozdziale zaprezentowane zostaną testy, które należy utworzyć dla przygotowanego dotąd fragmentu kodu. Dowiesz się również, jak tworzy się takie testy.

Testy jednostkowe

Najważniejszy rodzaj testów to testy jednostkowe. Pozwalają one na sprawdzenie, czy opracowane kontrolery (dyrektywy i usługi) mają prawidłową strukturę i konstrukcję oraz czy działają zgodnie z oczekiwaniami.

Zanim przejdziemy do poszczególnych testów jednostkowych, warto spojrzeć na szkielet przeznaczony dla wszystkich testów jednostkowych dotyczących kontrolera:

```
describe('Kontrolery', function() {
  var $scope, ctrl;
  // W teście należy wskazać moduł.
  beforeEach(module('guthub'));
  beforeEach(function() {
    this.addMatchers({
      toEqualData: function(expected) {
        return angular.equals(this.actual, expected);
      }
    });
  });

  describe('ListCtrl', function() {....});

  // Miejsce na opisanie pozostałych kontrolerów.

});
```


Kup ksi k Pole ksi k

Przygotowany szkielet (tutaj nadal wykorzystujemy styl Jasmine do tworzenia testów) wykonuje kilka zadań.

1. Tworzy globalnie (przynajmniej dla testu) dostępny obiekt `scope` i kontroler, a więc nie trzeba się przejmować tworzeniem nowej zmiennej dla każdego kontrolera.
2. Inicjalizuje moduł używany przez aplikację (w omawianym przykładzie jest to GutHub).
3. Dodaje specjalne dopasowanie nazywane `equalData`. Pozwala ono na przeprowadzanie asercji na obiektach źródła (na przykład przepisach kulinarnych) zwracanych przez usługę `$resource` lub na wywołanie RESTful.

Pamiętaj o konieczności dodania specjalnego dopasowania nazywanego `equalData` za każdym razem, gdy zachodzi potrzeba stosowania asercji na zwróconych obiektach `ngResource`. Wiąże się to z faktem, że zwrócone obiekty `ngResource` mają metody dodatkowe, których zwykłe wykonanie zakończy się niepowodzeniem, ponieważ oczekiwane są wywołania `equalData`.

Mając przygotowany szkielet, spójrzmy na gotowy test jednostkowy przeznaczony dla kontrolera `ListCtrl`:

```
describe('ListCtrl', function() {
  var mockBackend, recipe;
  // _$httpBackend_ to nazwa taka sama jak $httpBackend. Zastosowany zapis służy do odróżnienia
  // zmiennych wstrzykniętych od zmiennych lokalnych.
  beforeEach(inject(function($rootScope, $controller, _$httpBackend_, Recipe) {
    recipe = Recipe;
    mockBackend = _$httpBackend_;
    $scope = $rootScope.$new();
    ctrl = $controller('ListCtrl', {
      $scope: $scope,
      recipes: [1, 2, 3]
    });
  }));

  it('Wynikiem powinna być lista przepisów kulinarnych', function() {
    expect($scope.recipes).toEqual([1, 2, 3]);
  });
});
```

Jak zapewne pamiętasz, kontroler `ListCtrl` należy do najprostszych w aplikacji. Konstruktor kontrolera pobiera po prostu listę przepisów, a następnie zapisuje je w obiekcie. Wprawdzie można do tego utworzyć test, ale wydaje się to zbędne. W omawianym przykładzie mimo wszystko utworzyliśmy test, ponieważ testy jednostkowe są wspaniałe!

