

Bez wysiłku poznaj niesamowite możliwości Ruby on Rails i twórz wspaniałe aplikacje

Head First Ruby on Rails



Metody wyszukujące
Rails pozwolą Ci
zapanować nad
danymi



Zintegrowanie
aplikacji z Google
Maps sprawi, że już
nigdy się nie zgubisz



Zobaczysz, jak Suzy sprawdziła
swoich adoratorów i uniknęła
kolejnego nieudanego wieczoru

Edycja
polska



Opanujesz
Embedded Ruby
i przejmiesz kontrolę
nad swoimi
aplikacjami



Dodasz do swojego
światła Rails technologie
Ajax i XML

O'REILLY®

David Griffiths

Helion

Tytuł oryginału: Head First Rails

Tłumaczenie: Anna Trojan

ISBN: 978-83-246-6056-8

© Helion S.A. 2010.

Authorized translation of the English edition of Head First Rails

© 2009 O'Reilly Media, Inc.

This translation is published and sold by permission of O'Reilly Media, Inc.,
the owner of 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 kopowanie
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 właścicielami.

Autor oraz Wydawnictwo HELION dołożyły wszelkich starań, by zawarte
w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej
odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne
naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION
nie ponoszą 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)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

http://helion.pl/user/opinie?hfror_ebook

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/hfror.zip>

Printed in Poland.

- Poleć książkę na Facebook.com
- Kup w wersji papierowej
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » [Nasza społeczność](#)

Pochwały dla książki *Head First Ruby on Rails*

„Książka *Head First Ruby on Rails* kontynuuje tradycje serii *Head First*, dostarczając przydatnych, S praktycznych informacji, które pozwolą czytelnikowi na szybkie poczynienie postępów w nauce. S Jest to doskonała pozycja dla osób poznających dopiero Rails, jak i dla tych, które chcą sobie S przyswoić najnowsze dostępne możliwości”.

— **Jeremy Durham, programista stron internetowych**

„Chciałbym, by książka ta była dostępna, kiedy to ja zaczynałem swoją przygodę z Rails. S Mogłaby mi niezwykle pomóc”.

— **Mike Isman, programista stron internetowych**

„Uwielbiam książki z serii *Head First*. Są nie tylko niezwykle cenne jako materiał edukacyjny, S ale także zabawne!”.

— **LuAnn Mazza, korektor merytoryczny książki**

„Książka *Head First Ruby on Rails* jest świetnym, obszernym wprowadzeniem do tworzenia S stron internetowych w duchu Web 2.0. Pozycja ta pokaże Ci, jak szybko i łatwo można S tworzyć rozbudowane witryny następnej generacji”.

— **Matt Proud, administrator systemów oraz programista**

„*Head First Ruby on Rails* to książka, którą chciałbym mieć pod ręką, kiedy to ja S uczyłem się Rails. Pokaże Ci wszystkie niezbędne do pracy elementy w humorystyczny S i konkretny sposób”.

— **Eamon Walshe, trener metodologii Agile**

Pochwały dla książki *Head First Ajax*

„Ajax to coś więcej niż złożenie istniejących technologii, wprowadzenie niewielkich zmian do aplikacji S internetowej i oświadczenie, że teraz oparta jest ona na Ajaksie. W książce *Head First Ajax* Rebecca S M. Riordan przeprowadzi Cię przez wszystkie etapy budowania aplikacji opartej na Ajaksie i pokaże, S że Ajax to coś więcej niż tylko »ten niewielki element asynchroniczny« — to lepsze podejście S do projektowania witryn internetowych”.

— **Anthony T. Holdener III, autor książki *Ajax: The Definitive Guide***

„Książki z serii *Head First* nie tylko się czyta. Te książki się »wykonuje«. A to ogromna różnica”.

— **Pauline McNamara, konsultant techniczno-pedagogiczny S
uniwersyteckich projektów eLearningowych, Szwajcaria**

„Autor w świetny sposób naucza różnych aspektów technologii Ajax, powracając do poprzednich S lekcji bez powtarzania się i wprowadzając często pojawiające się problemy w sposób, który pozwala S czytelnikom na ich samodzielne rozwiązywanie. W obszarach, w których brakuje jeszcze ustalonych S praktyk, czytelnikowi przedstawiane są wszystkie możliwości i zachęca się go do samodzielnego S podjęcia decyzji”.

— **Elaine Nelson, projektant stron internetowych**

„Masz problemy z Ajaksem? Poradzisz sobie z nimi dzięki tej książce! Przyswoisz sobie podstawowe S pojęcia, a przy okazji będziesz się nieźle bawić”.

— **Bear Bibeault, architekt aplikacji internetowych**

Pochwały dla innych książek z serii

„Wczoraj otrzymałem tę książkę; zacząłem ją czytać... i nie mogłem przestać! Jest niesamowicie S cool. Jest zabawna, ale jednocześnie omawia wiele zagadnień i jest bardzo konkretna. S Naprawdę jestem pod wrażeniem”.

— **Erich Gamma, IBM Distinguished Engineer oraz współautor S książki Design Patterns**

„Jedna z najzabawniejszych, najmadrzej napisanych książek poświęconych projektowaniu S oprogramowania, jakie kiedykolwiek czytałem”.

— **Aaron LaBerge, VP Technology, ESPN.com**

„To, co kiedyś sprowadzało się do długiego procesu uczenia się metodą prób i błędów, S teraz zostało zredukowane do wciągającej lektury”.

— **Mike Davidson, CEO, Newsvine, Inc.**

„Idea elegancji projektu jest podstawą każdego rozdziału, a każda koncepcja przedstawiona S jest z równymi dawkami pragmatyzmu i inteligentnego humoru”.

— **Ken Goldstein, Executive Vice President, Disney Online**

„Kocham *Head First HTML with CSS & XHTML* — książka ta w dowcipnej formie pokaże S Ci wszystko, czego powinieneś się nauczyć”.

— **Sally Applin, projektant interfejsów użytkownika i artysta**

„Zazwyczaj kiedy czytam książkę lub artykuł poświęcony wzorcówm projektowym, od czasu S do czasu muszę na siłę przypominać sobie, że mam się skoncentrować na przyswojeniu treści. S Inaczej jest z tą książką. Choć może to zabrzmić dziwnie, ta książka sprawia, że nauka S wzorców projektowych staje się zabawą”.

Ta książka zdaje się wręcz krzyczeć: »Zrób to sam, mój drogi!«”.

— **Eric Wuehler**

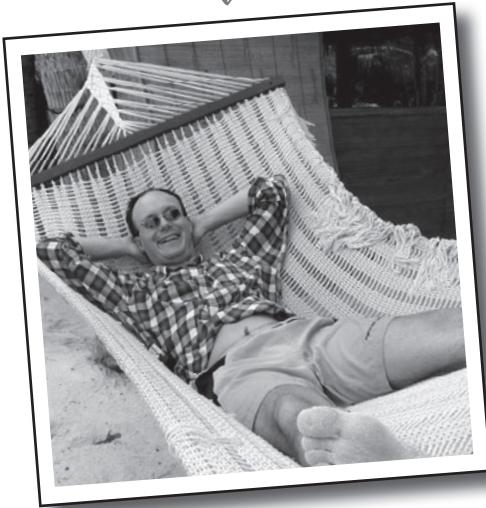
„Wprost kocham tę książkę. Tak naprawdę pocałowałem ją nawet na oczach własnej żony”.

— **Satish Kumar**

Dla Dawn, i pamięci mojej matki — Joan Beryl Griffiths

Autor Head First Ruby on Rails

David Griffiths



David Griffiths zaczął programować w wieku 12 lat, po obejrzeniu filmu dokumentalnego poświęconego pracy Seymoura Paperta. W wieku 15 lat napisał implementację języka komputerowego Paperta — LOGO. Po ukończeniu studiów matematycznych na uniwersytecie zaczął pisać programy komputerowe oraz artykuły do czasopism. Obecnie pracuje jako trener metodologii Agile w Wielkiej Brytanii, pomagając ludziom tworzyć prostsze i bardziej wartościowe oprogramowanie. Swój wolny czas spędza na podróżach z cudowną żoną Dawn.

Spis treści (skrócony)

Wprowadzenie	21
1. Naprawdę szybkie Rails. <i>Początki</i>	33
2. Aplikacje Rails — stworzone, by nimi zarządzać. <i>Poza rusztowaniem</i>	81
3. Wszystko się zmienia. <i>Wstawianie, uaktualnianie i usuwanie</i>	139
4. Prawda czy konsekwencje? <i>Wyszukiwanie w bazie danych</i>	189
5. Zapobieganie błędom. <i>Sprawdzanie poprawności danych</i>	223
6. Łączenie wszystkiego razem. <i>Tworzenie połączeń</i>	255
7. Ograniczanie ruchu. <i>Ajax</i>	299
8. Wszystko wygląda teraz inaczej... <i>XML i różne reprezentacje</i>	343
9. Kolejne kroki. <i>Architektura REST i Ajax</i>	393
10. Rails w świecie rzeczywistym. <i>Prawdziwe aplikacje</i>	437
Skorowidz	455

Spis treści (z prawdziwego zdarzenia)



Wprowadzenie

Przedstawienie swojego mózgu na Rails. A zatem tutaj Ty próbujesz się czegoś nauczyć, podczas gdy Twój mózg próbuje oddać Ci przysługę, starając się, by to, czego się nauczyłeś, nie zostało zapamiętane. Twój mózg myśli sobie: „Lepiej zostawić miejsce na ważniejsze rzeczy, takie jak to, których dzikich zwierząt należy unikać i czy jazda na snowboardzie nago jest złym pomysłem”. Jak zatem możesz zmusić swój mózg do zaakceptowania przekonania, że Twoje życie uzależnione jest od poznania Rails?

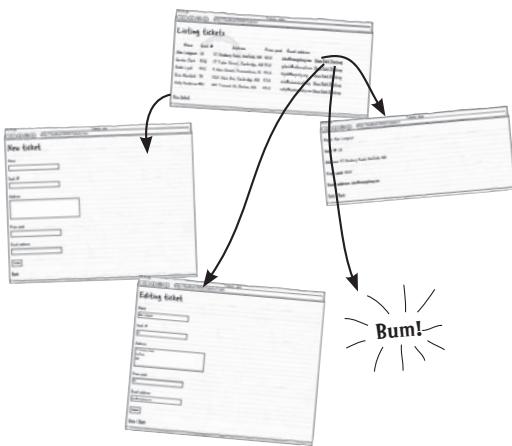
Dla kogo przeznaczona jest ta książka?	22
Wiemy, co sobie myślisz	23
Metapoznanie — myślenie o myśleniu	25
Oto, co możesz zrobić, by skłonić swój mózg do posłuszeństwa	27
Ważne informacje	28
Zespół korektorów merytorycznych	30
Podziękowania	31

Początki

1

Naprawdę szybkie Rails

Chcesz szybko zacząć pisać aplikacje internetowe? Powinieneś zatem poznać Rails. Rails to najfajniejsza i najszybsza platforma programowania, jaka istnieje. Pozwala tworzyć w pełni funkcjonalne aplikacje internetowe szybciej, niż kiedykolwiek wydawało się to możliwe. Początki są łatwe — wystarczy zainstalować Rails i zacząć przewracać strony książki. Zanim się zorientujesz, o lata świetlne wyprzedzisz swoich konkurentów!



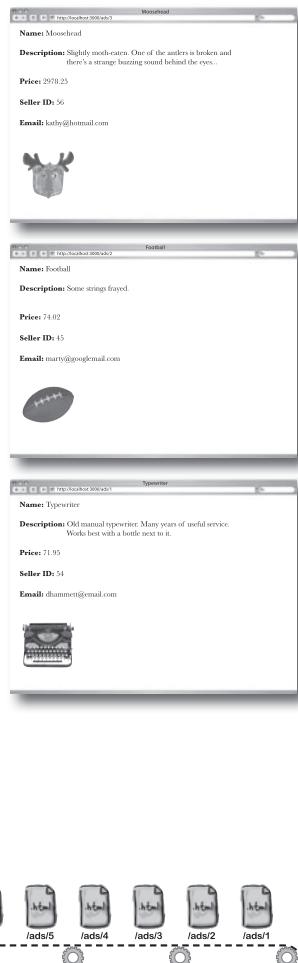
Aplikacja musi robić wiele rzeczy	35
Co jest potrzebne aplikacji?	36
Rails służy do tworzenia aplikacji bazodanowych, takich jak system sprzedaży biletów	38
Nową aplikację tworzy się za pomocą polecenia rails	39
Teraz do domyślnej aplikacji trzeba dodać własny kod	41
Rusztowanie to kod GENEROWANY	42
W bazie danych nie ma jeszcze tabel!	46
Tabelę tworzy się dzięki wykonaniu migracji	47
Pięknie! Uratowałeś pracę kumpla!	51
By zmodyfikować aplikację, musisz przyjrzeć się jej architekturze	52
Trzy części Twojej aplikacji: model, widok i kontroler	53
Cała prawda o Rails	54
Trzy typy kodu przechowywane są w OSOBNYCH folderach	57
Trzeba zmodyfikować pliki WIDOKU	58
Edycja kodu HTML w widoku	59
Aplikacja musi teraz przechować większą liczbę informacji	63
Migracja to po prostu skrypt w języku Ruby	64
Rails może generować migracje	65
Nadaj swojej migracji odpowiednią nazwę, a Rails napisze za Ciebie kod	66
Migrację należy wykonać za pomocą rake	67
Sama zmiana bazy danych nie wystarczy	68
Dlaczego Rails mówi do mnie po angielsku?	75
Uczymy Rails języków obcych	76

Poza rusztowaniem

2

Aplikacje Rails — stworzone, by nimi zarządzać

Co tak naprawdę dzieje się w Rails? Widziałeś już, jak **rusztowania** generują mnóstwo kodu i pomagają pisać aplikacje internetowe w sposób niesamowicie szybki, ale co, jeśli pragniesz czegoś innego? W tym rozdziale zobaczyś, jak można **przejąć kontrolę** nad programowaniem w Rails, i będziesz miał okazję zajrzeć pod maskę tej platformy. Przekonasz się, w jaki sposób Rails decyduje o tym, który **kod** należy wykonać, jak **dane** wczytywane są z bazy danych i jak generowane są **strony internetowe**. Pod koniec rozdziału będziesz w stanie publikować dane tak, jak **sam** zechcesz.



Rusztowanie robi O WIELE za dużo	85
Zaczynamy od wygenerowania modelu MeBay...	86
...a następnie utworzymy tabelę za pomocą polecenia rake	87
Ale co z kontrolerem?	88
Widok tworzony jest przez szablon strony	90
Szablon strony zawiera kod HTML	91
Trasa mówi Rails, gdzie znajduje się strona	93
Widok nie ma danych do wyświetlenia	100
Co zatem powinna pokazywać strona?	101
Kontroler przesyła ogłoszenie do widoku	102
Rails zmienia rekord w obiekcie	104
Dane znajdują się w pamięci, a strona internetowa je widzi	105
Jest problem — ludzie nie potrafią znaleźć żądanych stron	109
Trasy wykonywane są w kolejności	112
By przesyłać dane do widoku, będziesz potrzebował kodu kontrolera	114
Strona indeksująca potrzebuje danych ze WSZYSTKICH rekordów	115
Metoda Ad.find(:all) wczytuje całą tabelę naraz	116
Dane zwracane są jako obiekt zwany tablicą	117
Tablica to ponumerowana sekwencja obiektów	118
Wczytanie wszystkich ogłoszeń za pomocą pętli for	122
Potrzebny nam kod HTML dla każdego elementu tablicy	123
Rails konwertuje szablony stron na kod języka Ruby	124
Pętle można dodawać do szablonów stron za pomocą scriptletów	125
Z każdym przejściem pętli strona generuje jeden odnośnik	126
Jak wygląda wygenerowany kod HTML?	127
Ale my mamy dwa szablony stron... czy powinniśmy zmieniać kod każdego z nich?	130
A co z nową treścią statyczną wysłaną przez MeBay?	133

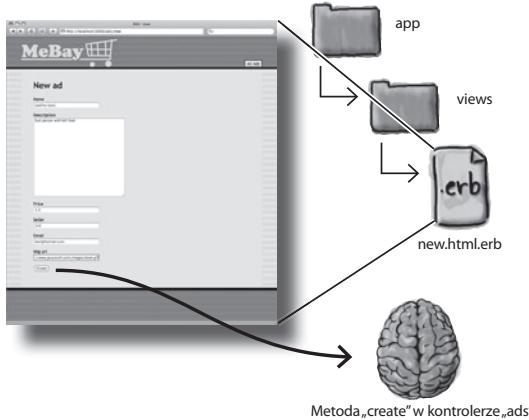
Wstawianie, uaktualnianie i usuwanie

3

Wszystko się zmienia

Zmiana to część życia — szczególnie w przypadku danych. Na razie widziałeś, jak można szybko wyczarować aplikację Rails dzięki rusztowaniu, a także jak napisać własny kod w celu publikacji danych z bazy. Ale co zrobić, kiedy chcemy, by użytkownicy mogli edytować dane w zaplanowany przez nas sposób? Co jeśli rusztowanie nie robi tego, co chcemy *my*? W tym rozdziale nauczysz się **wstawiać, uaktualniać i usuwać** dane dokładnie tak, jak tego chcesz. A przy okazji zobaczyz również, jak tak naprawdę działa Rails, i być może nauczysz się również czegoś o bezpieczeństwie.

Ludzie chcą sami publikować ogłoszenia w Internecie	140
Wiesz już, jak budować aplikację publikującą dane z bazy	141
Zapisywanie danych działa dokładnie ODWROTNIE do ich odczytywania	142
Potrzebny nam formularz służący do dodawania danych oraz metoda akcji zapisująca te dane	143
Czy formularze i obiekty są ze sobą powiązane?	145
Rails może tworzyć formularze powiązane z obiektami modelu	146
Obiekt formularza @ad nie został utworzony	150
Obiekt formularza musi zostać utworzony przed wyświetleniem formularza	151
Obiekt ogłoszenia formularza zostanie utworzony w akcji new kontrolera	152
Każdy szablon strony ma teraz odpowiadającą mu metodę kontrolera	153
Formularz nie odsyła obiektu, odsyła DANE	155
Rails musi przekształcić dane na obiekt przed ich zapisaniem	156
Metoda create kontrolera krok po kroku	157
Kontroler musi zapisać rekord	158
Nie twórz nowej strony, użij istniejącej	164
Jak jednak akcja kontrolera może wyświetlać stronę INNEJ akcji?	165
Przekierowania pozwalają kontrolerowi określić, który widok zostanie wyświetlony	166
Ale co się dzieje, kiedy ogłoszenie należy po opublikowaniu poprawić?	169
Uaktualnienie ogłoszenia przypomina utworzenie go... tylko jest trochę inne	170
Zamiast tworzyć ogłoszenie, musimy je odnaleźć; zamiast je zapisać, musimy je uaktualnić	171
Ograniczanie dostępu do funkcji	178
...teraz jednak stare ogłoszenia trzeba usunąć	181
Wykonanie tego samodzielnie dało Ci możliwość zrobienia więcej, niż potrafi rusztowanie	187



Wyszukiwanie w bazie danych

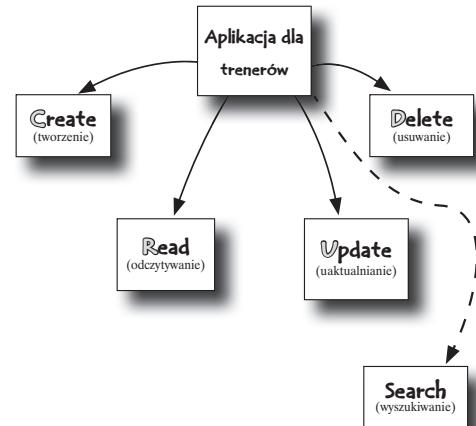
4

Prawda czy konsekwencje?

Każda decyzja ma swoje konsekwencje. W Rails wiedza o tym, jak podejmować dobre decyzje, może zaoszczędzić Ci zarówno czasu, jak i wysiłku. W tym rozdziale przyjrzymy się, jak **wymagania użytkownika** wpływają na wybory, jakich dokonujesz, już od samego początku tworzenia Twojej aplikacji. Czy powinieneś użyć rusztowania, czy lepiej zmodyfikować wygenerowany kod? Czy powinieneś tworzyć wszystko od nowa? Bez względu na wybór, kiedy nadejdzie pora dalszego dostosowania aplikacji do własnych potrzeb, będziesz musiał nauczyć się obsługi **wyszukiwania w bazie danych — dostępu do danych** w sposób, który ma sens zarówno z Twojego punktu widzenia, jak i z punktu widzenia **potrzeb Twoich użytkowników**.

Dbaj o siebie z Rubyville Health Club	190
Aplikacja w zasadzie wygląda dość podobnie...	193
Poprawimy rusztowanie	194
Zaprojektowanie opcji wyszukiwania	195
Zacznijmy od utworzenia formularza	196
Dodanie wyszukiwania do interfejsu	199
Jak możemy znaleźć rekordy klientów?	207
Potrzebne nam jedynie te rekordy, gdzie client_name = łańcuch wyszukiwania	208
Dla każdego atrybutu istnieje metoda wyszukująca	209
Musimy dopasować albo nazwisko klienta, albo trenera	214
Metody wyszukujące piszą zapytania do bazy danych	215
Musimy być w stanie zmodyfikować warunki wykorzystane w zapytaniu SQL	216
Kod SQL podaje się za pomocą :conditions	217

Interes świetnie się kręci,
ale mamy kłopot z prześledzeniem
wszystkich prywatnych zajęć fitness
naszych klientów. Myślisz, że dasz
radę pomóc?



Sprawdzanie poprawności danych

5

Zapobieganie błędom

Każdy popełnia błędy... ale wielu z nich można zapobiec! Nawet przy najlepszych chęciach użytkownicy nadal będą wprowadzać niepoprawne dane do Twojej aplikacji internetowej i to Ty będziesz musiał poradzić sobie z konsekwencjami. Wyobraź sobie, co by było, gdyby istniała jakaś metoda **zapobiegania występowaniu błędów**. Do tego właśnie służą **validatory**. Czytaj dalej, a pokażemy Ci, jak można dodać **sprytnie sprawdzanie błędów w Rails** do Twojej aplikacji internetowej, tak byś mógł **przejąć kontrolę** nad tym, jakie dane są dozwolone, a jakich należy się wystrzegać.

Uwaga — pojawiły się niepoprawne dane	224
Kod sprawdzający poprawność danych przynależy do MODELU	226
Na potrzeby prostego sprawdzania poprawności danych Rails wykorzystuje validatory	227
Jak działają validatory?	228
Sprawdźmy, czy coś jest liczbą	230
Użytkownicy pomijają niektóre pola formularzy	232
Jak sprawdzamy obowiązkowe pola?	233
Validatory są proste i działają dobrze	236
W MeBay wydarzyło się coś dziwnego	239
Validatory sprawdzają, jednak nie wyświetlają błędów	240
Jeśli tworzysz własne strony, musisz także pisać własny kod komunikatów o błędach	243
Kontroler musi wiedzieć, czy wystąpił błąd	244
Nadal musimy wyświetlić komunikaty o błędach!	248
System MeBay wygląda przepięknie	250



Tworzenie połączeń

6

Łączenie wszystkiego razem

Niektóre rzeczy lepsze są razem niż osobno. Posmakowałeś zatem niektórych kluczowych składników Rails. Tworzyłeś całe aplikacje internetowe, a także brałeś to, co wygenerowała platforma Rails, i przystosowywałeś do swoich potrzeb. W prawdziwym świecie życie może jednak być bardziej skomplikowane. Czytaj dalej... czas zacząć budować wielofunkcyjne strony internetowe! I nie tylko to — czas zacząć sobie radzić ze skomplikowanymi powiązaniami między danymi, a także przejąć kontrolę nad danymi, pisząc własne walidatory.



Stary... Zarezerwowałem lot na imprezę na plaży, ale wylądowałem na historycznej wyprawie do starej kolonii tredowatych!



Linie Coconut Airways potrzebują nowego systemu rezerwacji	256
Chcemy widzieć loty i rezerwacje miejsc razem	258
Zobaczmy, co daje nam rusztowanie dla miejsc	259
Na stronie lotu musi się znaleźć formularz rezerwacji oraz lista miejsc	260
Jak możemy podzielić zawartość strony na odrębne pliki?	261
ERb SKŁADA nasze strony	265
Jak można utworzyć szablon częściowy formularza rezerwacji?	266
Teraz musimy dołączyć szablon częściowy do szablonu strony	267
Musimy przekazać szablonowi częściowemu miejsce!	270
Zmienne lokalne można przekazywać do szablonu częściowego	271
Niezbędny jest nam szablon częściowy dla listy miejsc	278
Ludzie trafiają na niewłaściwe loty	280
Powiązanie łączy ze sobą modele	281
Jak jednak definiujemy powiązanie?	283
Niektóre osoby mają jednak za duży bagaż	285
Musimy napisać WŁASNY validator	286
Potrzebne nam jest ODWROTNE powiązanie	289
System wystartował w Coconut Airways	296

Ajax

7

Ograniczanie ruchu

Każdy chce uzyskać z życia jak najwięcej... podobnie z aplikacją. Bez względu na to, jak jesteś dobry w obsłudze Rails, czasami tradycyjne aplikacje internetowe sobie nie radzą. Bywa, że użytkownicy pragną czegoś bardziej **dynamicznego**, czegoś, co odpowiada na wszystkie ich kaprysy. Ajax pozwala tworzyć **szynkne aplikacje internetowe z doskonałym czasem reakcji**, zaprojektowane tak, by użytkownik mógł **czerpać z Internetu jak najwięcej**. Rails ma wbudowany własny zestaw bibliotek Ajaksa, które tylko czekają na to, aż ich użyjesz! Pora **szynko i łatwo dodać do aplikacji fantastyczne możliwości oferowane przez technologię Ajax** i zachwycić jeszcze większą liczbę użytkowników.

Linie Coconut Airways mają nową ofertę	300
Które części strony najbardziej się zmieniają?	301
Czy przeglądarka nie uaktualnia zawsze całej strony?	306
Co INNEGO może wykonać żądanie?	307
Najpierw musimy dołączyć biblioteki Ajaksa...	308
...a następnie dodać odnośnik „Odśwież” oparty na Ajaksie	309
Przeglądarka musi prosić o uaktualnienie	314
Czy jednak POWINNIŚMY nakazywać przeglądarkę nieustanne proszenie?	315
Licznik obsługuje się podobnie jak przycisk czy odnośnik	316
Cała prawda o Ajaksie	320
Ktoś ma kłopot ze swoim wieczorem kawalerskim	321
Formularz musi wykonać żądanie oparte na Ajaksie	322
Formularz musi pozostawać pod KONTROLĄ JavaScriptu	323
Musimy zastąpić metodę create	325
Jaki efekt ma ten kod?	326
Teraz pojawił się problem z rezerwacjami lotów	331
Potrafimy uaktualnić jedną część strony naraz	332
Kontroler musi zamiast HTML zwracać kod w JavaScriptie	333
Co generuje Rails?	337
Jeśli nie powiesz, gdzie umieścić odpowiedź, zostanie ona wykonana	338



XML i różne reprezentacje

8

Wszystko wygląda teraz inaczej...

Nie da się zawsze wszystkich zadowolić. A może jednak? Dotychczas widzieliśmy, jak można wykorzystać Rails do szybkiego i łatwego tworzenia aplikacji internetowych, które idealnie pasują do pewnego zbioru wymagań. Co jednak zrobić, kiedy pojawiają się inne wymagania? Co powinniśmy zrobić, jeśli niektóre osoby chcą otrzymać proste strony internetowe, inne interesuje mashup z aplikacji firmy Google, a jeszcze inne chcą, aby aplikacja była dostępna w czytniku kanałów RSS? W tym rozdziale będziemy tworzyć różne reprezentacje tych samych danych, co da nam maksymalną elastyczność przy minimalnym wysiłku.



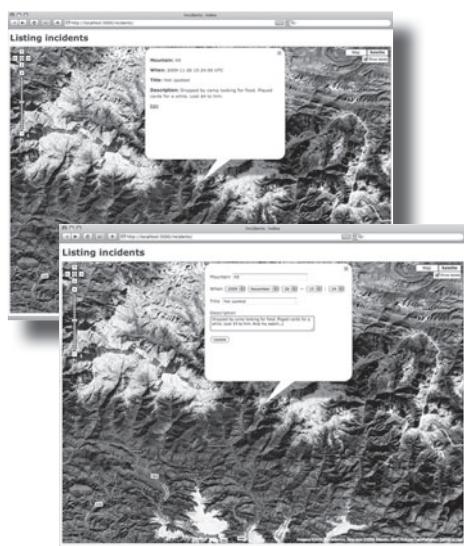
Zdobywanie szczytów świata	344
Użytkownicy nienawidzą interfejsu aplikacji!	345
Dane muszą się znaleźć na mapie	346
Musimy utworzyć nową akcję	347
Nowa akcja wydaje się działać...	348
Nowa strona potrzebuje mapy... w tym właśnie rzecz!	349
Jakiego typu kod jest nam potrzebny?	350
Kod ten działa jedynie dla serwera lokalnego	351
Teraz potrzebne nam dane mapy	352
Co zatem powinniśmy wygenerować?	354
Wygenerujemy kod XML z modelu	355
Obiekt modelu może generować kod XML	356
Jak powinien wyglądać taki kod kontrolera?	357
Tymczasem na wysokości kilku tysięcy metrów...	362
Musimy generować XML oraz HTML	363
XML i HTML to po prostu reprezentacje	365
W jaki sposób powinniśmy decydować, z którego formatu skorzystać?	366
Jak działa strona z mapą?	370
Kod jest gotowy do opublikowania	372
Kanały RSS to po prostu kod XML	380
Utworzmy akcję o nazwie news	381
Musimy zmienić strukturę kodu XML	384
Użyjemy nowego typu szablonu — XML Builder	385
Teraz dodajmy kanały RSS do stron	389
Zdobyleś szczyt!	391

Architektura REST i Ajax

9

Kolejne kroki

Czas skonsolidować umiejętności w zakresie korzystania z aplikacji typu mashup. Dotychczas widzieliśmy, jak w celu pokazania danych geograficznych można dodać do naszych aplikacji mapy z serwisu **Google Maps**. Co jednak, jeśli chcemy **rozszerzyć istniejącą już funkcjonalność?** Czytaj dalej, a przekonasz się, jak można wzbogacić aplikacje typu **mashup o bardziej zaawansowane cudeńka oparte na Ajaksie**. Co więcej, przy okazji nauczysz się też nieco o architekturze **REST**.



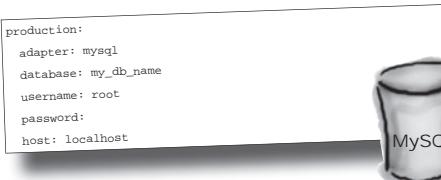
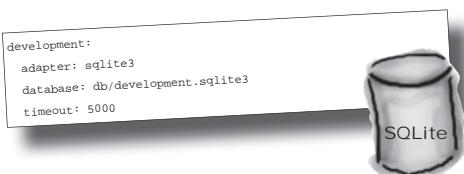
Zdarzeń jest zbyt dużo!	394
Mapa mogłaby pokazywać więcej szczegółów	395
Moglibyśmy rozszerzyć funkcjonalność mapy za pomocą Ajksa	396
Jak jednak możemy przekształcić stronę indeksującą?	397
Co będzie musiała wygenerować akcja show?	398
Nowa funkcjonalność mapy jest pełnym sukcesem!	403
Musimy utworzyć żądania wykorzystujące Ajksa	404
Szablon częściowy mapy pozwala nam wybrać akcję new	406
Jak możemy UDOWODNIĆ, że zdarzenie zostało zapisane?	411
Formularz musi uaktualnić zawartość elementu <div> wyskakującego okna	412
Lawina!	417
Jak działa to teraz...	418
Moglibyśmy umieścić odnośnik „Edit” w oknie wyskakującym	419
Zaczniemy od zmodyfikowania akcji edit	420
Na stronie show potrzebny nam jest także nowy odnośnik	422
Jak stosuje się metodę pomocniczą link_to?	423
Na pomoc spieszysy odnośnik oparty na Ajksie	427
Używamy niewłaściwej trasy!	429
Na wybór trasy ma wpływ metoda HTTP	430
Czym jest zatem metoda HTTP?	431
Witryna Head First Climbers Cię potrzebuje!	434

10

Prawdziwe aplikacje

Rails w świecie rzeczywistym

Nauczyłeś się już wiele o Ruby on Rails. By jednak zastosować tę wiedzę w prawdziwym świecie, będziesz musiał zastanowić się nad kilkoma sprawami. W jaki sposób połączyć aplikację z inną bazą danych? Jak testuje się aplikacje Rails? Jak można wydobyć maksimum możliwości z Rails oraz języka Ruby? I skąd można dowiedzieć się o najświeższych nowościach w świecie Rails? Czytaj dalej, a pokażemy Ci kierunek, dzięki któremu jeszcze bardziej rozwiniiesz swoje umiejętności programistyczne.



S

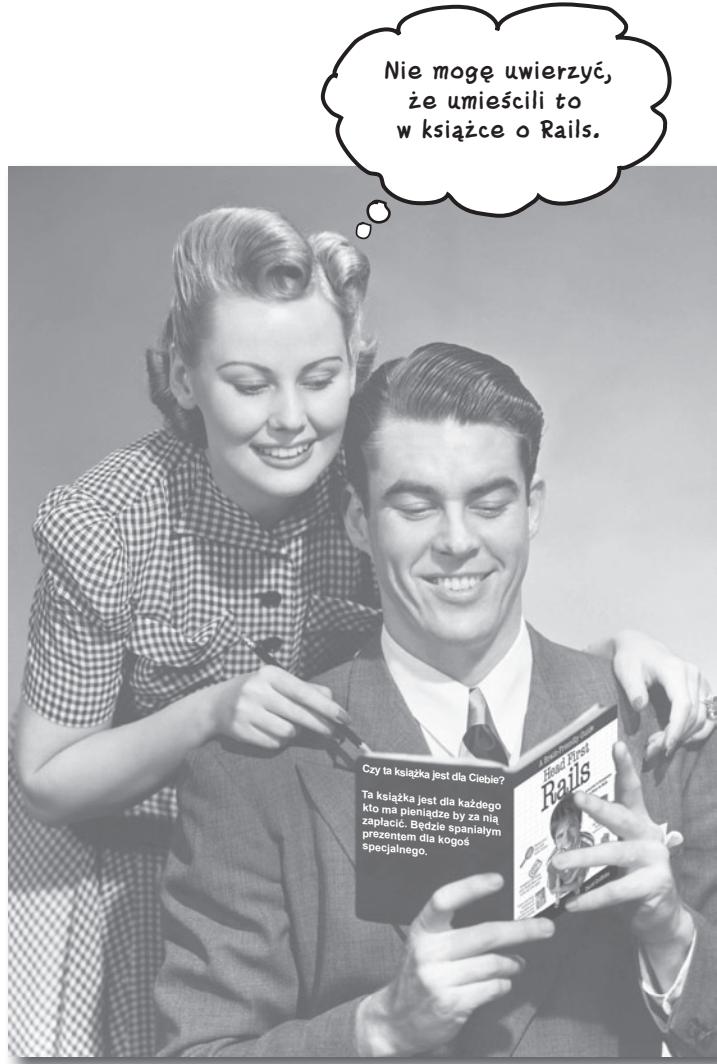
Patrz! Eksperymenty z językiem Ruby!	441
Aplikacje internetowe muszą być testowane	442
Jakie rodzaje testów są dostępne?	443
Udostępnienie aplikacji użytkownikom	444
Jak zmienia się bazę danych?	445
Czym jest architektura REST?	446
Aplikacje internetowe pobłędziły	447
Życie na krawędzi	448
Uzyskanie dodatkowych informacji	449
Nieco dodatkowej lektury...	450
Książki Head First o podobnej tematyce	451
Koniec wycieczki...	453

Skorowidz

455

Jak korzystać z tej książki

Wprowadzenie



W tym rozdziale odpowiem na palcze pytanie:
„A więc dlaczego UMIEŚCILI to w książce o Rails?“.

Dla kogo przeznaczona jest ta książka?

Jeśli możesz odpowiedzieć twierdząco na wszystkie z poniższych pytań:

- 1 Czy swobodnie posługujesz się językiem **HTML**?
- 2 Czy masz jakieś doświadczenie z językiem programowania, takim jak **Java**, **C#** czy **PHP**?
- 3 Czy chcesz tworzyć **fantastyczne aplikacje** internetowe w **ułamku czasu**, jaki zajmowało Ci to wcześniej?

ta książka przeznaczona jest właśnie dla Ciebie.

Kto prawdopodobnie powinien odłożyć tę książkę?

Jeśli możesz odpowiedzieć twierdząco na którykolwiek z poniższych pytań:

- 1 Czy jesteś osobą, która **nie ma żadnego doświadczenia w pracy z językiem HTML**?

- 2 Czy jesteś **doświadczonym programistą Rails, który szuka kompletnego podręcznika**?
- 3 Czy **boisz się spróbować czegoś nowego**?
Czy przedzej zdecydowałbyś się na leczenie kanałowe zęba, niż pomieszał paski i kropki w ubiorze? Czy uważasz, że książka techniczna nie może być poważna, jeśli dokonuje antropomorfizacji klientów i serwerów?

ta książka nie jest przeznaczona dla Ciebie.



W tym przypadku nie martw się.
Siegnij najpierw po książkę „Head First HTML with CSS & XHTML. Edycja polska” autorstwa Elisabeth Freeman oraz Erica Freeman, a później wróć do tej książki.

[Uwaga od działu marketingu: ta książka przeznaczona jest dla każdego posiadacza karty kredytowej.]

Wiemy, co sobie myślisz

„Czy to w ogóle może być poważna książka o Rails?”

„O co chodzi z tą całą grafiką?”

„Czy w ogóle mogę się czegokolwiek w ten sposób *nauczyć*? ”

Wiemy, co myśli Twój mózg

Twój mózg łaknie nowości. Ciągle poszukuje, wypatruje i *oczekuje* czegoś niezwykłego. Został zbudowany w ten sposób i dzięki temu pomaga Ci przeżyć.

Co zatem Twój mózg robi ze wszystkimi zwykłymi, rutynowymi, normalnymi rzeczami, z jakimi się spotykasz? Robi wszystko, co może, by zapobiec przeszkadzaniu mu w tym, co jest jego *prawdziwą* pracą — czyli odnotowywaniu rzeczy *istotnych*. Nie kłopocze się zapamiętywaniem wszystkich nudnych rzeczy, gdyż one nigdy nie przejdą przez jego filtr rzeczy „w oczywisty sposób nieistotnych”.

A skąd mózg *wie*, co jest ważne? Powiedzmy, że idziesz sobie na spacer i nagle przed Tobą pojawia się tygrys. Co się dzieje wewnętrz Twojej głowy i ciała?

Rozbudzają się neurony. Wybuchają emocje. *Pobudzone zostają neuroprzekaźniki.*

I w ten sposób Twój mózg wie...

To musi być ważne! Nie zapomnij o tym!

Wyobraź sobie jednak, że jesteś w domu czy w bibliotece. To strefa bezpieczna, ciepła, wolna od tygrysów. Uczysz się. Przygotowujesz się do egzaminu. Albo próbujesz opanować jakieś skomplikowane zagadnienie, co wedle Twojego szefa powinno Ci zająć tydzień, maksymalnie dziesięć dni.

I jest tylko jeden problem. Twój mózg stara się wywiadczyć Ci wielką przysługę. Stara się zapewnić, by te w oczywisty sposób nieistotne treści nie zaśmieciły i tak już wypełnionej przestrzeni. Przestrzeni, którą wykorzystasz o wiele lepiej, przechowując w niej naprawdę *ważne* rzeczy. Takie jak tygrysy. Albo jak niebezpieczeństwo pożaru. Albo nazwiska zwycięzców trzech ostatnich edycji „Idola”. I nie da się po prostu powiedzieć swojemu mózgowi: „Ej, ty, dziękuję bardzo, ale bez względu na to, jak nudna nie byłaby ta książka i jak niski jest teraz mój poziom emocji według skali Richtera, naprawdę *chcę* zapamiętać te rzeczy”.

Twój mózg uważa, że TO jest ważne.



Czytelnika serii „Head First” traktujemy jak ucznia.

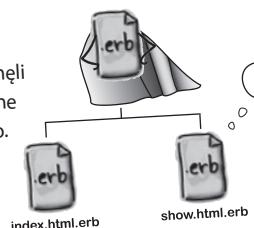
Co zatem trzeba zrobić, by się czegoś nauczyć? Po pierwsze, musisz się czegoś **dowiedzieć**, a później tego nie **zapomnieć**. Nie chodzi o wbijanie Ci różnych faktów do głowy. Wedle najnowszych odkryć w dziedzinie nauki poznania, neurobiologii i psychologii edukacyjnej do uczenia się potrzeba czegoś więcej niż tylko tekstu umieszczonego na stronie. Wiemy, co stymuluje Twój mózg.

Oto niektóre z zasad uczenia się w serii Head First:



Pomoce wizualne. Obrazy o wiele łatwiej jest zapamiętać niż same słowa; sprawiają one, że proces uczenia się staje się o wiele bardziej wydajny (w studiach przypominania i transferu zanotowano poprawę o 89%). Ułatwiają one również rozumienie wielu zagadnień. Wystarczy **umieścić słowa wewnątrz czy obok grafiki**, do której się odnoszą (a nie pod nią czy na innej stronie), a istnieje dwa razy większa szansa, że uczący się rozwiąże problem powiązany z tym zagadnieniem.

Konwersacyjny i spersonalizowany styl. W ostatnich badaniach studenci osiągnęli o 40% lepszy wynik testów sprawdzających uczenie się, kiedy informacje były im podawane z wykorzystaniem zwrotów bezpośrednich i stylu konwersacyjnego w miejsce formalnego. Należy opowiadać historie, zamiast wykładać. Używać języka codziennego. Nie można podchodzić do wszystkiego zbyt poważnie. Na kogo zwróciłeś większość uwagi: na interesującego towarzysza imprezy czy wykładowcę?



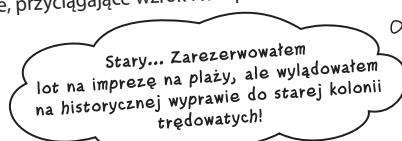
Uczący się powinien być zmuszony do głębszych przemyśleń. Innymi słowy: jeśli nie będziesz aktywnie ćwiczył neuronów, w Twojej głowie nic się nie będzie działało. Czytelnik musi być zmotywowany, zaangażowany, ciekawy, pełen chęci do rozwiązywania problemów, wyciągania wniosków i generowania nowej wiedzy. Z tego powodu konieczne są wyzwania, ćwiczenia i zmuszające do myślenia pytania, a także działania angażujące obie półkule mózgowe oraz różne zmysły.



Jazda próbna

Przyciąganie — i utrzymanie — uwagi czytelnika. Każdy z nas ma za sobą doświadczenie, które można podsumować jako: „Naprawdę chciałbym się tego nauczyć, ale trudno mi nie zasnąć na drugiej stronie”. Twój mózg zwraca uwagę na rzeczy niezwykłe, interesujące, dziwne, przyciągające wzrok i niespodziewane. Opanowanie nowego, trudnego, technicznego zagadnienia wcale nie musi być nudne. Twój mózg będzie uczył się o wiele lepiej, jeśli tak nie będzie.

Dotknietcie emocji. Obecnie wiemy już, że umiejętność uczenia się w dużej mierze uzależniona jest od ładunku emocjonalnego przekazywanych informacji. Pamiętamy to, co ma dla nas znaczenie. Pamiętamy coś, kiedy coś czujemy. Nie, nie mówimy tutaj o chwytyjących za serce opowieściach o chłopcu i jego piesku. Mówimy o emocjach takich, jak zaskoczenie, ciekawość, rozbawienie, „ale o co tutaj chodzi?” i uczucie satysfakcji przychodzące po rozwiązaniu zagadki, nauczeniu się czegoś, co inni uważają za trudne, czy uświadomieniu sobie, że wiemy coś, czego ten przemądrzały Robert „Wiem O Wiele Więcej Od Ciebie” z sąsiadniego działu wcale *nie wie*.



Metapoznanie — myślenie o myśleniu

Jeśli naprawdę chcesz się nauczyć i do tego chcesz się nauczyć szybciej i w bardziej pogłębiony sposób, zwróć uwagę na to, jak zwracasz uwagę. Pomyśl o tym, jak myślisz. Naucz się tego, jak się uczysz.

Większość z nas nie uczęszczała w młodości na kursy dotyczące metapoznania czy teorii uczenia się. *Oczekiwano* od nas, że będziemy się uczyć, ale rzadko *uczono*, jak to robić.

