O'REILLY®

Ember.js

dla webdeveloperów

POZNAJ ATUTY EMBER.JS!

Helion

Jesse Cravens, Thomas Q Brady

Tytuł oryginału: Building Web Apps with Ember.js

Tłumaczenie: Andrzej Stefański

ISBN: 978-83-283-0610-3

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/emberw.zip

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

Printed in Poland.

Spis treści

ROZDZIAŁ 4.

Prototyp aplikacji Rock'n'Roll Call: szablony

W rozdziale 3. poznaliśmy nowoczesne narzędzia wspomagające pracę przy przygotowywaniu szkieletu złożonej aplikacji internetowej. Gdy mamy już dobrze opanowane podstawy, nadszedł czas, by zacząć pisać kod.

W tym rozdziale skupimy się na systemie do obsługi szablonów — Handlebars.js — który domyślnie jest dołączany do aplikacji z frameworkiem Ember. Zwykle niełatwo jest ustalić, od czego powinno się zaczynać omawianie tego typu systemu, z wieloma zależnymi od siebie „kurami" i „jajkami". My zdecydowaliśmy się rozpocząć od szablonów, ponieważ naszym zdaniem od tego powinieneś zaczynać tworzenie ambitnego projektu. Po tym, gdy przewrócisz ostatnią stronę tego rozdziału, będziesz wiedział, jak wprowadzić „żywe" (powiązane dwustronnie) zmienne do szablonu HTML, jak automatycznie generować (i aktualizować) listy HTML (znaczniki `UL` i `OL`) z referencji do tablic, a nawet jak wykorzystywać w swoich szablonach konstrukcje logiczne (`if/then/else`).

Rock'n'Roll

Gdy pojawia się potrzeba napisania aplikacji demonstrującej nowy framework lub nowy sposób pisania aplikacji, okazuje się, że jedynym chyba rozwiązaniem jest napisanie aplikacji do zarządzania listą zadań. Zastanawialiśmy się, czy ma na to wpływ atmosfera w Dolinie Krzemowej. Mieszkamy w Austin w Teksasie, w okolicy czasem nazywanej Krzemowymi Wzgórzami — określenie to jest pochodną dużej ilości firm technologicznych, które częściowo lub w całości przeniosły tu swoje biura z Kalifornii. Firmy

te umiejscowiły się tutaj z wielu powodów, między innymi dla dużo tańszej ziemi i siły roboczej, ale nie możemy się oprzeć wrażeniu, że znaczenie miało również tempo życia. Aby zademonstrować swoje nastawienie, postanowiliśmy, że nasza przykładowa aplikacja nie będzie dotyczyła produktywności. Zbudujemy coś w stylu The Internet Movie Database (*imdb.com*), ale w celu indeksowania zespołów i muzyków, a nie filmów, reżyserów i aktorów. Aplikację tę nazwiemy *Rock'n'Roll Call*. A niezależnie od niej istnieje już wspaniała aplikacja demonstracyjna Ember, obsługująca listę rzeczy do zrobienia na portalu TodoMVC, którą można zobaczyć pod adresem *http://todomvc.com/examples/emberjs*.

Aby tego typu aplikacja była użyteczna, musi obsługiwać co najmniej poniższe opcje:

Użytkownik musi mieć możliwość wyszukiwania artystów (utworów) po nazwie (tytule).
: W idealnym przypadku nie powinno być konieczne określanie, czy szuka się artysty, czy utworu; użytkownik powinien mieć możliwość wpisania szukanego terminu i kliknięcia *Szukaj*.

Wyniki wyszukiwania powinny być wyświetlane z oznaczeniami, jakiego typu element (artysta czy utwór) został odnaleziony.
: Nie chcemy kłopotać użytkowników koniecznością zaznaczania, czy chcą szukać artysty, czy utworu, ale przy wyświetlaniu rezultatów powinny prawdopodobnie być udostępnione filtry pozwalające użytkownikowi na tym etapie ograniczyć zakres wyświetlanych informacji do odnalezionych artystów lub utworów.

Wyniki wyszukiwania powinny prowadzić do strony z większą ilością informacji na temat znalezionego elementu.
: Byłoby wspaniale, gdyby ta strona zawierała krótki opis znalezionego wyniku — biografię artysty lub historię utworu, a także linki do bardziej szczegółowych informacji lub nawet utworów dostępnych online.

Dla zabawy dajmy użytkownikom możliwość śledzenia popularności muzyki i artystów, których szukają, dzięki czemu użytkownicy będą mogli ocenić, jak powszechne są ich zainteresowania.
: Będziemy musieli gdzieś zapisywać te dane, a w idealnym przypadku będziemy potrzebowali jakiegoś ciekawego sposobu wizualizowania takich danych.

Oczywiście do uruchomienia aplikacji będziemy potrzebowali dużej ilości danych na temat muzyki. Prawdopodobnie największą korzyścią z tego, że postanowiliśmy przygotować taką aplikację demonstracyjną zamiast aplikacji do zarządzania zadaniami, jest fakt, iż praktycznie nie jest możliwe

samodzielne zbudowanie kompletnej aplikacji tego typu. Aplikacje do zarządzania zadaniami to wyspy — nie potrzebują żadnych danych, które nie zostały wygenerowane przez użytkownika. Nasza aplikacja, tak jak większość aplikacji, które będziesz budować w realnym świecie, będzie korzystać z komunikacji z usługami sieciowymi — zewnętrznymi serwerami dostarczającymi danych. W tym przypadku przygotowaliśmy aplikację komunikującą się z The Echo Nest (*http://the.echonest.com*), wspaniałą usługą *music intelligence* z bogatym zestawem opcji, olbrzymią i ciągle rosnącą bazą danych, wspaniałą dokumentacją oraz API dla JavaScript.

Uwaga, spoiler! Oto w jaki sposób zamierzamy to wszystko zrobić:

1. Utworzymy szablon, który będzie miał pole `TextField` frameworka Ember połączone ze zmienną o nazwie `searchTerms`, przekazującej dane do metody `action` zdefiniowanej w klasie `ApplicationController`, która przekierowuje do ścieżki `SearchResultsRoute`. Zdefiniowany dla tej ścieżki `SearchResultsController` odpyta Echo Nest API o dane znajdujące się w zmiennej `searchTerms`.

   `SearchResultsController` odpyta Echo Nest API dwukrotnie, raz zakładając, że użytkownik szuka nazwy artysty, i drugi raz szukając nazwy utworu.

2. Nasz szablon `search-results` przetworzy oddzielnie wyniki wyszukiwań, tworząc nazwy klas, które pozwolą nam zwizualizować wyniki wyszukiwania artystów i utworów w inny sposób.

   Nasz szablon `search-results` będzie zawierał kilka znaczników `checkbox` frameworka Ember oraz miejsce do wyświetlenia naszej listy z wynikiem wyszukiwania artystów i utworów z warunkowym wyświetlaniem zależnym od wybranych wartości tych kontrolek.