Kup ksi k   Pole ksi k

Znacznie ciekawiej robi się w przypadku usługi `MultiRecipeLoader`. Wymieniona usługa jest odpowiedzialna za pobranie listy przepisów kulinarnych z serwera i przekazanie ich jako argumentu (kiedy zastosowana jest prawidłowa konfiguracja za pomocą usługi `$route`):

```
describe('MultiRecipeLoader', function() {
  var mockBackend, recipe, loader;
  // _$httpBackend_ to nazwa taka sama jak $httpBackend. Zastosowany zapis służy do odróżnienia
  // zmiennych wstrzykniętych od zmiennych lokalnych.
  beforeEach(inject(function(_$httpBackend_, Recipe, MultiRecipeLoader) {
    recipe = Recipe;
    mockBackend = _$httpBackend_;
    loader = MultiRecipeLoader;
  }));

  it('Wynikiem powinno być wczytanie listy przepisów kulinarnych', function() {
    mockBackend.expectGET('/recipes').respond([{id: 1}, {id: 2}]);

    var recipes;

    var promise = loader();
    promise.then(function(rec) {
      recipes = rec;
    });

    expect(recipes).toBeUndefined();
    mockBackend.flush();
    expect(recipes).toEqualData([{id: 1}, {id: 2}]);
  });
});
// Miejsce na opisanie pozostałych kontrolerów.
```

Test usługi `MultiRecipeLoader` odbywa się przez przygotowanie usługi `Http`
`↪Backend` w naszym teście. Obiekt pochodzi z pliku *angular-mocks.js* i jest dołączany w trakcie przeprowadzania testów. Po prostu wstrzyknięcie go do metody `beforeEach()` jest wystarczające, aby można było konfigurować oczekiwania. W drugim, znacznie ciekawszym teście oczekiwanie zostało zdefiniowane jako wywołanie `server GET` do *recipes*, a wynikiem powinna być tablica obiektów. Następnie używamy dopasowania w celu sprawdzenia, czy uzyskany wynik jest dokładnie zgodny z oczekiwaniami. Zwróć uwagę na wywołanie `flush()` w obiekcie makiety, przekazujące odpowiedź pochodzącą z serwera. Tego rodzaju mechanizm można wykorzystać do przetestowania przepływu kontroli i sprawdzenia, jak aplikacja działa przed otrzymaniem odpowiedzi z serwera i po jej otrzymaniu.

Pomijamy kontroler `ViewCtrl`, ponieważ jest niemal identyczny z `ListCtrl`, poza dodatkiem w postaci metody `edit()`. Wymieniona metoda jest bardzo łatwa do przetestowania: wystarczy wstrzyknąć usługę `$location` do testu i sprawdzić jej wartość.

Kup ksi k Pole ksi k

Przechodzimy teraz do kontrolera `EditCtrl`, który z perspektywy testów jednostkowych ma dwa interesujące punkty. Funkcja `resolve()` jest podobna do używanej już poprzednio i może być przetestowana w dokładnie ten sam sposób jak wcześniej w rozdziale. Zamiast tego zobaczysz teraz, jak można przetestować metody `save()` i `remove()`. Spójrzmy więc na wymienione testy (przyjęto założenie o użyciu szkieletu przedstawionego w poprzednim przykładzie):

```
describe('EditCtrl', function() {
  var mockBackend, location;
  beforeEach(inject(function($rootScope,
      $controller,
      _$httpBackend_,
      $location,
      Recipe) {
    mockBackend = _$httpBackend_;
    location = $location;
    $scope = $rootScope.$new();

    ctrl = $controller('EditCtrl', {
      $scope: $scope,
      $location: $location,
      recipe: new Recipe({id: 1, title: 'Przepis'})
    });
  }));

  it('Wynikiem powinien być zapisany przepis kulinarny', function() {
    mockBackend.expectPOST('/recipes/1',
      {id: 1, title: 'Przepis'}).respond({id: 2});

    // Ustawienie innej wartości, aby mieć gwarancję jej zmiany podczas testu.
    location.path('test');

    $scope.save();
    expect(location.path()).toEqual('/test');

    mockBackend.flush();

    expect(location.path()).toEqual('/view/2');
  });

  it('Wynikiem powinno być usunięcie przepisu kulinarnego', function() {
    expect($scope.recipe).toBeTruthy();
    location.path('test');

    $scope.remove();

    expect($scope.recipe).toBeUndefined();
    expect(location.path()).toEqual('/');
  });
});
```


Kup ksi k Pole ksi k

W pierwszym teście sprawdzane jest działanie funkcji `save()`. W szczególności upewniamy się, że operacja zapisu powoduje wykonanie do serwera żądania `POST` wraz z obiektem. Następnie, po udzieleniu odpowiedzi przez serwer, przechodzimy na stronę zawierającą nowo zapisany przepis kulinarny.