Zakładamy jednak, że jeśli trzymasz tę książkę w dłoniach, naprawdę chcesz nauczyć się Rails. I najprawdopodobniej nie chcesz spędzić na tym zbyt wiele czasu. Jeśli chcesz skorzystać z tego, co czytasz w tej książce, musisz *zapamiętać*, o czym czytasz. A żeby to zrobić, musisz to *zrozumieć*.

By wydobyć maksimum z tej książki, czy też *dowolnej* książki lub procesu uczenia się, musisz wziąć odpowiedzialność za swój mózg. Twój mózg i *te* informacje.

Sztuczka polega na sprawieniu, by Twój mózg uznał nowy materiał, którego się uczysz, za Naprawdę Ważny. Niezbędny do Twojego przeżycia. Tak ważny jak tygrys. W przeciwnym razie będziesz stale walczył ze swoim mózgiem, który będzie robił, co tylko potrafi, byś nie zapamiętał nowych informacji.

A więc JAK sprawić, by Twój mózg zaczął traktować Rails tak jak głodnego tygrysa?

Istnieje powolna i żmudna droga albo droga szybsza i bardziej efektywna. Powolna droga polega na powtarzaniu. Wiesz oczywiście, że jesteś w stanie nauczyć się i zapamiętać nawet najnudniejsze rzeczy, jeśli ciągle będziesz sobie wbijał do głowy jedną i tę samą informację. Po wystarczającej liczbie powtórzeń Twój mózg mówi: „*Nie wydaje* mi się, by było to dla niego ważne, ale on *ciągle i ciągle, i ciągle* patrzy na tę samą rzecz, więc pewnie jest ona jednak ważna”.

Szybszym sposobem jest robienie *czegokolwiek, co zwiększa aktywność mózgu* — w szczególności różnego typu aktywność mózgu. Kwestie wymienione na poprzedniej stronie to właśnie element tego rozwiązania. Wszystko to są rzeczy, o których dowiedziono, że wspomagają korzystną dla nas pracę mózgu. Przykładowo badania pokazują, że umieszczenie słów w *obrębie* opisywanych przez nie obrazków (w przeciwieństwie do umieszczania ich w innym miejscu strony, na przykład w podpisie obrazka czy zwykłym tekście) zmusza mózg do próby zrozumienia, jakie są powiązania pomiędzy słowami i obrazkiem, co powoduje pobudzenie większej liczby neuronów. Większa liczba neuronów zostaje pobudzona = większe szanse na to, że nasz mózg *zrozumie*, iż jest to kwestia, na którą warto zwrócić uwagę i być może zapamiętać.

Styl konwersacyjny pomaga, ponieważ ludzie zazwyczaj zwracają większą uwagę, kiedy wydaje im się, że są uczestnikami rozmowy, gdyż oczekuje się od nich wtedy, że będą nadążali za informacjami i odpowiednio odpowiadali. Co jednak niesamowite, dla Twojego mózgu nie ma znaczenia, że „konwersacja” odbywa się pomiędzy Tobą a książką. Z drugiej strony jeśli książka pisana jest suchym, formalnym stylem, Twój mózg postrzega ją podobnie jak siedzenie na wykładzie w sali pełnej biernych słuchaczy. Można sobie wtedy pozwolić na drzemkę. Obrazki i styl konwersacyjny to jednak tylko początek...



Jak korzystać z tej książki

Oto, co zrobiliśmy:

Użyliśmy **obrazków**, ponieważ Twój mózg jest nastawiony na elementy wizualne, nie tekst. A jeśli chodzi o mózg, obraz *naprawdę* wart jest tyle co tysiąc słów.

Kiedy tekst i obrazki współdziałają ze sobą, osadzamy tekst w obrazkach, ponieważ mózg działa bardziej wydajnie, gdy tekst umieszczony jest *wewnątrz* elementu, do którego się odnosi, a nie znajduje się w podpisie czy też gdzieś na stronie.

Używamy **powtórzeń**, przekazując tę samą kwestię na *różne* sposoby i za pomocą różnych typów mediów, odnosząc się do *różnych* zmysłów, w celu zwiększenia szansy na to, że treść zostanie zakodowania w więcej niż jednym obszarze mózgu.

Wykorzystujemy ilustracje i obrazki w **zaskakujące** sposoby, ponieważ mózg zwraca uwagę na nowości. Wykorzystujemy obrazki i pomysły zawierające *jakiś komponent emocjonalny*, ponieważ mózg zwraca uwagę na biochemię emocji. To, co powoduje, że coś czujesz, łatwiej będzie Ci zapamiętać — nawet jeśli to uczucie to nic ponad **zaskoczenie, rozbawienie czy ciekawość**.

Wykorzystujemy spersonalizowany, **konwersacyjny styl**, ponieważ Twój mózg zwraca większą uwagę wtedy, gdy sądzi, że uczestniczysz w rozmowie, niż wtedy, gdy biernie słuchasz jakiejś prezentacji. Twój mózg zachowuje się tak nawet wtedy, gdy *czytasz*.

W książce znajduje się ponad 80 **ćwiczeń**, ponieważ Twój mózg lepiej się uczy i więcej pamięta, kiedy **robisz** rzeczy, a nie tylko o nich *czytasz*. Ćwiczenia te są wymagające, ale da się je wykonać, ponieważ właśnie taki poziom trudności preferuje większość osób.

Wykorzystujemy **różne style nauki**, ponieważ kiedy Ty możesz woleć procedury omawiane krok po kroku, ktoś inny woli najpierw spojrzeć z szerszej perspektywy, a jeszcze ktoś chce po prostu zobaczyć przykład. Bez względu na preferencje w zakresie uczenia się *każdy* skorzysta na zobaczeniu tej samej treści przedstawionej na różne sposoby.

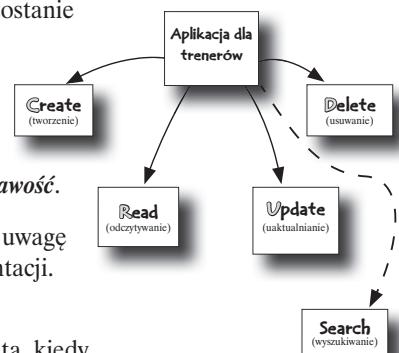
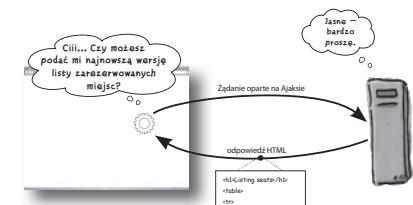
Zamieszczamy treść przeznaczoną dla **obu półkul mózgowych**, ponieważ im większa część Twojego mózgu zostanie zaangażowana, tym większa szansa, że nauczysz się czegoś i zapamiętasz, a także dłużej pozostaniesz skupiony. Ponieważ praca z jedną półkulą mózgu oznacza często, że druga ma szansę odpocząć, będziesz mógł uczyć się wydajniej i przez dłuższy czas.

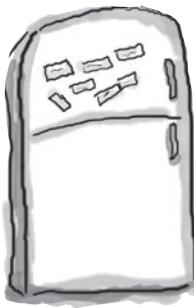
Umieszczamy w książce **historie** i ćwiczenia prezentujące *więcej niż jeden punkt widzenia*, ponieważ mózg uczy się w sposób pogłębiony, kiedy zmusi się go do ocen i sądów.

Zamieszczamy **wyzwania** — ćwiczenia oraz **pytania**, na które nie zawsze istnieje prosta odpowiedź, ponieważ Twój mózg uczy się i zapamiętuje, kiedy musi nad czymś *popracować*. Pomyśl o tym — nie można przecież wyrzebić własnego *ciała, patrząc* jedynie na inne osoby ćwiczące na siłowni. Dołożyliśmy szczególnych starań, by zapewnić, że kiedy ciężko pracujesz, pracujesz nad *właściwymi rzeczami*.

Że nie poświęcasz *jednego dodatkowego dendrytu* na przetworzenie trudnego do zrozumienia przykładu, czy próbując się przegryźć przez przeładowany żargonem lub przesadnie lakoniczny tekst.

Wykorzystujemy **ludzi**. W historiach, przykładach, obrazkach, ponieważ — no cóż... — ponieważ *Ty sam* jesteś człowiekiem. A Twój mózg zwraca większą uwagę na *ludzi* niż na *rzeczy*.





Wytnij to i przyklej na lodówce.

Oto, co możesz zrobić, by skłonić swój mózg do posłuszeństwa

Zrobiliśmy naszą część roboty. Reszta zależy od Ciebie. Poniższe wskazówki to tylko początek. Wsłuchaj się w swój mózg i sprawdź, co jest w Twoim przypadku skuteczne, a co nie. Próbuj nowych rzeczy.

1 Zwolnij. Im więcej zrozumiesz, tym mniej będziesz musiał nauczyć się na pamięć.

Nie ograniczaj się tylko do *czytania*. Zatrzymaj się i pomyśl. Kiedy w książce zadane jest pytanie, nie przeskakuj od razu do odpowiedzi. Wyobraź sobie, że ktoś naprawdę *zadaje* Ci to pytanie. W im bardziej pogłębiony sposób będzie pracował Twój mózg, tym większa szansa, że się czegoś nauczysz i coś zapamiętasz.

2 Wykonuj ćwiczenia. Rób własne notatki.

Umieściliśmy je tu dla Ciebie, ale gdybyśmy od razu je wykonali, przypominałoby to poproszenie kogoś, by wykonał za Ciebie pompuki. Na ćwiczenia nie możesz tylko *patrzeć*.

Zlap za ółówek. Istnieje mnóstwo dowodów na to, że aktywność fizyczna w *trakcie* uczenia się może poprawić skuteczność tego procesu.

3 Czytaj części „Nie istnieją głupie pytania”.

Czytaj je wszystkie. Nie są to opcjonalne wstawki z dodatkowymi informacjami, *to część podstawowej treści książki!* Nie pomijaj ich.

4 Niech książka ta będzie ostatnią rzeczą, jaką czytasz przed pójściem spać. A przynajmniej ostatnią rzeczą wymagającą wysiłku.

Część procesu uczenia się (w szczególności transfer do pamięci długotrwałej) ma miejsce już *po* odłożeniu książki. Twój mózg potrzebuje trochę czasu dla siebie, by móc przetworzyć informacje. Jeśli w czasie tego przetwarzania przekażesz mu nowe informacje, część z tego, czego właśnie się nauczyłeś, będzie stracona.

5 Mów o tym, co czytasz. Na głos.

Mówienie pobudza inną część mózgu. Jeśli próbujesz coś zrozumieć albo zwiększyć szansę na późniejsze pamiętanie czegoś, powiedz to na głos. A jeszcze lepiej: spróbuj to na głos komuś wytłumaczyć. Będziesz się szybciej uczył i być może odkryjesz informacje, o których nie wiedziałeś, kiedy czytałeś o nich w książce.

6 Pij wodę. Dużo wody.

Twój mózg działa najlepiej, kiedy otoczony jest dużą ilością płynów. Odwodnienie (które ma miejsce, zanim poczujesz się spragniony) zmniejsza funkcje poznawcze.

7 Słuchaj swojego mózgu.

Zwracaj uwagę na to, czy nie staje się przeładowany. Jeśli zauważysz, że zaczynasz pomijać niektóre informacje albo zapominasz, co właśnie przeczytałeś, czas na przerwę. Kiedy przekroczyłeś pewien punkt, nie będziesz się szybciej uczył, próbując wbić sobie do głowy kolejne informacje, i możesz w ten sposób zaszkodzić całemu procesowi nauki.

8 Wczuj się..

Twój mózg musi wiedzieć, że to, co robisz, *ma znaczenie*. Zaangażuj się w historie. Wymyśl własne podpisy do zdjęć. Chichotanie z nieszczególnie śmiesznego dowcipu jest *i tak* lepsze od nieodczuwania niczego.

9 Ćwicz pisanie aplikacji Rails!

Istnieje tylko jedna droga do prawdziwego opanowania programowania w Rails: **programowanie aplikacji w Rails**. I właśnie to będziesz robił w tej książce. Najlepszym sposobem na zrozumienie zagadnienia jest **wykonanie go**. Aktywność wzmacnia naturalne ścieżki neuronowe, dlatego damy Ci **wiele** możliwości praktycznego wypróbowania nowych umiejętności — każdy rozdział zawiera aplikacje, które będziemy tworzyć. Nie pomijaj ich zatem — duża część procesu uczenia się ma miejsce, kiedy samodzielnie budujesz te aplikacje. I nie martw się, jeśli popełniasz błędy. Tak naprawdę Twój mózg szybciej uczy się na błędach niż na sukcesach. Wreszcie pamiętaj, by upewnić się, że zrozumiesz, co się dzieje, zanim przejdziesz do kolejnej części książki. Każdy rozdział rozwija zagadnienia omówione wcześniej.

Jak korzystać z tej książki

Przeczytaj to

Niniejsza książka służy do nauki, nie jest materiałem źródłowym. Celowo odarliśmy ją ze wszystkiego, co mogłoby zakłócić proces uczenia się tego, nad czym pracujemy w danym miejscu książki.

I przy pierwszej lekturze musisz zacząć od początku, ponieważ książka ta czyni pewne założenia dotyczące tego, co już widziałeś i czego się nauczyłeś.

Zanim zaczniesz pracę z książką, musisz zainstalować Ruby on Rails na swoim komputerze.

Nie jest to książka z instrukcjami, dlatego nie ma w niej rozdziałów wskazujących, jak zainstalować Ruby on Rails na komputerze. Lepiej będzie poszukać tego rodzaju informacji w Internecie. Niezbędne będzie zainstalowanie Ruby on Rails w wersji 2.1 lub wyższej, a także bazy danych SQLite3. Można je znaleźć na stronie:

<http://www.rubyonrails.org/down>

Książka ta nie jest przewodnikiem encyklopedycznym.

Nie oczekuj, że zobaczyś w niej długie strony opisujące piętnaście sposobów zrobienia czegoś. Chcemy, byś zrozumiał wszystko dzięki **wykonywaniu**, dlatego od samego początku podajemy tylko tyle informacji, by posunąć Twój proces uczenia do przodu. Po skończeniu książki będziesz miał w głowie mentalny obraz tego, jak działa platforma Rails i jakie ma możliwości. Wtedy będziesz mógł wbić sobie do głowy materiały encyklopedyczne w sposób o wiele szybszy i bardziej znaczący, niż byłbyś to w stanie zrobić wcześniej. Psychologowie nazywają tę umiejętność **dzieleniem informacji na części**.

Wszystkie kody z niniejszej książki dostępne są na stronie internetowej.

W miarę postępowania dalej pokażemy Ci wszystkie kody, jakich będziesz potrzebował.

Programowanie obok czytania książki jest **dobrym pomysłem**; **świętym pomysłem** jest bawienie się kodem i realizowanie za jego pomocą swoich własnych pomysłów. Czasami możesz jednak chcieć skopiować kod wykorzystywany w każdym rozdziale, dlatego udostępniliśmy go na stronie internetowej książki. Aplikacje Rails są dość samodzielne, dlatego nie ma powodu, byś nie mógł mieć dostępu do kodu, który robi to, *co mówi książka*, obok swojej własnej podrasowanej wersji. Kody można pobrać ze strony internetowej:

<ftp://ftp.helion.pl/przyklady/hfror.zip>

Nie objaśniamy w pełni każdego fragmentu kodu.

Rails może wygenerować dla Ciebie mnóstwo kodu i nie chcemy, byś ugrzązł w opisach wiersza po wierszu. Opiszemy istotne fragmenty, które powinieneś znać, a później przejdziemy dalej.

Nie martw się — do końca książki wszystkie części złożą się w jedną całość.

To książka o Rails, a nie o języku Ruby.

Ruby to język, w którym napisana jest platforma Rails, i po drodze nauczyszmy Cię wystarczająco dużo o tym języku. Nie martw się — jeśli masz doświadczenie z innym językiem programowania, takim jak C# czy Java, świetnie sobie poradzisz. Rails to system o ogromnych możliwościach, w którym możesz osiągnąć wiele, mając ograniczoną znajomość języka Ruby.

Ćwiczenia NIE są opcjonalne.

Ćwiczenia i inne typy aktywności nie są dodatkiem. Stanowią część podstawowej zawartości książki. Niektóre z nich pomagają w zapamiętywaniu, inne w zrozumieniu, a jeszcze inne pomogą Ci zastosować to, czego się nauczyłeś. Nie pomijaj ćwiczeń.

Powtórzenia są celowe i ważne.

Jedną z cech wyróżniających książkę z serii *Head First* jest to, że chcemy, byś naprawdę się czegoś nauczył. Chcemy też, byś po skończeniu książki pamiętała, czego się nauczyłeś. Większość książek typu encyklopedycznego nie ma na celu zapamiętania i utrwalenia wiedzy, jednak celem tej książki jest nauczenie, dlatego zobaczysz, że niektóre zagadnienia pojawią się w niej więcej niż raz.

Nie zawsze będziemy pokazywać cały kod.

Czytelnicy mówią nam, że przebijanie się przez dziesięć niewiele się od siebie różniących wersji tego samego fragmentu kodu jest frustrujące, dlatego czasami pokażemy jedynie te części skryptu, które się zmieniły.

Rozdziały bazują na umiejętnościach, a nie technologiach.

Każdy rozdział dostarczy Ci umiejętności pozwalających na pisanie bardziej **zaawansowanych i wartościowych** aplikacji. Nie ma tu zatem osobnych rozdziałów traktujących o współpracy z bazami danych czy projektowaniu ładnych interfejsów. Zamiast tego każdy rozdział nauczy Cię trochę o bazie danych, trochę o interfejsie i trochę o kilku innych częściach Rails. Pod koniec każdego z nich będziesz w stanie powiedzieć: „Super — teraz potrafię budować aplikacje, które umieją X”.

Zespół korektorów merytorycznych

Andrew Bryan



Jeremy Durham



Matt Harrington



Mike Isman



LuAnn Mazza



Eamon Walshe



Korektorzy merytoryczni:

Andrew Bryan jest programistą i konsultantem biznesowym z Auckland w Nowej Zelandii. Obecnie pracuje dla firmy zajmującej się mediami internetowymi i reklamą w Bostonie, gdzie mieszka ze swoją czarującą żoną Angie.

Jeremy Durham buduje aplikacje internetowe z wykorzystaniem Ruby on Rails od 2005 roku, a jego kod jest częścią kilku bibliotek Ruby. Mieszka w Arlington w stanie Massachusetts z żoną i dwójgiem dzieci.

Matt Harrington jest absolwentem Northeastern University, a od 9. roku życia także programistą.

Mike Isman pracuje z platformą Ruby on Rails od czasu, gdy w 2006 roku dołączył do zespołu *eons.com*, jeszcze zanim opublikowano Rails 1.0. W czasie pracy dla Eons Mike pisał również mniejsze witryny internetowe oparte na Rails, w tym kalkulator długości życia dostępny na stronie *livingto100.com*. W 2004 roku ukończył informatykę na University of Rochester i od tego czasu zajmuje się programowaniem aplikacji internetowych.

LuAnn Mazza jest analitykiem komputerowym z Illinois.

Eamon Walshe jest trenerem metodologii Agile, a wcześniej pełnił rolę Distinguished Engineer w IONA Technologies. Jest fanem Rails, ponieważ platforma ta pozwala programistom skupić się na tym, co ważne — dostarczaniu prawdziwej wartości biznesowej i to szybko.

Podziękowania

Moim redaktorom:

Mam ogromny dług wdzięczności wobec moich redaktorów — **Bretta McLaughlina** oraz **Lou Barr**. Zawsze służyli mi radą i wsparciem i kiedy tylko natrafiałem na problem, który wydawał się całkowicie nierozerwiąwalny, potrafili nie tylko zidentyfikować, *co* dokładnie było nie tak, ale również *dłaczego* było nie tak, a następnie proponowali kilka sposobów naprawienia sytuacji.



↑
Lou Barr

Brett McLaughlin



Szczególne podziękowania należą się mojej żonie, autorce książki *Head First Statystyka*, **Dawn Griffiths**. Niniejsza książka zwyczajnie nie byłaby skończona na czas, gdyby nie jej ogromna praca wykonana nad finalną wersją.

Książka ta jest w takim samym stopniu moja jak jej.



↑
Dawn Griffiths

Zespołowi O'Reilly:

Caitrin McCullough oraz **Karen Shaner**, które troszczyły się o wszystko, od kontraktów po zawartość stron internetowych.

Brittany Smith, redaktorowi produkcji książki, za bycie kopalinią praktycznej pomocy.

Catherine Nolan, za cierpliwe poprowadzenie mnie przez pierwszą fazę książki.

Laurie Petrycki, za jej wiarę w tę książkę i pozwolenie mi na skorzystanie z jej biura w Cambridge.

A także **Kathy Sierra** oraz **Bertowi Batesowi**, twórcom serii *Head First*, których oryginalna wizja zmieniła sposób pisania książek technicznych.

I nie zapominając:

Brianowi Hanly, CEO firmy Exoftware, oraz **Steve'owi Harvey'owi**. Ich szczodre wsparcie i uprzejmość umożliwiły powstanie tej książki.

Wreszcie całemu **zespołowi korektorów merytorycznych**, którzy musieli wykonać ogromną pracę w bardzo krótkim czasie.

Jestem Wam winien więcej, niż kiedykolwiek będę w stanie oddać.

1. Początki



Naprawdę szybkie Rails



Chcesz szybko zacząć pisać aplikacje internetowe? Powinieneś zatem poznać **Rails**. Rails to najfajniejsza i najszybsza platforma programowania, jaką istnieje. Pozwala tworzyć **w pełni funkcjonalne aplikacje internetowe** szybciej, niż kiedykolwiek wydawało się to możliwe. Początki są łatwe — wystarczy **zainstalować Rails** i zacząć przewracać strony książek. Zanim się zorientujesz, **o lata świetlne wyprzedzisz swoich konkurentów!**

Piątek, godzina 9 rano

Pierwszy e-mail, jaki otwierasz, pochodzi od przyjaciela, który jest w opałach:

Hej — jak się masz?

Potrzebna mi *wielka* przysługa! Pamiętasz tę aplikację do sprzedaży biletów, nad którą — jak mówilem — pracowaliśmy? Nie wygląda to za dobrze. Siedzimy nad tym od tygodni! Nasz zespół naprawdę sobie nie radzi.

Czy myślisz, że mógłbyś utworzyć tę aplikację dla nas?

Potrzebna nam strona internetowa, która jest w stanie:

- wyświetlić wszystkie sprzedane bilety,
- utworzyć nową transakcję sprzedaży biletu,
- wczytać i wyświetlić pojedynczy bilet,
- uaktualnić szczegóły sprzedaży,
- usunąć transakcję sprzedaży biletu.

Wiem — wydaje się, że to gigantyczna liczba funkcji, ale szef mówi, że to minimum opcji, jakich potrzebują; dobrze wiesz, że z tym facetem trudno się kłócić!

A oto struktura danych:

Ticket (bilet):

name — imię i nazwisko kupującego (łańcuch znaków)

seat_id_seq — numer miejsca, na przykład E14 (łańcuch znaków)

address — adres kupującego (długi łańcuch znaków)

price_paid — cena sprzedaży biletu (liczba dziesiętna)

email_address — adres e-mail kupującego (łańcuch znaków)

Załączam szkice stron internetowych, żebyś wiedział, do czego zmierzamy.

Eeee, potrzebujemy tego na poniedziałek, inaczej poleczę ze stołka. Pomocy!

System jest zaprojektowany do użycia przez personel zatrudniony w hali koncertowej.

Baza danych będzie resetowana dla każdego koncertu, więc wystarczające będzie zapisanie szczegółów jednego koncertu naraz. Myślisz, że dasz radę pomóc?

Aplikacja musi robić wiele rzeczy

Poniżej znajdują się szkice stron. Czy pasują one do wymagań tego systemu?

Na pierwszej stronie powinny być wymienione wszystkie sprzedane bilety.

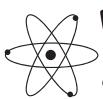
Na pierwszej stronie znajdzie się także przycisk, który pozwoli utworzyć nową transakcję sprzedaży biletu.

Obok każdego biletu na liście będzie się znajdować odnośnik podpisany „Show”, który wyświetla szczegóły pojedynczego biletu.

Oprócz odnośnika „Show” powinien tam być także odnośnik „Edit” wykorzystywany do aktualizacji transakcji sprzedaży biletu.

Wreszcie będzie się tam znajdować odnośnik „Destroy” służący do usunięcia transakcji sprzedaży biletu.

Name	Seat #	Address	Price paid	Email address	Action
Alan Langner	1A	37 Newbury Road, Ashfield, MA 400	\$100	alan@hanglong.com	Show Edit Destroy
Gordon Clark	99G	17 Tudor Street, Cambridge, MA 350	\$100	gclark@verizon.net	Show Edit Destroy
Bobbi Lyall	54C	9 Main Street, Provincetown, RI 490	\$100	blyall@optonline.net	Show Edit Destroy
Eric Macelink	7H	1528 Mass Ave, Cambridge, MA 370	\$100	eric@maccamazing.com	Show Edit Destroy
Molly Henderson	68J	665 Tremont St, Boston, MA 440	\$100	molly@redhandedsky.com	Show Edit Destroy



WYSIL
SZARE KOMÓRKI

Jakiego rodzaju oprogramowania będziesz potrzebował do utworzenia i uruchomienia aplikacji?

Co jest potrzebne aplikacji?

By uruchomić aplikację na serwerze hali koncertowej, potrzebujemy kilku elementów. Potrzebne nam są:

1 Platforma aplikacji.

Niezbędny nam będzie zbiór napisanego wcześniej kodu, który będzie stanowił podstawę aplikacji internetowej.

2 System bazy danych.

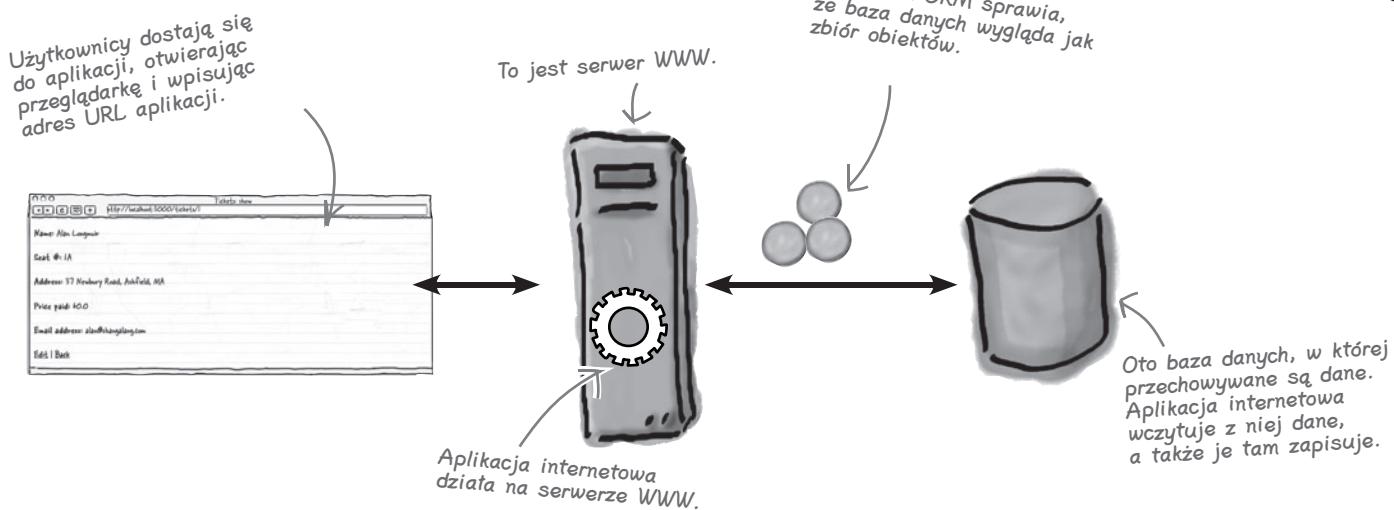
Potrzebna nam będzie jakąś baza danych, w której przechowamy dane.

3 Serwer WWW.

Musimy gdzieś uruchomić aplikację.

4 Biblioteka mapowania relacyjno-objektowego.

By ułatwić dostęp do bazy danych, większość aplikacji internetowych wykorzystuje obecnie do przekształcenia rekordów bazy danych w obiekty bibliotekę mapowania relacyjno-objektowego (ang. object-relational mapping, w skrócie O/R mapping lub ORM).



W czym zatem pomoże nam Rails?

Bez względu na język, w jakim tworzysz aplikację, najprawdopodobniej będziesz potrzebował wszystkich trzech elementów. Jedną z najlepszych rzeczy w Rails jest to, że platforma ta zawiera *całe* oprogramowanie, jakiego będziesz potrzebował — *dolaczane za darmo*.

Zobaczmy, jak to działa.

Łamigłówka



Platforma Rails ma wiele wbudowanych możliwości.

Twoje zadanie polega na odgadnięciu, które z elementów widocznych w basenie potrzebne nam będą w naszej aplikacji internetowej.

Później elementy te należy umieścić w pustych wierszach poniżej. Nie wszystkie elementy będą nam potrzebne.

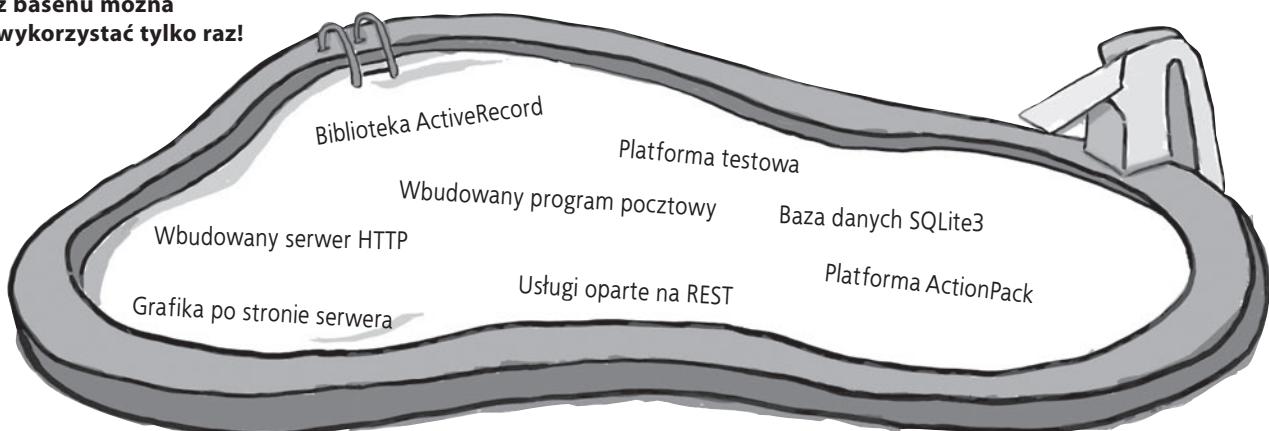
1

2

3

4

Uwaga: każdy element z basenu można wykorzystać tylko raz!



Rails służy do tworzenia aplikacji bazodanowych, takich jak system sprzedaży biletów

Sercem wielu aplikacji jest baza danych. Podstawowym celem istnienia tych aplikacji jest umożliwienie użytkownikom dostępu do zawartości bazy danych oraz edycji tych danych *bez konieczności* bezpośredniego korzystania z języka SQL.

Jakie problemy należy rozwiązać przy połączeniu bazy danych z aplikacją internetową?

Aplikacja internetowa musi zezwalać użytkownikowi na dostęp i modyfikację danych, dlatego Rails zawiera platformę aplikacji o nazwie **ActionPack**, która wspomaga generowanie interaktywnych stron internetowych współpracujących z bazami danych.

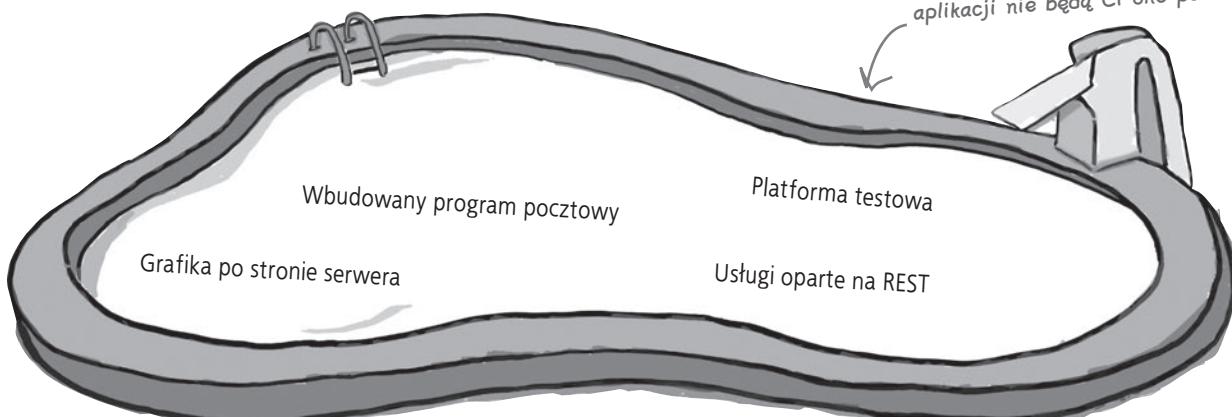
Po drugie, aplikacje internetowe muszą być uruchamiane na **serwerze WWW**, który będzie w stanie wyświetlać te strony, dlatego serwer taki wbudowany jest w Rails.

Po trzecie, niezbędną jest **baza danych**. Rails tworzy aplikacje, które skonfigurowane są do pracy ze zintegrowaną bazą danych **SQLite3**.

Po czwarte, niezbędną jest **biblioteka mapowania relacyjno-objektowego**; Rails udostępnia takową pod nazwą **ActiveRecord**. Dzięki temu baza danych wygląda jak zbiór prostych *obiektów* języka Ruby.

Oprócz tych narzędzi Rails zawiera również wiele **skryptów**, które wspomagają zarządzanie aplikacją. Kiedy będziesz tworzył aplikację internetową opartą na bazie danych, wkrótce przekonasz się, że

Rails daje Ci wszystko, czego potrzebujesz.



Łamigłówka: Rozwiążanie



Platforma Rails ma wiele wbudowanych możliwości. Twoje zadanie polega na odgadnięciu, które z elementów widocznych w basenie potrzebne nam będą w naszej aplikacji internetowej. Później elementy te należy umieścić w pustych wierszach poniżej. Nie wszystkie elementy będą nam potrzebne.

Platforma ActionPack

Baza danych SQLite3

Wbudowany serwer HTTP

Biblioteka ActiveRecord

*W niektórych systemach operacyjnych
będziesz musiał zainstalować ją
niezależnie od Rails.*

Rails daje Ci także wszystkie te
elementy, jednak w przypadku tej
aplikacji nie będą Ci one potrzebne.

Nową aplikację tworzy się za pomocą polecenia rails

Jak zatem zacząć pracę z Rails?

Utworzenie nowej aplikacji internetowej w Rails jest tak naprawdę bardzo proste. Wystarczy otworzyć okno wiersza poleceń lub terminala i wpisać do niego **rails tickets**, gdzie *tickets* to nazwa aplikacji, którą chcesz utworzyć.

```
Plik Edycja Okno Pomoc
> rails tickets
```

Wpisz po prostu „rails tickets”
w wierszu poleceń.

Co to robi?

Wpisanie `rails tickets` w sprytny sposób generuje aplikację internetową w nowym folderze o nazwie *tickets*. Co więcej, wewnątrz folderu *tickets* Rails generuje całe mnóstwo dalszych folderów i plików, które tworzą podstawową strukturę nowej aplikacji.

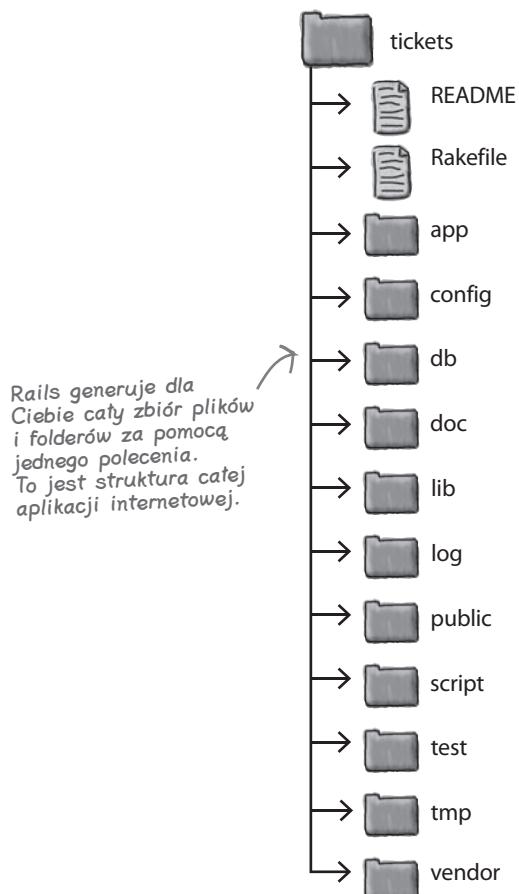
Oznacza to, że tak naprawdę utworzyłeś całą podstawową aplikację za pomocą jednego krótkiego polecenia.



Rails generuje mnóstwo plików i folderów, ale nie ma się tym co martwić.

Wszystkie mają swoje uzasadnienie i do końca książki zrozumiesz, do czego służą.

Zrób tak!





Jazda próbna

Ponieważ utworzona przed chwilą aplikacja jest aplikacją *internetową*, by zobaczyć, jak działa, będziesz musiał uruchomić wbudowany serwer WWW.

W wierszu poleceń czy terminalu wejdź do folderu *tickets* i wpisz **ruby script/server**.

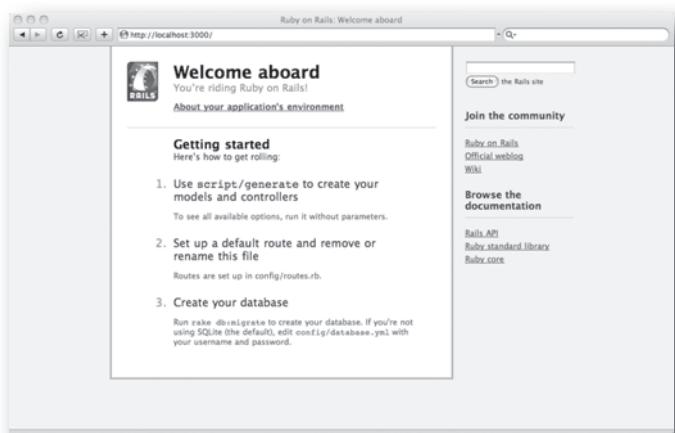
Wejdź do folderu aplikacji... →
... i uruchom serwer WWW.

```
Plik Edycja Okno Pomoc  
> cd tickets  
> ruby script/server
```

To jest konsola. Wchodzisz do niej za pomocą wiersza poleceń w systemie Windows lub terminala w systemach Linux lub Mac.

Na ekranie pojawi się kilka poleceń, które potwierdzają, że serwer działa. Teraz możesz zobaczyć domyślną stronę główną, otwierając w przeglądarce adres:

<http://localhost:3000/>



To jest domyślna strona główna serwera WWW.



Ciekawostki

Rails domyślnie uruchamia serwer WWW na porcie 3000. Jeśli chcesz użyć innego portu, takiego jak 8000, wykonaj polecenie:

```
ruby script/server -p 8000
```

Teraz do domyślnej aplikacji trzeba dodać własny kod

Rails od razu tworzy podstawową strukturę aplikacji, jednak będziesz musiał dodać kod robiący to, czego chcesz Ty. Każda aplikacja jest inna, ale czy Rails ma jakieś narzędzia czy sztuczki, które ułatwiają tworzenie własnego kodu?

Faktycznie, tak właśnie jest. Czy zauważłeś, że platforma Rails utworzyła dla Ciebie całą strukturę plików, prawie jakby wiedziała, czego będziesz potrzebować? Dzieje się tak, ponieważ aplikacje Rails mają bardzo silne konwencje nazewnictwa.

Aplikacje Rails zawsze przestrzegają konwencji

Wszystkie aplikacje Rails mają tę samą podstawową strukturę plików i wykorzystują spójne nazewnictwo poszczególnych elementów. Dzięki temu aplikacje są łatwiejsze do zrozumienia, jednak równocześnie oznacza to, że wbudowane narzędzia Rails będą rozumieć, jak działa Twoja aplikacja.

Dlaczego jest to tak istotne? Skoro narzędzia wiedzą, jaką strukturę ma aplikacja, można ich użyć do zautomatyzowania wielu zadań programistycznych. W ten sposób Rails może wykorzystać konwencje do wygenerowania dla Ciebie kodu bez konieczności konfiguracji aplikacji. Innymi słowy, Rails przedkłada *konwencję nad konfiguracją*.

Przyjrzyjmy się jednemu z narzędzi Rails o największych możliwościach — rusztowaniu.

Nie istnieją, grupie pytania

P: Ciągle piszecie o „Ruby” albo o „Rails”. Jaka jest między nimi różnica?

O: Ruby to język programowania. Rails to zbiór skryptów języka Ruby. Dlatego serwer WWW, platforma aplikacji ActionPack, a także wbudowane skrypty narzędzi to wszystko skrypty języka Ruby, które są jednocześnie częścią Rails.

P: W jaki sposób mogę zmodyfikować stronę główną mojej nowej witryny?

O: Zerknij do kodu HTML znajdującego się w pliku index.html w katalogu public aplikacji. Katalog public zawiera całą statyczną zawartość aplikacji.

P: A co jeśli chcę użyć innego serwera WWW? Czy mogę to zrobić?

O: W czasie tworzenia aplikacji używanie wbudowanego serwera WWW ma sens. Jeśli jednak chcesz wdrożyć gotową wersję aplikacji na innym serwerze WWW, możesz to zrobić.

Zasada Rails:

Konwencja
ważniejsza od
konfiguracji

P: Czy ma znaczenie, w którym folderze jestem, kiedy wykonuję polecenie ruby script/server?

O: Oczywiście, że tak. Musisz znajdować się w folderze zawierającym aplikację.

P: Co kompiluje mój kod?

O: Ruby jest językiem interpretowanym, jak JavaScript. Oznacza to, że kompilacja nie jest potrzebna. Możesz po prostu zmienić kod i natychmiast go wykonać.

Rusztowanie to kod GENEROWANY

Co zatem musi zrobić nasza aplikacja? Spójrzmy jeszcze raz do wiadomości od przyjaciela:

To ten sam e-mail
co wcześniej.

Hej — jak się masz?

Potrzebna mi *wielka* przysługa! Pamiętasz tę aplikację do sprzedaży biletów, nad którą — jak mówiłem — pracowaliśmy? Nie wygląda to za dobrze. Siedzimy nad tym od tygodni! Nasz zespół naprawdę sobie nie radzi.

Czy myślisz, że mógłbyś utworzyć tę aplikację dla nas?

Potrzebna nam strona internetowa, która jest w stanie:

- wyświetlić wszystkie sprzedane bilety,
- utworzyć (Create) nową transakcję sprzedaży biletu,
- wczytać (Read) i wyświetlić pojedynczy bilet,
- uaktualnić (Update) szczegóły sprzedaży,
- usunąć (Delete) transakcję sprzedaży biletu.

Aplikacja musi wykonywać wszystkie te operacje. Musi być w stanie tworzyć, odczytywać, uaktualniać i usuwać dane.

Wiem — wydaje się, że to gigantyczna liczba funkcji, ale szef mówi, że to minimum opcji, jakich potrzebują; dobrze wiesz, że z tym facetem trudno się kłócić!
A oto struktura danych:

Ticket (bilet):

name — imię i nazwisko kupującego (łańcuch znaków)
seat_id_seq — numer miejsca, na przykład E14 (łańcuch znaków)
address — adres kupującego (długi łańcuch znaków)
price_paid — cena sprzedaży biletu (liczba dziesiętna)
email_address — adres e-mail kupującego (łańcuch znaków)

Załączam szkice stron internetowych, żebyś wiedział, do czego zmierzamy.

Eeee, potrzebujemy tego na poniedziałek, inaczej poleczę ze stołka. Pomocy!

Operacje te
muszą działać
na tej strukturze
danych biletu.

Musimy zatem utworzyć strony internetowe, które pozwolą nam *tworzyć* (*Create*), *odczytywać* (*Read*), *uaktualniać* (*Update*) i *usuwać* (*Delete*) bilety. Ponieważ pierwsze litery tych operacji w języku angielskim to **C**, **R**, **U** i **D**, są one znane jako *operacje CRUD*. W aplikacjach opartych na bazach danych operacje te wykonywane są stosunkowo często — tak często, że Rails udostępnia sposoby szybkiego generowania całego kodu i wszystkich stron, jakie będą potrzebne. Wszystko to wykonywane jest za pomocą *rusztowania* (ang. *scaffolding*).