3. Wyniki wyszukiwania Echo Nest zawierają unikatowy identyfikator elementów spełniających wyniki wyszukiwania. Nasz szablon `search-results` powinien zawierać pola `link-to`, które będą odnośnikami do ścieżek stworzonych specjalnie w celu wyświetlenia szczegółowych danych na temat artysty (`ArtistRoute`) oraz utworu (`SongRoute`). Te ścieżki będą generować dodatkowe zapytanie do Echo Nest API, pozwalające pobrać informacje na temat wybranego elementu i przekazujące unikatowy identyfikator związany z odnośnikiem, jaki użytkownik kliknął w wynikach wyszukiwania.

   Uzbrojony w model wygenerowany z odpowiedzi Echo Nest widok `ArtistView` lub `SongView` powinien wyświetlić, pobrać obrazki, filmy oraz tekstowy opis elementu z odpowiedzi Echo Nest, a następnie wypełnić nimi szablon `artist` lub `song`.

4. Gdy użytkownik klika wynik wyszukiwania, `SearchResultController` powinien zapisać rekord w lokalnym magazynie wraz ze znacznikiem czasu i unikatowym identyfikatorem, wyświetlaną nazwą, typem i oceną *hotttnesss* — stosowaną wewnętrznie przez Echo Nest miarą popularności elementu. Później zajmiemy się zapisywaniem tych danych zdalnie.

   Odnośnik w głównym panelu nawigacyjnym powinien prowadzić użytkownika do `ActivityRoute` oraz szablonu `activity`, który powinien zawierać komponent wykorzystujący D3 do wizualizacji aktywności użytkownika — wszystkich danych pobranych podczas korzystania przez niego z aplikacji. Ponieważ będziemy wizualizować parametr o nazwie *hotttnesss* (z ang. „gorąco"), to naszym zdaniem mapa termiczna będzie właściwym rozwiązaniem.

Zaczynamy od HTML

Jeśli interesowałeś się choć trochę tematem nowoczesnych metodologii tworzenia stron internetowych, musiałeś słyszeć rozmowy o tym, że tradycyjne sposoby pracy i podejścia tracą na znaczeniu. „Kaskadowe" staje się coraz bardziej zawstydzającym słowem. Użyteczność wszystkich wykorzystywanych narzędzi i elementów tworzonych podczas prac nad aplikacją jest podawana w wątpliwość. Na przykład szkicowanie projektu dynamicznej aplikacji internetowej w Photoshopie, tworzenie wyglądu Twojego serwisu dla różnych szerokości i wysokości, jakie przeglądarka może wybrać (lub ograniczyć), to niekończąca się historia. Projektowanie dla idealnych urządzeń — wielkich monitorów stacjonarnych i potężnych procesorów — po prostu już nie wystarcza.

Jeśli to możliwe, najlepszym rozwiązaniem jest projektowanie w przeglądarce. Jeśli jesteś projektantem, który potrafi napisać wystarczającą ilość kodu HTML i CSS, by zrobić makietę, to wspaniale. Jeśli jesteś deweloperem i musisz współpracować z projektantem, który nie czuje się tak pewnie w HTML i CSS, w Twoim najlepszym interesie leży jak najwcześniejsze przygotowanie i jak najczęstsze aktualizowanie takiej makiety. I to jest jedna ze wspaniałych rzeczy przy typowym zastosowaniu Ember: Twoje szablony są pisane w HTML-u.

W przypadku tej aplikacji zamiast rozpoczynać od szkicowania lub rysowania projektów, przygotowaliśmy makietę, wykorzystując HTML i CSS. Mimo że pracujemy w tej branży od wielu lat, było to dla nas nowe doświadczenie. Przez większość czasu nasza praca przypominała pracę podwykonawcy na budowie biegającego z planem, młotkiem i masą dodatków. Tym razem

jednak czuliśmy się bardziej jak rzeźbiarze — ponieważ HTML był tak plastyczny, że mogliśmy wszystko dostrajać, dopóki nie zaczęło wyglądać tak, jak chcieliśmy.

Podstawy Handlebars.js

Możesz rozpocząć swój projekt, po prostu tworząc najprostszy statyczny HTML. Później wrócisz i wstawisz w odpowiednie miejsca znaczniki, warunki i zmienne Handlebars. Zacznijmy od stworzenia głównej zawartości naszej aplikacji: nagłówka, dostępnej globalnie części treści strony oraz stopki. Możemy po prostu dodać kod taki jak poniżej — wykorzystujący Twitter Bootstrap — bezpośrednio do naszego pliku *app/templates/application.hbs*:

```
<div class="wrapper">
  <div class="navbar navbar-inverse" role="navigation">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse"
            data-target=".navbar-ex1-collapse">
        <span class="sr-only">Przełącz nawigację</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a href="#" class="navbar-brand">Rock'n'Roll Call</a>
    </div>

    <div id="navbar-collapse-1" class="collapse navbar-collapse">
      <ul class="nav navbar-nav search-lockup">
        <li class="search-group">
        <input class="search-input" placeholder="Nazwa artysty lub utworu">
        <button class="btn btn-primary"><i class="glyphicon
            glyphicon-play"></i></button>
        </li>
      </ul>
      <ul class="nav navbar-nav navbar-right" >
        <li><a href="#">Aktywność</a></li>
      </ul>

    </div>
  </div>

  <div class="container-fluid" >
    <div class="row-fluid" >
      <!-- tutaj zawartość strony -->
    </div>
  </div>
</div>
<footer>
  <p>
```

```
    <i class="glyphicon glyphicon-copyright-mark" ></i> Helion 2015,
      <em>Ember.js dla webdeveloperów</em>
  </p>
  <p>
    Autorzy: Jesse Cravens <a href="http://twitter.com/jdcravens"
>@jdcravens</a>
    i Thomas Q. Brady <a href="http://twitter.com/thomasqbrady"
>@thomasqbrady</a>
  </p>
</footer>
```

A ponieważ nie jest to książka na temat CSS, pójdziemy na skróty i dodamy od razu cały znajdujący się w pliku *app/styles/style.scss* arkusz stylów dla tej strony, a także plik graficzny *app/images/stage.jpg*. Pliki te można znaleźć w materiałach dołączonych do książki, umieszczonych pod adresem *ftp://ftp.helion.pl/przyklady/emberw.zip*.

Twitter Bootstrap

Może zastanawiasz się, w jaki sposób Twitter Bootstrap znalazł się w Twojej aplikacji. Został on dodany przez Yeoman w rozdziale 3., po tym, gdy po pytaniu:

```
Would you like to include Twitter Bootstrap for Sass?
(Y/n)
```

wybrałeś *Y*.