Drugi test jest jeszcze prostszy. Po prostu sprawdzamy, czy wywołanie funkcji `remove()` powoduje usunięcie wskazanego przepisu kulinarnego, a następnie przekierowujemy użytkownika na stronę docelową. Jest to łatwe do wykonania dzięki wstrzyknięciu usługi `$location` do testu i jej użyciu.

Pozostała część testów jednostkowych dla kontrolerów wykorzystuje te same wzorce, a więc można je tutaj pominąć. Ogólnie rzecz ujmując, testy jednostkowe opierają się na kilku aspektach:

- zagwarantowanie, że kontroler (lub bardziej prawdopodobnie obiekt `scope`) osiągnie prawidłowy stan na końcu procesu inicjalizacji;
- potwierdzenie wykonania prawidłowych wywołań serwera i osiągnięcie właściwego stanu przez obiekt `scope` w trakcie wspomnianych wywołań serwera oraz po ich zakończeniu (do tego celu w testach jednostkowych używany jest obiekt makiety);
- wykorzystanie funkcji wstrzykiwania zależności we frameworku AngularJS, aby uzyskać uchwyt do elementów i obiektów używanych przez kontroler. Pozwala to upewnić się, że kontroler ustawia prawidłowy stan.

Testy scenariuszy

Gdy testy jednostkowe zakończą się powodzeniem, może pojawić się pokusa zakończenia pracy. Jednak praca programisty AngularJS nie kończy się, zanim nie będą przeprowadzone testy scenariuszy. Wprawdzie testy jednostkowe dają gwarancję, że każdy najmniejszy fragment kodu JavaScript działa zgodnie z oczekiwaniami, ale jednocześnie warto się upewnić o wczytaniu szablonów, prawidłowym powiązaniu kontrolerów i poprawnej reakcji na kliknięcia elementów szablonu.

Dokładnie do tego celu służą testy scenariuszy we frameworku AngularJS. Pozwalają one na:

- wczytanie aplikacji;
- przejście na konkretną stronę;
- kliknięcie i wprowadzenie tekstu;
- upewnienie się o prawidłowej reakcji aplikacji.

Kup ksi k Pole ksi k

Na jakiej zasadzie działa test scenariusza dla strony wyświetlającej listę przepisów kulinarnych? Przede wszystkim przed rozpoczęciem rzeczywistego testu trzeba poczynić pewne przygotowania.

Aby ten test scenariusza działał, konieczne jest przygotowanie serwera WWW, który będzie miał możliwość akceptacji żądań wykonywanych przez aplikację GutHub, a także pobierania listy przepisów kulinarnych z testowanej aplikacji oraz ich przechowywania. Możesz zmodyfikować kod i wykorzystać zapisywaną w pamięci listę przepisów — wymaga to usunięcia `$resource` dla przepisu i zmiany usługi na obiekt zawierający dane w formacie JSON. Ewentualnie można ponownie wykorzystać i zmodyfikować serwer WWW przedstawiony w poprzednim rozdziale bądź też użyć narzędzia Yeoman!

Po przygotowaniu i uruchomieniu serwera WWW obsługującego aplikację wystarczy utworzyć i wykonać przedstawiony poniżej test:

```
describe('GutHub App', function() {
  it('Wynikiem powinna być lista przepisów kulinarnych', function() {
    browser().navigateTo('/index.html');
    // Domyślna lista przepisów kulinarnych w aplikacji GutHub składa się jedynie z dwóch pozycji.
    expect(repeater('.recipes li').count()).toEqual(2);
  });
});
```