Magnesiki z kodem

Istnieje proste polecenie, które można wydać w konsoli w celu wygenerowania kodu rusztowania. Zobacz, czy będziesz w stanie ułożyć magnesiki tak, by uzupełnić to polecenie.

`ruby script/generate ticket name:`

.....:.....:.....:.....

seat_id_seq

string

scaffold

price_paid

decimal

string

address

email_address

text

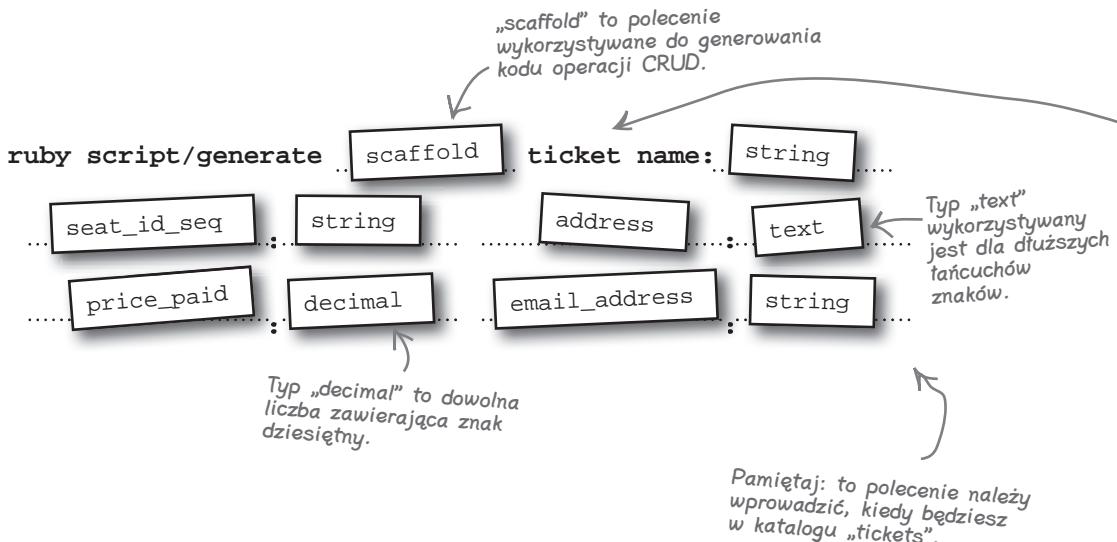
string

Polecenie scaffold tworzy kod



Magnesiki z kodem: Rozwiążanie

Istnieje proste polecenie, które można wydać w konsoli w celu wygenerowania kodu rusztowania. Zobacz, czy będziesz w stanie ułożyć magnesiki tak, by uzupełnić to polecenie.



Co zatem robi polecenie scaffold?

Polecenie scaffold tworzy kod, który pozwala użytkownikowi tworzyć, odczytywać, uaktualniać oraz usuwać dane z bazy.

Kiedy masz aplikację opartą na bazie danych, która potrzebuje tworzyć, odczytywać, uaktualniać i usuwać dane z bazy, rusztowanie może oszczędzić Ci sporo czasu i wysiłku.

Wpisz polecenie scaffold dla tabeli ticket do konsoli i zobacz, co się stanie:

Zrób tak!

```
Plik Edycja Okno Pomoc
> ruby script/generate scaffold ticket name:string
  seat_id_seq:string address:text price_paid:decimal
  email_address:string
```



Jazda próbna

Teraz pora sprawdzić, czy aplikacja naprawdę działa. By zobaczyć nową stronę z biletami, wpisz w przeglądarce adres:

<http://localhost:3000/tickets>

Odpowiada to nazwie podanej w poleceniu „`scaffold`”. Widzisz, jak platforma Rails sama zrobiła z tego liczbę mnoga?

Hm... to zdecydowanie nie
wylada dobrze.

Co zatem poszło nie tak? Pomimo że kod rusztowania został wygenerowany poprawnie, serwer WWW wyświetla błąd. Wszystko, co otrzymujemy, to komunikaty o błędach.

Action Control

http://localhost:3000/tickets/

ActiveRecord::StatementInvalid in TicketsController#index

SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets"

RAILS_ROOT: /Users/davidg/Desktop/chap1-scaffold/tickets

[Application Trace](#) | [Framework Trace](#) | [Full Trace](#)

```
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_.rb:1:in `new': SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets" (ActiveRecord::StatementInvalid)
```

```
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_.rb:1:in `new': SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets" (ActiveRecord::StatementInvalid)
```

```
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_.rb:1:in `new': SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets" (ActiveRecord::StatementInvalid)
```

```
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_.rb:1:in `new': SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets" (ActiveRecord::StatementInvalid)
```

```
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_.rb:1:in `new': SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets" (ActiveRecord::StatementInvalid)
```

```
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_.rb:1:in `new': SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets" (ActiveRecord::StatementInvalid)
```

```
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_.rb:1:in `new': SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets" (ActiveRecord::StatementInvalid)
```

```
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_.rb:1:in `new': SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets" (ActiveRecord::StatementInvalid)
```

```
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_.rb:1:in `new': SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets" (ActiveRecord::StatementInvalid)
```

```
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_.rb:1:in `new': SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets" (ActiveRecord::StatementInvalid)
```

```
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_.rb:1:in `new': SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets" (ActiveRecord::StatementInvalid)
```

```
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_.rb:1:in `new': SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets" (ActiveRecord::StatementInvalid)
```

```
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/base.rb:586:in `connection'
```

```
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/base.rb:134:in `connection'
```

```
app/controllers/tickets_controller.rb:5:in `index'
```

```
/Library/Ruby/Gems/1.8/gems/actionpack-2.1.2/lib/action_controller/base.rb:1:in `new': SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets" (ActiveRecord::StatementInvalid)
```

```
/Library/Ruby/Gems/1.8/gems/actionpack-2.1.2/lib/action_controller/base.rb:1:in `new': SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets" (ActiveRecord::StatementInvalid)
```

```
/Library/Ruby/Gems/1.8/gems/actionpack-2.1.2/lib/action_controller/base.rb:1:in `new': SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets" (ActiveRecord::StatementInvalid)
```

```
/Library/Ruby/Gems/1.8/gems/actionpack-2.1.2/lib/action_controller/filters.rb:1:in `new': SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets" (ActiveRecord::StatementInvalid)
```



WYSIL SZARE KOMÓRKI

Zastanów się nad komunikatem o błędzie widocznym w przeglądarce.
Jak myślisz, dlaczego aplikacja nie działa?

W bazie danych nie ma jeszcze tabel!

Aplikacja powinna wyświetlić pustą listę sprzedanych biletów, ale tak nie zrobiła. Dlaczego nie? Powinna była wczytać listę z tabeli bazy danych o nazwie **tickets**, jednak nie utworzyliśmy jeszcze żadnych tabel.

Czy powinniśmy połączyć się z bazą danych i utworzyć tabelę? W końcu baza danych znajduje się tu, w aplikacji. Ale właściwie dlaczego mamy to robić? Przecież przekazaliśmy Rails wystarczająco dużo informacji, by platforma ta utworzyła tabelę za nas. Spójrzmy raz jeszcze na polecenie scaffold:

```
Plik Edycja Okno Pomoc
> ruby script/generate scaffold ticket name:string
   seat_id_seq:string address:text price_paid:decimal
   email_address:string
```

Uwaga: rusztowanie nosi nazwę „ticket” (liczba pojedyncza), natomiast tabela będzie się nazywała „tickets” (liczba mnoga).

tickets	
name	string
seat_id_seq	string
address	text
price_paid	decimal
email_address	string

Kiedy wykonywaliśmy polecenie scaffold, przekazaliśmy Rails wystarczającą ilość informacji dotyczących struktury danych, a w Rails istnieje ważna zasada: *Nie powtarzaj się*. Jeśli powiesz coś Rails raz, nie powinieneś musieć tego powtarzać.

Zasada Rails:

Jak zatem zmusić Rails do utworzenia tabeli?



Ciekawostki

Rails ma wbudowaną bazę danych SQLite3. Gdzie ona zatem jest?

Baza danych znajduje się w folderze db, w pliku development.sqlite3.

**Nie
powtarzaj
się**

Znajomi programiści mogą nazywać tę zasadę „DRY”, od pierwszych liter jej angielskiego odpowiednika („Don't Repeat Yourself”).

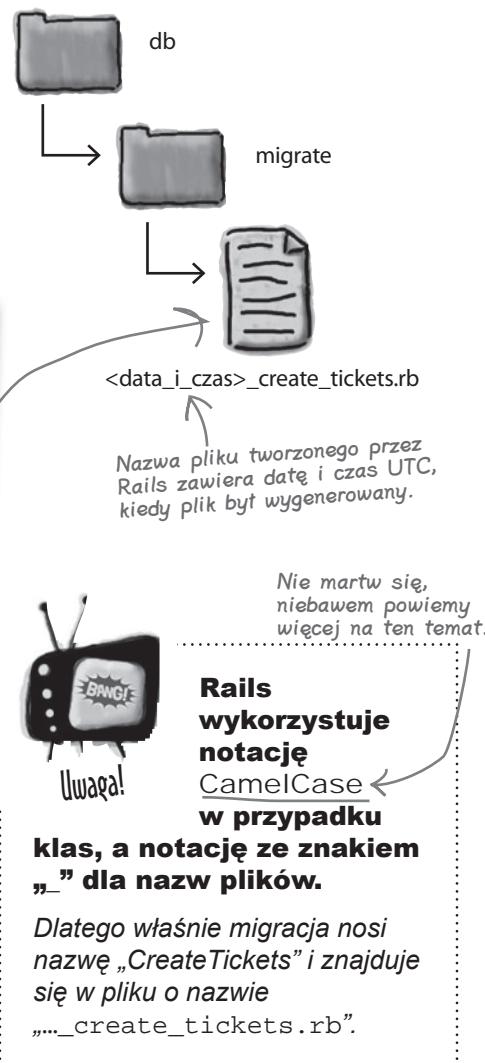
Tabelę tworzy się dzięki wykonaniu migracji

Kiedy platforma Rails wygenerowała rusztowanie, utworzyła również niewielki skrypt w języku Ruby, noszący nazwę **migracja** (ang. *migration*) i służący do tworzenia tabeli. Migracja to skrypt zmieniający strukturę dołączonej bazy danych.

Zajrzyj do folderu *db/migrate*. Powinieneś znaleźć tam plik o nazwie *<data_i_czas>_create_tickets.rb*, gdzie *<data_i_czas>* to data i czas UTC utworzenia pliku. Jeśli otworzysz ten plik w edytorze tekstu, powinien on wyglądać mniej więcej następująco:

```
class CreateTickets < ActiveRecord::Migration
  def self.up
    create_table :tickets do |t|
      t.string :name
      t.string :seat_id_seq
      t.text :address
      t.decimal :price_paid
      t.string :email_address
      t.timestamps
    end
  end
  def self.down
    drop_table :tickets
  end
end
```

Oto zawartość pliku migracji.



Migracja to niewielki skrypt języka Ruby. Zamiast wykonywać ten skrypt bezpośrednio, należy go wykonać za pomocą innego narzędzia Rails, o nazwie **rake**. By wykonać migrację, należy wpisać **rake db:migrate** w wierszu poleceń. Poniższe polecenie wykonuje kod migracji i tworzy tabelę:

```
Plik Edycja Okno Pomoc
> rake db:migrate
```



Dlaczego migracja zawiera w nazwie datę i czas?



Jazda próbna

Upewnij się, że utworzyłeś tabelę tickets za pomocą polecenia rake.

Teraz wróć do przeglądarki i odśwież stronę:

`http://localhost:3000/tickets`

Twoja aplikacja internetowa działa! W kilka minut możesz utworzyć kilka testowych rekordów:

The screenshot shows a web browser window titled "Tickets: index". The URL in the address bar is "http://localhost:3000/tickets/". The page displays a table with the following data:

Name	Seat id	seq	Address	Price paid	Email address	Action
Alan Longmuir	1A		37 Newbury Road, Ashfield, MA	60.0	alan@shangalang.com	Show Edit Destroy
Gordon Clark	43G		17 Tudor Street, Cambridge, MA	35.0	gclark@rollermail.com	Show Edit Destroy
Bobbi Lyall	54C		9 Main Street, Provincetown, RI	43.0	blyall@baycity.org	Show Edit Destroy
Eric Manclark	7H		1326 Mass Ave, Cambridge, MA	37.0	eric@mananamail.org	Show Edit Destroy
Nelly Henderson	8BJ		665 Tremont St, Boston, MA	64.0	nelly@byebyebaby.com	Show Edit Destroy

A hand-drawn annotation on the left side of the screenshot says: "Oto kilka rekordów, jakie dodaliśmy. Ty też dodaj kilka!" (Here are some records we added. You can add some too!).

Oto kilka rekordów, jakie dodaliśmy. Ty też dodaj kilka!



Czekaj! To niemożliwe!
Wpisaliśmy kilka poleceń
do konsoli i to
spowodowało utworzenie
całej aplikacji?

**Tak — zbudowaliśmy o wiele
więcej niż tylko stronę główną.
Zbudowaliśmy cały system.**

Rusztowanie wygenerowało cały zbiór stron, które pozwalają użytkownikom tworzyć, modyfikować i usuwać informacje dotyczące biletów. By zobaczyć, jak działa cała aplikacja, utworzymy i zmodyfikujemy kolejny rekord.

New ticket

Name

Seat id seq

Address

Price paid

Email address

[Back](#)

Kliknięcie odnośnika „New ticket” na stronie internetowej przeniesie Cię do formularza, w którym możesz utworzyć nowy bilet.

Po przestaniu formularza będziesz w stanie wczytać nowy bilet z bazy danych i wyświetlić go.

Tickets: show

Name: Alan Longmuir
Seat id seq: 1A
Address: 37 Newbury Road, Ashfield, MA
Price paid: 60.0
Email address: alan@shangaleng.com

[Edit](#) | [Back](#)

Przycisk „Edit” na stronie wyświetlającej bilet pozwala uaktualnić dowolne informacje.

Editing ticket

Name

Seat id seq

Address

Price paid

Email address

[Show](#) | [Back](#)

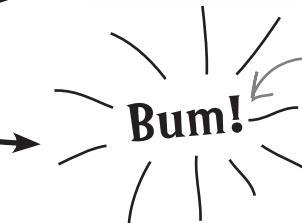
Kliknięcie tącza „Back” przeglądarki przenosi Cię z powrotem na stronę główną...

Listing tickets

Name	Seat id seq	Address	Price paid	Email address	
Alan Longmuir	1A	37 Newbury Road, Ashfield, MA	60.0	alan@shangaleng.com	Show Edit Destroy
Gordon Clark	43G	17 Tudor Street, Cambridge, MA	35.0	gclark@rollermail.com	Show Edit Destroy
Bobbi Lyall	54C	9 Main Street, Provincetown, RI	43.0	blyall@baycity.org	Show Edit Destroy
Eric Mancalk	7H	1326 Mass Ave, Cambridge, MA	37.0	eric@mananamail.org	Show Edit Destroy
Nelly Henderson	88J	665 Tremont St, Boston, MA	64.0	nelly@byebyebaby.com	Show Edit Destroy

[New ticket](#)

...gdzie możesz wybrać bilet do usunięcia.





CELNE SPOSTRZEŻENIA

- Polecenie

```
rails <nazwa aplikacji>
```

generuje dla Ciebie aplikację znajdująca się w folderze `<nazwa aplikacji>`. Rails tworzy również foldery oraz pliki stanowiące podstawową strukturę aplikacji.

- Rails zawiera wbudowany serwer WWW. By go uruchomić, należy użyć polecenia:

```
ruby script/server
```

Domyślna strona główna znajduje się pod adresem:

```
http://localhost:3000/
```

- Aplikacje Rails są zgodne z zasadą: „Konwencja ważniejsza od konfiguracji”.
- Operacje tworzenia, odczytywania, aktualniania

i usuwania wykonywane na bazie danych znane są pod nazwą operacji CRUD.

- Rusztowanie tworzy dla Ciebie kod CRUD. By utworzyć rusztowanie dla obiektu `thing`, należy wykonać:

```
ruby script/generate scaffold thing
  <nazwa kolumny 1>:<typ kolumny 1>
  <nazwa kolumny 2>:<typ kolumny 2>
  ...
```

- By zobaczyć rusztowanie, należy wybrać w przeglądarce adres:

```
http://localhost:3000/things
```

- Aplikacje Rails są zgodne z zasadą: „Nie powtarzaj się”.
- Migracja to skrypt zmieniający strukturę dołączonej bazy danych. Migrację wykonuje się za pomocą polecenia:

```
rake db:migrate
```

Nie istniejąca głupie pytania

P: Niektóre polecenia rozpoczynają się od `rails`, inne od `ruby`, a jeszcze inne od `rake`. Jaka jest między nimi różnica?

O: Polecenie `rails` wykorzystywane jest do utworzenia nowej aplikacji. Z kolei `ruby` to interpreter języka Ruby, który wykorzystywany jest do wykonywania skryptów narzędzi przechowywanych w folderze `scripts`. Polecenia `ruby` i `rake` używane są w zasadzie do wszystkich zadań w Rails.

P: Czym zatem jest `rake`?

O: `rake` to polecenie użyte przez nas do wykonania migracji bazy danych. Oznacza „Ruby make” i wykorzystywane jest do niektórych z tych samych zadań, do jakich w językach takich, jak C i Java służą, odpowiednio, `make` oraz `ant`. Kiedy `rake` otrzymuje zadanie do wykonania (na przykład migrację), jest w stanie w sprytny sposób przeanalizować aplikację i zdecydować, które skrypty ma wykonać. Jest zatem nieco sprytniejsze od `ruby` i wykorzystywane jest do bardziej skomplikowanych zadań, takich jak modyfikowanie struktury bazy danych czy wykonywanie testów.

P: Nie rozumiem zasady: „Konwencja ważniejsza od konfiguracji”. Co ona oznacza?

O: Wiele języków programowania daje Ci mnóstwo opcji, z których możesz wybierać, tak jak wybierasz wyposażenie nowego samochodu. W przypadku języka z dużą liczbą dostępnych opcji konieczne jest przechowanie wyborów programisty — zazwyczaj w dużych plikach XML. Rails ma inne podejście. W Rails wszystko nazywane jest w spójny sposób i przechowywane w ustandaryzowanym miejscu. Jest to nazywane podejściem „konwencjonalnym” — nie dlatego, że jest staromodne, ale dlatego, że jest zgodne z pewnymi „konwencjami” czy „standardami”.

P: Nie mogę zatem zmienić sposobu działania Rails?

O: W Rails możesz zmienić właściwie wszystko, jeśli jednak będziesz przestrzegał konwencji, wkrótce zauważysz, że tworzenie aplikacji pójdziesz Ci szybciej, a inne osoby uznażą Twój kod za łatwiejszy do zrozumienia.

Pięknie! Uratowałeś pracę kumpla!

Twoje krótkie spotkanie z Rails uratowało stołek przyjaciela... przynajmniej na razie. Wygląda na to, że właśnie przyszedł kolejny e-mail:

Wielkie dzięki!

Niesamowite jest widzieć działającą aplikację — i jeszcze wszystko poszło Ci tak szybko! Rails wygląda na fantastyczną technologię. Samo to, jak wszystko pokazuje się od razu po edycji kodu... Żadnej komplikacji. Żadnego wdrażania. To musi być super!

Naprawdę uratowałeś mój stołek.

Tylko jedna uwaga — podpisy dla „seat_id_seq” powinny być bardziej czytelne dla człowieka, coś w stylu „Seat #”. Myślisz, że dałoby się to zrobić?

Jak zatem możemy zmienić podpisy?

Platforma Rails szybko wygenerowała dla nas aplikację internetową, co oszczędziło nam wiele czasu i wysiłku. Co jednak zrobić, jeśli chcemy wprowadzić niewielkie zmiany do wyglądu wygenerowanych stron?

Jak łatwe jest modyfikowanie stron wygenerowanych dla nas przez Rails?

By zmodyfikować aplikację, musisz przyjrzeć się jej architekturze

Rusztowanie po prostu generuje dla nas kod. Po wygenerowaniu kodu dostosowanie go do własnych potrzeb należy do Ciebie. A jeśli spojrzysz do folderu aplikacji, zobaczysz, że jest w nim mnóstwo wygenerowanego kodu, który możesz chcieć dostosować do własnych wymagań.

Jeśli zatem musisz wprowadzić zmiany do aplikacji — na przykład zmodyfikować podpisy stron — od czego możesz zacząć?



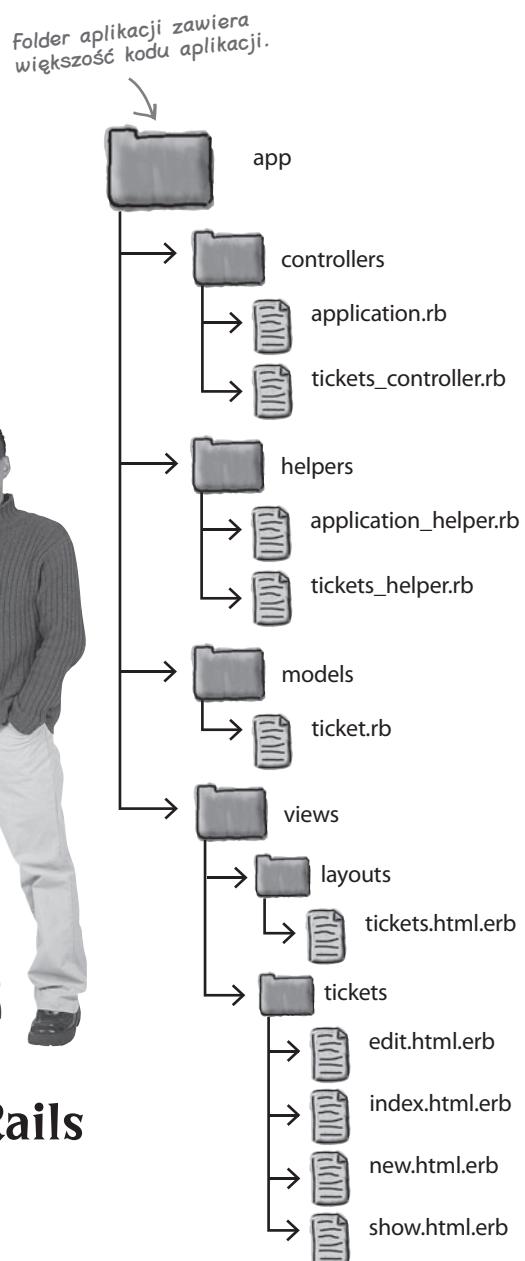
Polegaj na konwencjach Rails.

Pamiętasz, jak powiedzieliśmy, że aplikacje Rails zgodne są z zasadą: „Konwencja ważniejsza od konfiguracji”? To ułatwi nam modyfikację aplikacji. Dlaczego? Ponieważ kod podzielony jest zgodnie z jego **funkcją**. Oznacza to, że skrypty Ruby robiące podobne rzeczy znajdują się w *podobnych miejscach*.

Jeśli zatem potrzebujesz zmienić zachowanie aplikacji Rails, powinieneś być w stanie zidentyfikować, gdzie należy poprawić kod, a następnie go zmodyfikować.

Oczywiście, żeby to jednak zrobić, musisz zrozumieć...

Standardową architekturę Rails



Trzy części Twojej aplikacji: model, widok i kontroler

Prawie cały kod w aplikacjach Rails mieści się w jednej z trzech kategorii:

1 Kod modelu

Kod modelu (ang. *model*) zarządza zapisem danych do bazy oraz ich odczytem. **Obiekty** kodu modelu reprezentują rzeczy istniejące w *domenie systemu* — tak jak **bilety** w systemie biletów.

Oznacza to po prostu problemy biznesowe, jakie aplikacja stara się rozwiązać.



2 Kod widoku

Widok (ang. *view*) to część aplikacji **prezentowana** użytkownikowi. Z tego powodu czasami nazywana jest również **warstwą prezentacyjną**. W przypadku aplikacji internetowej widok przede wszystkim generuje strony internetowe.

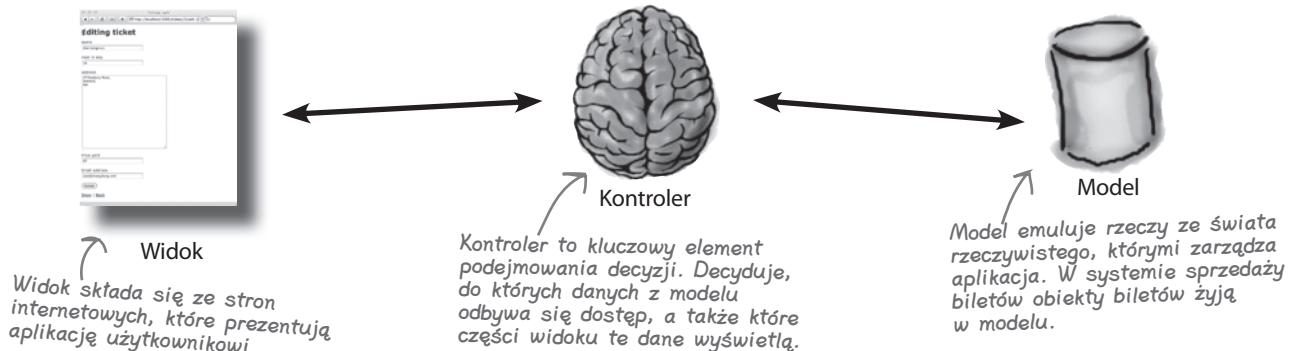


3 Kod kontrolera

Kontroler (ang. *controller*) to prawdziwy *mózg* aplikacji. Decyduje o tym, w jaki sposób użytkownik wchodzi w **interakcję** z systemem, kontrolując, do których danych z modelu odbywa się dostęp i które części widoku je zaprezentują..



Różne typy kodu są w aplikacji Rails połączone w następujący sposób:





Cała prawda o Rails

Wywiad tygodnia:

Pytamy najbardziej pożądaną platformę internetową o to, co nią kieruje

Head First: Witaj Rails, ogromnie się cieszymy, że mogłas do nas wpaść.

Rails: Cieszę się, że mogę tu być.

Head First: Musiało ci być ciężko znaleźć wolną chwilę w tak napiętym terminarzu.

Rails: Jasne, jestem niesamowicie zajęta. Z tymi wszystkimi połączeniami z bazą danych, logiką aplikacji i stronami, jakie muszę prezentować, nie zostaje mi zbyt wiele czasu dla siebie. Ale daję sobie jakoś radę — mam niezłych współpracowników.

Head First: Zastanawiałem się nad jedną rzeczą — mam nadzieję, że się nie obrazisz... Kiedy tworzyś nową aplikację, po co jest w niej od razu tyle katalogów?

Rails: Co mogę powiedzieć? Pomocna ze mnie dusza. Z czasem nauczyłam się, co ludzie chcą robić w swoich aplikacjach. Nie podoba mi się, gdy ludzie ręcznie muszą tworzyć ciągle te same rzeczy.

Head First: Ale czy to nie jest nieco... no, mylące?

Rails: Proszę cię. Jestem osobą przestrzegającą konwencji. Żadnych niespodzianek. Kiedy raz nauczysz się, jak ze mną współpracować, zobacysz, że szybko się do mnie przyzwyczaisz.

Head First: Słyszałem, że nie lubisz, by cię konfigurowano.

Rails: Możesz mnie konfigurować w dowolny sposób, jednak większość osób woli pracować w ten sam sposób co ja. Konwencja ważniejsza od konfiguracji. Jasne?

Head First: No tak — to w końcu jedna z twoich zasad, prawda?

Rails: Właśnie — ta i jeszcze: „Nie powtarzaj się”.

Head First: I jeszcze co?

Rails: „Nie powtarzaj się”?

Head First: I jeszcze co?

Rails: Nie... Hej, zabawny z ciebie facet!

Nieistniejąca grupa pytań

P: Gdzie w mojej aplikacji powinna się znajdować logika biznesowa?

O: Cóż, zależy, co rozumiesz pod tym pojęciem. Niektóre osoby definiują logikę biznesową jako reguły związane z zarządzaniem danymi. W tym przypadku logika biznesowa mieści się w modelu. Inne osoby definiują logikę biznesową jako reguły definiujące sposób działania systemu — na przykład jakie możliwości ma aplikacja i jaka jest kolejność dostępu do nich. W tym przypadku logika biznesowa mieści się

w kontrolerze. W dalszej części książki będziemy używali pojęć „logika modelu” oraz „logika aplikacji”, by rozróżnić te dwie sytuacje.

P: Jaka jest różnica między widokiem a kontrolerem?

O: Widok decyduje o tym, jak *wygląda* aplikacja, a kontroler decyduje o tym, jak *działa*. Widok definiuje zatem kolor przycisku na stronie, a także widoczny na nim tekst, natomiast kontroler decyduje, co dzieje się po naciśnięciu przycisku.

P: Jaki kod będę pisał najczęściej?

O: To zależy od aplikacji i programisty. Jeśli zobacysz, że najczęściej dodajesz kod do którejś z trzech części aplikacji, możesz się zastanowić, czy kolejny dodawany fragment przynależy do prezentacji, interakcji, czy też modelowania.

JAKI JEST MÓJ CEL?

Dopasuj opis kodu do części aplikacji, z którą łączy się ten kod.

Projekt kart w pasjansie internetowym.

W systemie bankowości elektronicznej ten kod decyduje o tym, czy chcesz przelać pieniądze na konto, czy z konta.

Obiekt „spotkanie” w aplikacji terminarza.

W systemie bloga kod ten decyduje, czy komentarze mają być wyświetlane jako tabela, czy jako lista.

Ten kod rejestruje ofertę w internetowym portalu aukcyjnym.

Kod decyduje, że musisz się zalogować do aplikacji udostępniającej pocztę elektroniczną.

Menu składające się z odnośników.



Model



Widok



Kontroler

★ JAKI JEST MÓJ CEL? ★

ROZWIĄZANIE

Dopasuj opis kodu do części aplikacji, z którą łączy się ten kod.

Projekt kart w pasjansie internetowym.

W systemie bankowości elektronicznej ten kod decyduje o tym, czy chcesz przelać pieniądze na konto, czy z konta.

Obiekt „spotkanie” w aplikacji terminarza.

W systemie bloga kod ten decyduje, czy komentarze mają być wyświetlane jako tabela, czy jako lista.

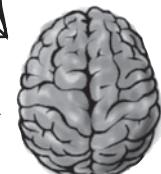
Ten kod rejestruje oferty w internetowym portalu aukcyjnym.

Kod decyduje, że musisz się zalogować do aplikacji udostępniającej pocztę elektroniczną.

Menu składające się z odnośników.



Widok



Kontroler

Trzy typy kodu przechowywane są w OSOBNYCH folderach

Rails woli zatem konwencję od konfiguracji i wykorzystuje architekturę MVC (*Model-View-Controller*, czyli Model-Widok-Kontroler). I co z tego wynika?

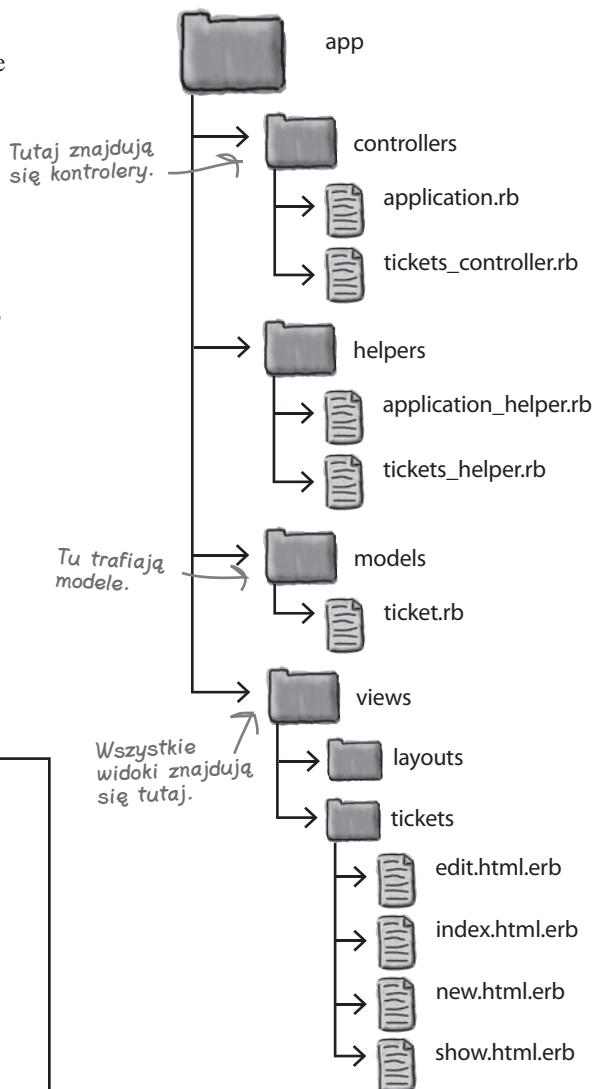
W jaki sposób architektura MVC pomoże nam zmienić podpisy na naszych stronach i tym samym poprawić aplikację? Przyjrzymy się raz jeszcze plikom wygenerowanym przez rusztowanie. Ponieważ kod jest w jasny sposób podzielony na trzy odrębne typy — model, widok oraz kontroler — Rails umieszcza każdy typ w osobnym folderze.



Zaostrz ołówek

Na diagramie folderów znajdującym się po prawej stronie zaznacz pliki, o których sądzisz, że trzeba je będzie zmodyfikować w celu edycji podpisów na stronach.

Napisz następnie, **dłaczego** wybrałeś te właśnie pliki.



Podpisy znajdują się w widokach

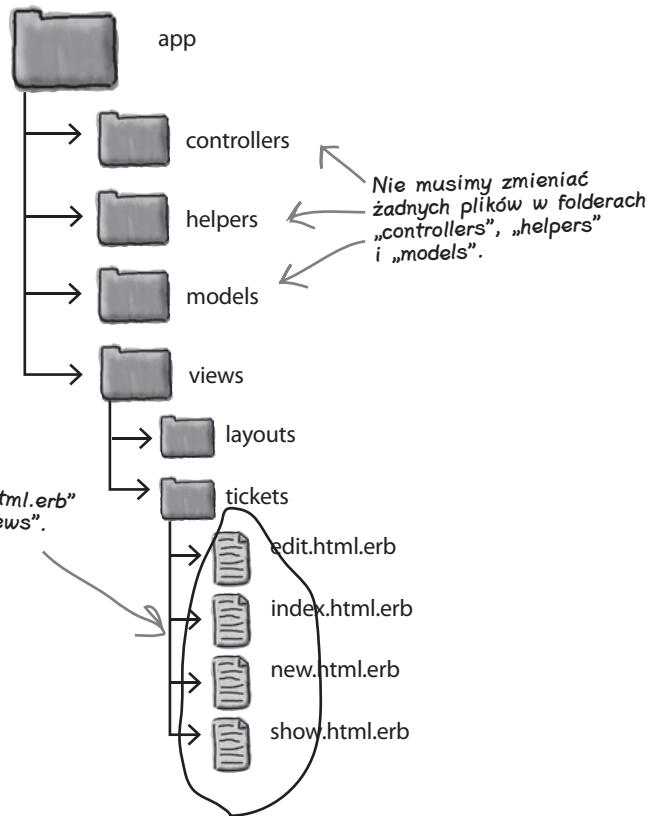


Rozwiążanie

Twoje zadanie polegało na zaznaczeniu plików, które trzeba zmodyfikować w celu zmiany podpisów na stronach.

Ponieważ musimy zmienić wygląd stron, musimy zmienić widoki. Pliki, które trzeba uaktualnić, znajdują się w folderze „views” i mają rozszerzenie „.html.erb”.

Podpisy na stronach możemy zmienić, modyfikując pliki „.html.erb” znajdujące się w folderze „views”.



Trzeba zmodyfikować pliki WIDOKU

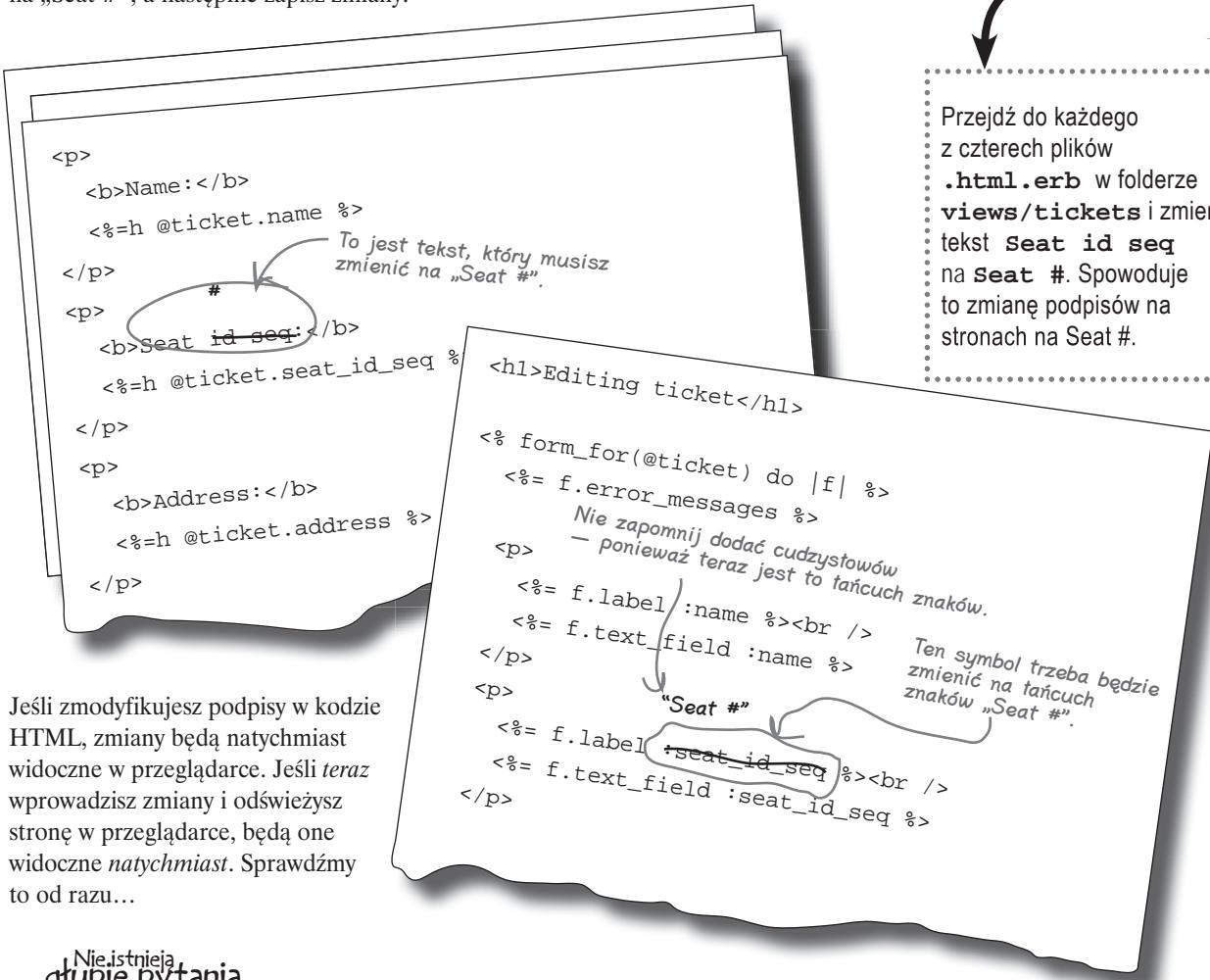
Jeśli chcemy zmienić podpisy na stronach internetowych, musimy zmodyfikować kod widoku. Cały kod widoku znajduje się w folderze *app/views*.

Pliki widoku generują strony internetowe i nazywane są *szablonami stron*. Czym zatem jest szablon strony i co zawierają te szablony?

Edycja kodu HTML w widoku

Jak zatem naprawdę wygląda szablon strony? Otwórz cztery pliki `.html.erb` z folderu `views/tickets` za pomocą edytora tekstu. Zawartość plików wygląda w dużej mierze jak kod HTML.

Chcemy zmienić podpisy dla pola `seat_id_seq` na `Seat #`. By to zrobić, odszukaj tekst „Seat id seq” we wszystkich czterech plikach, zmień go na „Seat #”, a następnie zapisz zmiany.



Jeśli zmodyfikujesz podpisy w kodzie HTML, zmiany będą natychmiast widoczne w przeglądarce. Jeśli teraz wprowadzisz zmiany i odświeżysz stronę w przeglądarce, będą one widoczne *natychmiast*. Sprawdźmy to od razu...

Nie istnieją głupie pytania

P: **Nazywacie**
`:seat_id_seq` **symbolem.**
Czym jest symbole?

Q: Symbol nieco przypomina łańcuch znaków. Łąncuch znaków otoczony jest cudzysłowem, natomiast symbol zawsze rozpoczyna się od dwukropka (`:`). Symbole wykorzystywane

są zazwyczaj w Rails do nazywania różnych rzeczy, ponieważ są nieco bardziej wydajne pod względem pamięci. W większości przypadków symbole i łańcuchy znaków mogą być wykorzystywane wymiennie.



Jazda próbna

Odśwież stronę znajdująca się pod adresem:

`http://localhost:3000/tickets/`

The screenshot shows a web application for managing tickets. At the top, there's a header "Tickets: new". Below it, a "New ticket" form with fields for Name, Seat #, Address, and Price paid. The "Name" field contains "Alan Longmuir", "Seat #" contains "1A", "Address" contains "37 Newbury Road, Ashfield, MA", and "Price paid" contains "60.0". Below the form is a "Listing tickets" table with four rows of data:

Name	Seat #	Address	Price
Alan Longmuir	1A	37 Newbury Road, Ashfield, MA	60.0
Gordon Clark	43G	17 Tudor Street, Cambridge, MA	35.0
Bobbi Lyall	54C	9 Main Street, Provincetown, RI	43.0

Teraz wszystkie podpisy brzmią „Seat #”. Dokładnie tego chcieliśmy.

Czy zauważyłeś, jak szybko zmiana pokazała się w aplikacji?

To dzięki temu, że platforma Rails zbudowana jest w języku Ruby, a kod w tym języku nie musi być komplikowany. Serwer WWW Rails może po prostu wykonać aktualny kod źródłowy.
Czy ma to jednak jakieś znaczenie?

Owszem, by wypróbować zmodyfikowany kod, musisz wykonać o wiele mniej kroków. Nie trzeba na przykład kompilować kodu ani łączyć go w pakiety i gdzieś wdrażać. Wystarczy tylko napisać kod i wykonać go. Cykl programowania w Rails jest bardzo szybki, a wprowadzenie zmian do aplikacji odbywa się również prędko.

Cykl programowania

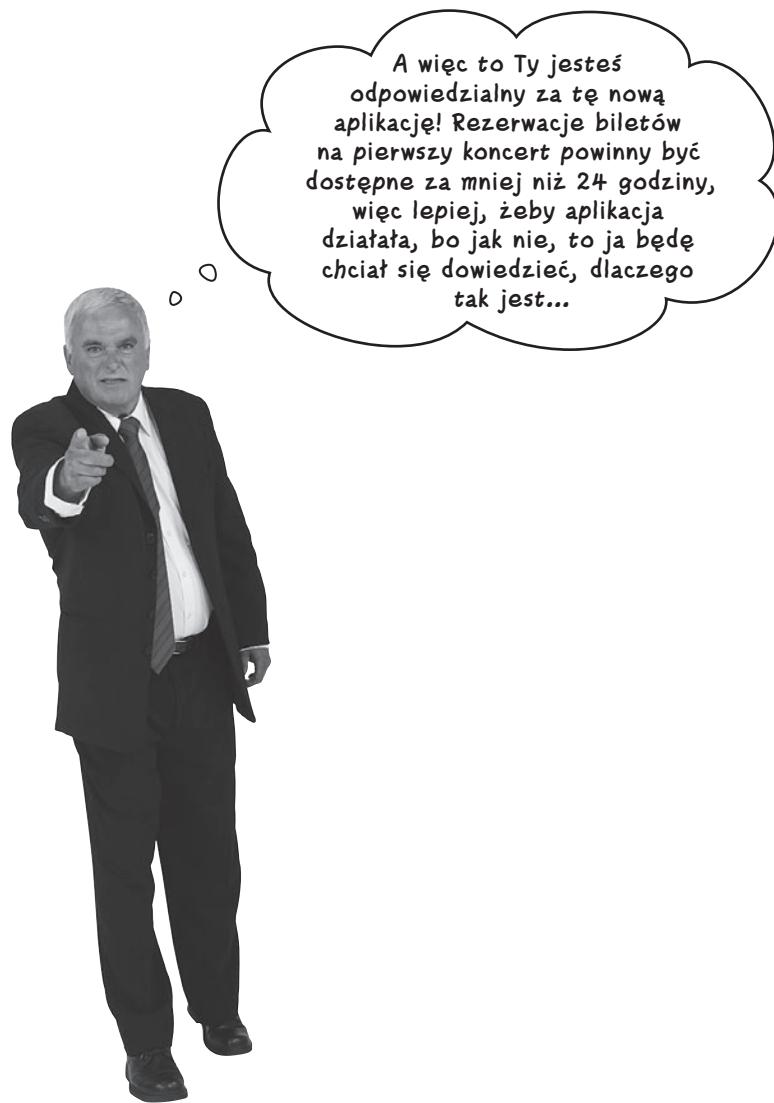
- Napisanie/poprawienie kodu
- ~~Kompilacja kodu~~
- ~~Utworzenie pakietu~~
- ~~Wdrożenie aplikacji~~
- Uruchomienie jej
- Powtórzenie

Te kroki są zbędne, kiedy programujesz w Rails.