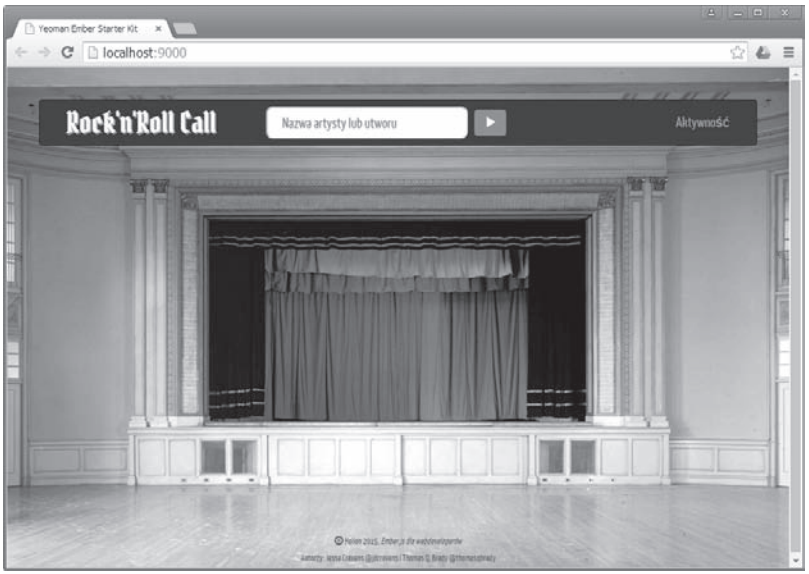
Rysunek 4.1 pokazuje, jak w tym momencie wygląda strona.

Czy powinienem korzystać ze znaczników <script>, czy z plików .hts?

W rozdziale 2. wbudowaliśmy szablony Handlebars w strony HTML wewnątrz znaczników `<script>` wyglądających tak:

```
<script type="text/x-handlebars" data-template-name="application" >
  <!-- tutaj kod szablonu -->
</script>
```

Było to możliwe, ponieważ korzystaliśmy z pełnej kompilacji Handlebars, która dołącza logikę w taki sposób, że w czasie działania wyszukuje znaczniki `<script>` i konwertuje je na obiekty JavaScript — w rzeczywistości fabryki — w pamięci (w `Em.TEMPLATES`, jeśli Cię to ciekawi), gdzie czekają gotowe do przetworzenia na kod HTML w zależności od potrzeb.



Rysunek 4.1. Szkielet naszej strony po dodaniu CSS

Jak można sobie wyobrazić, taka konwersja ze znaczników `<script>` na fabryki może obciążać komputer, szczególnie gdy Twoja aplikacja, a wraz z nią liczba szablonów, będzie rosnąć. To dlatego Handlebars pozwala wstępnie kompilować Twoje szablony i dołączać tylko okrojony plik *handlebars.runtime.js*, w którym nie ma logiki odpowiedzialnej za konwersję, co pozwala zaoszczędzić czas potrzebny do wyszukiwania szablonów oraz ograniczyć ilość danych pobieranych przez Twoją aplikację.

Dzięki pracy włożonej w przygotowanie Yeoman w rozdziale 2. możemy wykorzystać rajdową wersję Handlebars. Yo wygeneruje nam szablony Handlebars w oddzielnych plikach z rozszerzeniem *.hbs* podczas generowania widoków, a Grunt będzie dla nas pilnował tych szablonów, kompilując je do postaci klas JavaScript i łącząc je z resztą kodu JavaScript za każdym razem, gdy będziemy zapisywać zmiany.

Musisz jedynie powiadomić Grunt, że chcesz, by pilnował Twoich szablonów Ember, oraz wskazać mu miejsce, w którym się one znajdują, w taki sposób:

```
grunt.initConfig({
    yeoman: yeomanConfig,
    watch: {
```

```
        emberTemplates: {
          files: ' <%= yeoman.app %>/templates/**/*.hbs',
          tasks: ['emberTemplates', 'connect:livereload']
        }
    }
})
```

Odpowiadając na pytanie, można powiedzieć, że należy korzystać z plików *.hbs*, gdy tylko jest to możliwe przy zastosowaniach produkcyjnych. Znaczniki `<script>` sprawdzają się przy tworzeniu szkiców lub bardzo małych zadań.

Zmienne

Mając już stronę, która wykonuje to, czego potrzebujemy, i wygląda tak, jak chcemy, możemy zacząć zamieniać statyczne elementy na elementy programowe, wykorzystując Handlebars. Zacznijmy od nazwy naszej aplikacji. Powiedzmy, że chcesz modyfikować nazwę naszej aplikacji tak, by na przykład szwedzkim użytkownikom wyświetlać *Rock Upprop*. Aby to wykonać, będziemy potrzebowali przynajmniej dwóch rzeczy:

1. Musimy użyć Handlebars, by wyświetlić zmienną w miejscu naszej statycznej treści HTML.
2. Musimy gdzieś zdefiniować tę zmienną.

W praktyce definiowanie takiej zmiennej może okazać się dość skomplikowane. Na razie umieśćmy po prostu zmienną w naszym obiekcie `Rocknroll ↪callYeoman`. Przy okazji przyjrzyjmy się kodowi JavaScript utworzonemu przez Yeoman w naszym pliku *app/scripts/app.js*:

```
var RocknrollcallYeoman = window.RocknrollcallYeoman =
↪Ember.Application.create();

/* Order and include as you please. (Przestawiaj i dodawaj w razie potrzeby.) */
require('scripts/controllers/*' );
require('scripts/store' );
require('scripts/models/*' );
require('scripts/routes/*' );
require('scripts/views/*' );
require('scripts/router' );
```

Dodajmy taką linię zaraz po wywołaniu `Ember.Application.create()`:

```
RocknrollcallYeoman.applicationName = "Rock'n'Roll Call";
```

Później możemy dodać trochę więcej kodu wstawiającego do zmiennej tekst w obsługiwanych językach. Wprowadzona tutaj zmiana sprawia, że dopóki będziemy korzystać z kontrolek Handlebars we wszystkich miejscach, gdzie wyświetlamy nazwę naszej aplikacji, dopóty zmiana nazwy naszej aplikacji będzie wymagała wprowadzenia modyfikacji w jednym miejscu.

Wróćmy więc do naszego szablonu. Teraz możemy po prostu zmodyfikować linię:

```
<a href="#" class="navbar-brand">Rock'n'Roll Call</a>
```

do takiej postaci:

```
<a href="#" class="navbar-brand">{{RocknrollcallYeoman.applicationName}}</a>
```

Jak już wcześniej widzieliśmy, nie ogranicza się to do wstawienia zmiennej podczas ładowania strony. Jeśli zawartość zmiennej `RocknrollcallYeoman.appli ↪cationName` zmieni się podczas korzystania z aplikacji — na przykład jeśli użytkownik zmieni w opcjach wybrany język — zawartość wyświetlana w tym polu zostanie zaktualizowana automatycznie bez konieczności pisania dodatkowego kodu ponad to, co już zostało napisane.

Tworzenie odnośników za pomocą {{link-to}}

Zacznijmy od najprostszego przypadku. Prawdopodobnie zechcemy, by po kliknięciu logo można było wrócić do domyślnego stanu aplikacji, takiego jaki otrzymujemy po wpisaniu w przeglądarce podstawowego adresu. Nie utworzyliśmy jeszcze żadnych ścieżek ani kontrolerów, ale nie oznacza to, że nie zrobił tego Ember. Strona, którą oglądamy w przeglądarce, to... zgadniesz? Jest to domyślna ścieżka `IndexRoute` znajdująca się w domyślnej `ApplicationRoute`. Tutaj nasz użytkownik powinien wpisać tekst do wyszukania i wcisnąć *Enter* lub kliknąć odpowiedni przycisk, co powinno doprowadzić go do ścieżki z wynikami wyszukiwania. Może on też kliknąć odnośnik, by zobaczyć wizualizację swojej aktywności w wyszukiwarce, co powinno skierować go do `ActivityRoute`. Dlatego jeśli zechce on wrócić do strony, na którą w tej chwili patrzymy, powinien wrócić do `IndexRoute`. Dobrze, to dość proste. Zamień znacznik `<a>` na znacznik `link-to` Handlebars w poniższy sposób.