Kup ksi k Pole ksi k

Kup ksi k Pole ksi k

Skorowidz

A

analiza aplikacji, 101
anatomia aplikacji, 23
API, 150
API HTML5, 174
API jQuery, 165
aplikacje
  AJAX, 57
  mobilne, 61
  sieciowe, 11
architektura MVC, 14, 24
arkusze stylów CSS, 39
asynchroniczne wywołania metod, 130
atak typu XSRF, 147
atrybut
  href, 42
  multiple, 203
  ng-app, 19
  ng-change, 30
  ng-controller, 44
  ng-model, 20, 45, 193
  ng-repeat, 19, 37
  required, 66
  src, 42

B

bezpieczeństwo, 146
biblioteka
  jQuery, 110
  NodeJS, 76
  RequireJS, 92
  Socket.IO, 76, 129, 204
blok
  Config, 179
  Run, 179
błędy, 173
buforowanie odpowiedzi, 134

C

ciasteczka, 184

D

dane wejściowe, 65
debugowanie, 85
definiowanie kontrolerów, 111
deklaracja
  dyrektywy, 201
  kontrolera, 194
  zasobu, 140
dodawanie trasy, 90
dołączanie danych, 14, 20, 27
DOM, Document Object Model, 14, 63, 164
dostęp do
  konsoli, 87
  zasięgu, 160
dwukropek, 140
dyrektywa, 17, 149
  butterbar, 109, 116, 211
  errorMessage, 212
  expander, 167
  focus, 65, 95, 109
  ng-app, 24
  ng-bind, 29
  ng-bind-html, 189
  ng-bind-html-unsafe, 189
  ngbkFocus, 65
  ng-class, 40
  ng-click, 65, 122
  ng-controller, 120
  ng-disabled, 67
  ng-hide, 38
  ng-minlength, 121
  ng-model, 120
  ngPluralize, 186
  ng-repeat, 37, 41, 118, 196, 200

Kup ksi k

Pole ksi k

dyrektywa
ng-style., 40
ng-submit, 32, 118, 120
ng-view, 58
dyrektywy
dostęp do zasięgu, 160
funkcja compile(), 157
funkcja link(), 157
nazwa, 152
obsługa zdarzeń, 32
opcja priority, 153
opcja templates, 154
opcja transclude, 157
właściwość restrict, 152
dyskretny kod JavaScript, 33
działanie
dyrektywy, 110
filtru, 57
funkcji save(), 126
opcji dyrektywy, 164
testu, 99
usługi $location, 174

E

edycja przepisu, 118
element
<input>, 66
ng-app, 116
ng-href, 117
ng-repeat, 117
ng-show, 38
ng-view, 117
elementy
drzewa DOM, 36, 164
powtarzalne, 36
szablonu, 116
tablicy, 42

F

faza
kompilacji, 158
łączenia, 158
filtr, 55
filterService, 200
linky, 189, 190
filtrowanie, 196
daty i godziny, 188
listy, 80
formatowanie danych, 55
formularz, 29
formularz rejestracyjny, 65
framework
BDD, 79
Express, 75
funkcja
$http.get(), 130
$resource, 140
$scope. safeApply(), 173
$watch(), 30, 45–50
addExpander(), 168
callback, 208
callMe(), 50
compile(), 158
computeNeeded(), 31
controller(), 165
directive(), 63
done, 203
edit(), 118
equals(), 143
factory(), 53, 181
focus(), 64, 110
inheritedData(), 165
injector(), 165
link(), 64, 158
otherwise(), 58, 115
provider(), 53, 181
remove(), 21, 112
resolve(), 114, 118, 125
run(), 156
save(), 112
scope(), 165
select(), 194
selectRow(), 41
service(), 53, 181
StartUpController(), 31
stun(), 39
then(), 107, 143
totalCart(), 47
funkcje
jQuery, 165
typu getter, 175
typu setter, 175
usługi $location, 174

Kup ksi k Pole ksi k

G

granice aplikacji, 24
grupowanie zależności, 51

H

harmonijka, 168
hermetyzacja, 142

I

IDE, 73
identyfikator lokalizacji, 186
informacje o lokalizacji, 173
instalacja
  Karma, 96
  Yeoman, 89
integracja z IDE, 79
interceptor, 212
interfejs
  document.cookie, 184
  użytkownika, 19
internacjonalizacja aplikacji, 185, 188