Niedziela, godzina 8 rano

Wprowadziłes dwie poprawki, ale teraz dzwoni telefon...
Co u licha?





Aplikacja musi teraz przechować większą liczbę informacji

Wszystko było właściwie skończone do momentu, gdy Twój przyjaciel wspomniał, że należy zapisywać numery telefonów. Potrzebujemy więcej danych, a co to oznacza dla naszej aplikacji?

1 Potrzebne nam dodatkowe pole wyświetlane na każdej stronie.

Na szczęście wiemy, jak ulepszać szablon strony, więc nie powinno to stanowić problemu.

Musimy dodać takie dodatkowe pole do strony.

Listing tickets						Tickets index
Name	Seat #	Address	Price paid	Email address	Phone	
Alan Langmuir	1A	37 Newbury Road, Ashfield, MA 40.0		alan@changalang.com	555-287-0491	Show Edit Destroy
Gordon Clark	43G	17 Tudor Street, Cambridge, MA	35.0	g.clark@rollermail.com	555-547-1130	Show Edit Destroy
Bobbi Lyall	54C	9 Main Street, Provincetown, RI 43.0		b.lyall@baycity.org	555-437-6330	Show Edit Destroy
Eric Mandlark	7H	132a Mass Ave, Cambridge, MA 37.0		eric@mandlarkmail.org	555-227-9970	Show Edit Destroy
Nelly Henderson	88J	665 Tremont St, Boston, MA 44.0		nelly@byebabybabby.com	555-747-1077	Show Edit Destroy

2 Musimy przechować dodatkową kolumnę w bazie danych.

Musimy przechować dodatkową kolumnę w naszej bazie danych, jednak jak to zrobić?

Musimy dodać kolumnę „phone” do tabeli „tickets”

tickets	
name	string
seat_id_seq	string
address	text
price_paid	decimal
email_address	string
phone	string

Zaostrz ołówek



Potrzebujemy dodać kolumnę do tabeli bazy danych. Napisz poniżej, jaki **typ** skryptu wykorzystywaliśmy wcześniej w celu modyfikacji struktury bazy danych.

Zaostrz ołówek



Rozwiążanie

Miałeś napisać, jaki typ skryptu wykorzystywaliśmy wcześniej w celu modyfikacji struktury bazy danych.

Migracja

Migracja to po prostu skrypt w języku Ruby

Aby dodać kolumnę do tabeli, potrzebna nam migracja. Ale *czym* tak naprawdę jest migracja? Przyjrzyjmy się tej, która utworzyła naszą tabelę z biletami.

```
class CreateTickets < ActiveRecord::Migration
  def self.up
    create_table :tickets do |t|
      t.string :name
      t.string :seat_id_seq
      t.text :address
      t.decimal :price_paid
      t.string :email_address
      t.timestamps
    end
  end
  def self.down
    drop_table :tickets
  end
end
```

Musimy utworzyć kod przypominający ten, jednak zamiast tworzenia tabeli nasza migracja musi dodać kolumnę.

Halo? Jak niby mamy napisać kod modyfikujący tabelę? Nie wiemy, jak to zrobić!

**Musimy UTWORZYĆ kod,
ale to nie znaczy,
że musimy NAPISAĆ kod.**



Nie istniejąca grupa pytań

P: Część kodu migracji wygląda tak, jakby usuwała tabelę. Dlaczego tak jest?

O: Migracje mogą zrobić o wiele więcej, niż pokazujemy tutaj. Każda migracja może na przykład odwrócić wprowadzone przez siebie zmiany. Z tego powodu kod tworzący nową tabelę jest połączony z kodem mogącym tę tabelę usunąć. Nie musisz teraz wiedzieć więcej na ten temat.

P: Nie muszę rozumieć kodu? Czy żeby posługiwać się Rails nie należy rozumieć kodu w języku Ruby?

O: Im lepiej będziesz rozumiał Ruby, tym większą kontrolę będziesz miał nad Rails. W miarę lektury książki będziesz się dowiadywał coraz więcej o języku Ruby.

P: Jeśli migracja to tylko skrypt w języku Ruby, dlaczego muszę używać rake? Czemu nie mogę po prostu wykonać skryptu?

O: Dobre pytanie. Część Ruby zaprojektowana została tak, by bezpośrednio wykonać kod, a część nie. Migracji nie należy wykonywać w sposób bezpośredni. Powinny one być wykonywane za pomocą rake.

P: Świetnie — ale dlaczego?

O: Polecenie rake jest „sprytniejsze” od ruby. Kiedy wywołasz rake db:migrate, tak naprawdę mówisz do rake: „Upewnij się, że wszystkie migracje zostały wykonane”. rake może zadecydować, by nie wywoływać migracji,

jeśli nie ma takiej potrzeby. Ruby nie może samodzielnie podejmować takich decyzji.

P: Czy nie mogę po prostu ręcznie zmodyfikować tabeli tickets?

O: Moglibyś tak zrobić, jednak lepiej jest zarządzać strukturą bazy danych za pomocą migracji. Kiedy opublikujesz aplikację, będziesz musiał odtworzyć strukturę danych w produkcyjnej bazie danych. Jeśli skorzystasz z migracji, polecenie rake będzie w stanie dopasować struktury danych w bazie produkcyjnej do tego, czego potrzebuje Twoja aplikacja. Jeśli ręcznie zmodyfikujesz strukturę danych, rzeczy łatwo mogą się rozsynchronizować. Tak jak w większości sytuacji w Rails, jeśli będziesz się stosował do konwencji, ułatwisz sobie życie.

Rails może generować migracje

Przypomnij sobie, jak wygenerowaliśmy rusztowanie za pomocą polecenia:

```
ruby script/generate scaffold ticket name:string seat_id_seq:string
address:text price_paid:decimal email_address:string
```

generate to skrypt służący do tworzenia kodu w języku Ruby. Dobra wiadomość jest taka, że generate pisze *nie tylko* kod rusztowania. Generuje również migracje.

Założmy, że wpisałeś takie polecenie:

```
ruby script/generate migration PhoneNumber
```

Nie wpisuj go naprawdę.

Wygenerowałoby ono nowy pusty plik migracji. Moglibyśmy wtedy dodać kod w języku Ruby w celu zmodyfikowania tabeli. Problem polega na tym, że nie wiemy, jak napisać kod kończący migrację.

Co zatem możemy zrobić? I co może zrobić dla nas Rails?

Nadaj swojej migracji odpowiednią nazwę, a Rails napisze za Ciebie kod

Zauważłeś już pewnie, że nazwy są dla Rails naprawdę istotne. Kiedy tworzyliśmy rusztowanie o nazwie ticket, platforma Rails udostępniła nam aplikację pod adresem <http://localhost:3000/tickets> i wygenerowała migrację tworzącą tabelę o nazwie tickets.

Konwencje nazewnictwa są w Rails istotne, ponieważ oszczędzają Ci pracy. Tak samo jest z nazewnictwem migracji. Zamiast nadawać nowej migracji starą nazwę, spróbuj nazwać ją tak:

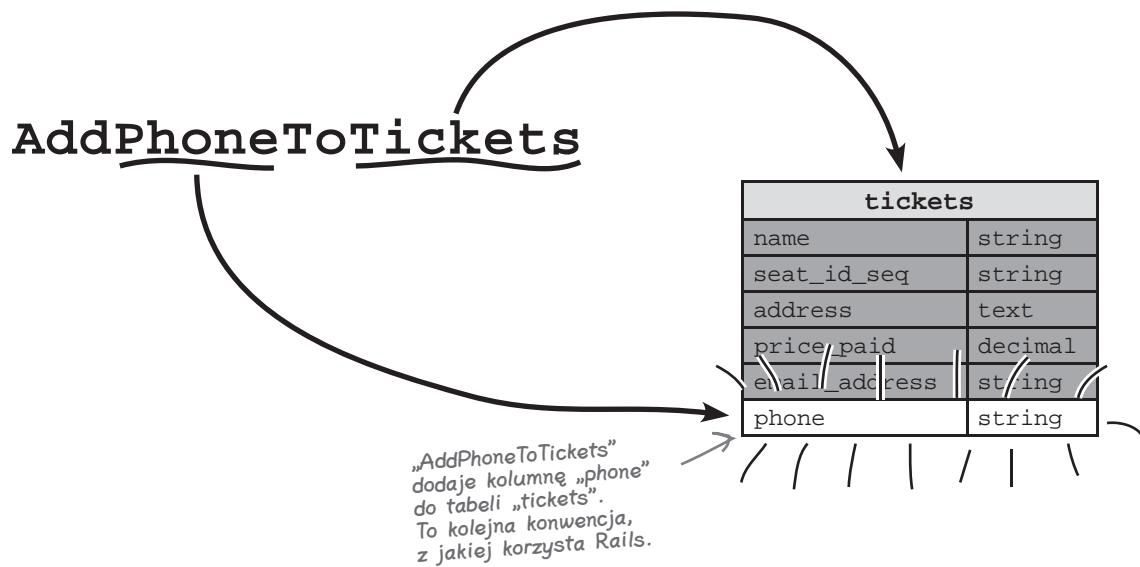
```
Plik Edycja Okno Pomoc
> ruby script/generate migration AddPhoneToTickets phone:string
```

Najważniejszą częścią jest ta nazwa.
Przyjmuje postać „Add... To...”

Dlaczego nazwa ma takie znaczenie?

Rails wie, że migracja o nazwie Add... To... będzie najprawdopodobniej dodawać określoną kolumnę do określonej tabeli, więc zamiast generować pustą migrację, którą będziesz musiał uzupełnić, **Rails napisze kod migracji za Ciebie**.

Musisz wykonanie to
polecenie.

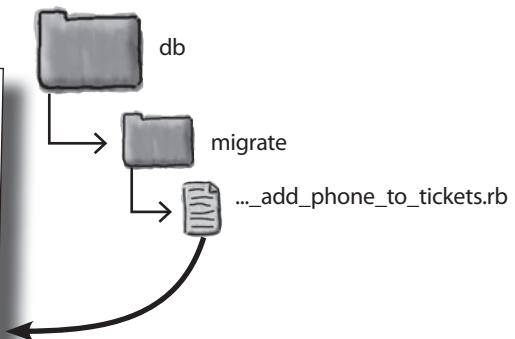


Migrację należy wykonać za pomocą rake

Oto migracja, jaką platforma Rails sprytnie dla Ciebie wygenerowała:

```
class AddPhoneToTickets < ActiveRecord::Migration
  def self.up
    add_column :tickets, :phone, :string
  end

  def self.down
    remove_column :tickets, :phone
  end
end
```



Kiedy wcześniej chcieliśmy wykonać migrację, korzystaliśmy z polecenia `rake`:

```
rake db:migrate
```

Ale czy możemy to zrobić teraz? W końcu nie chcemy, by polecenie `rake` wykonało znowu przez pomyłkę pierwszą migrację.

Data i czas mówią `rake`, które migracje należy wykonać i w jakiej kolejności

Rails zapisuje ostatnią datę i czas wszystkich wykonywanych migracji. Pozwala to poleceniu `rake` określić, które migracje zostały już wykonane, a które nie. Oznacza to, że przy każdym wykonaniu `rake db:migrate` **Rails wykona jedynie najnowsze migracje**.

Przetestujmy to. Wykonaj raz jeszcze `rake db:migrate` w celu dodania kolumny `phone` do tabeli `tickets`.

Zrób tak!

```
Plik Edycja Okno Pomoc
> rake db:migrate
```

Samą zmianą bazy danych nie wystarczy

Rusztowanie generuje kod, co jest świetne, ponieważ pozwala nam szybko zacząć. Jednak wadą takiego rozwiązania jest to, że gdy kod zostanie już wygenerowany, uaktualnienie go należy do programisty.

Dodaliśmy właśnie atrybut phone do bazy danych. Jednak ponieważ formularze zostały już wygenerowane przez rusztowanie, nie pobiorą one nowego pola z telefonem automatycznie. Musimy zatem wrócić do szablonów i dodać do nich odniesienia do numeru telefonu:

```
<p>
  <%= f.label :email_address %><br />
  <%= f.text_field :email_address %>

</p>
<p>
  <%= f.label :phone %><br />
  <%= f.text_field :phone %>

</p>
<p>
  <%= f.submit "Update" %>
</p>
```

To znajduje się wewnątrz pliku edit.html.erb.

```
<p>
  <b>Email address:</b>
  <%=h @ticket.email_address %>
</p>
<p>
  <b>Phone:</b>
  <%=h @ticket.phone %>
</p>
```

To jest kod dodany do pliku show.html.erb.

```

<p>
  <%= f.label :email_address %><br />
  <%= f.text_field :email_address %>
</p>

<p>
  <%= f.label :phone %><br />
  <%= f.text_field :phone %>
</p>
<p>

```

A to do pliku new.html.erb
— strony zawierającej
formularz służący
do tworzenia rezerwacji.

I wreszcie do pliku index.html.erb
generującego listę wszystkich biletów.
Kod musimy tutaj dodać w dwóch
miejscach: w nagłówku kolumny
orzaz w każdym wierszu tabeli.

Email address	Phone
<% for ticket in @tickets %>	<% h ticket.name %>
<% h ticket.seat_id_seq %>	<% h ticket.address %>
<% h ticket.price_paid %>	<% h ticket.email_address %>
<% h ticket.phone %>	<% link_to 'Show', ticket %>

Nie istniejąca
grupie pytania

P: Dlaczego w niektórych miejscach
jest napisane <%=h ... %>?

C: oznacza h?

O: h to metoda pomocnicza. Metody
pomocnicze wykorzystywane są najczęściej
do rzeczy takich, jak formatowanie
danych wyjściowych. Metoda pomocnicza
h znosi znaczenie specjalne pewnych
znaków w polu, takich jak < czy &. Zapobiega to przesłaniu za pomocą strony

tekstu zawierającego JavaScript czy inny
potencjalnie niebezpieczny kod.

P: Dlaczego w niektórych miejscach
użyte zostały łańcuchy znaków,
a w innych symbole?

O: łańcuchy znaków wykorzystywane są
w szablonach stron tam, gdzie wymagany
jest fragment zwykłego tekstu. Symbole
(słowa rozpoczynające się od dwukropków)
są często wykorzystywane w podpisach.

P: Dlaczego?

O: Symbole są wydajne pod względem
pamięci i większość metod (takich jak
f.label) przyjmujących parametry lubi
otrzymywać symbole zamiast łańcuchów
znaków. Jednak w większości przypadków
metody Rails pozwalają opcjonalnie
wykorzystywać łańcuchy znaków zamiast
symboli, jeśli są one łatwiejsze
do sformatowania.

Szef powraca



Ćwiczenie (nieco dłuższe)

Szef jest zadowolony z nowego sposobu działania aplikacji i teraz chce obok sprzedaży biletów zapisywać również imprezy. Oto struktura danych imprez:

Event (impreza):

artist — wykonawca (łańcuch znaków)

description — krótka biografia (tekst)

price_low — najtańsze bilety (liczba dziesiętna)

price_high — cena sprzedaży biletu (liczba dziesiętna)

event_date — kiedy się odbywa (data)

Jakie polecenie należy wpisać w konsoli, by utworzyć rusztowanie dla danych imprez?

.....
.....

Jakie polecenie wpisałbyś w celu utworzenia tabeli events w bazie danych?

.....

Szef chce, by podpisy na stronach brzmiały następująco: „Prices from” dla price_low, „To” dla price_high oraz „Date” dla event_date. Będziesz musiał zmodyfikować cztery szablony w celu wprowadzenia zmian. Wypisz zmiany dla pokazanego poniżej szablonu strony *new.html.erb*:

```
<h1>New event</h1>
<% form_for(@event) do |f| %>
  <%= f.error_messages %>
  <p>
    <%= f.label :artist %><br />
    <%= f.text_field :artist %>
  </p>
  <p>
    <%= f.label :description %><br />
    <%= f.text_area :description %>
  </p>
  <p>
    <%= f.label :price_low %><br />
    <%= f.text_field :price_low %>
  </p>
  <p>
    <%= f.label :price_high %><br />
    <%= f.text_field :price_high %>
  </p>
  <p>
    <%= f.label :event_date %><br />
    <%= f.date_select :event_date %>
  </p>
  <p>
    <%= f.submit "Create" %>
  </p>
<% end %>
<%= link_to 'Back', events_path %>
```

Plik *new.html.erb*

Podaj nazwy trzech pozostałych szablonów z katalogu *app/views/events*, jakie trzeba będzie zmienić.

Kolejne wymagania



Ćwiczenie
(nieco dłuższe)

Rozwiązanie

Szef jest zadowolony z nowego sposobu działania aplikacji i teraz chce obok sprzedaży biletów zapisywać również imprezy. Oto struktura danych imprez:

Event (impreza):
artist — wykonawca (łańcuch znaków)
description — krótka biografia (tekst)
price_low — najtańsze bilety (liczba dziesiętna)
price_high — cena sprzedaży biletu (liczba dziesiętna)
event_date — kiedy się odbywa (data)

Jakie polecenie należy wpisać w konsoli, by utworzyć rusztowanie dla danych imprez?

```
ruby script/generate scaffold event artist:string description:text price_low:decimal  
.....  
price_high:decimal event_date:date  
.....
```

Jakie polecenie wpisałbyś w celu utworzenia tabeli events w bazie danych?

```
rake db:migrate  
.....
```

Szef chce, by podpisy na stronach brzmiały następująco: „Prices from” dla price_low, „To” dla price_high oraz „Date” dla event_date. Będziesz musiał zmodyfikować cztery szablony w celu wprowadzenia zmian. Wypisz zmiany dla pokazanego poniżej szablonu strony *new.html.erb*:

```

<h1>New event</h1>
<% form_for(@event) do |f| %>
  <%= f.error_messages %>
  <p>
    <%= f.label :artist %><br />
    <%= f.text_field :artist %>
  </p>
  <p>
    <%= f.label :description %><br />
    <%= f.text_area :description %>
  </p>
  <p>      :prices_from ←
    <%= f.label :price_low %><br />
    <%= f.text_field :price_low %>
  </p>
  <p>      :to ←
    <%= f.label :price_high %><br />
    <%= f.text_field :price_high %>
  </p>
  <p>      :date ←
    <%= f.label :event_date %><br />
    <%= f.date_select :event_date %>
  </p>
  <p>
    <%= f.submit "Create" %>
  </p>
<% end %>
<%= link_to 'Back', events_path %>
```

Plik *new.html.erb*

„Prices from” także zadziała, ale użycie symbolu jest lepsze

Lub „To”

Lub „Date”

Podaj nazwy trzech pozostałych szablonów z katalogu *app/views/events*, jakie trzeba będzie zmienić.

edit.html.erb, show.html.erb oraz index.html.erb



Jazda próbna

Aplikacja ma teraz na stronach z biletami komplet informacji kontaktowych:

The screenshot shows a web application interface for managing tickets. On the left, a modal window titled "Editing ticket" is open, showing fields for Name (Alan Longmuir), Seat # (1A), Address (37 Newbury Road, Ashfield, MA), Price paid, Email address, and Phone. On the right, a list of tickets is displayed in a table format with columns for Name, Seat #, Address, Price paid, Email address, and Phone. Each row has links for Show, Edit, and Destroy.

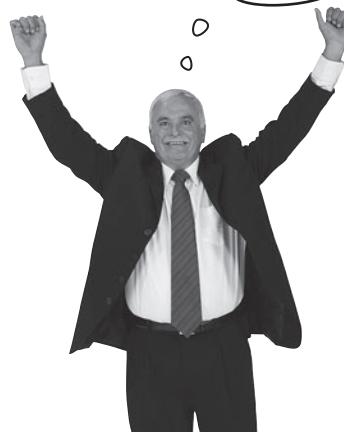
Name	Seat #	Address	Price paid	Email address	Phone
Alan Longmuir	1A	37 Newbury Road, Ashfield, MA	60.0	alan@shangalang.com	555-267-0488
Gordon Clark	43D	17 Tudor Street, Ashfield, MA	60.0	gclark@rollermail.com	555-267-0488
Bobbie Lyall	54C	9 Main Street, Provincetown, RI	35.0	blyall@baycity.org	555-947-1130
Eric Mancuso	7H	1326 Mass Ave, Cambridge, MA	37.0	eric@mahanmail.org	555-437-6338
Nelly Henderson	68J	865 Tremont St, Boston, MA	64.0	nelly@byebyebaby.com	555-227-9990

Zapisywane są również wszystkie imprezy:

The screenshot shows a web application interface for managing events. On the left, a modal window titled "Editing event" is open, showing fields for Artist (Lyle Lovett), Description (American singer-songwriter and actor.), Prices from (\$1.0), To (\$1.0), Date (2009-10-22), and Create. On the right, a list of events is displayed in a table format with columns for Artist, Description, Prices from, To, Date, and Edit / Back. Each row has links for Edit and Back.

Artist	Description	Prices from	To	Date	Edit / Back
Lyle Lovett	American singer-songwriter and actor.	\$1.0	\$1.0	2009-10-22	Edit / Back
Trey Anastasio					
Great Big Sea					
Boston Symphony					

Jest świetnie!
Wygląda na to,
że nas
uratowałeś!



Dlaczego Rails mówi do mnie po angielsku?

Prawdopodobnie od dłuższej chwili nurtuje Cię pewien problem: *dłaczego aplikacje Rails są w języku angielskim* i co można zrobić, by *używały innego języka*? W trakcie pracy w żadnym miejscu nie definiowałaś przecież nagłówków stron czy podpisów przycisków, a tymczasem wszystkie są w języku angielskim. Dlaczego?

Cóż, odpowiedź jest dość prosta. Tworzenie aplikacji w Rails wymaga minimalnego nakładu pracy użytkownika, a **pewne zdefiniowane standardowe operacje i elementy są częścią samej platformy**. Wystarczy, że podasz nazwę aplikacji i za pomocą rusztowania utworzysz odpowiednią strukturę danych, a platforma Rails wygeneruje dla Ciebie pliki widoku ze standardowymi tekstami, ale... *wszystkie te teksty są w języku angielskim*.

Nawet jeśli zaraz na początku spróbujesz nadać aplikacji oraz poszczególnym polom tabeli bazy danych polskie nazwy, uzyskasz co najwyżej przedziwnego potwora językowego (a do tego będziesz się musiał zdrowo nakombinować z automatyczną odmianą wyrazów w liczbie mnogiej — kto by się domyślił, że liczba mnoga od „bilet” to „bilets”, prawda?):

Nazwisko	Numer miejsca	Adres	Cena	Adres email	Telefon
Amadeusz Leśny	1A	ul. Leśny Dukt 15, Zielony Las	60.0	madek@skrzynkapoczta.org	Show Edit Destroy
Teodor Niedźwiedź	43G	ul. Czosnkowa 15, Bór Wielki	35.0	teodor@all-bears.com	Show Edit Destroy

Na szczęście nie oznacza to, że tworzenie aplikacji Rails w innym języku jest niemożliwe. Jest możliwe — wymaga jedynie nieco większego nakładu pracy.

Kluczem są pliki widoku

Pewnie nie zdajesz sobie z tego sprawy, ale potrafisz już samodzielnie wprowadzić odpowiednie zmiany. Pamiętasz, jak zmienialiśmy podpisy pól formularza w aplikacji służącej do sprzedaży biletów? No właśnie. Co stoi na przeszkodzie, by podpisy te zmienić na teksty w innym języku, takim jak polski?

Zrób tak!

Otwórz w edytorze tekstu wszystkie pliki widoku — *edit.html.erb*, *index.html.erb*, *new.html.erb* oraz *show.html.erb*. W podobny sposób jak wcześniej zmień wszystkie podpisy formularzy na ich odpowiedniki w języku polskim. Pamiętaj, że polskie teksty musisz także wprowadzić do standardowych elementów HTML tworzących stronę — *<h1>* czy *<th>* — oraz przycisków i odnośników generowanych przez Rails za pomocą metod pomocniczych, takich jak *link_to* oraz *f.submit*.

Uczymy Rails języków obcych

A oto kilka przykładów zmian, jakie będziesz musiał wprowadzić:

The diagram shows two code snippets side-by-side. On the left is the code for `index.html.erb`, which displays a list of tickets with columns for name and place number. An annotation above it says: "To fragment strony `index.html.erb`. Pamiętaj o zmianie tekstów elementów HTML!" (This is a fragment of the page `index.html.erb`. Remember to change the text of the HTML elements!). On the right is the code for `new.html.erb`, which contains form fields for name, address, price, email, and phone, along with a submit button and a link to the main page. An annotation above it says: "To część kodu strony `new.html.erb`. Pokazuje, jak należy zmienić podpisy pól formularza, przycisku oraz odnośnika." (This is part of the code for `new.html.erb`. It shows how to change the labels for the form fields, the button, and the link). A small car icon is positioned below the snippets.

```
<h1>Lista biletów</h1>


| Imię i nazwisko | Numer miejsca |
|-----------------|---------------|
|-----------------|---------------|


```

```
<p>
  <%= f.label :telefon %><br />
  <%= f.text_field :phone %>
</p>
<p>
  <%= f.submit "Utwórz" %>
</p>
<% end %>
<%= link_to 'Powrót do strony głównej', tickets_path %>
```

Jazda próbna

Po wprowadzeniu zmian odśwież stronę `http://localhost:3000/tickets`. Teraz możesz już sprzedawać swoją aplikację właścielowi jakiejś hali koncertowej w Polsce i nie martwić się o to, że zatrudnione tam osoby nie będą potrafiły jej obsługiwać!

The screenshot shows two browser windows. The top window displays a list of tickets with columns for name, place number, address, price, email, and phone. The bottom window shows a form for creating a new ticket, with fields for name, place number, address, price, email, and phone, along with a submit button and a link to the main page.

Imię i nazwisko	Numer miejsca	Adres	Cena	Adres e-mail	Telefon
Amadeusz Leśny	1A	ul. Leśny Dukt 15, Zielony Las	60.0	mak@skrzynkapocztowa.org	0329078678
Teodor Niedźwiedź	43G	ul. Czosnkowa 15, Bór Wielki	35.0	teodor@all-bears.com	0509789098
Teofil Królikowski	54C	ul. Polna 19, Brzozowo	43.0	rabbit@adresemail.pl	0719876543
Eryk Kudłaty	7H	ul. Browarna 8, Wielkie Miasto	37.0	eryk@thehairynes.com	0699726354
Edward Poziomka	88J	al. Kasztanowa 115, Mroki	64.0	dzio@stforek.org	0661234567

[Nowy bilet](#)

[Zakończono](#)

Imię i nazwisko: Teodor Niedźwiedź
Numer miejsca: 43G
Adres: ul. Czosnkowa 15, Bór Wielki
Cena: 35.0
Adres e-mail: teodor@all-bears.com
Telefon: 0509789098
[Edytuj](#) | Powrót do strony głównej

[Zakończono](#)

Kontynuacja...

Tickets: edit - Mozilla Firefox

Plik Edycja Widok Historia Zakładki Narzędzia Pomoc

http://localhost:3000/tickets/2/edit

Edycja biletu

Imię i nazwisko

Numer miejsca

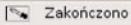
Adres

Cena

Adres e-mail

Telefon

[Pokaz](#) | [Powrót do strony głównej](#)

 Zakończono

Kontynuacja...

The screenshot shows a Mozilla Firefox browser window with the title "Tickets: new - Mozilla Firefox". The address bar displays "http://localhost:3000/tickets/new". The page content is a form titled "Nowy bilet" (New ticket). The form fields include:

- Imię i nazwisko (Name): An input field.
- Numer miejsca (Seat number): An input field.
- Adres (Address): A large text area for entering address details.
- Cena (Price): An input field.
- Adres e-mail (Email address): An input field.
- Telefon (Phone): An input field.
- Utwórz** (Create): A button labeled "Utwórz" (Create).
- [Powrót do strony głównej](#) (Return to main page): A link labeled "Powrót do strony głównej" (Return to main page).

The status bar at the bottom of the browser window shows "Zakończono" (Completed).

Żeby nie wprowadzać dodatkowego zamieszania, wszystkie kolejne aplikacje w książce przedstawione będą w oryginalnych, angielskich wersjach językowych, w takiej formie, jak generowane są przez Rails. Nie powinno to jednak być dla Ciebie żadnym problemem — w końcu **wiesz już, co należy zmienić, by zmodyfikować język aplikacji Rails!**

Koncert jest wyprzedany!

Aplikacja przez cały tydzień działa idealnie, a w następny piątek wszystkie miejsca w hali koncertowej są wyprzedane.



CELNE SPOSTRZEŻENIA

- Rails korzysta z architektury Model-Widok-Kontroler, zwanej również architekturą MVC — od angielskich odpowiedników tych terminów.
- Rails generuje osobne foldery dla kodu modelu, widoku oraz kontrolera.
- Wszystkie zmiany wprowadzane do aplikacji można zobaczyć natychmiast po ich zapisaniu i odświeżeniu strony w przeglądarce. To dzięki temu, że platforma Rails zbudowana jest w języku Ruby, a kod w tym języku nie musi być kompilowany.

- Zmiany w strukturze tabel można wprowadzić za pomocą migracji. By wygenerować_migration, która dodaje kolumnę do tabeli, użyj następującego polecenia:

```
ruby script/generate migration  
Add<kolumna>To<tabela>  
<kolumna>:<typ danych>
```

- Migrację wykonuje się za pomocą polecenia:

```
rake db:migrate
```



Niezbędnik programisty Rails

Masz za sobą rozdział 1. i teraz
do swojego niezbędnika programisty
Rails możesz dodać umiejętność
tworzenia aplikacji Rails.

Narzędzia Rails

`rails` nazwa-aplikacji

Tworzy aplikację

`ruby script/server`

Uruchamia aplikację

`ruby script/generate scaffold...`

Generuje jądro CRUD dla modelu

`ruby script/generate migration`

Generuje migracje zmieniające strukturę bazy danych

`rake db:migrate`

Wykonuje nową migrację na bazie danych

2. Poza rusztowaniem

Aplikacje Rails — stworzone, by nimi zarządzać

Pewnie będzie w szoku,
kiedy zauważysz, że użyłam
dopalacza...



Co tak naprawdę dzieje się w Rails? Widziałeś już, jak rusztowania generują mnóstwo kodu i pomagają pisać aplikacje internetowe w sposób niesamowicie szybki, ale co, jeśli pragniesz czegoś innego? W tym rozdziale zobaczyś, jak można **przejąć kontrolę** nad programowaniem w Rails, i będziesz miał okazję zajrzeć pod maskę tej platformy. Przekonasz się, w jaki sposób Rails decyduje o tym, który **kod** należy wykonać, jak **dane** wczytywane są z bazy danych i jak generowane są **strony internetowe**. Pod koniec rozdziału będziesz w stanie publikować dane tak, jak **sam** zechcesz.

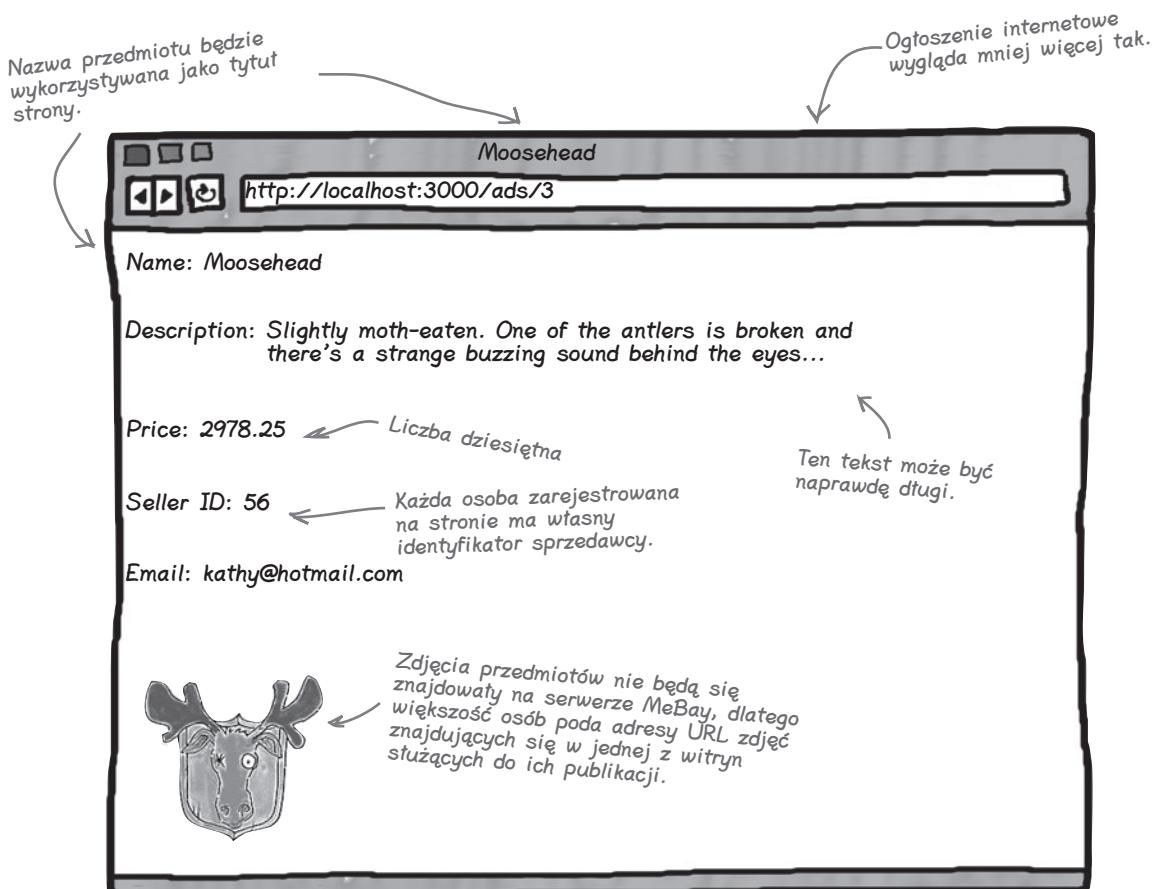
MeBay Inc. potrzebuje Twojej pomocy

MeBay Inc. to firma handlowa pomagająca ludziom sprzedawać ich niechciane przedmioty w Internecie. Firmie potrzebna jest nowa wersja witryny i to Ty będziesz musiał jej pomóc.

By umieścić ogłoszenie na stronie, sprzedający dzwoni do MeBay i podaje swój identyfikator oraz charakterystykę przedmiotu, który chce sprzedać. MeBay posiada własny system wprowadzania danych, a Twoja aplikacja będzie potrzebna do publikowania ogłoszeń z MeBay w Internecie.

MeBay chce przechowywać ogłoszenia w bazie danych

Wszystkie ogłoszenia zawierają te same typy informacji, a MeBay chce je przechować w bazie danych. Dane będą wstawiane do tabel, które utworzysz przy okazji budowania aplikacji. Potrzebne będzie coś takiego:



Zaostrz ołówek



Założmy, że planowałeś użyć rusztowania Rails, by stworzyć tę witrynę. Wypełnij luki w poniższym diagramie architektury aplikacji.

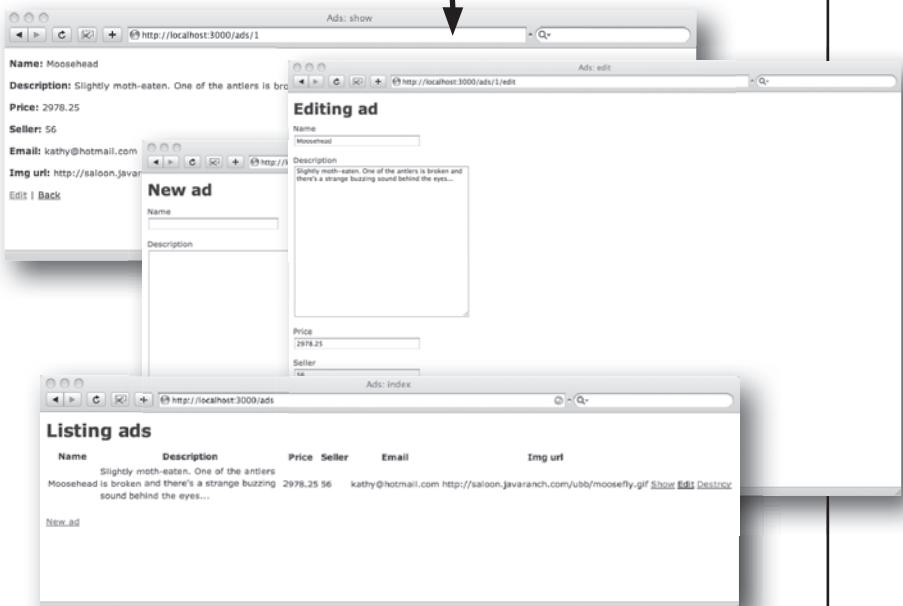
Najpierw należy utworzyć nową aplikację Rails o nazwie mebay za pomocą następującego polecenia:



Model bazy danych



zawiera logikę aplikacji



składa się ze stron internetowych, które pozwalają użytkownikowi na

oraz danych.

Czy użycie rusztowania dla tej strony wiąże się z jakimiś problemami?

-
-
-
-
-

Rusztowania mają swoje ograniczenia

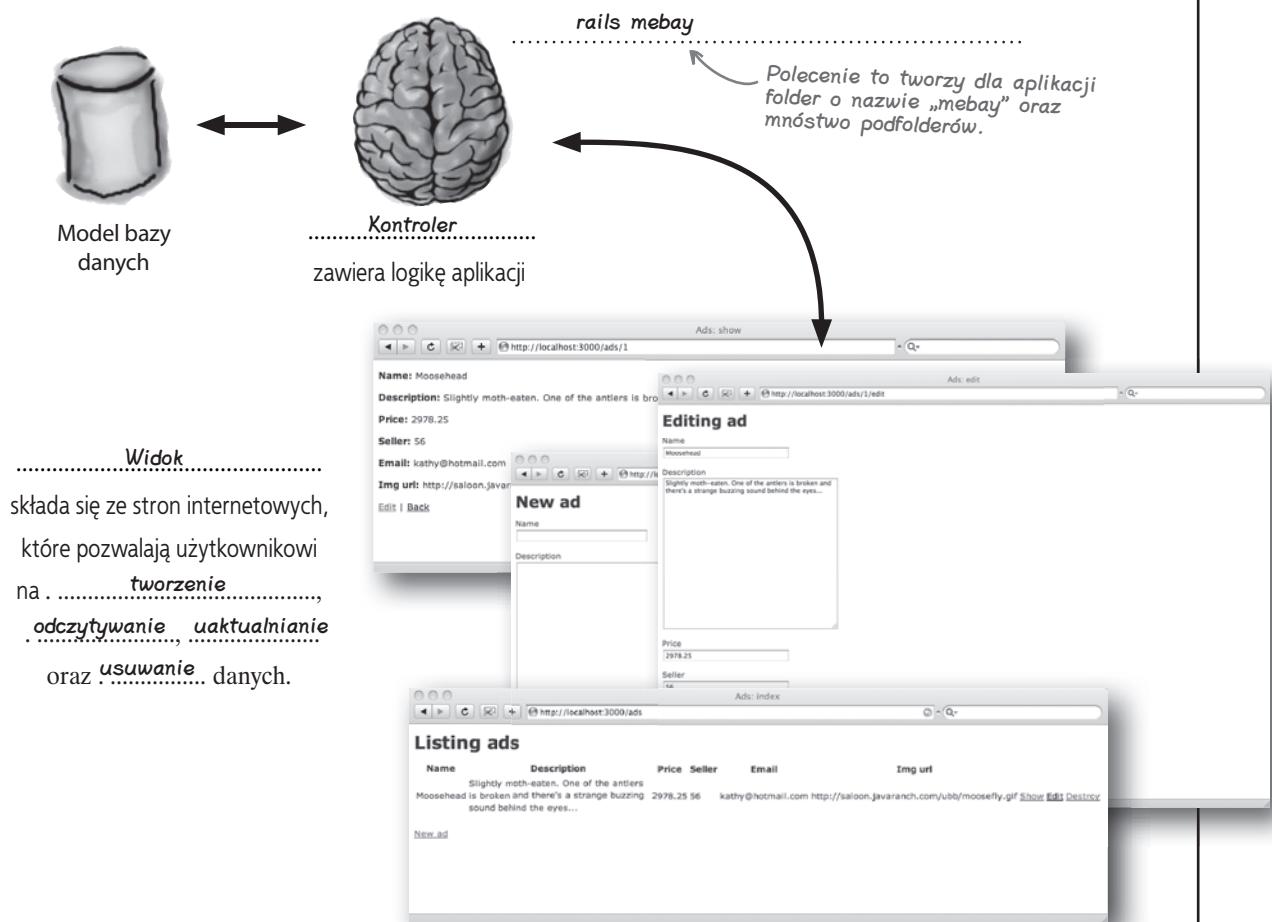
Zaostrz ołówek



Rozwiążanie

Założymy, że planowałaś użyć rusztowania Rails, by stworzyć tę witrynę. Wypełnij luki w poniższym diagramie architektury aplikacji.

Najpierw należy utworzyć nową aplikację Rails o nazwie mebay za pomocą następującego polecenia:



Czy użycie rusztowania dla tej strony wiąże się z jakimiś problemami?

Problem polega na tym, że rusztowanie generuje kod pozwalający użytkownikowi na edycję danych, a MeBay chce jedynie publikować ogłoszenia. Firma nie chce, by ktoś zmieniał ceny i szczegóły ofert. Wszystkie dane będą wprowadzane przez własne systemy MeBay, dlatego — przynajmniej na razie — wystarczające będzie samo wyświetlanie.

Rusztowanie robi O WIELE za dużo

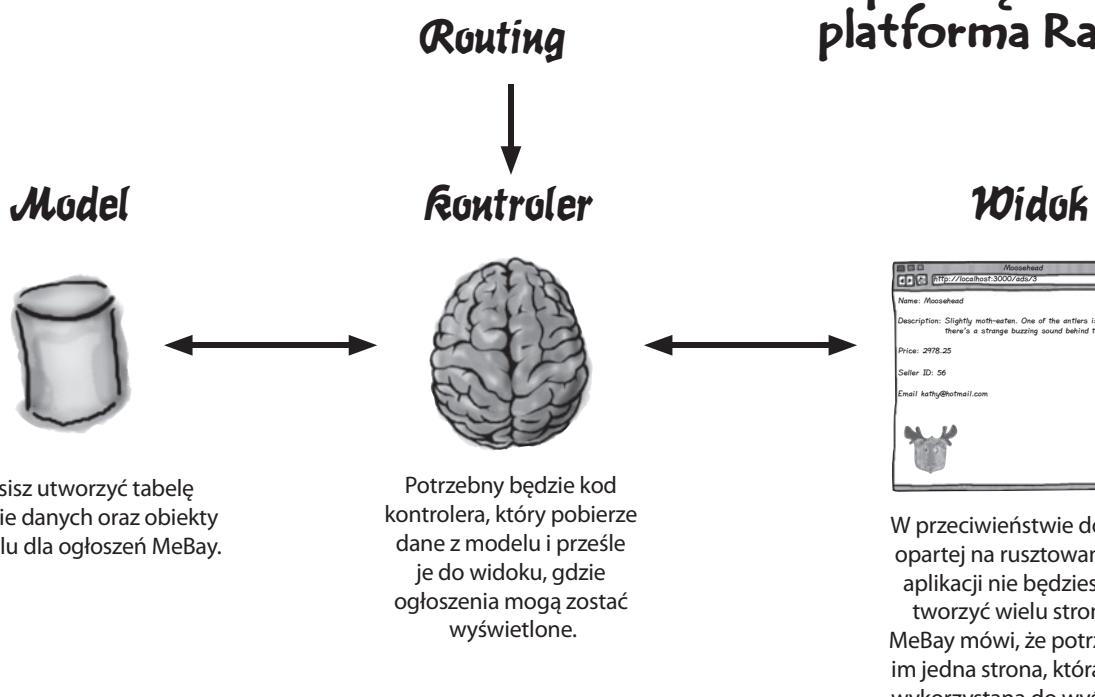
MeBay potrzebuje aplikacji, która robi *mnóstwo*, niż mogłyby robić aplikacje z rusztowaniem. Rusztowanie jest świetne, ale niektóre aplikacje są tak proste, że czasami lepiej jest budować je ręcznie.