Statyczny HTML:

```
<a href="#" class="navbar-brand" >{{RocknrollcallYeoman.applicationName}}</a>
```

należy zmienić na szablon Handlebars:

```
{{#link-to "index" class="navbar-
brand"}}{{RocknrollcallYeoman.applicationName}}{{/link-to}}
```

Prawdopodobnie rozumiesz tę składnię. Nawiasy {{ i }} po prostu zastępują < i > z HTML-a. Tak samo jak w HTML-u `link-to` składa się ze znacznika otwierającego `{{#link-to ...}}` oraz znacznika zamykającego `{{/link-to}}`. Wewnątrz znacznika otwierającego tak jak w HTML-u można zadeklarować atrybuty znacznika `a`, który zostanie umieszczony na stronie, i w tym przykładzie zadeklarowaliśmy nazwę klasy `navbar-brand`. Część, która może Cię zastanawiać, to ciąg znaków przed tą deklaracją nazwy klasy: `index`. Podczas gdy w znaczniku HTML deklarujesz cel atrybutem `href`, w znaczniku Handlebars `link-to` przekazujesz nazwę ścieżki jako pierwszy parametr. Konwencja nazewnictwa jest prosta, gdy tylko się ją zrozumie. Bierzesz nazwę ścieżki (w tym przypadku `IndexRoute`), zamieniasz pierwszą literę na małą, usuwasz ciąg *Route*, wstawiasz minus przed każdą wielką literą oprócz pierwszej, rozdzielasz słowa, jeśli jest więcej niż jedno, i zamieniasz wszystkie litery na małe. Opis może wydawać się skomplikowany, ale ma to sens. Zobaczmy na przykładach:

- Ścieżka IndexRoute powinna być przekazana do link-to jako index.
- Ścieżka `SearchResultsRoute` powinna być przekazana do `link-to` jako `search-results`.

Wszystkie informacje na temat konwencji nazewnictwa można znaleźć w rozdziale „Naming Conventions" przewodnika „Ember Guides" pod adresem *http://emberjs.com/guides/concepts/naming-conventions/*.

Wprowadzanie danych z {{input}}

Jak możesz sobie wyobrazić, zawartość pola `input` z tekstem do wyszukiwania będzie dość ważna. W przypadku samego pola `input` możliwe jest uzyskanie dostępu do jego zawartości, a nawet przechwycenie jego przesłania z poziomu samego języka JavaScript, ale dużo łatwiej jest pozwolić zająć się tym Handlebars i Ember. Zastąpmy nasz znacznik `input` znacznikiem Handlebars:

Statyczny HTML:

```
<input class="search-input" placeholder="Nazwa artysty lub utworu" >
```

należy zamienić na szablon Handlebars:

```
{{input type="text" class="search-input" placeholder="Nazwa artysty lub utworu"}}
```

W tym miejscu pole `input` nie będzie jeszcze zbyt wiele robić. Łączeniem jego wartości oraz zdarzeniem `submit` zajmiemy się w następnym rozdziale.

Zwróćmy teraz uwagę na wyniki wyszukiwania. Zaczynamy od makiety wykonanej w statycznym HTML-u:

```
<div class="container-fluid" >
  <div class="row-fluid" >
    <div class="search-results-wrapper clearfix" >
      <div class="search-facets col-md-2" >
        <h3>Pokazuj: </h3>
        <ul class="facets" >
          <li>
            <label>Artystów</label>
            <input type="checkbox" checked="checked" >
          </li>
          <li>
            <label>Utwory</label>
            <input type="checkbox" checked="checked" >
          </li>
        </ul>
      </div>

      <div class="results col-md-10" >
        <h3>Artyści</h3>
        <ul class="search-results artists" >
          <li><a href="#" >Tom Waits</a></li>
          <li><a href="#" >Tom Waits & Keith Richards</a></li>
          <li><a href="#" >Tom Waits & Keith Richards</a></li>
          <li><a href="#" >Tom Waits [Vocalist] & Orchestra [Orchestra]
            & Michael Riesman [Conductor] & Bryars, Gavin
            [Composer] </a></li>
          <li><a href="#" >Tom Waits [Vocals] & Gavin Bryars Ensemble
            [Ensemble] </a></li>
          <li><a href="#" >Tom Waits [Vocalist]; Orchestra [Orchestra];
            Michael Riesman [Conductor] </a></li>
          <li><a href="#" >Tom Waits [Vocals] & Gavin Bryars Ensemble
            [Ensemble] & Bryars, Gavin [Composer] </a></li>
          <li><a href="#" >Tom Waits [Vocalist], Orchestra [Orchestra] &
            Michael Riesman [Conductor] </a></li>
        </ul>

        <h3>Utwory</h3>
        <ul class="search-results songs" >
          <li><a href="#" >"Tom Waits" - Panic Strikes a Chord</a></li>
          <li><a href="#" >"Tom Waits" - Doug Kuony</a></li>
          <li><a href="#" >"Tom Waits" - The Moonband</a></li>
          <li><a href="#" >"Tom Waits" - The Moonband</a></li>
          <li><a href="#" >"Tom Waits" - Spaghetti Western</a></li>
          <li><a href="#" >"Tom Waits" - The Passionate & Objective
          ↪Jokerfan</a></li>
          <li><a href="#" >"Tom Waits" - Mike Macharyas</a></li>
          <li><a href="#" >"Tom Waits" - Junkyard Poets</a></li>
          <li><a href="#" >"Tom Waits" - The Fall of Troy</a></li>
          <li><a href="#" >"Tom Waits" - Anouk</a></li>
        </ul>
      </div>
```

```
    </div>
  </div>
</div>
```

Na razie dodamy te wyniki do pliku *app/templates/index.hbs*. Następnie w pliku *app/templates/application.hbs* zmienimy linię z komentarzem:

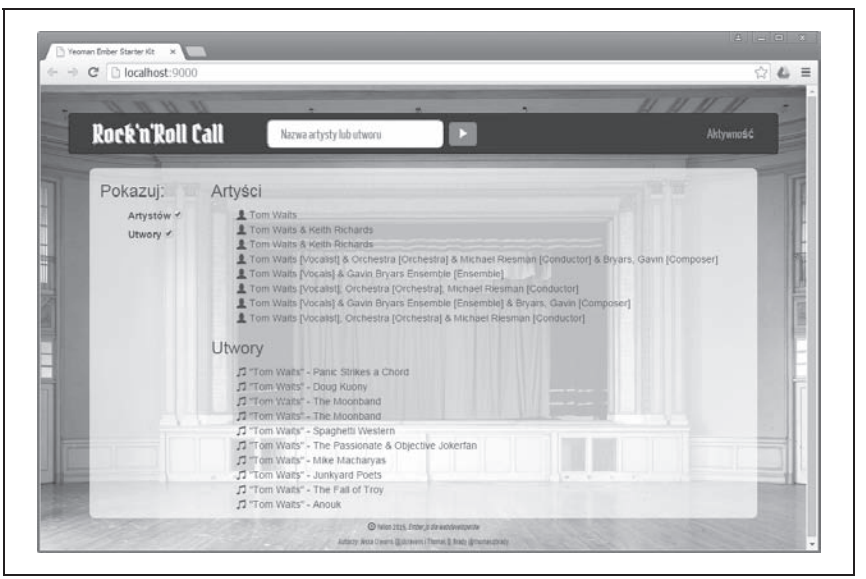
```
<!-- tutaj zawartość strony -->
```

na znacznik *handlebars*:

```
{{outlet}}
```

aby wskazać frameworkowi Ember, w którym miejscu należy wyświetlić szablon związany z bieżącą ścieżką (*index*).