K

karta
  kredytowa, 139
  Model, 86
  Performance, 86
katalog
  app, 93
  config, 93
  test, 93
klasa HelloController, 13
kod
  lokalizacji, 186
  serwera, 75
  usługi, 105
kompilacja, 82
kompilator Closure Compiler, 83
komponent nasłuchujący, 183
komunikacja
  między kontrolerami, 196
  między zasięgami, 182
  z serwerami, 61, 129
  z usługami, 211
konfiguracja
  modułu, 180
  routingu, 113
  środowiska programistycznego, 92
  testów jednostkowych, 96
  zasięgu, 161
  żądania, 131
konstruktor kontrolera, 167
kontroler, 14, 25, 43, 103, 166
  CartController, 21, 47
  EditCtrl, 111, 120, 125
  FilterCtrl, 196
  HelloController, 13
  IngredientsCtrl, 113
  ListCtrl, 111, 117, 123, 196
  NamesListCtrl, 136
  NewCtrl, 112
  RootController, 212
  SearchController, 185
  ViewCtrl, 111
kontrolka datepicker, 191, 194
koszyk na zakupy, 18, 48

L

lista, 36
logika
  aplikacji, 14, 65
  biznesowa, 142
lokalizacja, 185, 186
luka w zabezpieczeniach, 146

Ł

łącza
  bezwzględne, 177
  href, 116
  względne, 177

M

menedżer NPM, 77
metoda, *Patrz* funkcja
metody
  konfiguracji modułu, 180
  modułu AngularJS, 178
  obiektu Recipe, 106
  obiektu zdarzenia, 184

Kup ksi k   Pole ksi k

model, 14, 25, 102
moduł
  główny, 179
  guthub, 114
  ngResource, 138
  ngSanitize, 189
  Sanitize, 188
modyfikacja
  ciasteczka, 148
  żądania, 132
monitorowanie
  elementów, 50
  zmian, 45
MVC, 14, 24

N

nagłówek
  Authorization, 213
  DO NOT TRACK, 133
nagłówki
  HTTP, 133
  uwierzytelnienia, 211
narzędzia, 73, 84
narzędzie
  Ant, 92
  Batarang, 85
    karta Model, 86
    karta Performance, 86
    właściwości elementów, 87
    zależności usługi, 87
  Karma, 76–78, 96
  RequireJS, 92
  Scenario Runner, 80, 82
  WebStorm, 73
  Yeoman, 70, 75
    dodawanie tras, 90
    funkcje, 88
    instalacja, 89
    testy, 91
    tworzenie projektu, 90
nasłuchiwanie zdarzeń, 183
nawias klamrowy, 28
nazwa dyrektywy, 152
ngResource, 142
notacja
  {{ }}, 20
  interpolacji, 39
NPM, Node Package Manager, 77

O

obiekt
  $scope, 44
  config, 131, 135
  Recipe, 106
  resolve, 114
  scope, 136
  zdarzenia, 184
obiektowy model dokumentu, 14
obiekty wstrzymane, 107
obietnica, 107, 143
obsługa
  błędów, 145, 210
  HTML5, 63
  kodów stanu, 212
  liczby mnogiej, 186
  lokalizacji, 187
  łączy, 176
  przekierowań, 211
  RequireJS, 96
  zdarzeń, 32, 33
ochrona przed luką, 147
oczyszczanie kodu HTML, 188
opakowanie kontrolki jQuery, 191
opcje
  dyrektywy, 151
  właściwości require, 167
operacje
  bitowe, 42
  logiczne, 42
  matematyczne, 42
  po stronie serwera, 142
optymalizacja, 83
  Simple, 83
  zaawansowana, 83
organizacja projektu, 70, 92

P

pasek
  nawigacyjny, 35
  tytułu, 162
plik
  angular.js, 187
  app.js, 94
  controller.js, 60
  controllers.js, 64, 95