Dlaczego tak jest? No cóż, jeśli sam piszesz kod, aplikacja będzie **prostsza i łatwiejsza** do utrzymania. Wadą takiego rozwiązania jest to, że aby ręcznie zbudować aplikację Rails, musisz zająć się *pod maską* i zrozumieć, jak naprawdę działa Rails.

Zacznijmy od przyjrzenia się temu, jaki kod będzie nam niezbędny na potrzeby aplikacji MeBay:

Musisz powiązać adresy URL w Internecie z kodem swojej aplikacji.

By zbudować aplikację bez rusztowania, musisz zrozumieć, jak naprawdę działa platforma Rails.



który kod napiszesz zatem jako pierwszy?

Zaczynamy od wygenerowania modelu MeBay...

Dobrym pomysłem jest rozpoczęcie od utworzenia kodu **modelu**, ponieważ struktura danych w modelu wpływa zarówno na kontroler, jak i na widok.

Tworzenie kodu modelu jest bardzo podobne do tworzenia rusztowania. Tak naprawdę jedną różnicą jest to, że zastępujesz słowo „scaffold” słowem „model”, jak poniżej:

Modele mają nazwy w liczbie pojedynczej — w tym przypadku „ad” (ogłoszenie), nie „ads” (ogłoszenia).

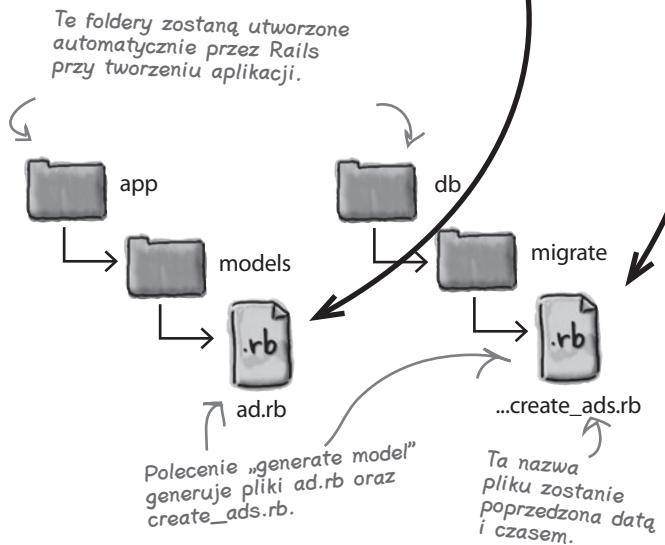
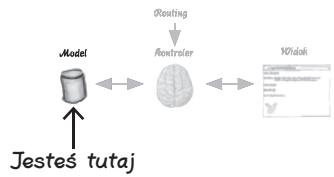
```
Plik Edycja Okno Pomoc
> ruby script/generate model ad name:string description:text
  price:decimal seller_id:integer email:string img_url:string
```

Polecenie generujące model tworzy dwa kluczowe skrypty wewnątrz podkatalogów *app* oraz *db*:

- **klasę modelu** (*app/models/ad.rb*) oraz
- **migrację danych** (*db/migrate/..._create_ads.rb*).

Migracja to skrypt języka Ruby, który łączy się z bazą danych i tworzy tabelę dla ogłoszeń. By wykonać ten skrypt i utworzyć tabelę, będziemy musieli użyć *rake*.

Dokładnie tak, jak zrobiliśmy to w rozdziale 1. Polecenie „rake” wie, które migracje należy wykonać w oparciu o ich datę i czas.



...a następnie utworzymy tabelę za pomocą polecenia rake

By utworzyć tabelę, będziemy musieli wywołać migrację za pomocą polecenia `rake db:migrate`:

Plik Edycja Okno Pomoc

```
> rake db:migrate
```

Pamiętaj, polecenie `rake db:migrate` tworzy tabelę w bazie danych za pomocą skryptu `..._create_ads.rb`, który utworzyliśmy przed chwilą z modelem.

Gdybyś jednak przyjrzał się dokładnie utworzonej tabeli, zobaczyłbyś dziwną rzecz. Rails *bez pytania* tworzy trzy dodatkowe kolumny w tabeli.

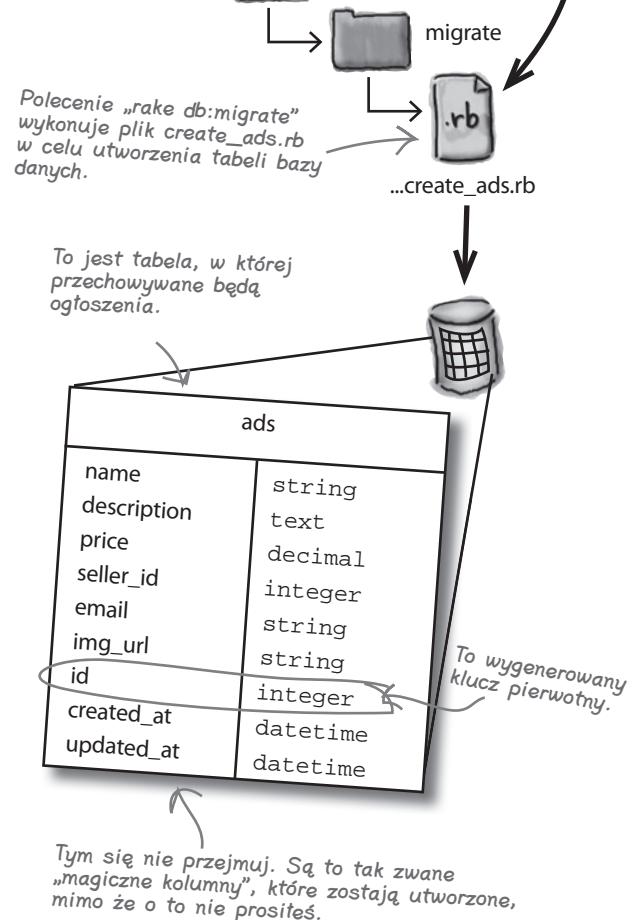
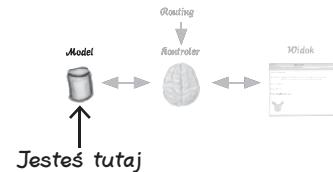
Są to „magiczne kolumny”: `id` (identyfikator), `created_at` (utworzone...) oraz `updated_at` (uaktualnione...). Kolumna `id` jest wygenerowanym kluczem pierwotnym, natomiast kolumny `created_at` oraz `updated_at` zapisują, kiedy dane zostały wprowadzone lub uaktualnione.

Zrób tak!

Polecenie `rake` tworzy dla Ciebie tabelę w bazie danych, jednak *nie wypełnia* tej tabeli danymi, z którymi mógłbyś eksperymentować.

Zanim przejdziemy dalej, potrzebne Ci będą jakieś dane w tabeli. Na szczęście uprzejmie ekipa z MeBay Inc. zostawiła kopię swoich danych testowych na naszym serwerze FTP. Pod adresem `ftp://ftp.helion.pl/przykłady/hfor.zip` — wśród przykładów znajdziesz plik `/mebay/db/development.sqlite3` zawierający potrzebne Ci dane. Zastąp nim plik znajdujący się w folderze `db` aplikacji mebay.

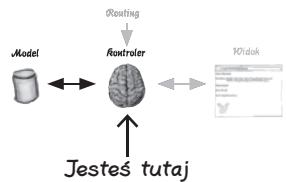
Pamiętaj, by to zrobić, bo inaczej później będziesz miał problemy.



Ale co z kontrolerem?

Sam model nie bardzo nam się do czegoś przyda. Potrzebny nam będzie kod wykorzystujący dane tworzone przez model — jest to zadanie dla **kontrolera**.

Tak jak rusztowania oraz modele, kontrolery także mają swoje własne generatory. Użyj poniższego polecenia **generate controller** w celu wygenerowania pustej klasy kontrolera:



Kontrolery mają nazwy w liczbie mnogiej.

```
Plik Edycja Okno Pomoc
> ruby script/generate controller ads
```

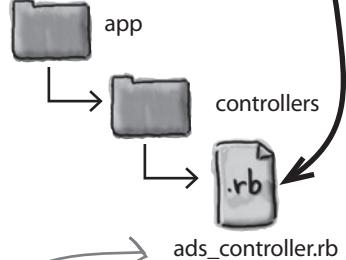
Polecenie to generuje **plik klasy** dla kontrolera: **app/controllers/ads_controller.rb**. Jeśli otworzysz plik za pomocą edytora tekstu, znajdziesz w nim kod w języku Ruby, taki jak poniższy:

```
To oznacza początek
kodu kontrolera.
Nazwa
kontrolera.

To oznacza: „Jest typem
kontrolera aplikacji”.
Pamiętaj – nazwa pliku i nazwa
kontrolera są podobne, jednak
kontroler wykorzystuje notację
CamelCase do rozdzielenia słów,
natomiast w nazwie pliku słowa
rozdziela znak „_”.
Tutaj trafi logika
aplikacji.

class AdsController < ApplicationController
end

Koniec kodu
kontrolera.
```



Za kilka stron wróćmy do tego, jaki kod należy dodać do tej klasy. Na razie cieszymy się z tego, że nie musieliszy pisać żadnego kodu w składni języka Ruby czy Rails.



Ciekawostki

Notacja CamelCase oznacza wykorzystywanie w identyfikatorach składających się z więcej niż jednego słowa wielkich liter pozwalających na rozróżnienie poszczególnych słów.

Nie istniejąca
grupie pytania

P: Czy polecenie `rake db:migrate` zawsze dodaje magiczne kolumny?

O: Dokładnie tak.

P: Nawet jeśli tabela utworzona zostaje przez rusztowanie?

O: Tak. Gdybyś przestudiował tabelę bazy danych z poprzedniego rozdziału, zobaczyłbyś w niej magiczne kolumny.

P: Czy mogę w jakiś sposób otworzyć bazę danych i przeanalizować tabele?

O: Tak — ale będzie Ci do tego potrzebne narzędzie. Dodatek do przeglądarki Firefox o nazwie SQLite Manager otwiera i odczytuje pliki `sqlite3` wykorzystywane przez Rails.

P: Zauważylem, że w poleceniu `generate model` użyliśmy `ad`, natomiast w poleceniu `generate controller` — `ads`. Czy to było celowe?

O: Tak. W Rails wszystkie modele mają nazwy w liczbie pojedynczej, natomiast kontrolery i tabele — w liczbie mnogiej. Oznacza to, że kiedy użyliśmy polecenia w celu wygenerowania modelu, zastosowaliśmy liczbę pojedynczą — `ad`, natomiast w poleceniu generującym kontroler występuała liczba mnoga — `ads`.

P: Jak ważne jest odpowiednie zastosowanie tej konwencji?

O: Bardzo ważne! Rails zależy od tych konwencji, dlatego kluczową sprawą jest, byś i Ty ich przestrzegał. Jeśli tego nie zrobisz, Rails może nie być w stanie odpowiednio skonfigurować dla Ciebie aplikacji i niektóre rzeczy mogą nie działać. Życie będzie o wiele łatwiejsze, jeśli będziesz przestrzegać konwencji oczekiwanych przez Rails.

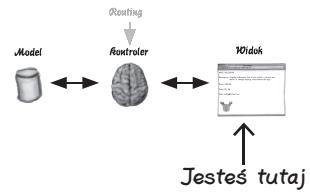
Modele mają
nazwy w liczbie
pojedynczej,
natomiast
kontrolery
i tabele
— w liczbie
mnożej.

Utworzyliśmy model i kontroler, teraz przejdźmy do widoku...

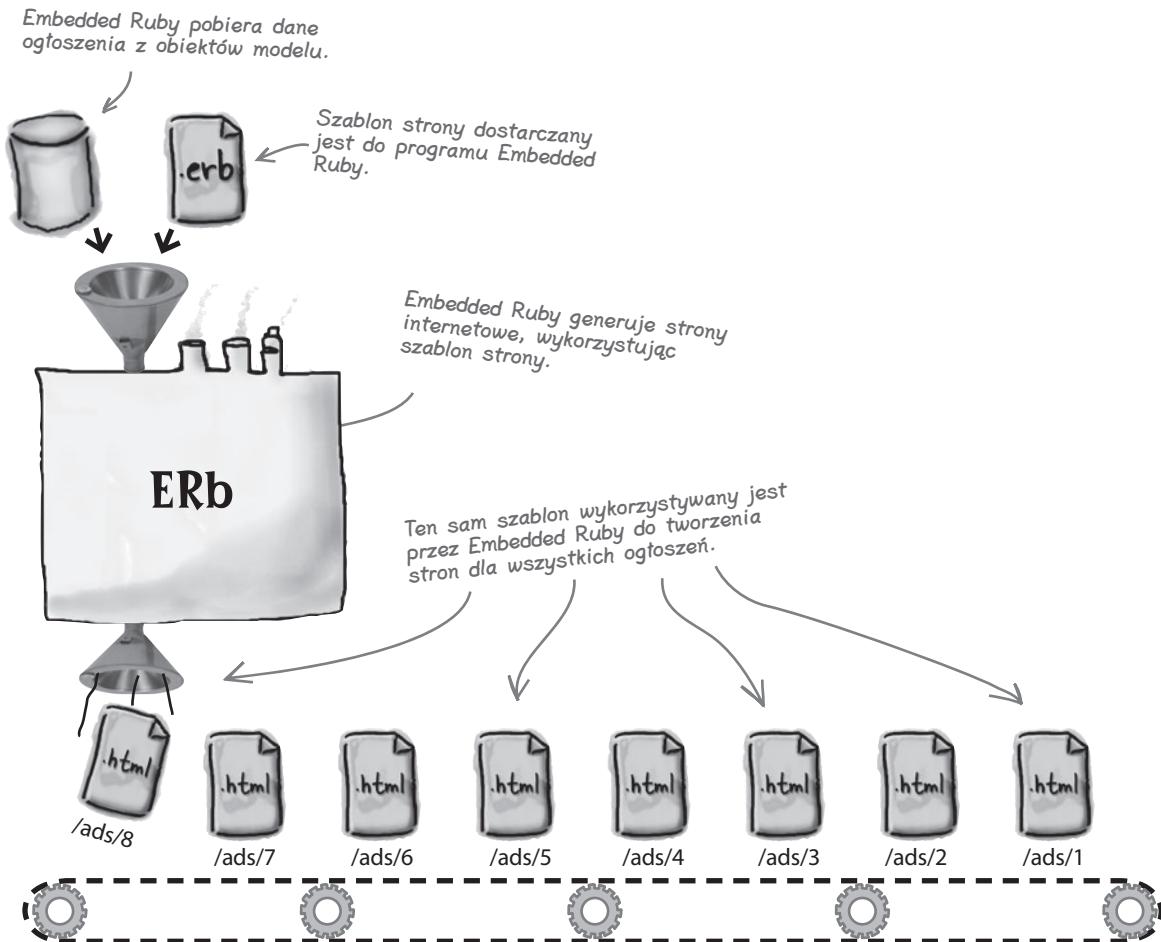
Widok tworzony jest przez szablon strony

Jaki kod widoku musimy wygenerować? Aplikacja MeBay potrzebuje pojedynczej strony internetowej; strona ta zostanie wykorzystana jako szablon dla wszystkich ogłoszeń publikowanych w witrynie. Z tego powodu strony w Rails są często nazywane **szablonami stron** (lub po prostu **szablonami**).

Strony internetowe tworzone są z szablonów przez Embedded Ruby (inaczej ERb); jest to część standardowej biblioteki języka Ruby. Jeśli ktoś poprosi o ogłoszenie numer 3, ERb wygeneruje stronę HTML dla ogłoszenia, wykorzystując szablon strony oraz dane z modelu.



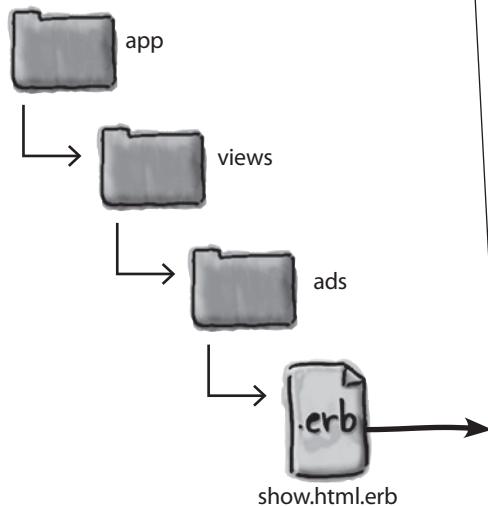
Jak zatem ERb tworzy strony internetowe?



Szablon strony zawiera kod HTML

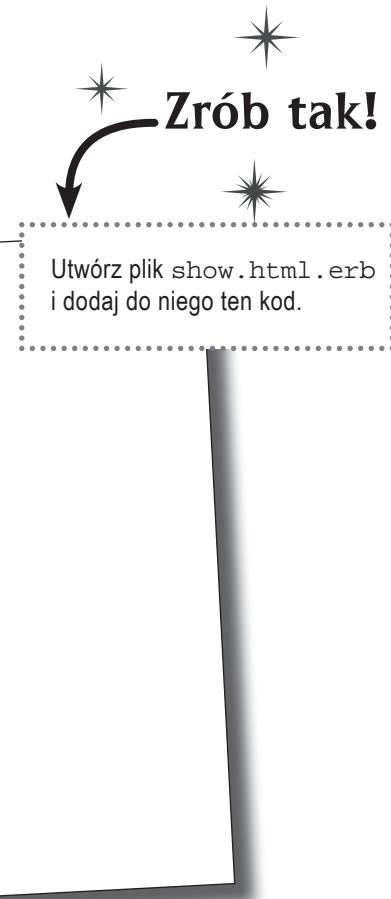
Kiedy generujesz model i kontroler, Rails generuje kod w języku **Ruby**. Widok jest jednak nieco inny. Aplikacja ma interfejs w języku **HTML**, dlatego logiczne jest, że **kod widoku jest także napisany w HTML**.

By utworzyć szablon ogłoszenia, otwórz edytor tekstu, utwórz plik o nazwie ***show.html.erb*** i zapisz go w folderze ***app/views/ads***. Oto, jak powinna wyglądać zawartość pliku ***show.html.erb***:



```

<html>
<head>
  <title>     </title>
</head>
<body>
<p>
  <b>Name:</b>
</p>
<p>
  <b>Description:</b>
</p>
<p>
  <b>Price:</b>
</p>
<p>
  <b>Seller Id:</b>
</p>
<p>
  <b>Email:</b>
</p>
</body>
</html>
  
```



Na razie szablon wygląda na dość pusty, ale za chwilę zobaczysz, jak kontroler może do niego sprytnie wstawiać wartości.

Jak zatem wygląda sama aplikacja internetowa?

Embedded
Ruby (ERb)
tworzy
strony
internetowe
z szablonu.



Jazda próbna

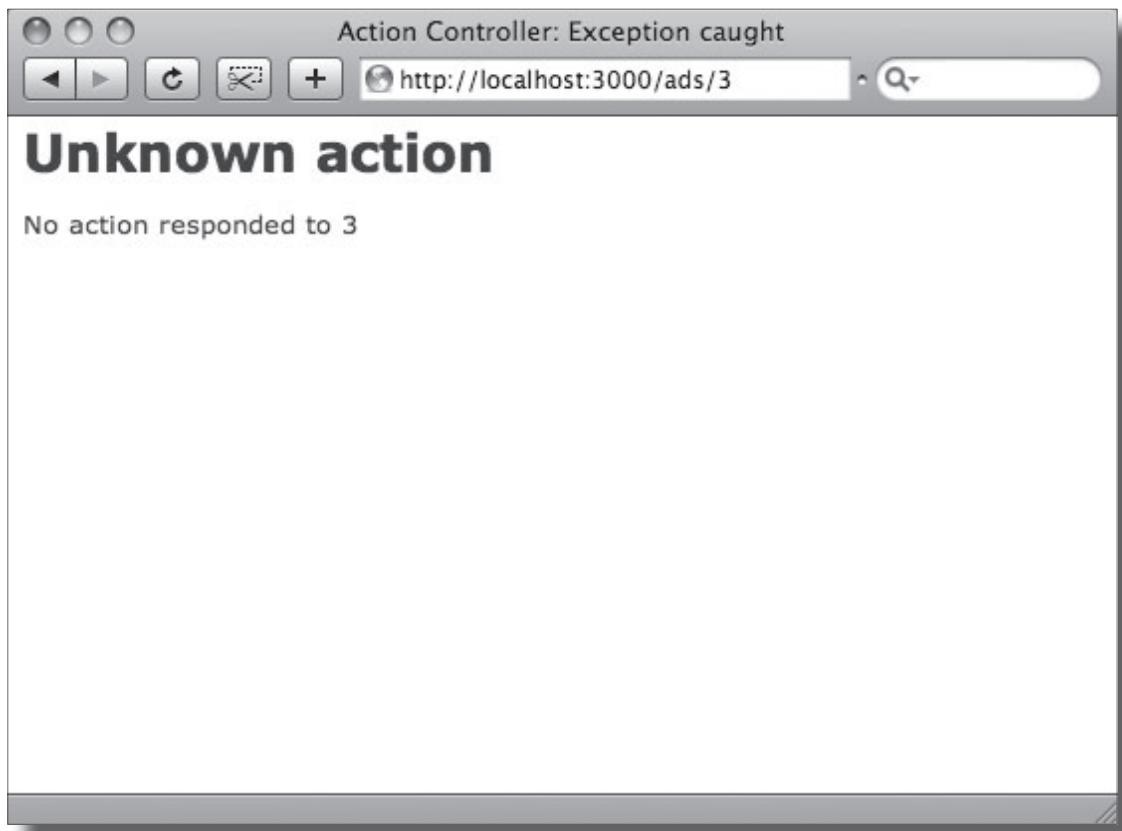
Uruchom serwer WWW za pomocą polecenia:

```
ruby script/server
```

i w przeglądarce otwórz stronę znajdująca się pod adresem:

```
http://localhost:3000/ads/3
```

Co się dzieje?



W aplikacji pojawia się błąd. Co się stało?

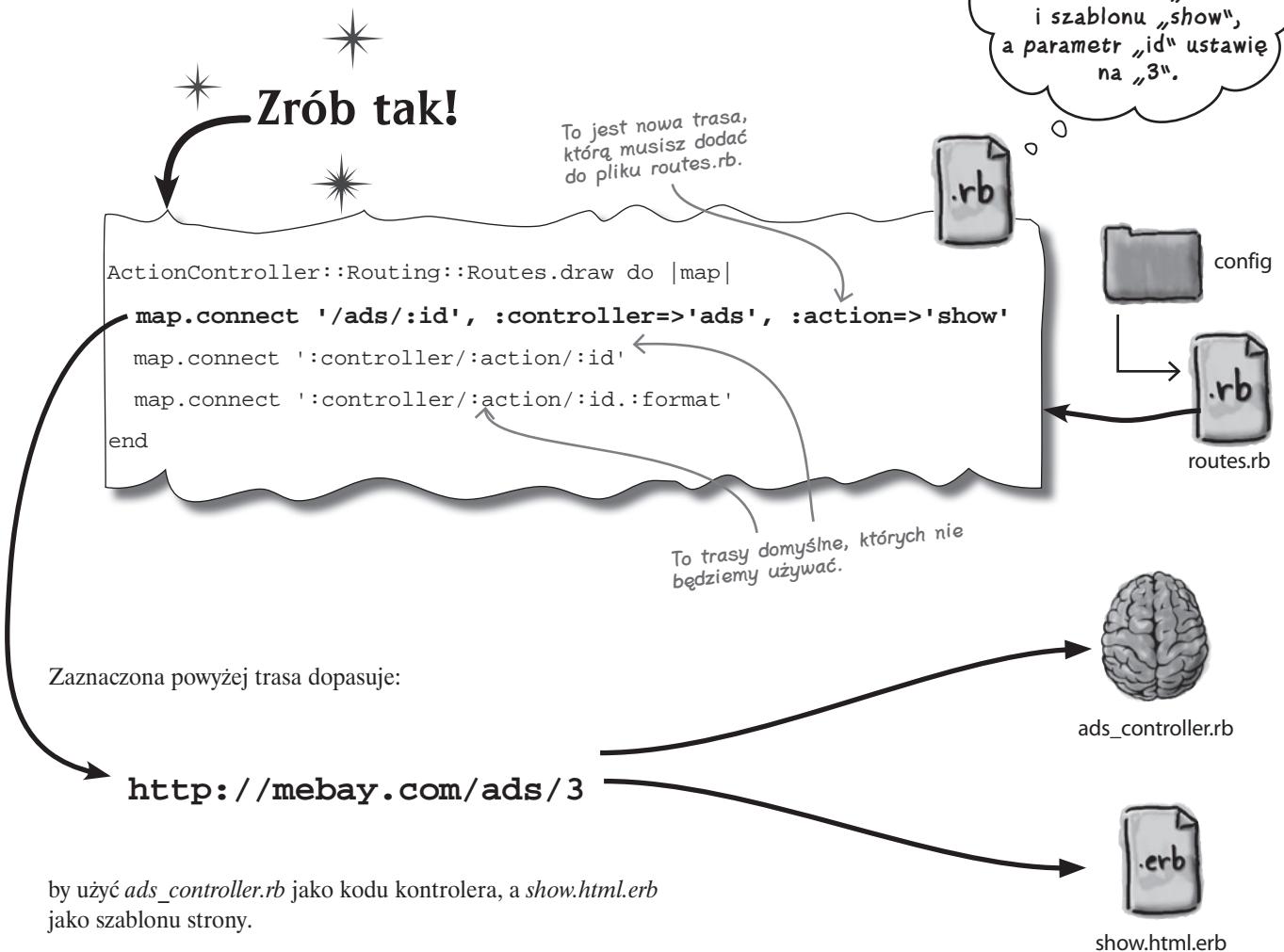
Ręcznie utworzyliśmy goły szkielet naszej aplikacji, jednak nie powiedzieliśmy Rails, jak ma używać nowego szablonu *show.html.erb*.

Jak możemy to zrobić?

Trasa mówi Rails, gdzie znajduje się strona

Rails potrzebuje reguły mówiącej, który kod należy wykonać dla określonego adresu URL. To jedno z niewielu miejsc, gdzie Rails rzeczywiście potrzebuje konfiguracji.

Reguły wykorzystywane przez Rails do odwzorowania ścieżek na kod nazywane są **trasami** (ang. *route*). Trasy definiowane są w programie Ruby w pliku *config/routes.rb*, gdzie musimy dodać nową trasę dla szablonu *show.html.erb*:

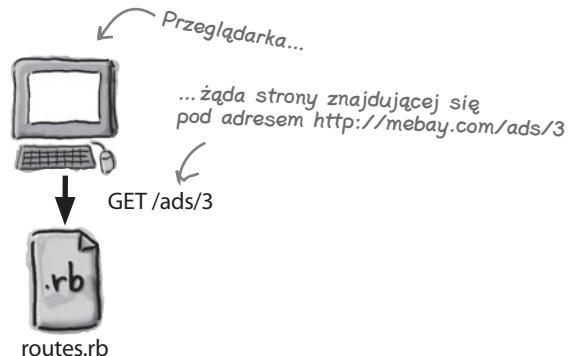


Ale jak to naprawdę zadziało? Co się stało?

Za kulisami z trasami

- 1 Kiedy Rails otrzymuje żądanie od przeglądarki, przekazuje ścieżkę żądania do programu routes.rb w celu odnalezienia pasującej trasy.

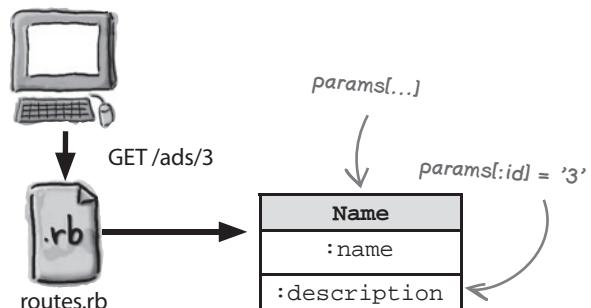
```
map.connect '/ads/:id',
:controller=>'ads', :action=>'show'
```



- 2 Jeśli pasująca trasa zawiera symbole, system routingu utworzy pasujące parametry w tabeli parametrów żądania params[...]. Przez symbol rozumiemy sekwencję liter poprzedzoną dwukropkiem (:).

System routingu widzi sekwencję liter poprzedzonych dwukropkiem jako symbol.

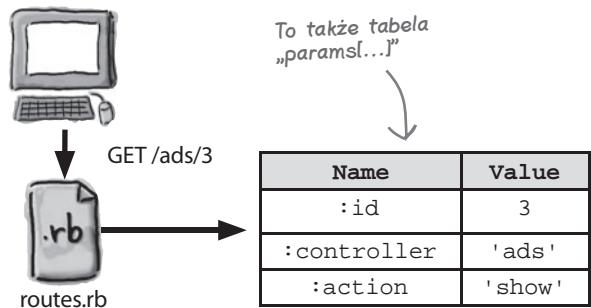
```
map.connect '/ads/:id',
:controller=>'ads', :action=>'show'
```



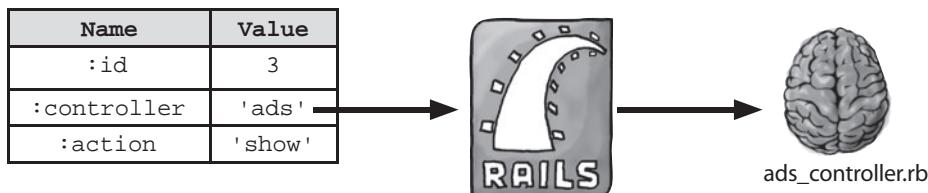
- 3 Trasa może również określać dodatkowe parametry, które zostaną wstawione do tabeli params[...]. Często podawane są tutaj :controller oraz :action. Jak myślisz, dlaczego?

```
map.connect '/ads/:id',
:controller=>'ads', :action=>'show'
```

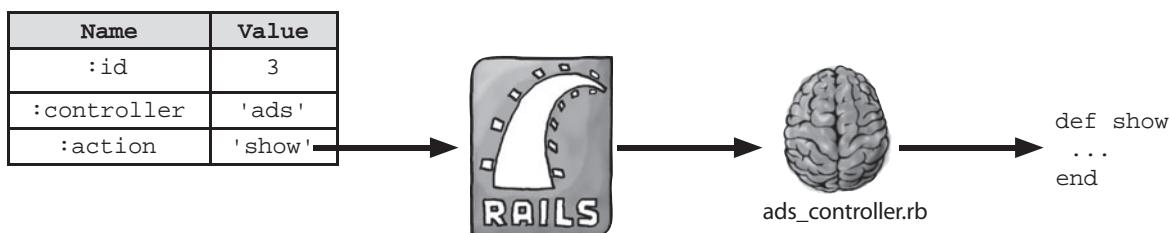
Trasa może również podawać dodatkowe parametry.



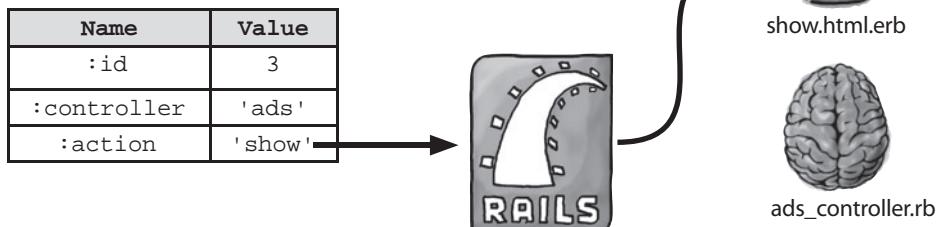
- 4 Kiedy program routes.rb zakończy działanie, platforma Rails sprawdzi wartość params[:controller] i wykorzysta ją do zadecydowania, jaki typ obiektu kontrolera ma utworzyć.



- 5 Po utworzeniu obiektu kontrolera Rails wykorzysta wartość przechowywaną w params[:action] w celu wyboru metody kontrolera, jaką należy wywołać.



- 6 Następnie, kiedy metoda kontrolera zakończy działanie, Rails wywołuje szablon strony pasujący do wartości params[:action]. Szablon strony generuje wówczas odpowiedź odsyłaną do przeglądarki.



Nieistniejąca głupie pytania

P: Czy nie byłoby szybciej wygenerować rusztowanie, a potem podać je edycji?

O: To zależy od aplikacji. MeBay chce jedynie niewielkiej części funkcjonalności. Jeśli aplikacja ma działać znaczco inaczej od rusztowania, szybciej będzie wygenerować po prostu model oraz kontroler, a następnie dodać własny kod.

P: Czy rusztowanie generuje również model?

O: Tak. Generator rusztowania wywołuje generatory dla modelu oraz kontrolera. Tworzy również szablony stron dla standardowych operacji tworzenia, odczytywania, aktualniania i usuwania.

P: Jakiego rodzaju parametrem jest :id?

O: To parametr żądania, taki jak wartości przesyłane przez formularze czy parametry przekazywane do adresu URL.

P: Czym jest żądanie?

O: Żądanie to to, co przeglądarka przesyła do serwera przy każdym kliknięciu odnośnika. Mówiąc serwerowi, jaką ścieżkę sobie życysz.

P: Czym zatem jest odpowiedź?

O: Odpowiedź to treść zwracana przez serwer do przeglądarki wraz z innymi informacjami, takimi jak typ MIME tej treści.

P: Dlaczego Rails potrzebuje konfigurowania tras? Czemu nie można zastosować standardowych ścieżek?

O: Rails zawsze woli konwencję od konfiguracji, z wyjątkiem sytuacji, w których system musi się porozumieć ze światem zewnętrznym. Format adresów URL wpływa na to, jak świat zewnętrzny widzi aplikację, dlatego Rails pozwala Ci na konfigurację tego elementu.

Kiedy używasz rusztowania, Rails generuje trasy za Ciebie, jednak nadal warto wiedzieć, jak one działają, na wypadek gdybyś musiał prześledzić błędy czy tworzyć własne trasy — jak w tej aplikacji.

P: Nadal nie do końca rozumiem, kiedy używać notacji CamelCase. O co w tym chodzi?

O: Notacja CamelCase oznacza po prostu wykorzystywanie w identyfikatorach składających się z więcej niż jednego słowa wielkich liter. Jest tak nazywana, ponieważ słowa rozpoczynające się wielkimi literami wyglądają jak wielbłądzie garby (ang. camel — wielbłąd).

W Rails nazwa pliku i nazwa kontrolera są podobne, jednak kontroler wykorzystuje notację CamelCase do rozdzielenia słów. Nazwy plików wykorzystują znaki „_”, co pozwala rozróżnić fragment kodu i plik.

P: Co wywołujemy najpierw — kontroler czy widok?

O: Kontroler zawsze wywoływany jest przed widokiem.

P: Dlaczego szablon strony ma rozszerzenie .html.erb? Czy nie jest to normalny plik HTML?

O: Szablon może być po prostu plikiem HTML, jednak szablony mogą również zawierać dodatkowe instrukcje, które zostaną przetworzone przez system Embedded Ruby. Wszystkie pliki, które mają przetwarzać system Embedded Ruby, mają na końcu nazwy pliku rozszerzenie .erb.

P: Dlaczego szablony znajdują się w folderze o nazwie views/ads, a kontrolery nie są umieszczane w folderze controllers/ads?

O: Wyobraź sobie, że chcesz zmodyfikować obiekt, a także zobaczyć obiekt. Otrzymasz stronę „edit” oraz stronę „view”. Jednak żądania „edit” oraz „view” przejdą przez jeden kontroler. Modele mają zatem jeden kontroler, ale potencjalnie kilka stron. Z tego powodu szablony stron znajdują się w swoim własnym podkatalogu i może ich być kilka.

P: Słyszałem, że niektórzy mówią o „obiektach biznesowych” i „obiektach domenowych”. Czy Rails je posiada?

O: Tak, ponieważ obiekty biznesowe oraz obiekty domenowe to po prostu inna nazwa obiektów modelu.

Jaka jest moja trasa?

Konkurenci firmy MeBay mają już własną aplikację Rails korzystającą z poniższego zbioru tras:

```
map.connect '/shows/:title', :controller => 'shows', :action=> 'display'
map.connect '/cats/:name', :controller => 'cats', :action=> 'show'
map.connect '/gadgets/:type', :controller => 'gadgets', :action=> 'show'
```

Czy jesteś w stanie odgadnąć, który plik szablonu strony zostanie wykorzystany do wygenerowania kodu HTML dla każdego z podanych adresów URL? Narysuj linię łączącą adres URL z wykorzystywanym szablonem strony, a następnie zapisz nazwę oraz wartość parametru, który zostanie pobrany z każdego adresu URL.

Narysuj linię prowadzącą od każdego adresu URL do właściwego pliku szablonu.

1 http://yourbay.com/gadgets/display

Nazwa parametru: Wartość:.....

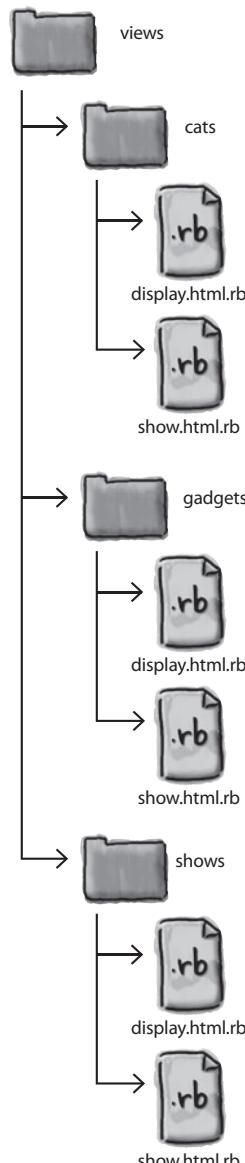
2 http://yourbay.com/shows/cats

Nazwa parametru: Wartość:.....

Tutaj wpisz nazwę oraz wartość parametru pobranego każdorazowo z adresu URL.

3 http://yourbay.com/cats/gadget

Nazwa parametru: Wartość:.....



Jaka jest moja trasa?

Rozwiążanie

Konkurenci firmy MeBay mają już własną aplikację Rails korzystającą z poniższego zbioru tras:

```
map.connect '/shows/:title', :controller => 'shows', :action=> 'display'  
map.connect '/cats/:name', :controller => 'cats', :action=> 'show'  
map.connect '/gadgets/:type', :controller => 'gadgets', :action=> 'show'
```

Czy jesteś w stanie odgadnąć, który plik szablonu strony zostanie wykorzystany do wygenerowania kodu HTML dla każdego z podanych adresów URL? Narysuj linię łączącą adres URL z wykorzystywanym szablonem strony, a następnie zapisz nazwę oraz wartość parametru, który zostanie pobrany z każdego adresu URL.

Ten adres URL
pasuje do tej reguły.

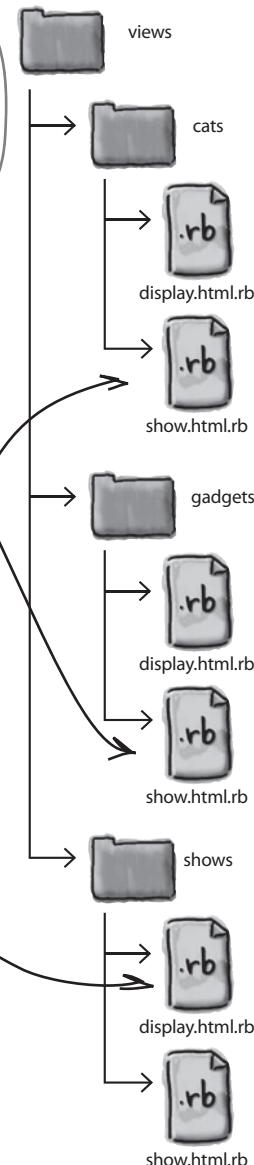
Pasująca reguła ma
w ścieżce żądania
parametr „:type”.

- 1 <http://yourbay.com/gadgets/display>
Nazwa parametru::type..... Wartość:'display'.....

Wartość parametru jest
częścią ścieżki znajdującej
się w tej samej pozycji
co parametr.

- 2 <http://yourbay.com/shows/cats>
Nazwa parametru::title..... Wartość:'cats'.....

- 3 <http://yourbay.com/cats/gadget>
Nazwa parametru::name..... Wartość:'gadget'.....





Jazda próbna

Otwórz w przeglądarce kilka stron:

`http://localhost:3000/ads/3` oraz

`http://localhost:3000/ads/5`

Strony te nie mają tytułów.

Obok podpisów nie ma wartości.

Ogłoszenia są puste!



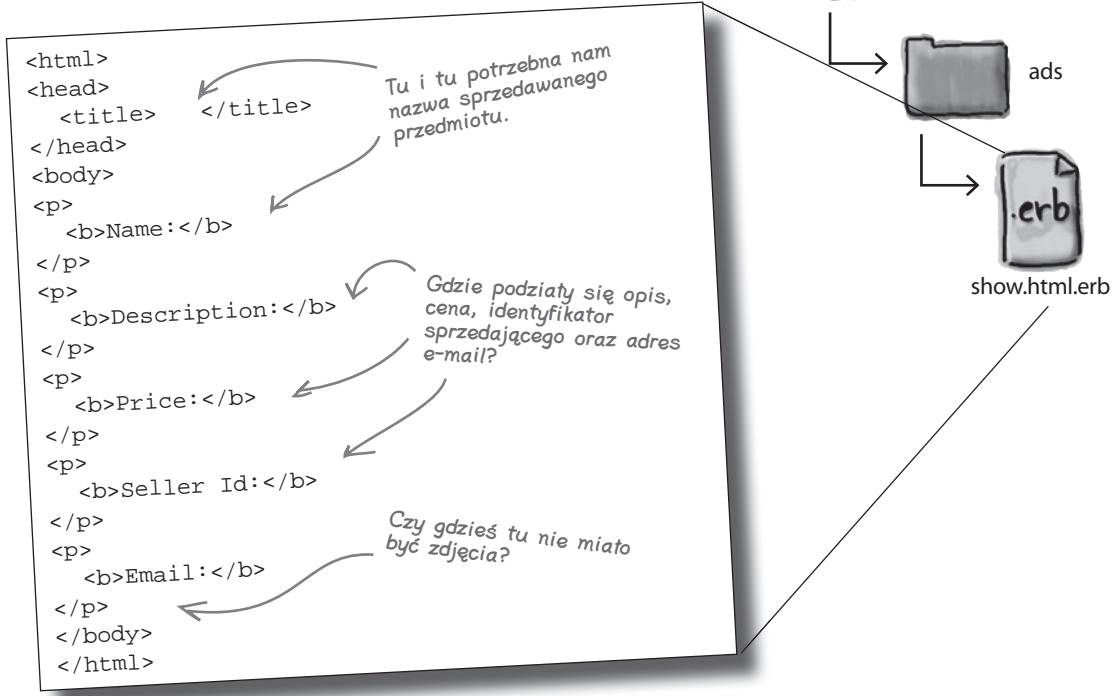
WYSIL
SZARE KOMÓRKI

Dlaczego na stronach nie ma szczegółów?
Czy to wina tras? Modelu? Widoku? Kontrolera?

Widok nie ma danych do wyświetlenia

Spójrz jeszcze raz do pliku *show.html.erb*.

Plik ten wykorzystany został do utworzenia strony dla każdego z ogłoszeń — i dokładnie to zrobił szablon:



Choć umieściliśmy najważniejsze części szkieletu HTML — podpisy, części body oraz head — na właściwych miejscach, zabrakło nam kilku elementów. Nie podaliśmy:

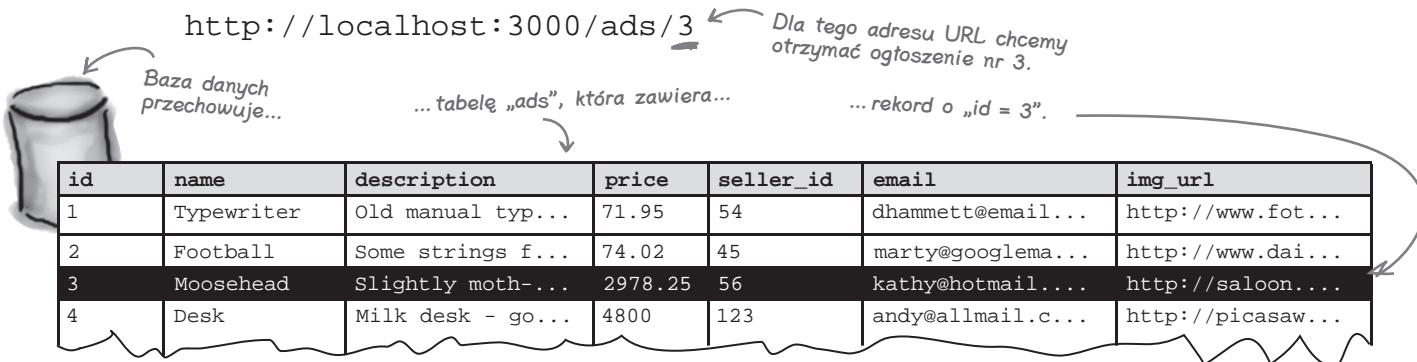
Jakie dane mają być wyświetlane

ani

W którym miejscu na stronie dane te mają zostać wstawione.