Rysunek 4.2 pokazuje wygląd strony w tej chwili.



Rysunek 4.2. Wyświetlony szablon z wynikami wyszukiwania

Ale czym są `{{outlet}}`, `IndexRoute` i `index.hbs`?

Nie wyjaśniliśmy jeszcze, jak można przechodzić między różnymi szablonami. Na razie jednak wystarczy wiedza, że wyświetlamy treści związane z `IndexRoute` oraz szablonem *index.hbs* w miejsce `{{outlet}}` w szablonie *application.hbs*. W kolejnym rozdziale omówimy tworzenie nowych szablonów, ścieżek i łączenie wszystkiego ze sobą.

Listy z {{each}}

Możliwe, że listy nienumerowane będą pierwszym elementem, w którym korzyść z używania szablonów będzie dla Ciebie oczywista. Handlebars obsługuje je w bardzo przyjazny sposób. Najpierw zredukujmy każdą z takich list do jednego elementu, a następnie zobaczymy, jak zwielokrotnić ten szablon dla każdego elementu otrzymanego w wynikach wyszukiwania. Lista artystów w wersji HTML wygląda tak:

```
<ul class="search-results artists">
        <li><a href="#">Tom Waits</a></li>
</ul>
```

Istotne jest to, że będziemy chcieli wykorzystać jeden znacznik `ul`, taki jak tutaj, ale zechcemy umieścić większą ilość znaczników `li` wewnątrz niego. Handlebars obsługuje znacznik `each`, który przechodząc przez kolejne elementy tablicy, tworzy coś w rodzaju wewnętrznego szablonu dla każdego elementu tablicy. Zacznijmy od stworzenia prostej tablicy, którą będziemy mogli wyświetlić. Do pliku *app.js* dodaj poniższe linie po deklaracji `applicationName`:

```
RocknrollcallYeoman.dummySearchResultsArtists = [
  {
    id: 1,
    name: 'Tom Waits',
    nickname: 'Tommy',
    type: 'artist',
    enid: 'ARERLPG1187FB3BB39'
  },
  {
    id: 2,
    name: 'Thomas Alan Waits',
    type: 'artist',
    enid: 'ARERLPG1187FB3BB39'
  },
  {
    id: 3,
    name: 'Tom Waits & Keith Richards',
    type: 'artist',
    enid: 'ARMPVNN13CA39CF8FC'
  }
];
```

Teraz możemy wykorzystać tę tablicę w naszym szablonie. Zastąp znacznik `li` z kodu HTML znacznikiem `each` Handlebars w taki sposób:

```
<ul class="search-results artists">
  {{#each RocknrollcallYeoman.dummySearchResultsArtists}}
    <li><a href="#">{{name}}</a></li>
  {{/each}}
</ul>
```

Magia, prawda? Nie jest nawet najważniejsze, że mechanizm ten przetwarza naszą zmienną globalną. Często w Ember i Handlebars naprawdę magiczne rzeczy dzieją się, gdy wartość zapisana w przekazanej zmiennej — w naszym przypadku globalnej zmiennej `dummySearchResultsArtists` — ulega zmianie, a Twoja lista automatycznie się aktualizuje.

Zauważ, że kontekst — lub zakres zmiennej — zmienia się wewnątrz znacznika `each`. Do węzła `text` w naszym znaczniku `a` odwołujemy się, po prostu pisząc `{{name}}`, a nie `App.dummySearchResultsArtists[index].name` czy coś w tym rodzaju. Wewnątrz pętli `each` kontekst wskazuje na bieżący obiekt z tablicy i możesz uzyskać dostęp do jego właściwości, odwołując się do nich za pomocą nazwy.

Co by było, gdybyśmy w każdym z naszych obiektów `dummySearchResults ↪Artists` mieli właściwość `nicknames`, która byłaby tablicą pseudonimów używanych przez artystę? Przypuśćmy, że chcielibyśmy wyświetlić wszystkie te pseudonimy w postaci oddzielnych wyników wyszukiwania w taki sposób, że „TAFKAP" — „The Artist Formely Known as Prince" — byłby oddzielnym elementem w wynikach wyszukiwania. Aby uniknąć nieporozumień, dopiszemy *alias [prawdziwe nazwisko lub oryginalna nazwa]*. W końcu ktoś może szukać artysty, wpisując jego pseudonim i nie wiedząc nawet o tym, że jest to pseudonim.

Mogłeś pomyśleć, że takie przełączanie kontekstu może w takim wypadku nie być zbyt wygodnym rozwiązaniem. Zobaczmy dlaczego. Oto pierwszy element naszego teoretycznego szablonu (bardzo byśmy tego chcieli, ale The Echo Nest nie podaje pseudonimów):

```
<ul class="search-results artists" >
  {{#each RocknrollcallYeoman.dummySearchResultsArtists}}
    {{#each ...
```

No dobra! Jak mamy odwoływać się do naszego lokalnego obiektu? Choć nie będziesz tego zbyt często używał, może nawet nie użyjesz nigdy, okazuje się, że możesz to zrobić w ten sposób:

```
<ul class="search-results artists">
  {{#each RocknrollcallYeoman.dummySearchResultsArtists}}
    <li><a href="#">{{this.name}}, alias "{{this.nickname}}"</a></li>
  {{/each}}
</ul>
```

Ale użycie `this` tutaj nie wygląda zbyt dobrze, dlatego mamy szczęście, że Handlebars umożliwia skorzystanie z innej składni znacznika `each`, co pozwoli nam nazwać zmienną w taki sposób:

```
<ul class="search-results artists">
  {{#each artist in RocknrollcallYeoman.dummySearchResultsArtists}}
```

```
        <li><a href="#">{{artist.nickname}}, alias "{{artist.name}}"</a></li>
    {{/each}}
</ul>
```

A jeśli zechcesz jeszcze bardziej skrócić i uprościć, nawet nie musisz używać `this` ani nazwy zmiennej. Handlebars domyślnie zakłada, że bieżący obiekt modelu z przetwarzanej tablicy jest bieżącym kontekstem:

```
<ul class="search-results artists">
  {{#each RocknrollcallYeoman.dummySearchResultsArtists}}
      <li><a href="#">{{nickname}}, alias "{{name}}"</a></li>
  {{/each}}
</ul>
```

Teraz jednak trafiliśmy na inny problem. Nasi artyści z pseudonimami ładnie się wyświetlają, ale artyści bez pseudonimów nie pojawiają się wcale. Gdy zajrzymy do kodu, okazuje się, że ma to sens. Jeśli nie masz tablicy o nazwie `nicknames`, omijasz również część kodu. Potrzebujemy czegoś w rodzaju klauzuli `if`. Prawdopodobnie nie zaskoczy Cię to, że Handlebars obsługuje taką klauzulę.