Kup ksi k   Pole ksi k

detail.html, 59
index.html, 58, 64, 95
karma.config.js, 78
list.html, 59
main.js, 95, 98
pliki
aplikacji, 70
JavaScript, 70
konfiguracyjne, 72, 78
szablonów HTML, 71
pokazywanie elementów, 38
pole
combo, 200
tekstowe, 29, 120
wyboru, 200
wyszukiwania, 199
porównania, 42
prawo Demeter, 17
produkty, 55
programowanie, 69
projekt jQuery-File-Upload, 201
prototypowe dziedziczenie, 26
przechwycenie odpowiedzi, 145
przedstawianie danych, 14
przekazywanie plików, 201
przenoszenie treści, 157
przepisy kulinarne, 102
przepisywanie łączy, 177
przycisk zerowania, 32
publikacja danych modelu, 44

S

Scenario Runner, 80, 82
schematy weryfikacji HTML, 150
serwer
Karma, 78
RESTful, 106
Socket.IO, 206
WWW, 75, 90
serwis GitHub, 62
strategie wiązania, 161
strefy czasowe, 188
stronicowanie, 207
styl Jasmine, 80, 123
style CSS, 39
szablon, 15, 17, 27, 54, 103
szablon po stronie klienta, 12

Ś

ścieżki app/img, 71
środowisko IDE, 73, 79

T

tabela, 36
tablica currentPageItems, 209
TDD, Test-driven development, 76
technika TDD, 76, 80
test, 99
ACID, 35
integracji, 72, 80
jednostkowy, 72, 79, 122, 135, 142
kontrolera, 136
metody, 125
scenariusza, 126, 127
typu E2E, 72, 80, 99
usługi, 124, 208
testowanie, 76
token, 148
transformacje
odpowiedzi, 134
żądania, 134
trasa, 44, 57, 114
tryb
hashbang, 175
HTML5, 175
tworzenie
aplikacji sieciowych, 11
dyrektyw, 109, 150
filtrów, 56
funkcji dyrektywy, 63
interfejsu użytkownika, 14
obiektu wstrzymanego, 107
obietnic, 108
paska, 162
projektu, 90, 91
szablonu, 12, 115
trasy, 57
usług, 53
zasięgu, 160
typ zasięgu, 161

Kup ksi k Pole ksi k

WebStorm, 74
wyrażenia, 45
Yeoman, 75
zasobów AngularJS, 138

U

uaktualnianie listy, 196
ukrywanie
  błędów, 212
  elementów, 38
uruchamianie
  aplikacji, 28, 75
  serwera, 90
  testów, 91
    ręczne, 81
    zautomatyzowane, 81
usługa, 53, 105
  $cookies, 185
  $cookieStore, 185
  $http, 61, 129, 137, 142
  $httpProvider, 135
  $location, 17, 57, 124, 171
    funkcje, 174
    integracja AngularJS, 173
    integracja HTML5, 174
    tryb hashbang, 175
    tryb HTML5, 175
  $q, 143
  $route, 57, 124
  $routeProvider, 57
  Authentication, 213, 214
  Error, 211
  errorService, 211
  filterService, 196
  MultiRecipeLoader, 124
  Pagination, 210
  Paginator, 208
  stronicowania, 207
uwierzytelnienie żądania, 213
użycie
  atrybutu ng-model, 45
  biblioteki Socket.IO, 204
  dyrektywy, 153
  dyrektywy focus, 65
  filtrów, 196
  funkcji $watch(), 45, 48
  kontrolerów, 43, 166
  Pythona, 76
  Scenario Runner, 82
  serwera WWW, 75, 81
  transformacji, 135
  usługi $location, 174