Co zatem powinna pokazywać strona?

Strona ogłoszenia powinna pokazywać dane dla ogłoszenia o numerze podanym w adresie URL. Oto przykładowy adres URL dla ogłoszenia numer 3:

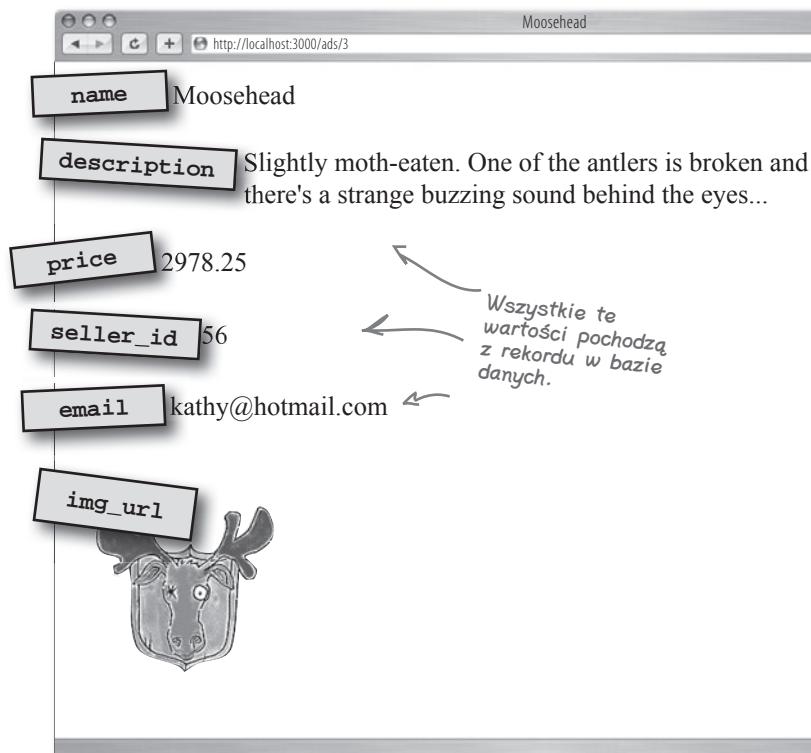


Pierwsze, co musisz zrobić, to nakazać modelowi wczytanie z tabeli ads rekordu o numerze identyfikatora równym numerowi identyfikatora z adresu URL. Jeśli użytkownik żąda strony dla ogłoszenia numer 3, modelowi należy przekazać, by wczytał rekord o `id = 3`.

Musimy wyświetlić dane we właściwym miejscu

Odczytanie danych to jedynie połowa sukcesu. Kiedy model odczyta dane, musi je przesłać do widoku. Widok musi wtedy wiedzieć, gdzie ma wyświetlić dane na każdej stronie. Każde z pól rekordu musi zostać wyświetlone obok odpowiadającego mu podpisu na stronie internetowej. Dodatkowo konieczne jest użycie wartości z kolumny `img_url` do wstawienia zdjęcia sprzedawanego przedmiotu na stronę.

Która część systemu odpowiedzialna jest za zażądanie odpowiednich danych z bazy, a następnie przesłanie tych danych do widoku?



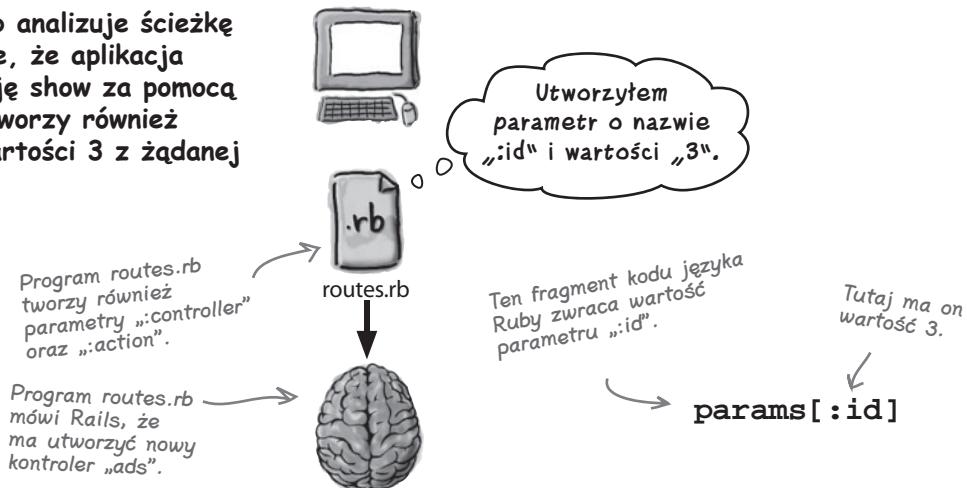
Kontroler przesyła ogłoszenie do widoku

Zobaczmy, jak będzie wyglądał i działał kod kontrolera:

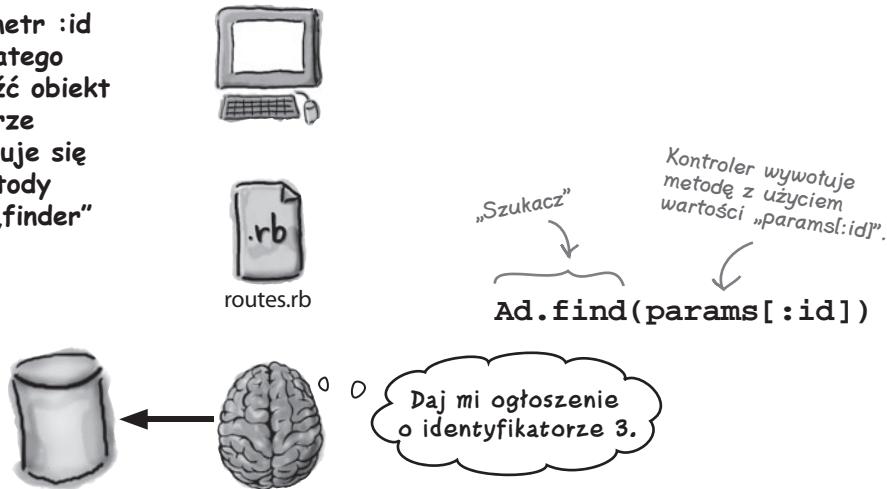
- 1 Kiedy przeglądarka użytkownika przesyła żądanie strony do aplikacji, Rails wywołuje program routes.rb, który decyduje, jaki kod należy wykonać.



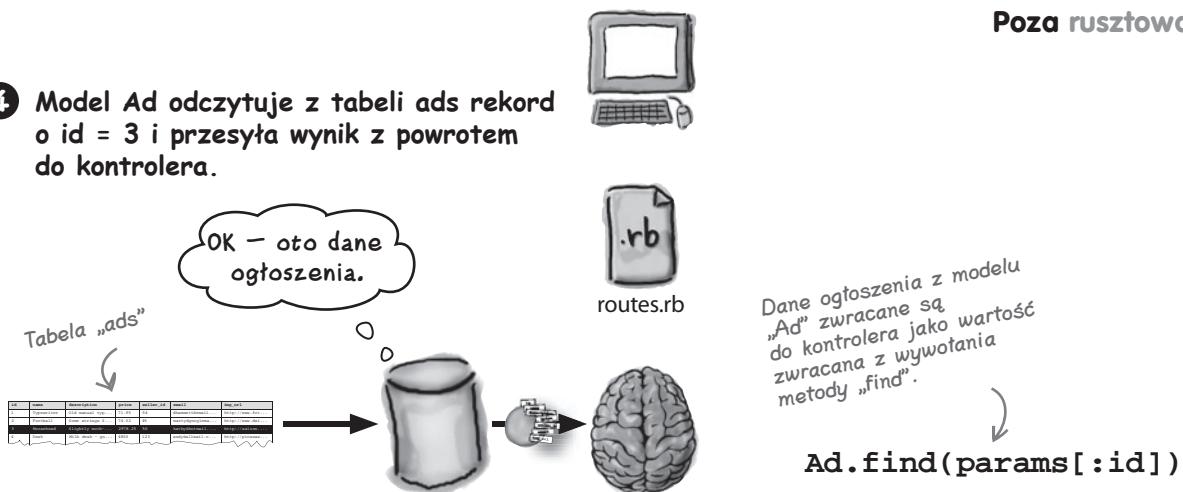
- 2 Program routes.rb analizuje ścieżkę żądania i decyduje, że aplikacja musi wykonać akcję show za pomocą kontrolera ads. Tworzy również parametr :id o wartości 3 z żądanej ścieżki.



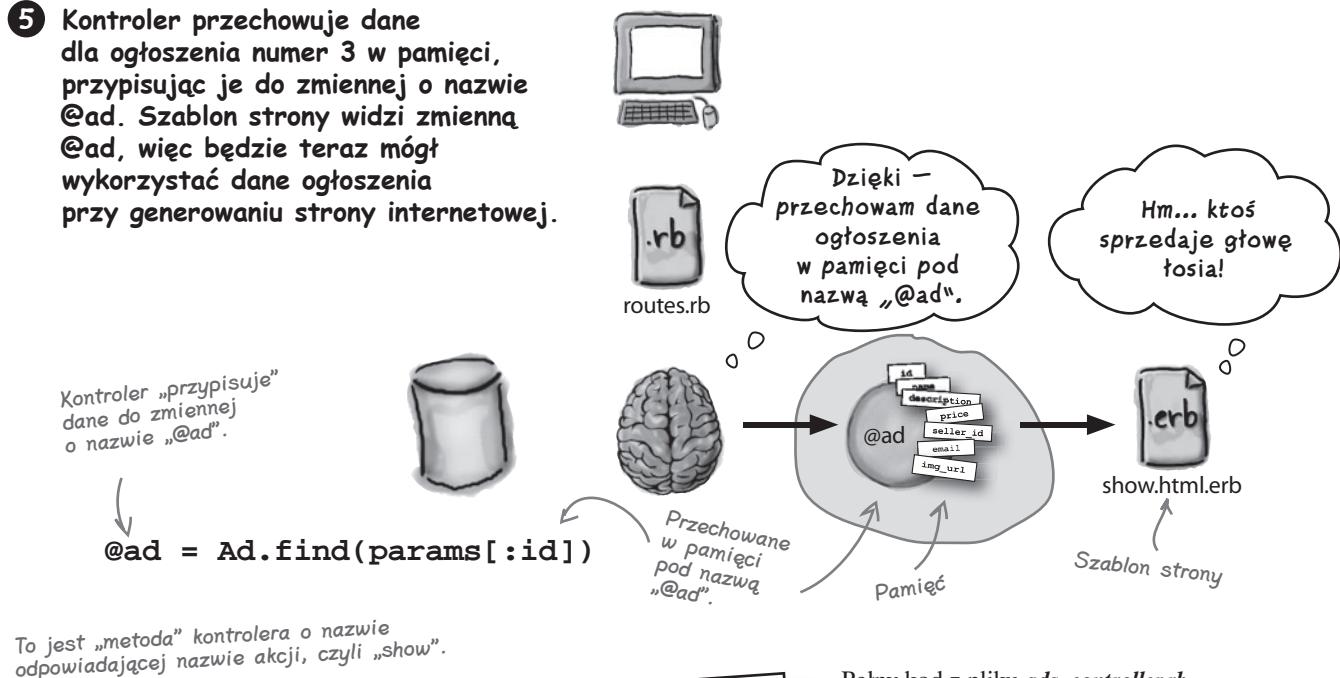
- 3 Kontroler widzi, że parametr :id ustawiony jest na „3”, dlatego każe modelowi Ad odnaleźć obiekt ogłoszenia o identyfikatorze id = 3. Kontroler komunikuje się z modelem za pomocą metody wyszukującej nazywanej „finder” („szukacz”).



- 4 Model Ad odczytuje z tabeli ads rekord o id = 3 i przesyła wynik z powrotem do kontrolera.



- 5 Kontroler przechowuje dane dla ogłoszenia numer 3 w pamięci, przypisując je do zmiennej o nazwie @ad. Szablon strony widzi zmienną @ad, więc będzie teraz mógł wykorzystać dane ogłoszenia przy generowaniu strony internetowej.



```
class AdsController < ApplicationController
  def show
    @ad = Ad.find(params[:id])
  end
end
```

Wpisz powyższy kompletny kod kontrolera.

Zrób tak!

Plik /app/controllers/ads_controller.rb

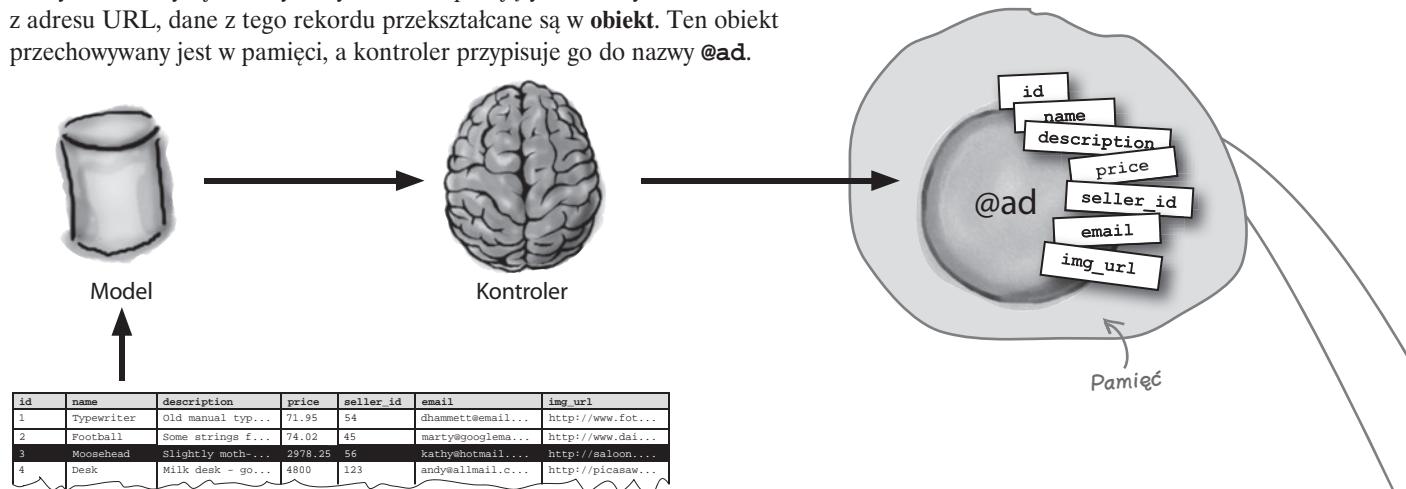
Pełny kod z pliku *ads_controller.rb* powinien trafić do metody o nazwie **show**, która pasuje do nazwy parametru `:action` utworzonego przez plik *routes.rb*.

Ale jak właściwie model odczytuje dane z bazy i jak szablon strony może z tych danych korzystać?

Od rekordu do obiektu

Rails zmienia rekord w obiekt

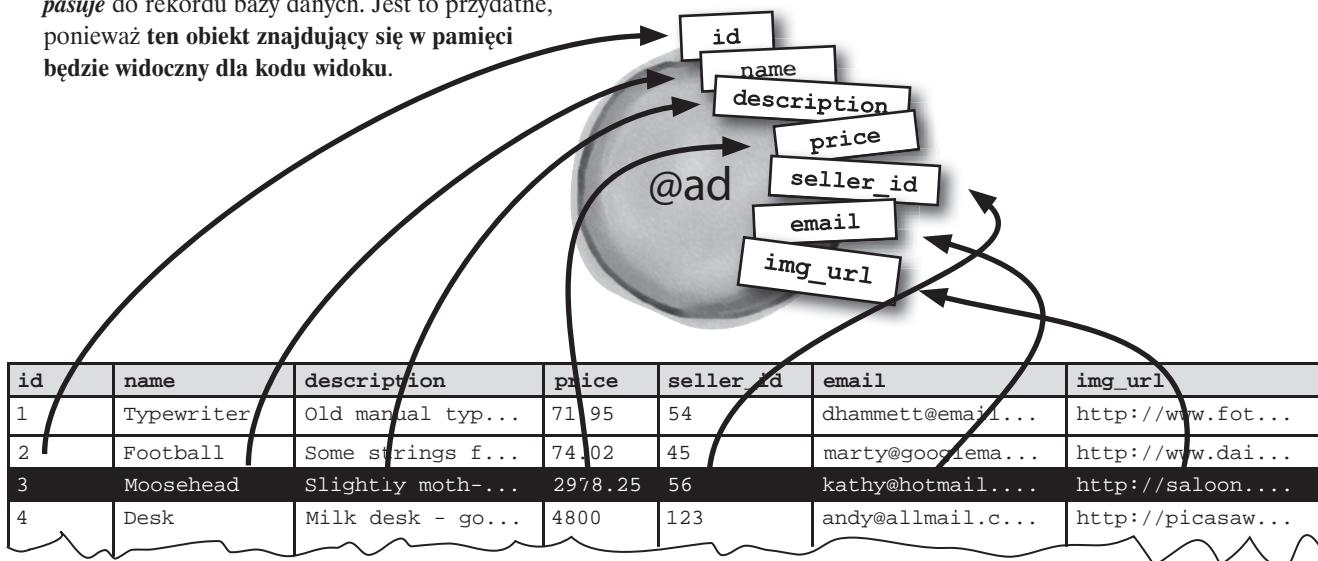
Kiedy Rails wczytuje z bazy danych rekord pasujący do identyfikatora z adresu URL, dane z tego rekordu przekształcane są w **obiekt**. Ten obiekt przechowywany jest w pamięci, a kontroler przypisuje go do nazwy `@ad`.



Rekord zawiera jednak kilka pól z danymi. W jaki sposób wszystkie dane zostają przechowane w jednym obiekcie?

Odpowiedź jest następująca: obiekt może mieć kilka **atrybutów**. Atrybut jest jak pole rekordu. Ma **nazwę** oraz **wartość**. Kiedy Rails odczytuje wartość pola `description` z rekordu bazy danych, przechowuje ją w atrybucie `@ad.description` obiektu `@ad`. Tak samo jest w przypadku pól `id`, `name`, `seller_id` i tak dalej.

W ten sposób obiekt modelu `@ad` **dokładnie pasuje** do rekordu bazy danych. Jest to przydatne, ponieważ **ten obiekt znajdujący się w pamięci będzie widoczny dla kodu widoku**.



Dane znajdują się w pamięci, a strona internetowa je widzi

Szablon strony (`show.html.erb`) nie jest po prostu przesyłany bezpośrednio do przeglądarki. Najpierw zostanie on przetworzony przez program **Embedded Ruby** (ERb) i dlatego właśnie nasz plik szablonu ma rozszerzenie `.erb`. Przyjrzyjmy się zatem dokładniej temu, jak ERb wczytuje obiekty z pamięci.

ERb wczytuje się w szablon, szukając niewielkich fragmentów osadzonego kodu w języku Ruby, nazywanych **wyrażeniami**. Wyrażenie otoczone jest znakami `<%=` oraz `%>`, a ERb zastępuje wyrażenie jego wartością. Dlatego jeśli znajdzie:

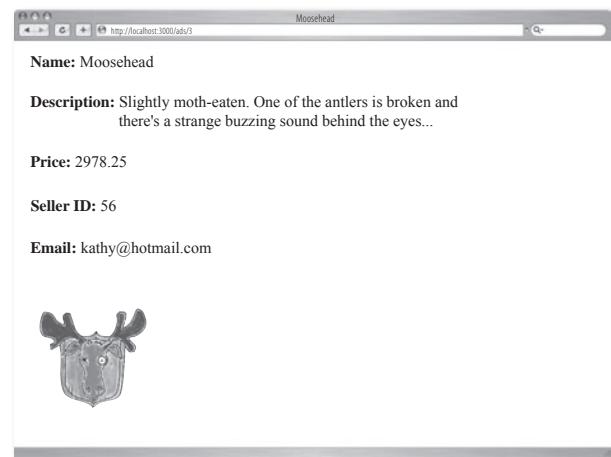
`<%= 1 + 1 %>`

gdzieś na stronie internetowej, Rails zastąpi to wyrażenie wartością 2 przed zwróceniem strony do przeglądarki.

Tak naprawdę chcemy teraz dotrzeć do wartości z obiektu `@ad` znajdującego się w pamięci, jak poniżej:

```
<html>
<head>
  <title><%= @ad.name %></title>
</head>
<body>
<p>
  <b>Name:</b><%= @ad.name %>
</p>
<p>
  <b>Description:</b><%= @ad.description %>
</p>
<p>
  <b>Price:</b><%= @ad.price %>
</p>
<p>
  <b>Seller ID:</b><%= @ad.seller_id %>
</p>
<p>
  <b>Email:</b><%= @ad.email %>
</p>
<p>
  
</p>
</body>
</html>
```

Ten szablon ERb z osadzonym kodem w Ruby wygeneruje kod HTML dla tej strony.



Przed odesaniem strony Rails zastępuje wszystkie znaczniki `<%= ... %>` wartościami ich obiektów.

A zatem — czy to działa?



Jazda próbna

By wypróbować aplikację, ekipa z MeBay wykorzystała swój system wprowadzania danych do wstawienia danych do bazy Rails.

Kiedy dane zostają umieszczone w bazie, od razu stają się dostępne za pośrednictwem Internetu. Jeśli zatem ktoś zażąda /ads/1 czy /ads/2, zobaczy stronę wygenerowaną przez odpowiednie dane z bazy.



Zespół wprowadzający dane w firmie MeBay



Name: Moosehead
Description: Slightly moth-eaten. One of the antlers is broken and there's a strange buzzing sound behind the eyes...
Price: 2978.25
Seller ID: 56
Email: kathy@hotmail.com

Name: Football
Description: Some strings frayed.
Price: 74.02
Seller ID: 45
Email: marty@googlemail.com

Name: Typewriter
Description: Old manual typewriter. Many years of useful service. Works best with a bottle next to it.
Price: 71.95
Seller ID: 54
Email: dhammett@email.com

Gratulacje!

Właśnie utworzyłeś swoją pierwszą stworzoną ręcznie aplikację Rails!

Choć zajęło Ci to nieco dłużej, niż gdybyś skorzystał z rusztowania, na każdym etapie miałeś pełną kontrolę nad sytuacją. Co więcej, zatrzymałś pod maskę Rails i przy okazji nauczyłeś się nieco o możliwościach tej platformy:



CELNE SPOSTRZEŻENIA

- **Trasy** mówią Rails, jaki kod należy wykonać po otrzymaniu żądania dla adresu URL.
- Kontroler wykorzystuje **identyfikator** (id) z adresu **URL** do wczytania odpowiednich danych z modelu.
- Model odczytuje bazę danych i zwraca dane jako **obiekt języka Ruby**.
- Kontroler nadaje obiekowi nazwę **w pamięci**, tak by mógł on zostać odnaleziony przez...
- ...**szablon strony**, który wykorzystuje osadzone wyrażenia języka Ruby do wstawienia danych do strony.

Łamigłówka

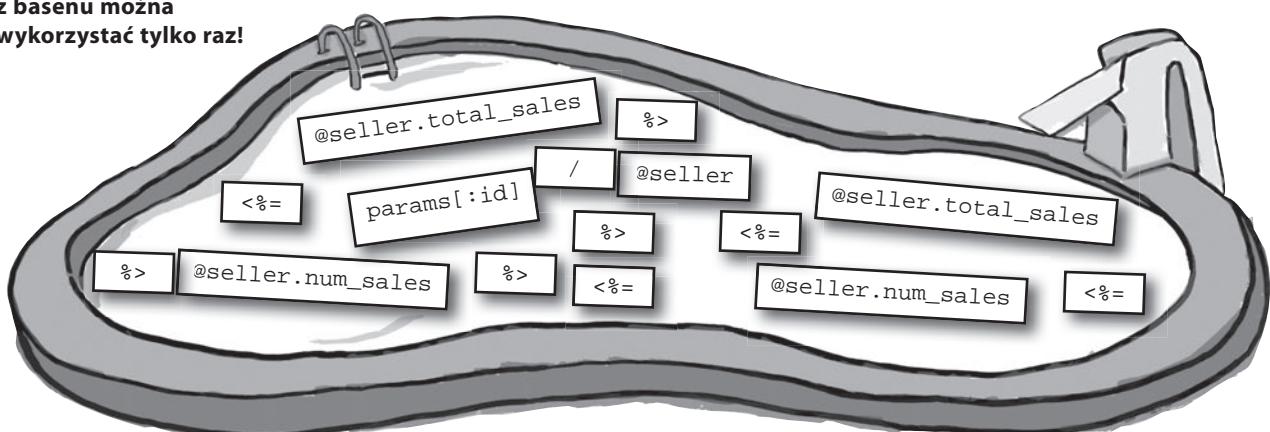


Firma MeBay chce wyświetlać informacje o sprzedających w /sellers/:id. Uzupełnij kontroler oraz szablon strony podanym niżej kodem.

```
def stats
    ..... = Seller.find(.....)
end

<p>
  <b>Number of sales:</b> .....
</p>
<p>
  <b>Total sales value:</b> .....
</p>
<p>
  <b>Average price:</b> .....
</p>
```

Uwaga: każdy element z basenu można wykorzystać tylko raz!



Łamigłówka: Rozwiążanie



Ten obiekt będzie miał
atrybuty odpowiadające
wartości sprzedawcy
w bazie danych.

```
def stats
  @seller = Seller.find(.. params[:id] ..)
end
```

Zwraca sprzedającego dla numeru
identyfikatora z adresu URL.

To będzie numer
z adresu URL, dlatego
dla użytkownika proszącego
o „/sellers/3” będzie to
numer 3.

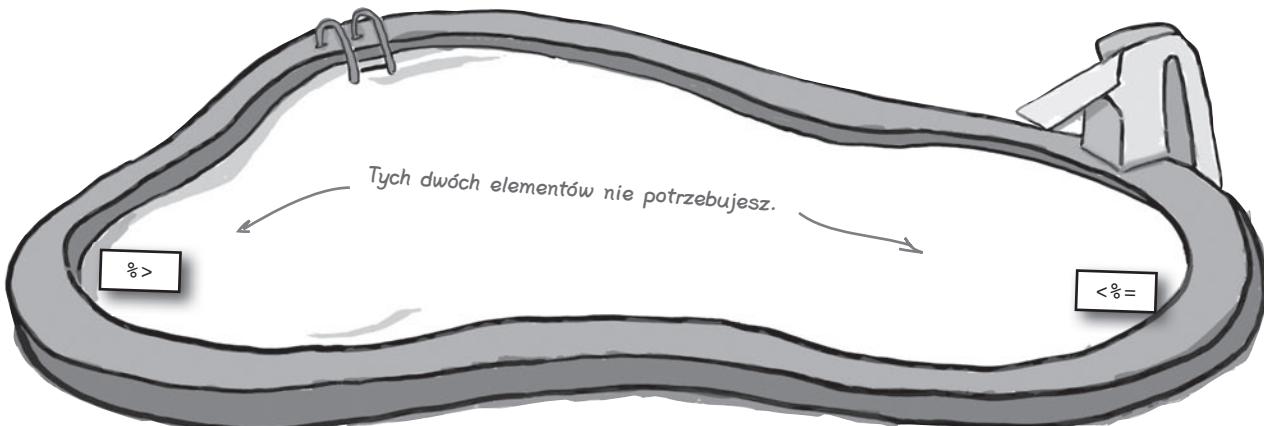
```
<p>
  <b>Number of sales:</b> ... <%= @seller.num_sales %>
</p>
<p>
  <b>Total sales value:</b> ... <%= @seller.total_sales %>
</p>
<p>
  <b>Average price:</b> ... <@seller.total_sales / @seller.num_sales %>
</p>
```

To wczyta dane z kolumny „num_sales”
z bazy danych.

To zwróci wartość kolumny „total_sales” z bazy danych.

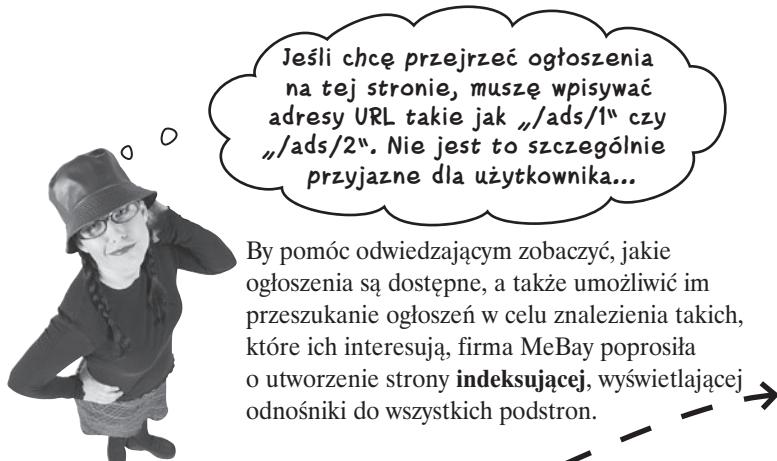
Wyrażenia mogą zawierać obliczenia,
dlatego ten znacznik zostanie
zastąpiony średnią wartością
sprzedaży.

Zauważ, że obliczenia
otaczają pojedynczy
zestaw znaczników.

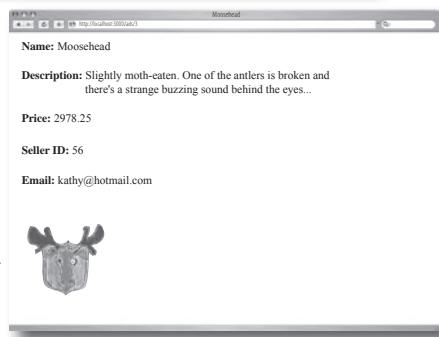
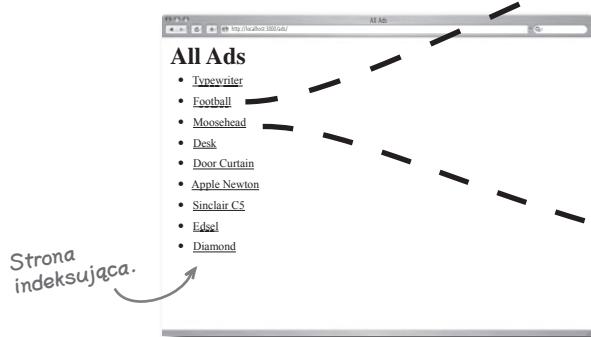
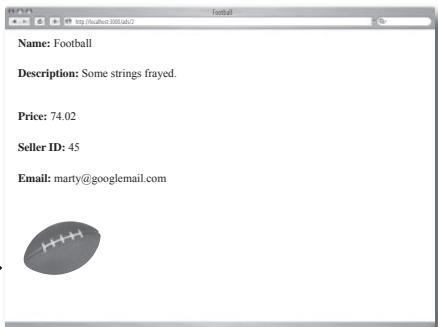


Jest problem — ludzie nie potrafią znaleźć żądanych stron

Choć istnieją już strony dla każdego ogłoszenia z bazy danych, brakuje łatwego sposobu umożliwiającego ludziom ich odnalezienie.



By pomóc odwiedzającym zobaczyć, jakie ogłoszenia są dostępne, a także umożliwić im przeszukanie ogłoszeń w celu znalezienia takich, które ich interesują, firma MeBay poprosiła o utworzenie strony **indeksującej**, wyświetlającej odnośniki do wszystkich podstron.



Zaostrz ołówek



Gdybyś chciał utworzyć nową stronę o nazwie „index”, jaka musiałaby być trasa, którą wywoływałaby strona <http://mebay.com/ads/>?

Jak nazywałby się kod kontrolera?

A jak szablon strony?

Trasa:

Kod kontrolera: Szablon strony:



Zaostrz ołówek

Rozwiążanie

Gdybyś chciał utworzyć nową stronę o nazwie „index”, jaka musiała być trasa, którą wywoływałaby strona <http://mebay.com/ads/>?

Jak nazywałby się kod kontrolera?

A jak szablon strony?

Trasa: `map.connect '/ads/', :controller=>'ads', :action=>'index'`

Kod kontrolera: `index` Szablon strony: `app/views/ads/index.html.erb`

Nie istnieja głupie pytania

P: Czym jest akcja?

O: Akcja to **zbiór operacji**, jakie aplikacja Rails wykonuje w odpowiedzi na żądanie ze strony użytkownika. Parametr `action` określa nazwę akcji. Cały Twój kod (na przykład metody kontrolera oraz plik szablonu strony) wykorzystują nazwy akcji, by platforma Rails mogła je odnaleźć.

P: Czy mogę użyć w Rails dowolnej bazy danych?

O: Obsługiwane są wszystkie najważniejsze bazy danych, takie jak SQLite3, MySQL oraz Oracle. Dodatkowo w większości przypadków nie musisz tworzyć zbyt wiele kodu specyficznego dla bazy danych. Dzięki temu możesz zmieniać systemy bazy danych bez niszczenia aplikacji czy pisania na nowo ogromnej ilości kodu.

P: Języki takie jak Java obok obiektów mają typy proste. Czy Ruby lub Rails również mają typy proste?

O: Nie. W języku Ruby nie ma typów prostych. Wszystko, z czym masz do czynienia w Ruby (w tym liczby, a nawet bloki kodu), to obiekty.

P: Czy szablon strony nie jest po prostu atrakcyjniejszą nazwą strony?

O: Nie. Szablon strony wykorzystywany jest do generowania stron, jednak nie jest samą stroną. Strony generowane są z szablonów.

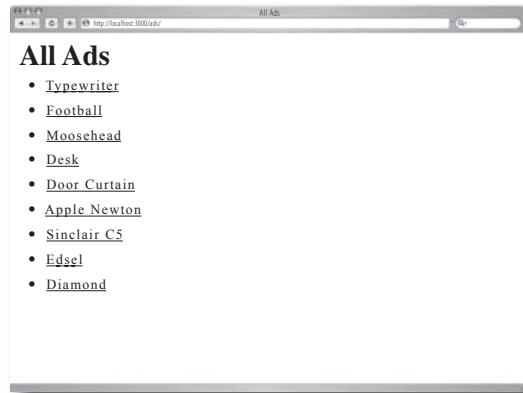
Zaostrz ołówek (ponownie)



Teraz mamy dwie trasy:

```
map.connect '/ads/:id', :controller=>'ads', :action=>'show'
map.connect '/ads/', :controller=>'ads', :action=>'index'
```

Która strona zostałaby wyświetlona dla każdego z adresów URL?

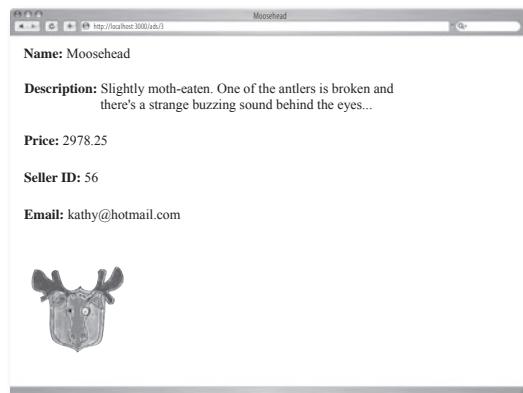


index.html.erb

/ads/3

/ads/something

/ads/



show.html.erb

Czy jest tu jakiś problem? Jeśli tak, jak byś go naprawił?

Myłce trasy



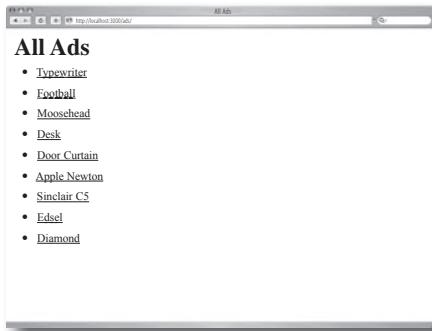
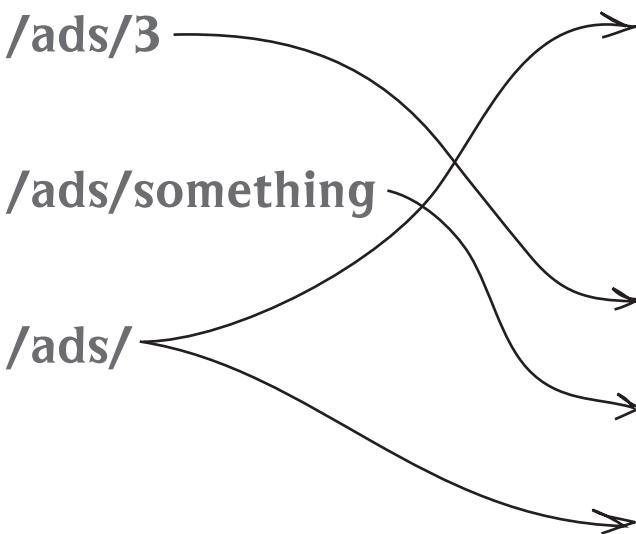
Zaostrz ołówek (ponownie)

Rozwiążanie

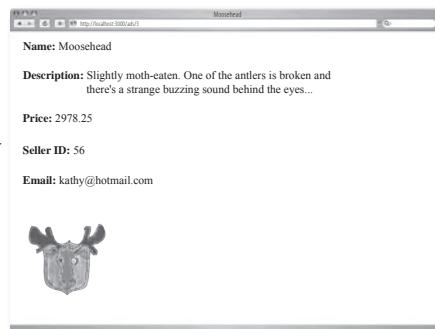
Teraz mamy dwie trasy:

```
map.connect '/ads/:id', :controller=>'ads', :action=>'show'  
map.connect 'ads/', :controller=>'ads', :action=>'index'
```

Która strona zostałaby wyświetlona dla każdego z adresów URL?



index.html.erb



show.html.erb

Czy jest tu jakiś problem? Jeśli tak, jak byś go naprawił?

Ścieżka „/ads/” pasuje do obu tras — trzeba ją zmienić tak, by pasowała tylko do jednej z nich.

Trasy wykonywane są w kolejności

Obie trasy dopasowują ścieżkę /ads. Rails unika dwuznaczności, wykorzystując jedynie pierwszą pasującą trasę, dlatego w celu uniknięcia zamieszania należy zmienić kolejność tras.

```
map.connect 'ads/', :controller=>'ads', :action=>'index'  
map.connect '/ads/:id', :controller=>'ads', :action=>'show'
```

To są trasy, które zostaną wykorzystane przez Rails. Teraz musisz uzupełnić kod.

Zrób tak!
Dodaj te trasy do pliku config/routes.rb.



Cata prawda o Routingu

Wywiad tygodnia:

Jak wygląda życie na najważniejszym skrzyżowaniu tras w Rails?

Head First: Ach, Routing. Bardzo nam miło, że mogłeś nam poświęcić kilka chwil swego cennego czasu.

Routing: Nie, kontroler ads... ads... O tak, właśnie ten.

Head First: Routingu?

Routing: Uff — odsuń się, kolego! Nadchodzi żądanie... [biip... biip]

Head First: Cóż, widać, że jesteś bardzo zajęty. Ale tak naprawdę, mimo że w aplikacji Rails zajmujesz bardzo ważne stanowisko, niektóre osoby nie są do końca pewne, czym się zajmujesz.

Routing: Hej — nie pracuję w tym zawodzie dla sławy. Pracuję, by kierować i obsługiwać. To właśnie ja. Jestem jak policjant kierujący ruchem drogowym, wiesz? Widzisz, jak tamtymi drzwiami nadchodzi żądanie?

Head First: Co — tym portem?

Routing: Właśnie tak. Co to będzie na tym serwerze? A tak, port numer 3000. Przeglądarka internetowa żąda — no nie wiem — powiedzmy, że /donuts/cream.

Head First: Tak?

Routing: Ale Rails nie wie, który kod ma wykonać, by na to żądanie odpowiedzieć. Więc przychodzi do mnie i ja patrzę na /donuts/cream i sprawdzam to na takim arkuszu z trasami, który tu mam...

Head First: Faktycznie, jest ich dość sporo.

Routing: No właśnie. Sprawdzam więc listę i szukam pierwszej trasy, która wygląda mniej więcej tak jak /donuts/cream. Mogę znaleźć, powiedzmy, /donuts/:flavor.

Head First: Ta trasa wygląda dość podobnie. Ale w jaki sposób pomaga ci to w skierowaniu żądania do odpowiedniego kodu?

Routing: No cóż, każde żądanie zawiera dokumenty, które muszę wypełnić. Zbiór nazw i wartości zwanych parametrami żądania. Widzisz?

Head First: No tak. Mnóstwo różnych rzeczy.

Routing: No właśnie. Wszystkie żądania je mają. Nazywane są params[...]. Patrzę więc na trasę, która mówi mi, że każda ścieżka pasująca do /donuts/:flavor musi użyć kontrolera donuts z — powiedzmy — akcją display.

Head First: Ma to sens.

Routing: Dodaję zatem więcej elementów do params[...], na przykład params[:controller] z wartością donuts i params[:action] z wartością display...

Head First: ...a Rails wykorzysta to do wybrania kodu, który ma być wykonany.

Routing: Dokładnie tak! Szybko się uczyysz, dzieciaku! Rails mówi: „Och, teraz widzę. Muszę użyć kontrolera donuts. Zaprawdę, powiadam ci, teraz taki utworzę”.

Head First: „Zaprawdę, powiadam ci...”?

Routing: No dobrze, może bez „zaprawdę...”. Czego by jednak nie powiedział, wie dobrze, że musi utworzyć obiekt kontrolera donuts. A ponieważ wartość params[:action] ustawniona jest na display, kiedy powstaje obiekt kontrolera donuts, Rails wywołuje na nim metodę display.

Head First: A co z :flavor, wspomnianym przez ciebie w kontekście trasy?

Routing: A, to. No właśnie. Jeśli więc żądanie dotyczyło /donuts/cream, co pasuje do /donuts/:flavor, dodaje kolejny parametr za pomocą param[:flavor] = 'cream'. Zapisuję zatem, czego żądano, na wypadek gdyby było to później ważne dla kodu z kontrolera.

Head First: Dziękuję, Routingu, to był prawdziwy...

Routing: Hej, odsuń się na moment! Przepraszam. To taki nerwus, co zawsze dwa razy kliką każdy odnośnik. Po jednym naraz! Po jednym naraz!

Head First: Dziękujemy...

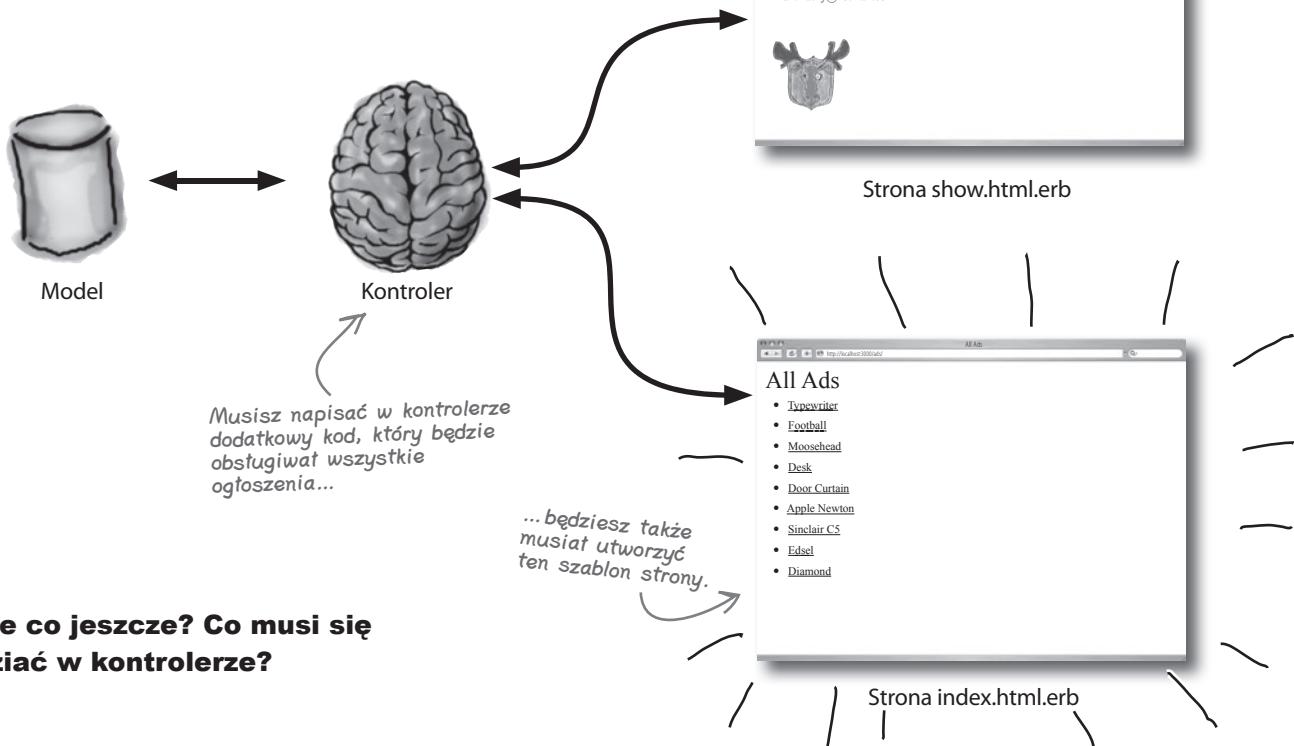
Routing: Nie ma o czym mówić. Słuchaj, mam tu teraz trochę roboty. Czemu nie pójdziesz dalej i nie sprawdzisz, co dzieje się w reszcie aplikacji? Właśnie, tam po lewej... Wydaje mi się, że jakiś nowy kod trafia właśnie do kontrolera ads...