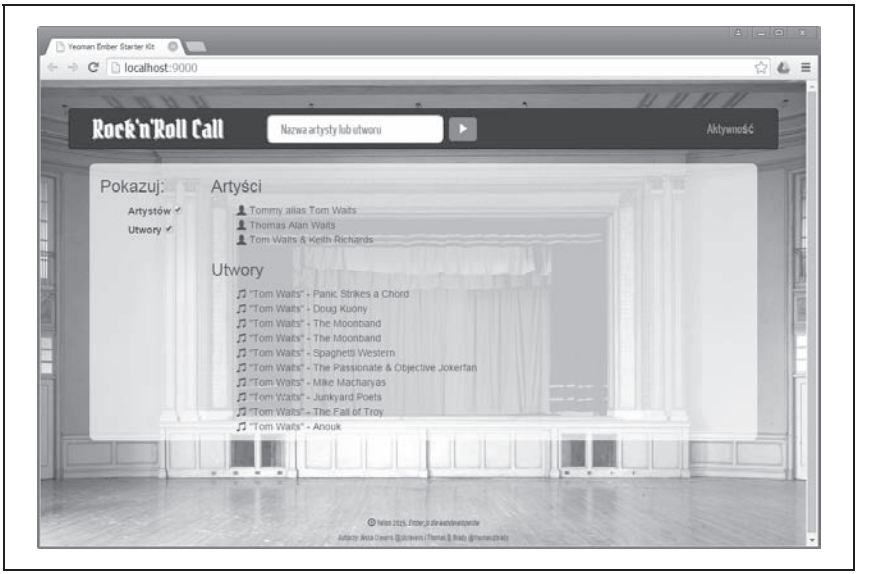
Warunki ze znacznikami {{if}} oraz {{else}}

Spróbujmy najpierw sprawdzać, czy wybrany artysta ma jakieś pseudonimy, i na tej podstawie zadecydujmy, czy wyświetlić je w pętli, czy skorzystać z prostszego szablonu. Może to wyglądać tak:

```
<ul class="search-results artists">
  {{#each RocknrollcallYeoman.dummySearchResultsArtists}}
    {{#if nickname}}
        <li><a href="#">{{nickname}} alias {{name}}</a></li>
    {{else}}
        <li><a href="#">{{name}}</a></li>
    {{/if}}
  {{/each}}
</ul>
```

Zamień znacznik `<ul>` w pliku *index.hbs* na powyższy kod. Rysunek 4.3 pokazuje, jak wygląda strona po takiej modyfikacji.

Możesz zastanawiać się, w jakiej sytuacji zawartość atrybutu `nicknames` jest uznawana za fałsz. Wszyscy oczekujemy takiego wyniku, gdy do tego atrybutu przypisana jest wartość podobnego typu: fałsz, `null`, niezdefiniowana. Co jednak w sytuacji, gdy mamy pustą tablicę: []? Spokojnie, Handlebars uzna każdą z tych wartości za fałsz.



Rysunek 4.3. Nasz szablon wyświetlania wyników z przykładowymi danymi i kodem zawierającym instrukcje warunkowe

Możesz już usunąć kod związany z obsługą pseudonimów, ponieważ był on dołączony jedynie dla demonstracji. W naszej aplikacji nie będziemy go używać, ponieważ korzystamy z API istniejącej usługi Echo Nest.

Obsługa działań użytkownika za pomocą {{action}}

Znacznik `{{action}}` służy do obsługi zdarzeń generowanych przez użytkownika korzystającego z aplikacji. Akcja zostanie przekazana do kontrolera powiązanego z bieżącą ścieżką. Gdy już skończymy omawiać ścieżki, nabierze to więcej sensu, ale z wprowadzenia w rozdziale 2. powinieneś znać przynajmniej podstawy.

Najczęstszym przypadkiem jest wykorzystanie akcji do przejmowania zdarzeń dotyczących kliknięcia odnośnika związanego ze znacznikiem `a` lub przycisku:

```
<button {{action 'robCos'}}>KLIKNIJ MNIE!</button>
```

Aby obsłużyć taką akcję, potrzebujemy odpowiedniego miejsca. Utwórzmy więc `IndexController` w pliku *app/scripts/controllers/index_controller.js*:

```
RocknrollcallYeoman.IndexController = Ember.Controller.extend({});
```

Tworząc kontroler, w rzeczywistości przesłaniamy istniejący `IndexController`, który Ember wygenerował dla nas automatycznie. Więcej informacji na temat automatycznego generowania znajduje się w następnym rozdziale.

Możemy teraz obsłużyć akcję w bieżącym kontrolerze, którym jest właśnie utworzony `IndexController`, ponieważ korzystamy z adresu startowego (*http://localhost:9000/*). Jeśli nie wszystko rozumiesz — cierpliwości; więcej na temat routingu dowiesz się w rozdziale 5.

```
RocknrollcallYeoman.IndexController = Ember.Controller.extend({
  actions: {
    viewedArtist: function(artist) {
      console.log('zaczekaj, przeglądam: ' + artist.name)
    }
  }
});
```

Możemy teraz korzystać ze znacznika `a` i specjalnie przygotowanych akcji, by przejąć obsługę kliknięć naszych użytkowników w pliku *index.hbs* w taki sposób:

```
<li><a {{action 'viewedArtist' this }} href="#">{{name}}</a></li>
```

Więcej informacji na temat tego, jak obsługujemy akcje, pojawi się później. W rozdziale 7. napiszemy bardziej szczegółowo, w jaki sposób trwale zapisywać dane z takich aktywności lokalnie i zdalnie.

Powiązane atrybuty

Korzystając z naszej dobrej passy, idźmy dalej. Załóżmy, że użytkownik kliknął nazwę artysty w wynikach wyszukiwania i przeszedł do strony opisującej wskazany element. W następnym rozdziale dowiesz się, jak utworzyć nową ścieżkę opisującą stan aplikacji i przejść do tej ścieżki. Tymczasem popatrzmy na ostateczny szablon i przeanalizujmy, co tutaj widzimy. Do pliku *app/templates/artist.hbs* dodaj taki kod:

```
<div class="entity-artist page-container">
  <div class="artist-bio-lockup clearfix">
    {{#if model.image}}
      {{#if model.license}}
        {{#if model.license.url}}
          <a {{bind-attr href="model.license.url"}}>
            <img {{bind-attr src="model.image.url"}} class="pull-right">
          </a>
```

```
        {{else}}
          <img {{bind-attr src="model.image.url"}} class="pull-right">
        {{/if}}
      {{else}}
        <img {{bind-attr src="model.image.url"}} class="pull-right">
      {{/if}}
    {{/if}}
    <h3 class="fancy" >{{model.name}}</h3>
    <h4>
      {{hotttnesss-badge model.hotttnesss}}
    </h4>
    <p class="bio pull-left" >Biografia(from {{model.biography.site}}):
    {{model.biography.text}}</p>
    <a {{bind-attr href="model.biography.url"}} class="pull-left">Więcej
    ↪informacji</a>
  </div>

  {{#if model.videos.length}}
  <div class="videos" >
    <h5>Filmy</h5>
    {{#each video in videos}}
            <a {{bind-attr href="video.url" }}><img {{bind-attr
                 src="video.image_url"}} class="video-thumbnail"></a>
    {{/each}}
  </div>
  {{/if}}
</div>
```