W

wczytywanie
  modułu, 179
  skryptu, 23
weryfikacja
  danych wejściowych, 65
  kodu HTML, 149
  pól formularza, 66
wiązanie select, 193
widok, 14, 25, 44
wielokrotne
  użycie kodu, 170
  użycie komponentów, 170
właściwości
  AngularJS, 89
  elementów, 87
  obiektu, 42
  obiektu zdarzenia, 184
właściwość
  $scope.isDisabled, 40
  $valid, 66
  innerHtml, 15
  require, 167
  window.location, 171, 173
wskazanie
  atrybutu select, 195
  lokalizacji, 57
wstrzykiwanie
  usługi, 96
  zależności, 16, 126
wtyczka
  Batarang, 85
  FileUpload, 203
wydajność aplikacji, 83
wykonywanie testów, 98
wyrażenia, 42
wyświetlanie
  listy, 196
  tekstu, 28, 118
wywołania
  API, 180
  zwrotne, 141

Kup ksi k Pole ksi k

wywołanie
$apply, 172
AngularJS, 23
factory(), 180
provider(), 181
scope.$apply, 172
select(), 194
service(), 181
wyzerowanie wartości pola tekstowego, 32
wzorzec Singleton, 183

X

XHR, 130
XSRF, 147

Z

zabezpieczenia JSON, 146
zagnieżdżenie zasięgów, 86
zależności, 51, 71, 179
RequireJS, 94
usługi, 87
zarządzanie
danymi, 14
modułami, 179
zależnościami, 92
zasada minimalnej wiedzy, 17
zasięg, 44, 86, 160, 182
globalny, 26
główny $rootScope, 182
nadrzędny, 161, 183
odizolowany, 160
potomny, 161, 183
zasoby
AngularJS, 138
RESTful, 106, 137
statyczne, 71
zasób karty kredytowej, 139
zdarzenia zasięgu, 183
zdarzenie
click, 166
loginRequired, 211
on-select, 194
select, 195
zintegrowane środowisko
programistyczne, IDE, 73
zlokalizowany zestaw reguł, 187
zmiana
elementów drzewa DOM, 63
widoków, 57
znacznik, *Patrz* element
znacznik semantyczny, 34
znak
dolara, 53
dwukropka, 140

Ż

żądanie, 131
CreditCard, 139
DELETE, 138
GET, 138
POST, 107, 138
XHR, 130
żywe szablony, 74

Kup książkę Poleć książkę

Kup ksi k

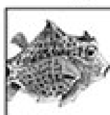
Pole ksi k

PROGRAM
PARTNERSKI
GRUPY WYDAWNICZEJ 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 WYDAWNICZA
Helion SA

Programowanie

O'REILLY®

AngularJS



AngularJS to hit ostatnich miesięcy w aplikacjach internetowych, wniósł bowiem do kodu JavaScript powiew świeżości oraz najlepsze praktyki znane z innych języków programowania. Architektura MVC, wstrzykiwane zależności, wiązanie danych to tylko niektóre z cech AngularJS. Jeżeli zaintrygowały Cię jego możliwości i chciałbyś zgłębić potencjał tego rozwiązania, to trafiłeś na doskonałą książkę!

Napisana przez inżynierów Google, pracujących na co dzień przy AngularJS, zawiera najświeższe informacje z pierwszej ręki. Sięgnij po tę książkę i przekonaj się, jak szybko stworzyć łatwą w utrzymaniu aplikację, korzystającą z nowoczesnych wzorców, komunikującą się wydajnie z serwerem oraz pokrytą automatycznymi testami. Zdobędziesz wiedzę na temat dyrektyw, kontrolerów oraz szablonów. Ponadto przekonasz się, jak tworzyć aplikacje wspierające wiele języków narodowych oraz w jaki sposób radzić sobie z ciasteczkami (cookies). To doskonała lektura dla wszystkich osób chcących dzięki AngularJS zmienić swoje podejście do tworzonego kodu JavaScript.

Dzięki tej książce:

- poznasz technikę wiązania danych
- wykorzystasz architekturę MVC w języku JavaScript
- poznasz dostępne zasięgi w AngularJS
- opanujesz komunikację z serwerami
- swobodnie wykorzystasz możliwości AngularJS

Twój przewodnik po świecie AngularJS!

ISBN 978-83-246-9990-2