Popatrz na powyższy fragment kodu kilka razy. I co? Widać tutaj tylko jeden nowy element Ember. Są nim atrybuty `bind-attr` w szablonie, które są otoczone nawiasami klamrowymi i znajdują się *wewnątrz* znaczników HTML w miejscu, w którym zazwyczaj znajdują się atrybuty. Jest tak, ponieważ w końcu zostaną one zamienione na atrybuty. Dyrektywa `bind-attr` pozwala dynamicznie przypisywać dowolny atrybut HTML z dostępnych zmiennych bieżącego kontekstu w czasie działania aplikacji — najczęściej w modelu, ale może być to też coś wygenerowanego przez kontroler, widok lub ścieżkę. Składnia jest dość prosta. Nie jest to znacznik, dlatego nie ma tutaj części otwierającej i zamykającej i dlatego też nie ma konieczności użycia znaku `#` ani zamykającego znaku ukośnika. Po prostu piszemy:

```
<[dowolny znacznik] {{bind-attr }}>
```

Następnie wskazujemy, jaki atrybut chcemy wstawić — w tym przypadku utwórzmy link roboczy:

```
<a {{bind-attr href=video.url}}>Zobacz na Vimeo</a>
```

Przyglądając się naszemu szablonowi, zobaczysz wiele przykładów uzupełniania atrybutów `href`, atrybutów `src` obrazków i jeden przykład atrybutu `data-`. Możesz wykorzystywać dowolne atrybuty w miarę potrzeb,

w tym atrybut `class`. Choć atrybut `class` jest trochę bardziej kłopotliwy, ponieważ często masz więcej niż jeden i każdy z nich będzie musiał być połączony z innymi danymi. Handlebars dostarcza wielu sposobów zarządzania mnóstwem kombinacji dynamicznych i statycznych połączeń z nazwami klas, wszystkie są opisane w dokumentacji Ember.js, w rozdziale „Binding Element Class Names" pod adresem *http://emberjs.com/guides/templates/binding-element-class-names/*.

Zobaczmy teraz, jak możemy utworzyć własny znacznik, by poprawić kod, którego użyliśmy wcześniej do wyświetlania oznaczenia *hotttnesss*.

Tworzenie własnych znaczników

Jak dotąd zajmowaliśmy się znacznikami istniejącymi wcześniej w Ember.js, takimi jak np. znacznik Handlebars `input`. Znacznik `input` jest w rzeczywistości dołączony do biblioteki Ember.js jako rozszerzenie Handlebars. W chwili pisania tego tekstu znajduje się on w linii nr 31 514 w pliku Ember.js 1.4.1+pre.af87bd20:

```
Ember.Handlebars.registerHelper('input' , function(options) {
  Ember.assert('You can only pass attributes to the `input` helper, not
  ↪arguments' , arguments.length < 2);

  var hash = options.hash,
      types = options.hashTypes,
      inputType = hash.type,
      onEvent = hash.on;

  delete hash.type;
  delete hash.on;

  if (inputType === 'checkbox' ) {
    return Ember.Handlebars.helpers.view.call(this, Ember.Checkbox, options);
  } else {
    if (inputType) { hash.type = inputType; }
    hash.onEvent = onEvent || 'enter' ;
    return Ember.Handlebars.helpers.view.call(this, Ember.TextField, options);
  }
});
```

Ciekawe jest to, że możemy tworzyć swoje własne znaczniki Handlebars, korzystając z tego samego wzorca, jaki został zastosowany do wykorzystania Handlebars jako rozszerzenie. Aby to pokazać, utworzymy znacznik `hotttnesss`, który będzie wyświetlał płomienie oraz liczbę odpowiadającą parametrowi *hotttnesss* z naszego modelu.

Gdybyśmy mieli robić to bez tworzenia znacznika, musielibyśmy dodać wiele elementów z ikonami do *app/templates/index.hbs* w taki sposób:

```
<h4>
  Hotness:
  {{#if model.hotttnesss}}
    <i class="hotttnesss">
      <i class="glyphicon glyphicon-fire hotttnesss0"></i>
      <i class="glyphicon glyphicon-fire hotttnesss1"></i>
      <i class="glyphicon glyphicon-fire hotttnesss2"></i>
      <i class="glyphicon glyphicon-fire hotttnesss3"></i>
      <i class="glyphicon glyphicon-fire hotttnesss4"></i>
      <i class="glyphicon glyphicon-fire hotttnesss5"></i>
      <i class="glyphicon glyphicon-fire hotttnesss6"></i>
      <i class="glyphicon glyphicon-fire hotttnesss7"></i>
      <i class="glyphicon glyphicon-fire hotttnesss8"></i>
      <i class="glyphicon glyphicon-fire hotttnesss9"></i>
    </i>
    <span class="hotttnesss-badge" {{bindAttr data-hotttnesss =
"model.hotttnesss"}}></span>
  {{/if}}
</h4>
```

Musielibyśmy też utworzyć styl dla każdej z tych ikon. Poniżej pokazujemy jeden taki styl dla `hotttnesss0`:

```
h4 .hotttnesss .hotttnesss0 {
  font-size: 190%;
  top: -45%;
  left: -45%;
  position: absolute;
  color: #FF0000;
  direction: rtl;
  unicode-bidi: bidi-override;
}
```

Tak więc zamiast dołączać wszystkie 10 elementów, możemy napisać znacznik, który dołączy odpowiedni kod HTML, wykorzystując informację z modelu. Zrobimy to, zamieniając parametr `hotttnesss` zapisany jako wartość z zakresu od 0 do 1 na wartość z zakresu od 1 do 10, by następnie utworzyć w pętli odpowiedni kod HTML i wyświetlić go. Dodaj poniższy kod na końcu pliku *app.js*:

```
Ember.Handlebars.helper('hotttnesss-badge', function(value, options) {
  var h = parseFloat(value);
  var hotttnesss_num = Math.round(h * 100);
  var hotttnesss_css = Math.ceil(h * 10) - 1;
  var html = "<h4>Hotness: ";
  if (hotttnesss_num > 0) {
    html += '<i class="hotttnesss">';
    for (var i=0; i<hotttnesss_css; i++) {
      html += '<i class="glyphicon glyphicon-fire hotttnesss'+i+'"></i>';
    }
```

```
    html += "</i>";
    html += '<span class="hotttnesss-badge">'+hotttnesss_css+'</span></h4>';
  } else {
    html += "0</h4>";
  }
  return new Handlebars.SafeString(html);
});
```

W końcu musimy też dodać właściwość `hotttnesss` do naszych przykładowych danych w *app/scripts/app.js*:

```
RocknrollcallYeoman.dummySearchResultsArtists = [
  {
    id: 1,
    name: 'Tom Waits',
    type: 'artist',
    enid: 'ARERLPG1187FB3BB39',
    hotttnesss: '1'
  },
  {
    id: 2,
    name: 'Thomas Alan Waits',
    type: 'artist',
    enid: 'ARERLPG1187FB3BB39',
    hotttnesss: '.89'
  },
  {
    id: 3,
    name: 'Tom Waits & Keith Richards',
    type: 'artist',
    enid: 'ARMPVNN13CA39CF8FC',
    hotttnesss: '.79'
  }
];
```

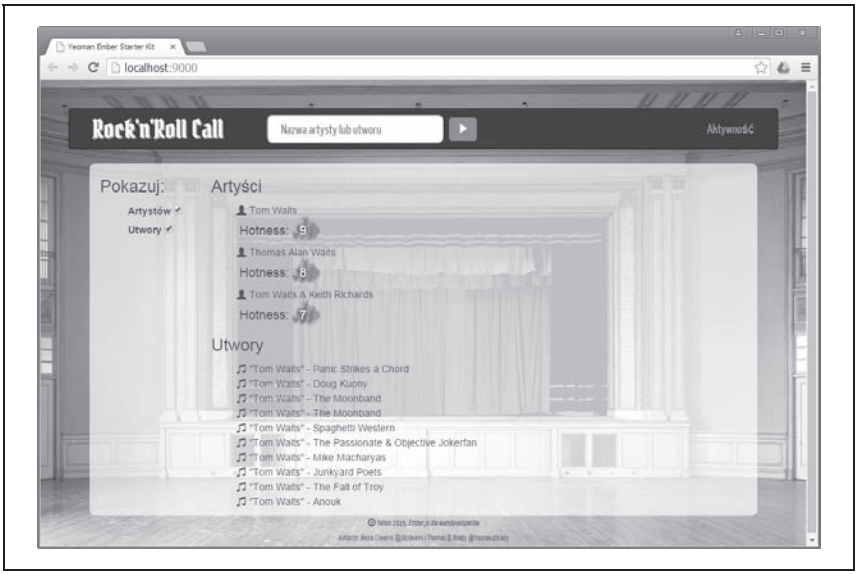
Możemy teraz dodać znacznik do szablonu w pliku *index.hbs* i przekazać dane z modelu jako pierwszy parametr w taki sposób:

```
{{hotttnesss-badge hotttnesss}}
```

Na rysunku 4.4 można zobaczyć, jak wygląda w tej chwili strona.

I w końcu się udało. Nie tylko ograniczyliśmy się do potrzebnego kodu, ale uprościliśmy też nasz szablon do jednej deklaracji znacznika.

Takie dane bardziej pasują do strony wyświetlającej szczegółowe informacje, do której możemy przejść, gdy chcemy dowiedzieć się więcej na temat wybranego elementu z wyników wyszukiwania. W następnym rozdziale przygotujemy te widoki i wtedy przeniesiemy znacznik *Hotness* na strony opisujące utwór oraz artystę.



Rysunek 4.4. Znacznik Hotness

Podsumowanie

Zrób sobie chwilę przerwy i pomyśl o tym, co do tej pory zrobiliśmy. Napisaliśmy zdecydowaną większość kodu HTML dla naszej witryny, przy czym dużą część w taki sposób, że kod jest tworzony dynamicznie, wstawiane są dane modelu, warunkowo wyświetlane mogą być części szablonu, a nawet w pętlach przetwarzane są tablice z modelu.

Zastanów się też nad tym, jakiego rodzaju pracę wykonaliśmy. Większą część tej pracy mógłby wykonać dowolny członek zespołu bez specjalnych kwalifikacji. Każdy, kto potrafi edytować HTML, mógłby wykonać większość pracy, a bardzo możliwe, że po włożeniu odrobiny wysiłku i nauczeniu się, jak należy korzystać ze zmiennych, połączonych atrybutów oraz instrukcji warunkowych, ten sam człowiek mógłby wykonać całą opisaną w tym rozdziale pracę bez Ciebie. Czyż nie jest to potężne narzędzie?

Wiedząc już to wszystko, przejdźmy do rzeczy, którymi będziesz się zajmował, gdy Twoi praktykanci będą przygotowywać kod HTML. Trzeba utworzyć prawdziwe modele oraz sprawić, by zapisane w nich dane w końcu pojawiały się w szablonach. W rozdziale 5. przyjrzymy się routerom i prostym modelom, by dowiedzieć się, jak to zrobić.

Skorowidz

E

F

G

H

I, J

K

L

M

N

O

P

Q

R

S, Ś

T

U, V

W

X

Y

Z, Ż

O'REILLY®

Ember.js dla webdeveloperów

Ember.js to biblioteka języka JavaScript, dzięki której możesz sprawnie budować aplikacje na podstawie wzorca MVC (ang. Model View Controller). Ember.js znakomicie sprawdza się przy tworzeniu jednostronicowych aplikacji internetowych (SPA, ang. Single Page Applications), a ponadto usprawnia automatyczne aktualizowanie szablonów czy dwustronne wiązanie danych. Jeżeli chcesz zgłębić tajniki tej biblioteki, ta książka jest dla Ciebie.

Dzięki niej poznasz potencjał Ember.js, jej atuty oraz techniki pracy. Lektura kolejnych rozdziałów pozwoli Ci poznać zasady działania generatora aplikacji Ember i szablonów, a także proces budowy modelu danych i kontrolerów. Na sam koniec dowiesz się, jak zapisywać dane po stronie klienta. Po zgłębieniu możliwości Ember.js nauczysz się przygotowywać serwer, który będzie przetwarzał żądania wysyłane z Twojej aplikacji. W tym celu wykorzystasz Rails MVC. W trakcie lektury zbudujesz własną aplikację o nazwie Rock'n'roll. Jesteś ciekaw, co potrafi? Sięgnij po tę książkę i przekonaj się sam!

Dzięki tej książce:

- poznasz możliwości Ember.js
- zbudujesz szkielet aplikacji i zaczniesz go uzupełniać
- przygotujesz odpowiedni model danych oraz kontrolery
- przygotujesz odpowiednie szablony i zaktualizujesz je w zależności od sytuacji
- wykorzystasz potencjał biblioteki Ember.js

Z Ember.js zaawansowane aplikacje internetowe są w Twoim zasięgu!

Jesse Cravens – główny inżynier technologii webowych w firmie Frog Design. Bierze udział we wszystkich etapach tworzenia zaawansowanych produktów i usług. Aktualnie zajmuje się głównie jednostronicowymi aplikacjami internetowymi, HTML5 oraz mobilnymi stronami internetowymi.

Thomas Q Brady – dyrektor technologiczny w Reaction, Inc. Pracuje nad oprogramowaniem, które ułatwia zarówno codzienną pracę w firmach, jak i tworzenie symulacji biznesowych czy prowadzenie interaktywnych kampanii marketingowych. Tworzy komercyjne aplikacje internetowe oraz korzysta z możliwości układu Arduino.

Helion
32284 numer katalogowy
księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900
Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci
Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl
ISBN 978-83-283-0610-3
9 788328 306103
Informatyka w najlepszym wydaniu
cena 39,90 zł