Kontroler kontroluje

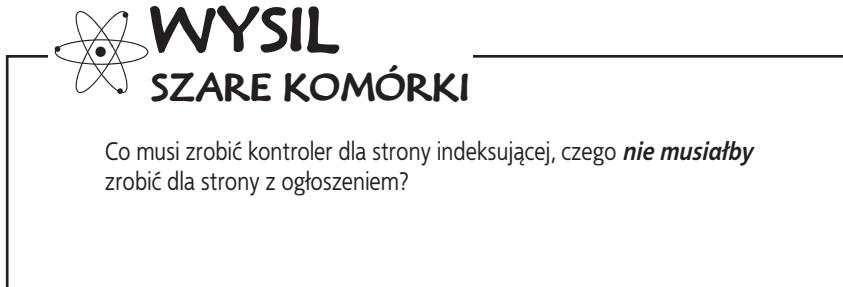
By przesyłać dane do widoku, będziesz potrzebował kodu kontrolera

Model jest już na miejscu; masz także już trasę dla niezbędnego nowego kodu kontrolera. Czy czegoś jeszcze brakuje?

No tak — dwóch elementów. Strona indeksująca potrzebuje odrębnego kodu w kontrolerze, ponieważ korzysta z wielu ogłoszeń i niezbędna jest nam nowa strona, która wyświetli je w widoku.



Ale co jeszcze? Co musi się dziać w kontrolerze?

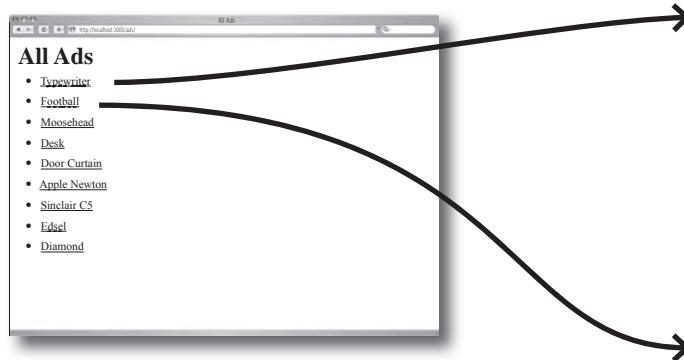


Strona indeksująca potrzebuje danych ze WSZYSTKICH rekordów

Stronie ogłoszenia wystarczą jedynie dane z pojedynczego rekordu, ale co ze stroną indeksującą? Będzie ona musiała odczytać dane z każdego rekordu z **całej** tabeli ads. Ale dlaczego?

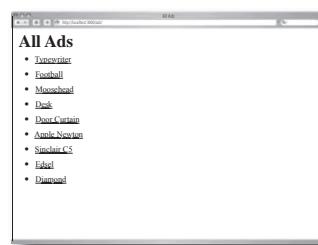
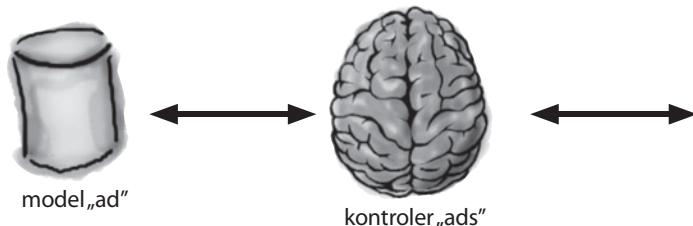
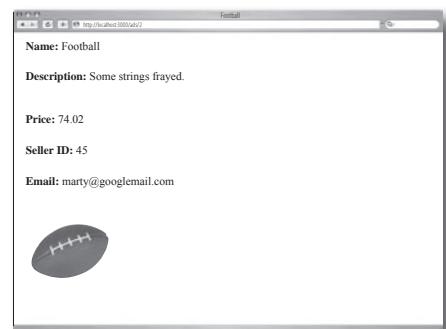
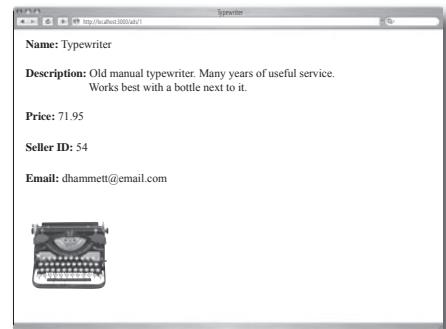
Przyjrzyj się projektowi strony indeksującej. Musi ona utworzyć odnośniki do **wszystkich** stron z ogłoszeniami, co oznacza, że musi znać nazwę oraz identyfikator **każdego** ogłoszenia znajdującego się w systemie.

Czy to jednak nie będzie problem? Dotychczas odczytywaliśmy tylko jeden rekord naraz. Teraz musimy odczytać całe mnóstwo rekordów naraz... właściwie *wszystkie* rekordy.



Jak można odczytać więcej niż jeden rekord?

Przed wyświetleniem strony kontroler wywoływany jest tylko **raz**, dlatego wszystkie rekordy muszą być odczytane w **całości** przed wywołaniem widoku.



Jak, Twoim zdaniem, kontroler może odczytać obiekty z modelu i przesłać je do widoku?

strona indeksująca ogłoszenia

Metoda Ad.find(:all) wczytuje całą tabelę naraz

Istnieje inna wersja **metody wyszukującej** `Ad.find(...)`, zwracająca dane z każdego rekordu całej tabeli ads:

```
def index
  @ads = Ad.find(:all)
end
```

Musisz dodać tę metodę do kontrolera.

Ten kod wczytuje wszystkie rekordy naraz.



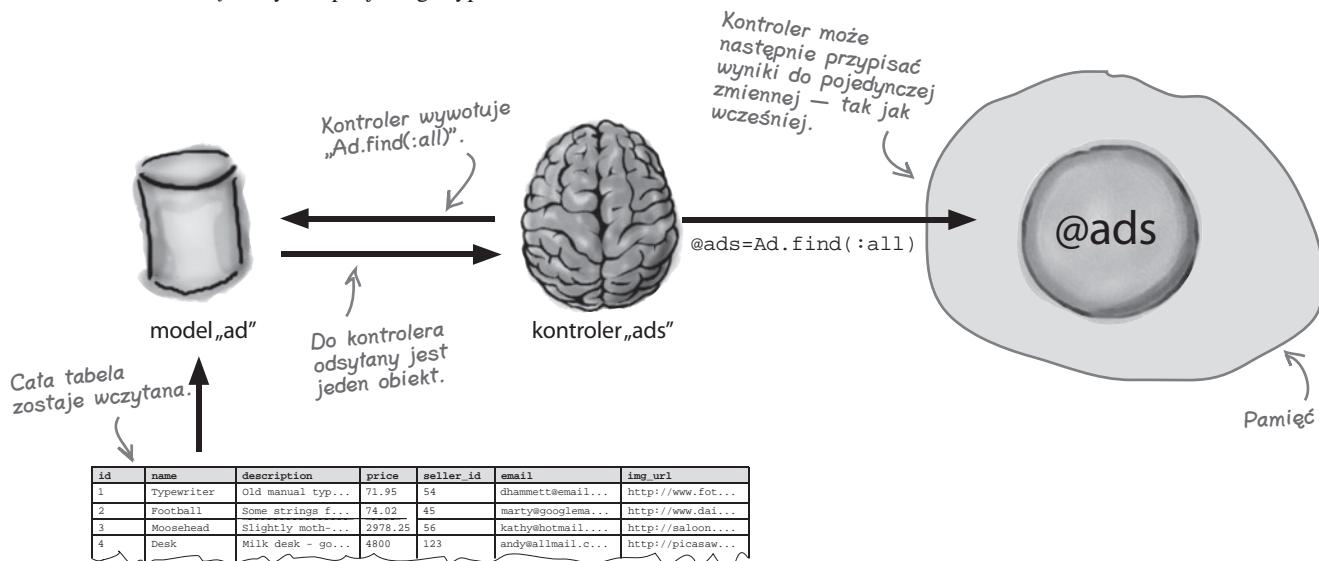
Jak to jednak działa? W końcu kiedy wczytywaliśmy pojedynczy rekord, wszystko było stosunkowo proste. Przekazywaliśmy identyfikator do modelu; model zwracał pojedynczy obiekt zawierający wszystkie dane z wiersza o odpowiednim identyfikatorze.

Teraz jednak nie wiemy, ile rekordów przyjdzie nam wczytać. Czy to nie oznacza, że będziemy potrzebować jakiegoś strasznie skomplikowanego kodu?

Na szczęście nie. W Rails odczytywanie każdego rekordu z tabeli jest podobne do odczytywania pojedynczego obiektu. Po wywoaniu metody `Ad.find(:all)` model zwraca **jeden obiekt** zawierający dane dla każdego rekordu z tabeli. Kontroler może przypisać obiekt do jednej zmiennej.

Jak jednak Rails może przechować wszystkie dane dla nieznanej liczby wierszy wewnętrz pojedynczego obiektu?

Robi tak dzięki użyciu specjalnego typu obiektu...

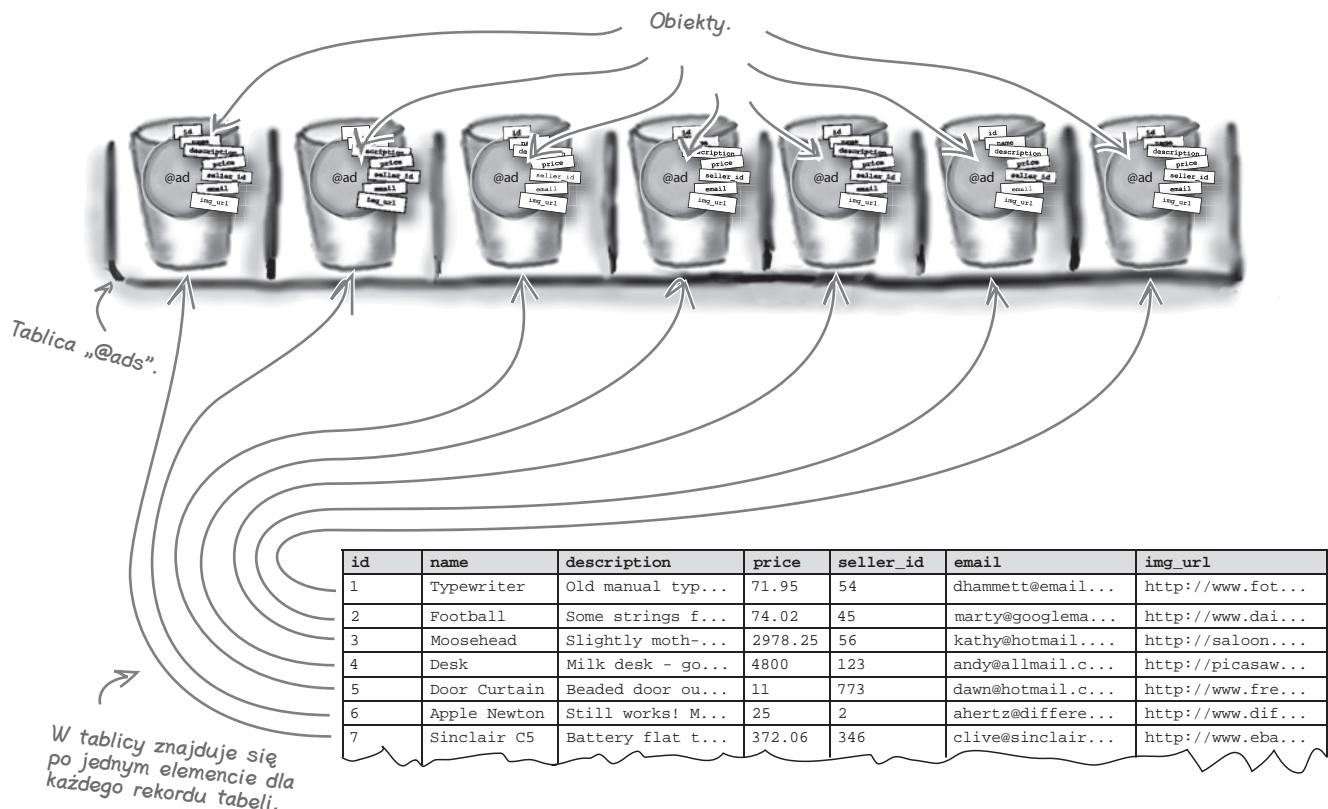


Dane zwracane są jako obiekt zwany tablica

Zamiast zwracać obiekt zawierający dane z pojedynczego rekordu, metoda `find(:all)` tworzy mnóstwo obiektów — po jednym dla każdego rekordu — a następnie opakowuje je w obiekt nazywany **tablica** (ang. *array*).

Metoda `Ad.find(:all)` zwraca pojedynczy obiekt tablicy, który z kolei zawiera tyle obiektów modelu, ile jest **wierszy** w tabeli bazy danych.

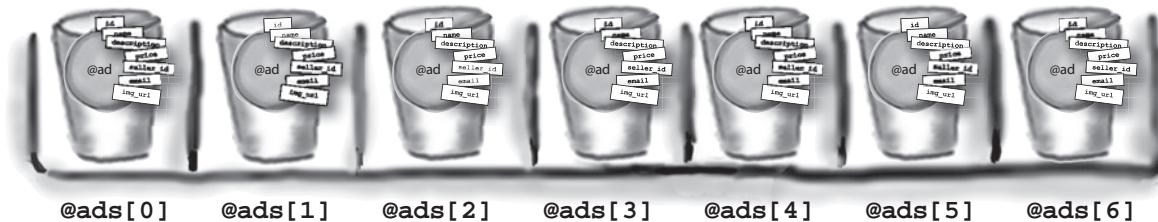
Kontroler może przechować pojedynczy obiekt tablicy w pamięci pod nazwą `@ads`. To ułatwia pracę szablonowi strony, ponieważ zamiast szukać nieznanej liczby obiektów modelu w pamięci, by uzyskać dostęp do wszystkich obiektów modelu, szablon musi jedynie znać nazwę tablicy.



W jaki sposób można jednak uzyskać dostęp do obiektów po tym, jak zostaną one umieszczone w tablicy?

Tablica to ponumerowana sekwencja obiektów

Tablica @ads przechowuje obiekty modelu jako sekwencję ponumerowanych przegródek, rozpoczynając od przegródki 0. Obiekty umieszczone w każdej z przegródek nazywane są *elementami* tablicy.



Poszczególne elementy tablicy można odczytać za pomocą numeru przegródki, w której się one znajdują.

@ads[4]

↑
Obiekt przechowywany
w przegródce 4 tablicy to
wiersz tabeli o id = 5.

Przegródki są zawsze numerowane od 0 w górę, a tablica może mieć dowolną wielkość, dlatego nie ma znaczenia, ile rekordów jest w tabeli, ponieważ wszystkie one mogą zostać umieszczone w pojedynczym obiekcie tablicy.



Uwaga!

Tablice
rozpoczynają się
od indeksu 0.

Oznacza to, że
pozycja każdego
elementu to wartość jego indeksu
plus jeden. W ten sposób @ads[0]
zawiera pierwszy element, @ads[1]
zawiera drugi i tak dalej.

Nie istnieją
głupie pytania

P: Dlaczego tablice zaczynają się od indeksu 0 zamiast 1?

O: Powodem są zaszłości historyczne. Większość języków programowania zawiera tablice i w większości przypadków ich indeksy rozpoczynają się od zera.

P: Czy kiedy umieszczam coś w tablicy, to tablica przechowuje osobną kopię tego czegoś?

O: Nie. Tablice przechowują jedynie referencje do obiektów przechowywanych w pamięci. Nie przechowują własnej kopii obiektu, pamiętają jedynie, gdzie się one znajdują.

P: Czy tablica naprawdę jest obiektem?

O: Tak. Tablica jest prawdziwym obiektem języka Ruby.

P: Jak duża może być tablica?

O: Nie ma ograniczenia rozmiaru tablicy, półki mieści się ona w pamięci.



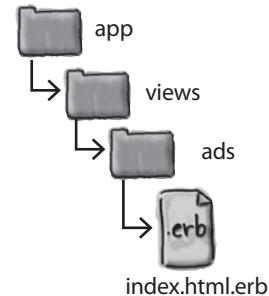
Ćwiczenie

Wstaw obiekty do strony, tak jakby w bazie danych znajdowały się tylko te trzy wiersze:

id	name	description	price	seller_id	email	img_url
1	Typewriter	Old manual typ...	71.95	54	dhammett@email...	http://www.fot...
2	Football	Some strings f...	74.02	45	marty@googlema...	http://www.dai...
3	Moosehead	Slightly moth-	2978.25	56	kathy@hotmail....	http://saloon....

Napisz swoją
odpowiedź tutaj.

Zapisz, jak mógłby wyglądać kod HTML pliku `index.html.erb`.



Wykorzystanie indeksów tablicy



Ćwiczenie Rozwiązanie

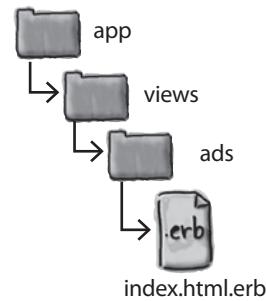
Wstaw obiekty do strony, tak jakby w bazie danych znajdowały się tylko te trzy wiersze:

id	name	description	price	seller_id	email	img_url
1	Typewriter	Old manual typ...	71.95	54	dhammett@email...	http://www.fot...
2	Football	Some strings f...	74.02	45	marty@googlema...	http://www.dai...
3	Moosehead	Slightly moth-...	2978.25	56	kathy@hotmail....	http://saloon....

Zapisz, jak mógłby wyglądać kod HTML pliku `index.html.erb`.

```
<html>
<head>
  <title>All Ads</title>
</head>
<body>
  <h1>All Ads</h1>
  <ul>
    <li><a href="/ads/<%= @ads[0].id %>"><%= @ads[0].name %></a>/li>
    <li><a href="/ads/<%= @ads[1].id %>"><%= @ads[1].name %></a>/li>
    <li><a href="/ads/<%= @ads[2].id %>"><%= @ads[2].name %></a>/li>
  </ul>
</body>
</html>
```

Twoj kod HTML może wyglądać nieco inaczej od tego. I dobrze. Dopóki masz w nim mniej więcej te same elementy w mniej więcej tej samej kolejności, dasz sobie radę.



Nie jestem pewien,
czy to ćwiczenie jest do końca
poprawne. A co jeśli w tabeli
nie znajdują się akurat trzy
rekordy?



W praktyce nie będziesz wiedział, ile jest ogłoszeń.

Powyższy kod wyświetla jedynie trzy ogłoszenia. A co jeśli, jest ich cztery, pięć czy trzy tysiące? Nie chcemy przecież zmieniać szablonu za każdym razem, gdy ogłoszenie zostanie dodane do bazy danych lub z niej usunięte.

Potrzebny nam jakiś sposób napisania kodu, który poradzi sobie z dowolną liczbą ogłoszeń w bazie danych.

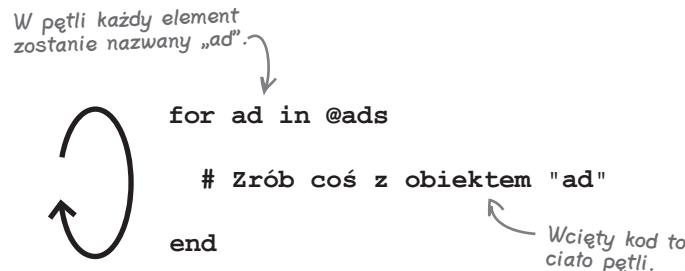


Wczytanie wszystkich ogłoszeń za pomocą pętli for

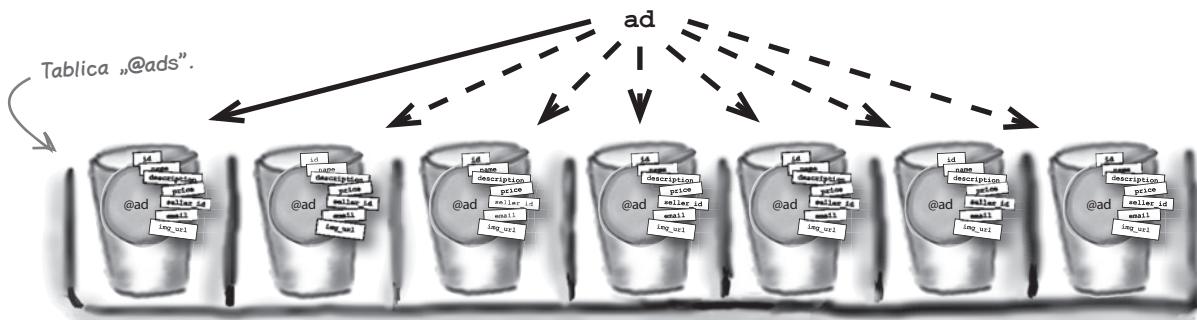
Pętla **for** języka Ruby pozwala wykonać ten sam fragment kodu w tym języku raz za razem. Można ją wykorzystać do odczytania elementów tablicy, po jednym na raz, a następnie wykonania fragmentu kodu na każdym z elementów.

Fragment kodu wykonywany za każdym razem nazywany jest **ciałem pętli**.

Ciało pętli zostanie wykonane dla każdego elementu tablicy po kolei, rozpoczynając od elementu 0:



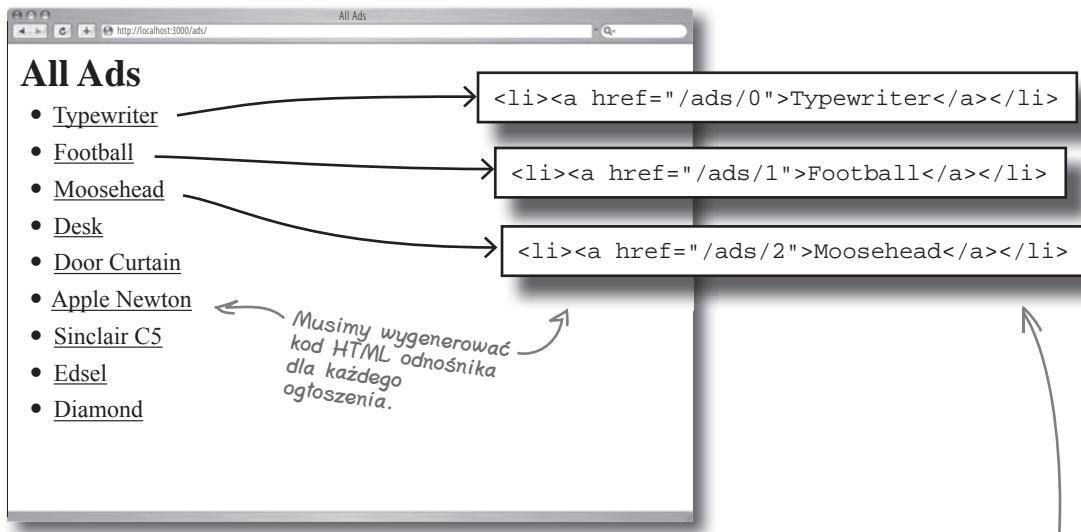
W powyższym kodzie za każdym razem, gdy wykonywane jest ciało pętli, bieżący element tablicy otrzymuje nazwę ad. W ten sposób ad odnosi się do każdego z obiektów modelu Ad, a wewnętrz pętli można uzyskać dostęp do wszystkich atrybutów obiektu modelu — szczegółów ogłoszenia, takich jak nazwa czy opis sprzedawanego przedmiotu.



Teraz musimy wygenerować kod HTML, który będzie tworzył odnośnik do strony internetowej ogłoszenia. Kod HTML jest jednak generowany przez szablon strony. Jak możemy w takiej sytuacji wykorzystać pętlę for?

Potrzebny nam kod HTML dla każdego elementu tablicy

Dla każdego obiektu ogłoszenia z tablicy @ads musimy w kodzie HTML wygenerować odnośnik.



By to osiągnąć, możemy użyć pętli `for`. Pętla pozwoli nam przejść po kolejce przez każde z ogłoszeń, po jednym na raz. Gdybyśmy wykorzystali ciało pętli do wygenerowania kodu HTML, utworzylibyśmy odnośniki dla każdego z ogłoszeń:

```
for ad in @ads
  <li><a href="/ads/<%= ad.id %>"><%= ad.name %></a></li>
end
```

Ten kod wygeneruje odnośnik dla każdego elementu.

To nie jest prawdziwy kod.

Problem polega na tym, że strony internetowe generujemy, umieszczając wyrażenia języka Ruby wewnętrz szablonów stron. *Kod HTML szablonu strony kontroluje, kiedy wywoływane są wyrażenia języka Ruby.* My jednak chcemy zrobić coś odwrotnego. Chcemy, by pętla `for` języka Ruby kontrolowała, kiedy generowany jest kod HTML.

Jak możemy zatem połączyć instrukcje sterujące, takie jak pętle `for`, z kodem HTML szablonów stron?

Rails konwertuje szablony stron na kod języka Ruby

Kiedy wcześniej chcieliśmy otrzymać wartości obiektów na stronach, wstawialiśmy je za pomocą `<%= ... %>`:

```
<%= @ad.name %>
```

ERb (Embedded Ruby) generuje stronę internetową z szablonu, zastępując każde wyrażenie jego wartością. Robi to dzięki konwersji całej strony na kod języka Ruby.

Wyobraź sobie, że w szablonie strony miałbyś tylko to:

```
<title><%= @ad.name %></title>
```

Znaczniki szablonu strony.

ERb generuje kod języka Ruby wyświetlający każde wyrażenie oraz każdy fragment kodu HTML. Powyższy kod szablonu zostanie zatem przekształcony na coś takiego:

```
print "<title>"  
print @ad.name  
print "</title>"
```

To jest pseudokod.
Prawdziwy kod jest trochę bardziej skomplikowany.

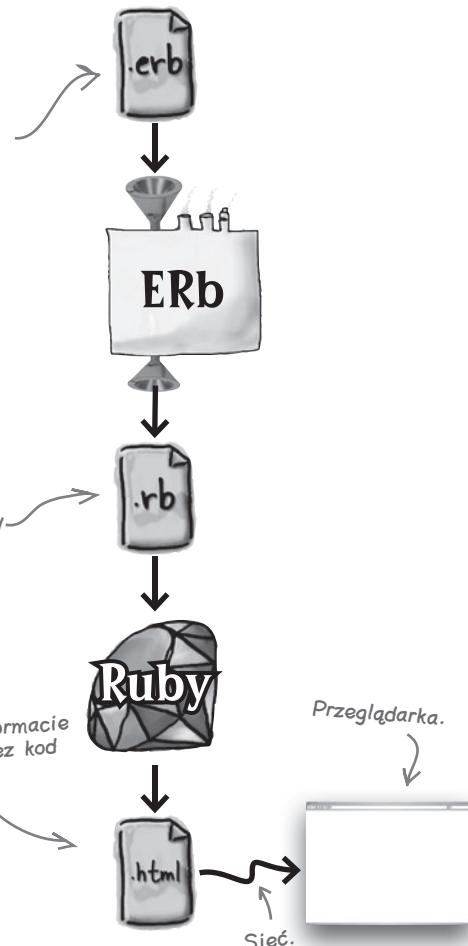
Kod języka Ruby wygenerowany przez ERb.

Kod języka Ruby zostaje następnie wykonany, a dane wyjściowe przesyłane są za pośrednictwem sieci do przeglądarki.

```
<title>Moosehead</title>
```

Dane wyjściowe w formacie HTML zwarcane przez kod w Ruby.

Jak powinien wyglądać kod języka Ruby, gdybyś chciał, by szablon wygenerował kod dla każdego obiektu tablicy?



Pętle można dodawać do szablonów stron za pomocą scriptletów

Zapomnijmy na moment o szablonach stron. Gdybyś miał napisać kod wyświetlający kod HTML dla każdego elementu tablicy, jak by on wyglądał? Móglby wyglądać mniej więcej tak:

```
for ad in @ads
    print '<li><a href="/ads/' + ad.id +
    print '>' + ad.name
    print '</a></li>'
end
```

Scriptlety i wyrażenia zwrócą ten kod.

Musimy wykonać pętlę na tablicy i wyświetlić kod HTML oraz wyrażenia dla każdego z jej elementów. Dotychczas widzieliśmy, jak system ERb generował jedynie polecenia `print`, jednak pętla `for` to nie to samo co polecenie `print`. Jak możemy przekazać do ERb fragmenty kodu w języku Ruby — takie jak pętla `for`?

Rozwiązaniem jest użycie *scriptletów*.

Scriptlet to znacznik zawierający fragment kodu języka Ruby. Wyrażenia otoczone są przez znaki `<%= ... %>`, natomiast scriptlety — przez `<% ... %>`. Scriptlety nie mają znaku `=` na początku.

By zobaczyć, jak działają scriptlety, spójrzmy na szablon strony tworzący kod powyżej pętli `for`:

```
<% for ad in @ads %>
<li><a href="/ads/<%= ad.id %>"><%= ad.name %></a></li>
<% end %>
```

Na początku scriptletu nie ma znaku „=”.

Wyrażenia zawierają znak „=”.

Na końcu scriptletu nie ma znaku „=”.

Powyższy kod wykorzystuje scriptlety do stworzenia kodu pętli, a wyrażenia — tam, gdzie mają być wstawione wartości. Sprawdźmy, jak będzie wyglądał szablon strony indeksującej, jeśli scriptlety wykorzystamy do wykonania pętli na tablicy `@ads`.

Wartości obiektów wstawiane są za pomocą `<%= ... %>`.

Kod wstawiany jest za pomocą `<% ... %>`.

Wyrażenie

Scriptlet

Z każdym przejściem pętli strona generuje jeden odnośnik

To jest kod, z którego będziesz korzystał w szablonie index.html.erb:

```
<html>
<head>
  <title>All Ads</title>
</head>
<body>
<h1>All Ads</h1>
<ul>
<% for ad in @ads %>
  <li><a href="/ads/<%= ad.id %>"><%= ad.name %></a></li>
<% end %>
</ul>
</body>
</html>
```

Kiedy Rails przetwarza szablon, kod HTML na górze i na dole pliku zostanie zwrócony dokładnie tak, jak oczekujemy. Ciekawa część znajduje się w środku strony. Każde przejście pętli generuje odnośnik HTML do odpowiedniej strony z ogłoszeniem.



Jak wygląda wygenerowany kod HTML?

Wyobraź sobie, że w bazie danych znajdują się tylko te trzy ogłoszenia.

Oznacza to, że kontroler utworzy tablicę @ads zawierającą trzy obiekty modelu. Kiedy szablon strony wykona pętlę na tablicy @ads, powinien zwrócić kod HTML wyglądający mniej więcej tak:

```
<html>
  <head>
    <title>All Ads</title>
  </head>
  <body>
    <h1>All Ads</h1>
    <ul>
      <li><a href="/ads/1">Typewriter</a></li>
      <li><a href="/ads/2">Football</a></li>
      <li><a href="/ads/3">Moosehead</a></li>
    </ul>
  </body>
</html>
```

Wygląda na to, że wygenerowaliśmy kod HTML wystarczający dla wszystkich ogłoszeń w bazie danych. Jeśli w bazie danych znajduje się więcej ogłoszeń, utworzona zostanie większa tablica @ads, a szablon wygeneruje dłuższy kod HTML.

Tyle teorii. Skoro utworzyliśmy już trasę, napisaliśmy akcję kontrolera, a szablon *index.html.erb* jest gotowy, czas wykonać kod.



Jazda próbna

Skoro trasa dla `/ads/` jest na miejscu, kontroler wczytuje wszystkie rekordy dzięki metodzie `Ad.find(:all)`, a szablon wykorzystuje scriptlet do osadzenia pętli `for` odczytująccej wszystkie obiekty modelu z tablicy `@ads`, czas przetestować nową stronę indeksującą.

- [Typewriter](#)
- [Football](#)
- [Moosehead](#)
- [Desk](#)
- [Door Curtain](#)
- [Apple Newton](#)
- [Sinclair CS](#)
- [Edsel](#)
- [Diamond](#)

Dobra robota!

Aplikacja jest gotowa, nowa witryna internetowa zostaje uruchomiona... a wszystko to udało Ci się osiągnąć bez rusztowania!

CELNE SPOSTRZEŻENIA



- Dane można wyświetlić dla **jednego rekordu**.
- Dane można wyświetlić dla **wszystkich rekordów** tabeli.
- Teraz potrafisz już pisać mnóstwo aplikacji tylko do odczytu!

Właśnie dostałeś wiadomość od ekipy z MeBay...

Funkcjonalność nowej strony dokładnie odpowiada oryginalnej specyfikacji. Wszyscy są zadowoleni. Tymczasem rano, w dzień, w którym strona ma zostać oficjalnie uruchomiona, otrzymujesz wiadomość:

Stary!

Wykonałeś niesamowitą robotę nad tą stroną. Jesteśmy naprawdę zadowoleni z tego, jak udało Ci się dopasować ją do naszej specyfikacji. Słyszeliśmy wcześniej, że aplikacje Rails zawsze wyglądają i działają tak samo!

Przy okazji, oto projekt pokazujący, jak będzie wyglądała strona. Myślimy, że taki będzie ostateczny wygląd aplikacji, ale jeśli coś się jeszcze zmieni, poinformujemy Cię później.

Jeszcze raz dzięki za całą ciężką pracę! :-)

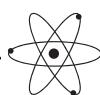
Do wiadomości dołączona jest próbka strony internetowej oraz zbiór arkuszy stylów i obrazków. Zmiana wyglądu aplikacji nie może być taka trudna, prawda?



Pobierz to!



Pobierz arkusze stylów oraz obrazki z serwera FTP:
<ftp://ftp.helion.pl/przyklady/hfror.zip>



WYSIL
SZARE KOMÓRKI

Można by zmodyfikować po prostu szablony stron, tak by wyglądały jak próbka strony pochodząca od projektanta. Czy będzie to problemem?

Ale my mamy dwa szablony stron... czy powinniśmy zmieniać kod każdego z nich?

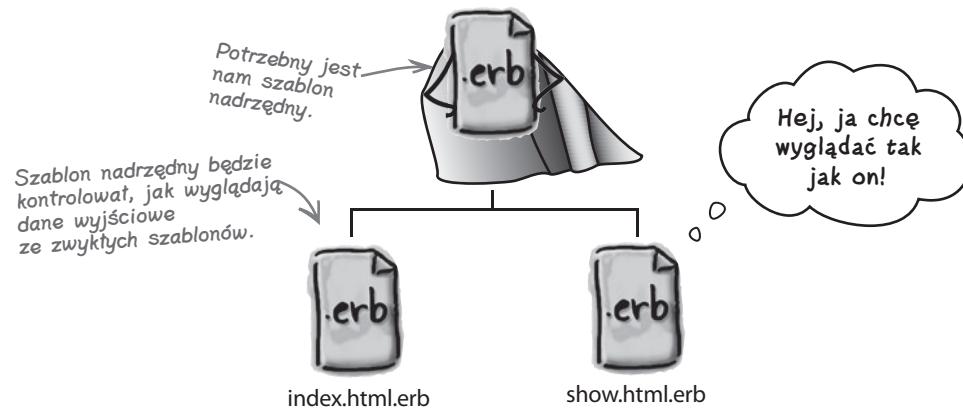
Mamy dwa szablony stron, dlatego jeśli zmienimy kod HTML w obydwu tak, by pasował do próbki strony z MeBay, będziemy musieli **zduplikować kod**.

Czy ma to jednak jakieś znaczenie? W końcu w witrynie MeBay znajdują się tylko *dwa* typy stron internetowych. To jeszcze nie *tak* źle, prawda?

Problem polega na tym, że aplikacja może z czasem urosnąć i wymagać coraz większej liczby opcji oraz szablonów stron. A do tego jeszcze ten komentarz, że projekt może się jeszcze zmienić. Im więcej razy zduplikujesz wygląd strony, w tym większej liczbie miejsc będziesz musiał utrzymywać ten sam kod HTML. Z czasem aplikację będzie coraz trudniej utrzymać.

Jakie zatem jest właściwe rozwiązanie? Cóż, oczywistym rozwiązaniem jest usunięcie duplikacji. Większość witryn internetowych ma ustandardyzowany wygląd dla większości podstron. Mają one standardowy, wzorcowy kod HTML otaczający zasadniczą treść każdej podstrony.

Potrzebny nam będzie jakiś sposób zdefiniowania **szablonu nadzawanego** — jednego szablonu, który będzie kontrolował wygląd grupy pozostałych szablonów.



Układ stron definiuje STANDARDOWY wygląd całego zbioru szablonów

Na szczęście w Rails istnieje taki szablon nadzawany, który nazywany jest **układem stron** (ang. *layout*). Układ strony definiuje kod HTML opakowujący wszystkie szablony należące do określonego modelu.

Zobaczmy, jak będzie to działało z nowym projektem.

Zasada Rails:
Nie powtarzaj się.

Czy już o tym nie mówiliśmy?



Gotowy do podania Kod szablonu nadziednego

Oto przykładowa strona HTML od projektanta, która została przekształcona na układ strony.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Ads: <%= controller.action_name %></title>
  <%= stylesheet_link_tag 'default.css' %>
</head>
<body>
  <div id="wrapper">
    <div id="header">
      <div>

        <h1>MeBay</h1>
        <ul id="nav">
          <li><a href="/ads/">All Ads</a></li>
        </ul>
      </div>
    </div>

    <div id="content">
      <%= yield %>
    </div>
    <div id="clearfooter"></div>
  </div>
  <div id="footer"></div>
</body>
</html>
```

Musisz umieścić go w dobrym miejscu, zapisując jako:

`app/views/layouts/ads.html.erb`

Ta nazwa mówi Rails, że układ strony należy zastosować do wszystkich szablonów stron należących do modelu ad.

W kodzie umieściliśmy kilka wyrażeń określających arkusze stylów i nadających stronie tytuł w oparciu o nazwę kontrolera. Co jednak ważniejsze, układ strony zawiera poniższy znacznik:

`<%= yield %>`

Zaostrz ołówek



Czy wstawienie danych wyjściowych aktualnych szablonów stron do układu strony może wiązać się z jakimiś problemami? Jeśli tak jest, napisz poniżej, z jakimi.

.....

Modyfikacja szablonów

Zaostrz ołówek



Rozwiążanie

Czy wstawienie danych wyjściowych aktualnych szablonów stron do układu strony może wiązać się z jakimiś problemami? Jeśli tak jest, napisz poniżej, z jakimi.

Szablony stron zawierają zbyt dużo — już umieszczono w nich cały standardowy kod HTML.

Standardowy kod HTML z szablonów

musi zostać USUNIĘTY

Spójrz na istniejący plik `index.html.erb`.

Zawiera on standardowe elementy HTML, takie jak `<head>` czy `<title>`:

```
<html>
<head>
  <title>All Ads</title>
</head>
<body>
  <h1>All Ads</h1>
  <ul>
    <% for ad in @ads %>
      <li><a href="/ads/<%= ad.id %>"><%= ad.name %></a></li>
    <% end %>
  </ul>
</body>
</html>
```



Teraz, gdy standardową część kodu udostępnia układ strony, szablony muszą zostać okrojone, tak by wyświetlały jedynie główną treść strony:

```
<h1>All Ads</h1>
<ul>
  <% for ad in @ads %>
    <li><a href="/ads/<%= ad.id %>"><%= ad.name %></a></li>
  <% end %>
</ul>
```

Zrób tak!

Zmodyfikuj pliki `index.html.erb` oraz `show.html.erb` w taki sposób, by usunąć z nich standardowy kod HTML.

A co z nową treścią statyczną wysłaną przez MeBay?

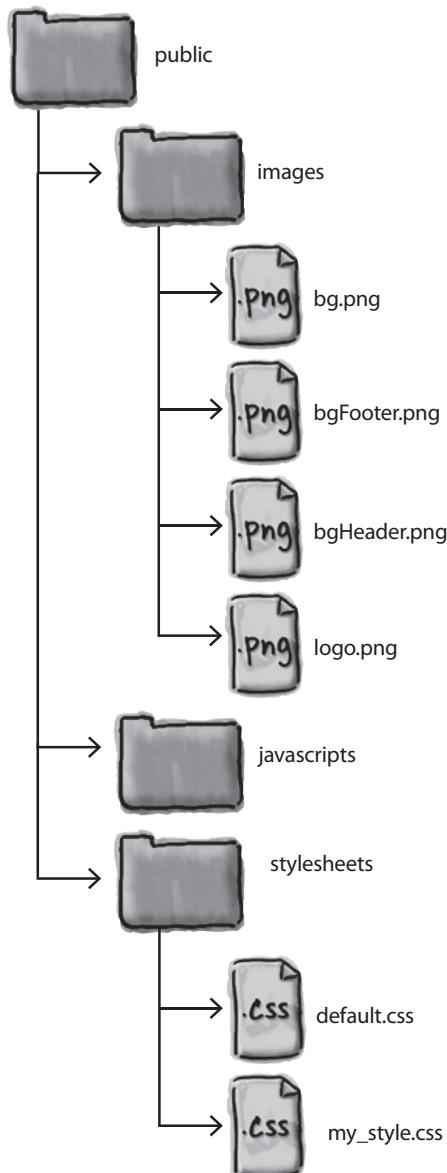
Jak na razie aplikacja Rails generowała jedynie zawartość dynamiczną. Prawie wszystko pochodziło z szablonów stron. Kiedy jednak zajmujemy się kosmetyczną częścią strony, często potrzebujemy zawartości **statycznej**, takiej jak arkusze stylów, obrazki czy skrypty w języku JavaScript. W jaki sposób możemy dołączyć zawartość statyczną do aplikacji?

Platforma Rails przygotowała folder przeznaczony specjalnie dla plików statycznych. Nosi on nazwę *public*.

Kiedy tworzyś aplikację, Rails od razu umieszcza w folderze *public* kilka plików. Pamiętasz, jak pierwszy raz uruchomiłeś aplikację Rails i patrzyłeś na jej stronę główną? Pliki standardowej strony powitalnej znajdują się w folderze *public*.

Większość aplikacji Rails przechowuje swoje obrazki, arkusze stylów i skrypty języka JavaScript, odpowiednio, w folderach *public/images*, *public/stylesheets* oraz *public/javascripts*.

Po zapisaniu dodatkowych obrazków oraz arkuszy stylów z e-maila powinieneś być gotów do pracy.





Jazda próbna

Otwórz w przeglądarce stronę znajdująca się pod adresem:

`http://localhost:3000/ads`

All Ads

- Typewriter
- Football
- Mousehead
- Pest
- Door curtain
- Apple Newton
- Sinclair C5
- Edsel
- Diamond

Ads: show

Name:Typewriter
Description:Old manual typewriter. Many years useful service. Works best with a bottle next to it.
Price:71.95
Seller Id:54
Email:dhammett@email.com

Jest pięknie.
Wykonałem swoje
zadanie... na razie.



Kiedy będziesz przeglądał witrynę, jej standardowy wygląd zostanie zastosowany do wszystkich podstron. A jeśli później dodasz więcej szablonów lub zmodyfikujesz kod HTML układu strony, aplikacja zachowią spójny wygląd.

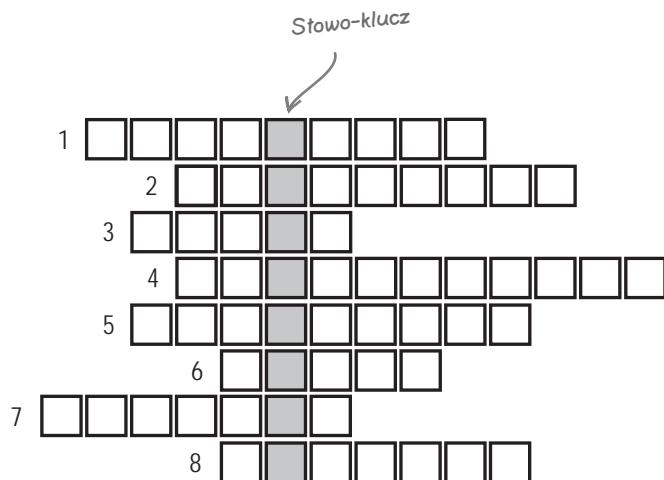


Krzyżówka bez rusztowania

Uzupełnij krzyżówkę odpowiedziami na każde z pytań-wskazówek w celu odkrycia słowa-klucza.

Wskazówka dla słowa-klucza:

Powód, dla którego możesz chcieć ręcznie utworzyć aplikację, zamiast korzystać z rusztowania.



Wskazówki

1. <% @ja.jestem? %>
2. <%= @ja.jestem? %>
3. Dla tego elementu możesz wykorzystać szablon strony.
4. Jeśli tworzysz prostą aplikację, możesz tego nie potrzebować.
5. Może przesyłać dane z modelu do widoku.
6. Konwertuje dane z bazy na obiekty języka Ruby.
7. Strukturę danych uaktualnia się za pomocą `rake db:`
8. Obiekt zawierający wiele obiektów.



Krzyżówka bez rusztowania

Uzupełnij krzyżówkę odpowiedziami na każde z pytań-wskazówek w celu odkrycia słowa-klucza.

Wskazówka dla słowa-klucza:

Powód, dla którego możesz chcieć ręcznie utworzyć aplikację, zamiast korzystać z rusztowania.

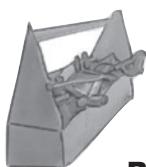
Słowo-klucz

The crossword grid consists of 8 numbered rows. Row 1: SCRIPTURE. Row 2: WYRAZENIE. Row 3: WIADOK. Row 4: RUSZTOWANIE. Row 5: KONTROLER. Row 6: MODEL. Row 7: MIGRATĘ. Row 8: TABLICA. Letters highlighted in grey include Y, A, Z, E, N, I, E, D, O, K, U, S, Z, T, O, W, A, N, I, E, O, D, E, L, R, G, R, A, T, E, A, B, L, I, C, A. An arrow points from the word 'Słowo-klucz' to the grid.

1	S	C	R	I	P	T	L	E	T
2	W	Y	R	A	Z	E	N	I	E
3	W	I	D	O	K				
4	R	U	S	Z	T	O	W	A	N
5	K	O	N	T	R	O	L	E	R
6	M	O	D	E	L				
7	M	I	G	R	A	T	E		
8	T	A	B	L	I	C	A		

Wskazówki

1. <% @ja.jestem? %>
2. <%= @ja.jestem? %>
3. Dla tego elementu możesz wykorzystać szablon strony.
4. Jeśli tworzysz prostą aplikację, możesz tego nie potrzebować.
5. Może przesyłać dane z modelu do widoku.
6. Konwertuje dane z bazy na obiekty języka Ruby.
7. Strukturę danych uaktualnia się za pomocą rake db:
8. Obiekt zawierający wiele obiektów.



Niezbędnik programisty Rails

Masz za sobą rozdział 2. i teraz
do swojego niezbędnika programisty
Rails możesz dodać umiejętność
ręcznego tworzenia aplikacji Rails
tylko do odczytu.

Narzędzia Rails

Model można wygenerować za pomocą:

`ruby script/generate model...`

a kontroler za pomocą:

`ruby script/generate controller...`

Narzędzia języka Ruby

Jeśli `my_array` jest tablicą języka Ruby,
pierwszy element uzyskasz dzięki:

`my_array[0]`

Pętla po wszystkich elementach wykonuje się
za pomocą:

```
for element in my_array
  # Zrób coś z elementem
end
```


3. Wstawianie, uaktualnianie i usuwanie



Zmiana to część życia — szczególnie w przypadku danych. Na razie widziałeś, jak można szybko wyczarować aplikację Rails dzięki rusztowaniu, a także jak napisać własny kod w celu publikacji danych z bazy. Ale co zrobić, kiedy chcemy, by użytkownicy mogli edytować dane w zaplanowany *przez nas* sposób? Co jeśli rusztowanie nie robi tego, co chcemy *my*? W tym rozdziale nauczysz się **wstawać, uaktualniać i usuwać** dane dokładnie tak, jak tego chcesz. A przy okazji zobaczysz również, jak tak *naprawdę* działa Rails, i być może nauczysz się również czegoś o bezpieczeństwie.

Ludzie chcą sami publikować ogłoszenia w Internecie

Użytkownicy kochają witrynę MeBay, jednak jest jeden problem. Ponieważ firma MeBay bała się dać użytkownikom zbyt szeroki dostęp do danych, sprzedający musieli telefonicznie podawać szczegóły swoich ofert pracownikom firmy, a następnie *czekali*, aż administratorzy systemu utworzą dla nich nowe ogłoszenia. W miarę wzrostu liczby sprzedających przedmioty wzrósł również czas oczekiwania. Wiele osób przenosi teraz swoje ogłoszenia do konkurencyjnych portali.

Firma MeBay poddała się wymaganiom klientów. Po wielu dyskusjach zdecydowano, że ludzie powinni móc **dodawać swoje własne ogłoszenia** w witrynie za pomocą strony wyglądającej tak:

New ad

Name:

Description:

Price:

Seller ID:

Email:

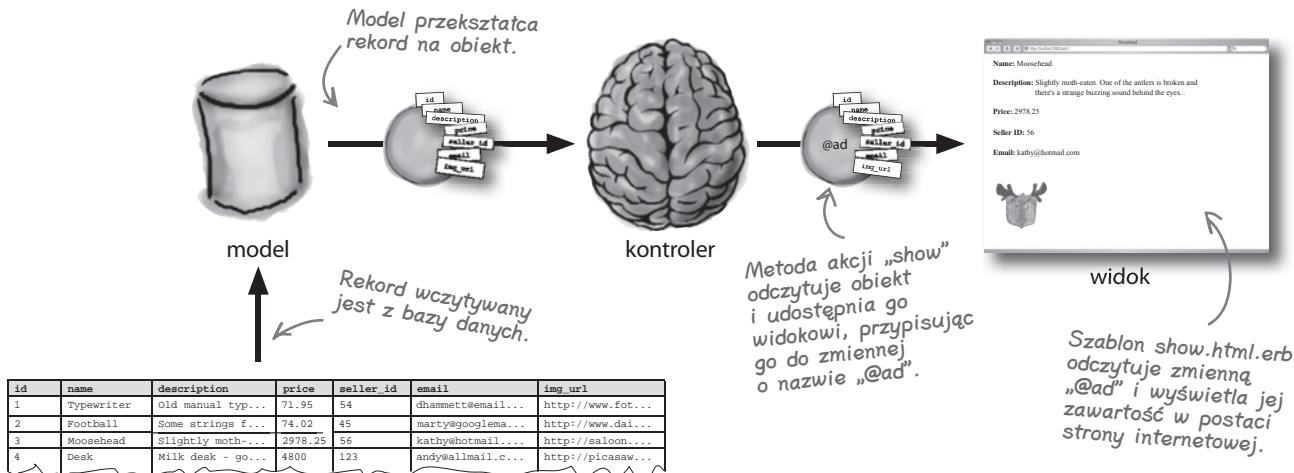
Img URL:

Create

Kolejny szkic projektowy
z firmy MeBay.

Wiesz już, jak budować aplikację publikującą dane z bazy

W obecnej wersji aplikacji ogłoszenia wędrują tylko w jednym kierunku. Rekordy ogłoszeń są odczytywane z bazy danych za pomocą modelu, który przekształca je w obiekty ogłoszeń przesypane do widoku przez kontroler. Działa to mniej więcej tak:



Zaostrz ołówek



Narysuj swój diagram tutaj.

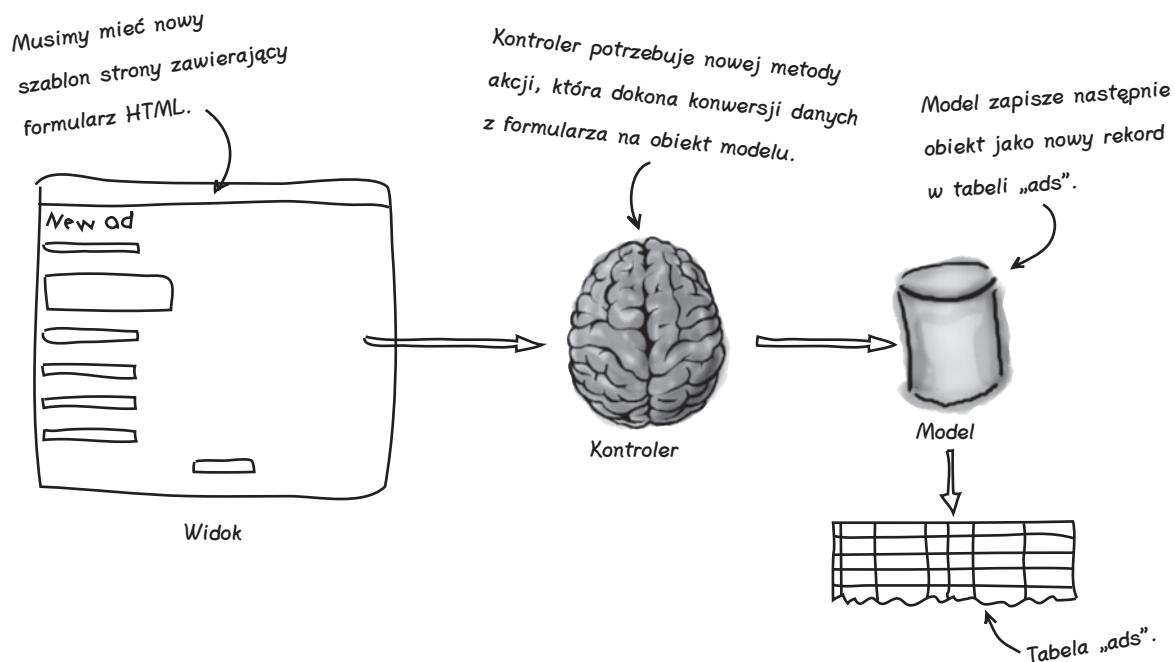
Narysuj diagram pokazujący, jak Twoim zdaniem, będzie działała nowa opcja dodawania ogłoszeń. Pamiętaj, by uwzględnić główne komponenty aplikacji i dodać notatki opisujące, co będzie robił każdy komponent.



Zaostrz ołówek

Rozwiążanie

Narysuj diagram pokazujący, jak, Twoim zdaniem, będzie działała nowa opcja dodawania ogłoszeń. Pamiętaj, by uwzględnić główne komponenty aplikacji i dodać notatki opisujące, co będzie robił każdy komponent.



Zapisywanie danych działa dokładnie ODWROTNIE do ich odczytywania

Zapisywanie danych w bazie jest podobne do publikowania ogłoszeń z bazy danych, tylko działa w drugą stronę. Zamiast szablonu strony *wyświetlającego* ogłoszenie potrzebujemy szablon do **dodawania** ogłoszenia. Zamiast metody akcji kontrolera przesyłającej ogłoszenie do strony potrzebna nam metoda kontrolera odczytująca dane ze strony i zamieniająca je w obiekt. A zamiast modelu wczytującego rekord i konwertującego go na obiekt potrzebny nam model konwertujący obiekt w nowy rekord bazy danych.

Potrzebny nam formularz służący do dodawania danych oraz metoda akcji zapisująca te dane

Potrzebny nam nowy szablon strony tworzący formularz HTML. Ponieważ będzie on wykorzystywany do wprowadzania nowych ogłoszeń, nazwiemy ten szablon `new.html.erb`.

Strona „New ad” („Nowe ogłoszenie”) będzie dostępna pod adresem:

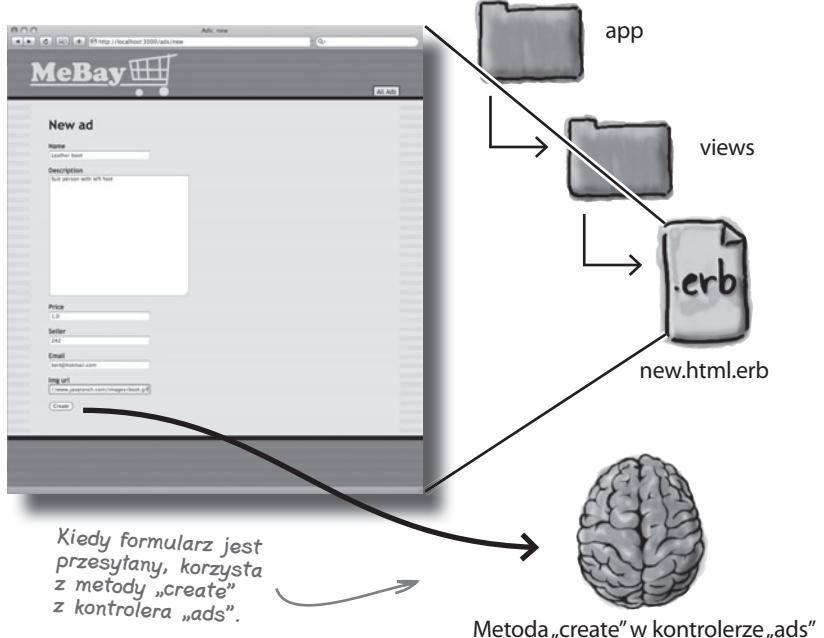
`http://mebay.com/ads/new`

natomiast formularz zostanie przesłany do:

`http://mebay.com/ads/create`

Musimy zatem utworzyć również nową trasę w pliku `routes.rb`.

Pamiętaj, ta trasa mówi Rails, którego fragmentu kodu należy użyć w celu obsługiżenia żądania z przeglądarki.



Zaostrz ołówek



Metoda kontrolera będzie musiała utworzyć obiekt modelu ogłoszenia z danych w formularzu. Czy widzisz w modelu atrybut, który nie ma swojego odpowiednika w polu formularza? Dlaczego tak jest?

Brakujący atrybut:

Powód jego nieobecności:

Jak będzie wyglądała trasa łącząca /ads/new z plikiem new.html.erb, /ads/create z metodą create w kontrolerze ads?

.....

.....

Identyfikatory są generowane

Zaostrz ołówek

Rozwiążanie



Metoda kontrolera będzie musiała utworzyć obiekt modelu ogłoszenia z danych w formularzu. Czy widzisz w modelu atrybut, który nie ma swojego odpowiednika w polu formularza? Dlaczego tak jest?

Brakujący atrybut: Atrybut „id”

Powód jego nieobecności: To nie użytkownik decyduje o tym, jaki będzie identyfikator — zostanie on automatycznie wygenerowany przez system.

Jak będzie wyglądała trasa łącząca /ads/new z plikiem new.html.erb, a /ads/create z metodą create w kontrolerze ads?

map.connect '/ads/new', :controller=>'ads', :action=>'new'

Ponieważ metoda będzie się nazywać „create”...

map.connect '/ads/create', :controller=>'ads', :action=>'create'

...plik będzie się nazywać new.html.erb.

Musimy dodać te trasy do pliku routes.rb.

Na górze Twojego pliku config/routes.rb powinno się teraz znajdować:

```
ActionController::Routing::Routes.draw do |map|
  map.connect '/ads/new', :controller=>'ads', :action=>'new'
  map.connect '/ads/create', :controller=>'ads', :action=>'create'
  map.connect '/ads/', :controller=>'ads', :action=>'index'
  map.connect '/ads/:id', :controller=>'ads', :action=>'show'
```

Nowe trasy powinny zostać dodane na górze pliku, by zapobiec pomieszaniu ich z trasą „/ads/:id”.

Wygląda na to, że formularze i obiekty zawierają wiele tych samych typów informacji. Zastanawiam się, czy istnieje jakiś głębszy związek pomiędzy formularzem a obiektem...



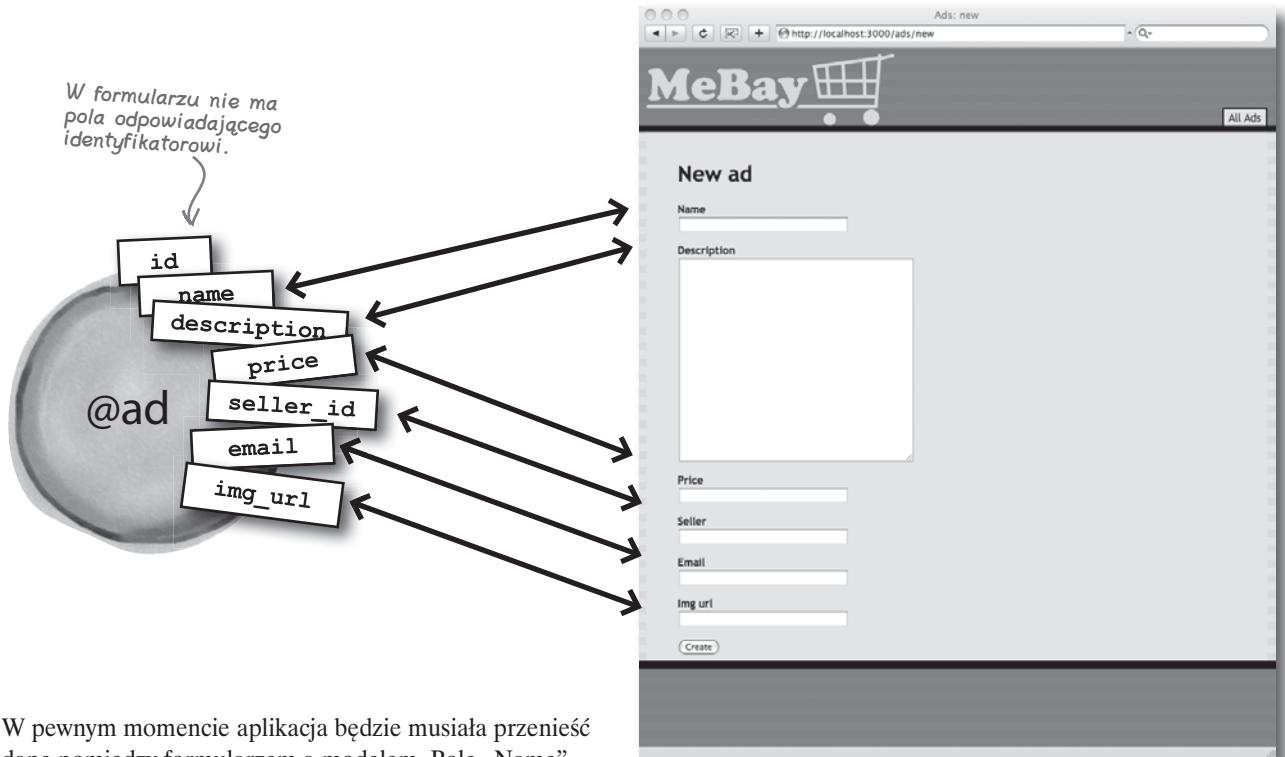
Pomiędzy wieloma częściami aplikacji Rails istnieją bliskie związki.

W końcu model zawiera dane dla aplikacji, widok pozwala użytkownikowi na dostęp do tych danych, a kontroler udostępnia logiczne spojwisko łączące wszystko ze sobą.

Czy jednak istnieje jakiś specjalny związek między formularzem a modelem?

Czy formularze i obiekty są ze sobą powiązane?

Poza wygenerowanym identyfikatorem pola formularza odpowiadają atrybutom obiektu ad.



W pewnym momencie aplikacja będzie musiała przenieść dane pomiędzy formularzem a modelem. Pole „Name” odpowiada atrybutowi name, pole „Description” — atrybutowi description i tak dalej.

Co się jednak dzieje, jeśli model tworzy obiekty z wartościami domyślnymi w atrybutach? Czy kod generujący wartości domyślne w formularzu powinien duplikować kod modelu?

Czy kiedy dane z formularza otrzymywane są przez kontroler, formularz powinien traktować pola jak pojedyncze wartości? Czy też może wszystkie wartości pól powinny być ze sobą powiązane, jak atrybuty obiektu?

Czy Rails może skorzystać z powiązania pomiędzy polami formularza a obiektem modelu przy tworzeniu formularza?

Rails może tworzyć formularze powiązane z obiektami modelu

Rails może wykorzystać obiekt modelu do pomocy w utworzeniu formularza. Oznacza to dwie rzeczy:

- 1 Wartości pól formularza zostaną ustawione na wartości przechowywane w atrybutach obiektu @ad. W przypadku formularza służącego do dodawania ogłoszenia nie ma to zbyt dużego znaczenia, ponieważ nowe ogłoszenia są puste.
- 2 Pola formularza otrzymają nazwy w jawnym sposób wiążące je z obiektem modelu.

W jaki sposób nazwa może wiązać pole formularza z obiektem?

Spójrzmy, co oznaczałoby to dla formularza dodającego ogłoszenia. Poniżej znajduje się kod HTML, który zostanie wygenerowany dla formularza opartego na obiekcie Ad:

Kod ten zostaby wygenerowany przez szablon show.html.erb.

```
<b>Name</b><br />
<input id="ad_name" name="ad[name]" type="text" />
<b>Description</b><br />
<textarea id="ad_description" name="ad[description]"></textarea>
<b>Price</b><br />
<input id="ad_price" name="ad[price]" type="text" />
<b>Seller</b><br />
<input id="ad_seller_id" name="ad[seller_id]" type="text" />
<b>Email</b><br />
<input id="ad_email" name="ad[email]" type="text" />
<b>Img url</b><br />
<input id="ad_img_url" name="ad[img_url]" type="text" />
```



WYŁĘŻ UMYSŁ

Kiedy porównasz nazwy pól oraz odpowiadające im atrybuty, jak sądzisz, w jaki sposób Rails zaprezentuje dane formularza kontrolerowi?

Masz jeszcze kilka stron, by się nad tym zastanowić...

Nazwy pól	Atrybuty obiektu
ad[name]	name
ad[description]	description
ad[price]	price
ad[seller_id]	seller_id
ad[email]	email
ad[img_url]	img_url



Magnesiki z kodem — formularz

Czas napisać szablon strony new.html.erb.

Znacznik `<% form_for %>` wykorzystywany jest do wygenerowania formularza za pomocą obiektu formularza. Uzupełnij pola formularza, używając poniższych magnesików:

Żadnego standardowego kodu HTML... wciąż korzystamy z szablonu nadzędzennego.

```

<h1>New ad</h1>
<% form_for(@ad, :url=>{:action=>'create'}) do |f| %>
  <p><b>Name</b><br /><%= f.text_field :name %></p>
  <p><b>Description</b><br /><%= ..... %></p>
  <p><b>Price</b><br /><%= ..... %></p>
  <p><b>Seller</b><br /><%= ..... %></p>
  <p><b>Email</b><br /><%= ..... %></p>
  <p><b>Img url</b><br /><%= ..... %></p>
  <p><%= ..... %></p>
<% end %>

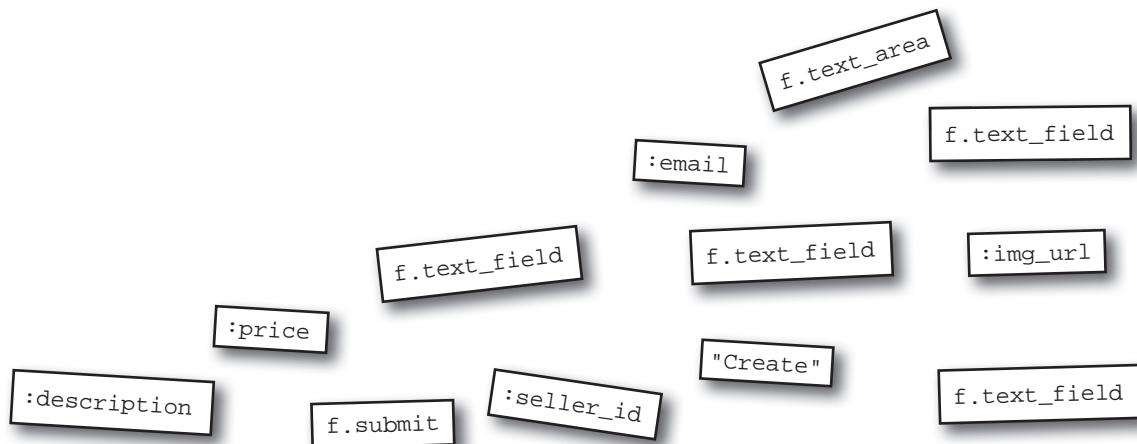
```

Znaczniki formularza to scriptlety, dlatego nie używamy w nich znaku `=`. Użyj po prostu `<%` oraz `%>`.

Akcja, do jakiej zostanie przestany formularz.

Pola formularza są za pomocą takich wyrażeń.

Obiekt, na którym oparty jest formularz.

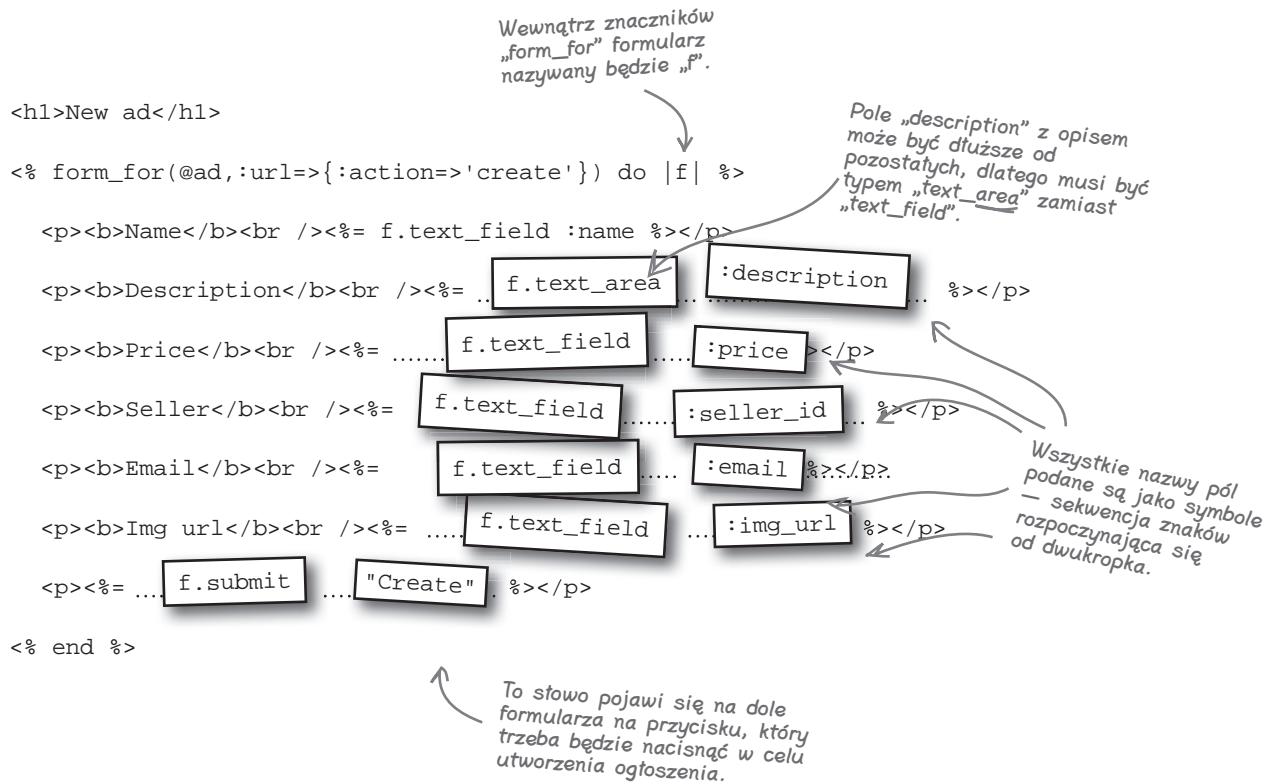




Magnesiki z kodem — formularz: Rozwiązańe

Czas napisać szablon strony new.html.erb.

Znacznik `<% form_for %>` wykorzystywany jest do wygenerowania formularza za pomocą obiektu formularza. Uzupełnij pola formularza, używając poniższych magnesików:



Zrób tak!

Zapisz ten kod w pliku o nazwie
`app/views/ads/new.html.erb`

Nie istnieja
głupie pytania

P: Dlaczego formularz nie zawiera żadnych pól odpowiadających kolumnom `id`, `updated_at` oraz `created_at`?

O: Te pola zostaną wypełnione automatycznie przez Rails. Pole `id` stanie się automatycznie wygenerowaną liczbą, natomiast pola `updated_at` oraz

`created_at` otrzymają datę i czas zapisu lub uaktualnienia rekordu w bazie danych. To są nasze magiczne kolumny, pamiętasz?



Jazda próbna

Szablon strony przeznaczony dla formularza `new` jest na swoim miejscu i powinien wygenerować kod HTML w taki sam sposób, jak generowaliśmy strony `show`.

Na miejscu znajduje się również trasa łącząca ścieżkę `/ads/new` z szablonem `new`.

Sprawdźmy zatem, czy wszystko działa, przechodząc na stronę:

```
http://localhost:3000/ads/new
```

```
Called id for nil, which would mistakenly be 4 -- if you really wanted the id of nil, use object_id
```

```
1: <h1>New ad</h1>
2: <% form_for(@ad,:url=>{:action=>'create'}) do |f| %>
3:   <p><b>Name</b><br /><%= f.text_field :name %></p>
4:   <p><b>Description</b><br /><%= f.text_area :description %></p>
5:   <p><b>Price</b><br /><%= f.text_field :price %>
```

Na stronie jest błąd!



**WYSIL
SZARE KOMÓRKI**

Na stronie formularza coś jest nie tak. Spójrz na zwrócony komunikat o błędzie i sprawdź, czy jesteś w stanie stwierdzić, co poszło źle.

Obiekt formularza @ad nie został utworzony

Problem ten spowodowany został przez obiekt @ad. Domyślnie zmienne takie jak @ad otrzymują specjalną wartość zwaną **nil**, co oznacza „*brak wartości*”. Jeśli zmienna @ad została ustawiona na nil zamiast na obiekt Ad, nie będzie miała atrybutów, takich jak **@ad.name**, **@ad.description** i tak dalej.

Czy skoro @ad nie ma atrybutów, może to być problemem dla formularza?

I to jeszcze jak! Formularz oparty jest na obiekcie @ad i uzyskuje dostęp do każdego z atrybutów tego obiektu w celu wygenerowania początkowych wartości pól. W momencie wywołania pierwszego atrybutu zwieracana jest wartość nil, co powoduje błąd.

Rails tworzy zmienną „@ad” z domyślną wartością „nil”, czyli inaczej: z brakiem wartości. Inne atrybuty, takie jak „@ad.name”, nie są zatem dostępne.

@ad = nil

Jak możemy uniknąć tego problemu?

```
<h1>New ad</h1>
<% form_for(@ad, :url=>{:action=>'create'}) do |f| %>
  <p><b>Name</b><br /><%= f.text_field :name %></p>
  <p><b>Description</b><br /><%= f.text_area :description %></p>
  <p><b>Price</b><br /><%= f.text_field :price %></p>
  <p><b>Seller</b><br /><%= f.text_field :seller_id %></p>
  <p><b>Email</b><br /><%= f.text_field :email %></p>
  <p><b>Img url</b><br /><%= f.text_field :img_url %></p>
  <p><%= f.submit "Create" %></p>
<% end %>
```

Początkowa wartość każdego z pól uzyskuje dostęp do odpowiadającego temu polu atrybutu obiektu „@ad”. Ale ponieważ „@ad” ustawione jest na „nil”, pojawia się błąd.

Obiekt formularza musi zostać utworzony przed wyświetleniem formularza

Kiedy generowana jest strona z formularzem, początkowe wartości każdego z pól formularza pochodząć będą z atrybutów powiązanych z formularzem obiektu.

Jak myślisz, dlaczego może to być problem?

Problem polega na tym, że nowy obiekt ogłoszenia *musi istnieć przed* wygenerowaniem formularza. Oczywiście dopóki użytkownik nie wypełni szczegółów ogłoszenia, obiekt nie zostanie zapisany w bazie danych — ale mimo to musi on zostać utworzony *przed* wywołaniem szablonu strony.



WYSIL SZARE KOMÓRKI

W którym miejscu aplikacji utworzyłbyś nowy obiekt ogłoszenia? Jeśli będzie to szablon albo metoda, jaką nosi nazwę?

Obiekt ogłoszenia formularza zostanie utworzony w akcji new kontrolera

- 1 Obiekt ogłoszenia formularza musi zostać utworzony przed wykonaniem szablonu new.html.erb.**

Obiekt utworzony zostaje w metodzie kontrolera o nazwie new, wykonanej przed wywołaniem szablonu new.html.erb.



Przeglądarka żąda
http://localhost:3000/
ads/new

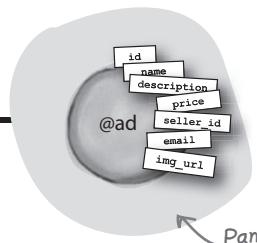
Wywołana zostaje metoda „new”
w kontrolerze „ads”.

```
def new  
...  
end
```

Jeśli w kontrolerze utworzysz metodę „new”, zostanie ona wywołana przez Rails przed wywołaniem szablonu new.html.erb.

- 2 Jak zatem tworzy się nowy obiekt ogłoszenia?**

Metoda Ad.new zwraca nowy obiekt, przypisywany następnie do zmiennej @ad. Nowy obiekt nie zostanie automatycznie zapisany do bazy danych, jednak potrzebny jest nam wyłącznie w pamięci, gdzie można go wykorzystać do wygenerowania formularza HTML.



Metoda „new” tworzy nowy obiekt.

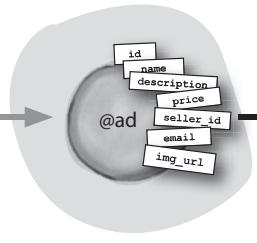
Musisz dodać ten kod do kontrolera.
def new

```
@ad = Ad.new
```

end

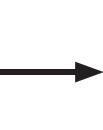
Wszystkie atrybuty nowego obiektu będą miały wartość domyślną „nil”.

- 3 Kiedy obiekt zostaje przypisany do zmiennej @ad w pamięci, szablon new.html.erb będzie mógł go użyć do wygenerowania formularza HTML wewnętrz strony internetowej /ads/new.**



Metoda „new” tworzy nowy obiekt.

Plik new.html.erb



Brak błędów

Szablon new.html.erb wykorzystuje obiekt do wygenerowania strony internetowej.

Każdy szablon strony ma teraz odpowiadającą mu metodę kontrolera

Kontroler ads ma teraz po jednej metodzie dla każdego z plików szablonów stron. Rails zawsze wywoła metodę kontrolera przed wygenerowaniem strony z szablonu.

Na akcję składa się **kombinacja** metody kontrolera oraz szablonu strony. Właśnie dlatego nazwa akcji pojawia się w nazwie metody kontrolera *oraz* w nazwie pliku szablonu strony.

Akcja to metoda kontrolera oraz szablon strony.

Zrób tak!

```
class AdsController < ApplicationController
  def new
    @ad = Ad.new
  end

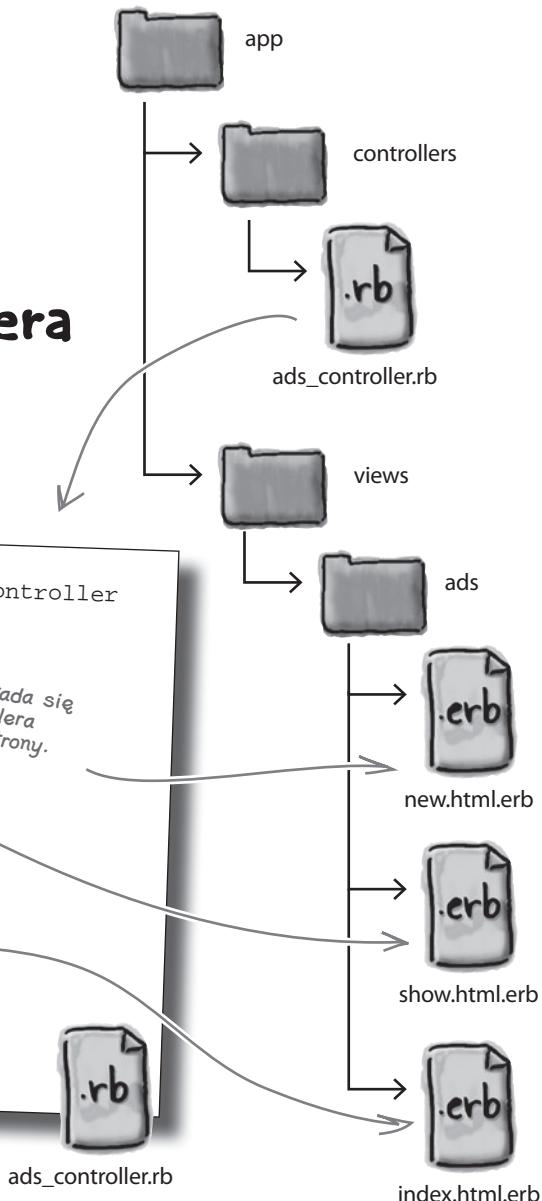
  def show
    @ad = Ad.find(params[:id])
  end

  def index
    @ads = Ad.find(:all)
  end
end
```

Akcja „new” składa się z metody kontrolera oraz szablonu strony.

Tak powinien teraz wyglądać Twój skrypt
ads_controller.rb.

Teraz, kiedy masz już kontroler tworzący nowy obiekt ogłoszenia przed wykonaniem szablonu new.html.erb, czas sprawdzić, czy kod działa.





Jazda próbna

Skoro kod kontrolera jest gotowy, czas ponownie przetestować aplikację, otwierając w przeglądarce stronę:

`http://localhost:3000/ads/new`

Teraz wprowadź jakieś dane:

New ad

Name
Leather boot

Description
Suit person with left foot

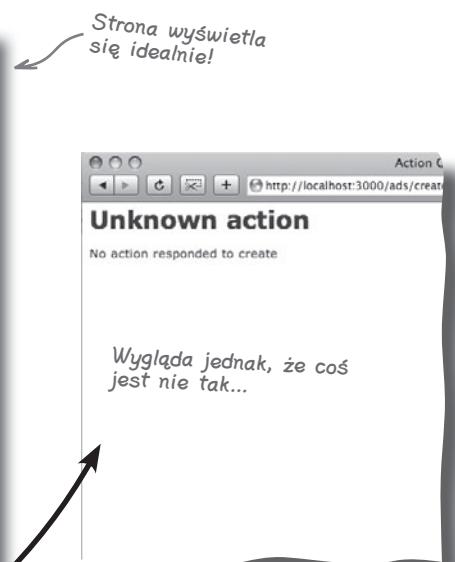
Price
1.0

Seller
242

Email
bert@hotmail.com

Img url
/www.javaranch.com/images/boot.gif

Create



Strona formularza wyświetlana jest tak, jak powinna, jednak gdy wprowadzamy dane i przesyłamy formularz, Rails zwraca błąd informujący nas, że nie utworzyliśmy jeszcze akcji `create`. Akcji `create?` To właśnie ta akcja miała otrzymać ogłoszenie z formularza.

Co akcja ta ma robić? Powinna odczytać dane z formularza i wykorzystać je do utworzenia nowego ogłoszenia.

Co jednak formularz odsyła do aplikacji?

Formularz nie odsyła obiektu, odsyła DANE

Formularz został wygenerowany za pomocą obiektu Ad. Ale co tak naprawdę odsyłane jest do serwera, kiedy zostanie przesłany formularz?

Ponieważ wykorzystywany jest protokół HTTP, nie można za pośrednictwem sieci przesyłać obiektu formularza. Zamiast tego odsyłane są dane.

Jak jednak SFORMATOWANE są te dane?

Pomyśl raz jeszcze o sposobie działania systemu routingu. Kiedy nadchodzi żądanie, Rails przesyła jego szczegóły do systemu routingu, który wstawia wartości do struktury danych nazywanej params [. . .], z wartościami dla akcji oraz kontrolera.

Struktura danych params [. . .] nie powstała wyłącznie z myślą o routingu. Może także być wykorzystywana do przechowywania wszelkich danych przesyłanych do aplikacji przez formularz internetowy.

Pola formularza zapisywane są w tabeli params [. . .] wraz z nazwą :ad. Następnie wartość zmiennej :ad staje się tak naprawdę kolejną tabelą wartości, odwzorowującą nazwy pól na ich wartości:

Tablica asocjacyjna „params”.

Nazwa	Wartość														
:controller	'ads'														
:action	'create'														
:ad	<table border="1"> <thead> <tr> <th>Nazwa</th><th>Wartość</th></tr> </thead> <tbody> <tr> <td>:name</td><td>Leather boot</td></tr> <tr> <td>:description</td><td>Suit person with left foot</td></tr> <tr> <td>:price</td><td>1.0</td></tr> <tr> <td>:seller_id</td><td>242</td></tr> <tr> <td>:email</td><td>bert@hotmail.com</td></tr> <tr> <td>:img_url</td><td>http://www.javaranch.com/images/boot.gif</td></tr> </tbody> </table>	Nazwa	Wartość	:name	Leather boot	:description	Suit person with left foot	:price	1.0	:seller_id	242	:email	bert@hotmail.com	:img_url	http://www.javaranch.com/images/boot.gif
Nazwa	Wartość														
:name	Leather boot														
:description	Suit person with left foot														
:price	1.0														
:seller_id	242														
:email	bert@hotmail.com														
:img_url	http://www.javaranch.com/images/boot.gif														

Wartość zmiennej „:ad” to tabela „params[:ad]”.

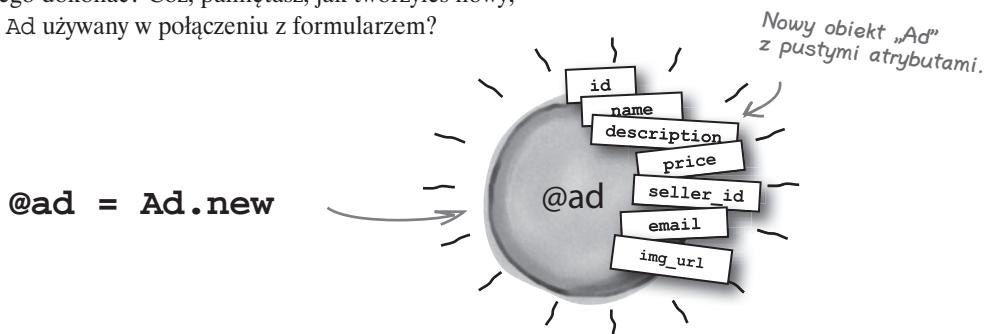
Co jednak dzieje się z tymi danymi, kiedy otrzyma je kontroler? I w jaki sposób możemy tak naprawdę UŻYĆ tych danych?

Rails musi przekształcić dane na obiekt przed ich zapisaniem

Rails w komunikacji z bazą danych używa tylko obiektów, dlatego zanim ogłoszenie zostanie zapisane w bazie danych, konieczne jest znalezienie jakieś metody konwersji danych na obiekt Ad.

Model może tworzyć obiekty z surowych danych

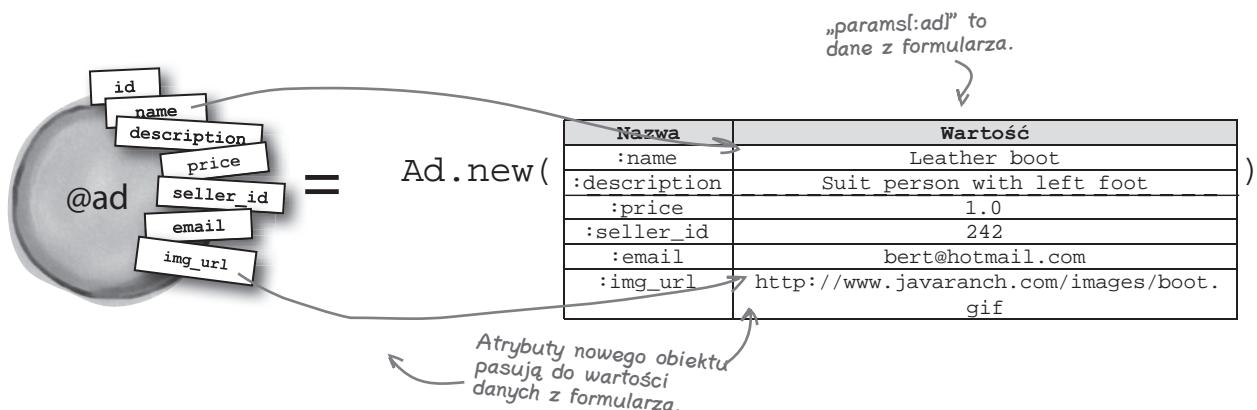
Jak można tego dokonać? Cóż, pamiętasz, jak tworzyłeś nowy, pusty obiekt Ad używany w połączeniu z formularzem?



Metodę `Ad.new` można również wywołać ze zbiorem wartości z tablicy asocjacyjnej, które będą wykorzystane do zainicjalizowania atrybutów nowego obiektu Ad.

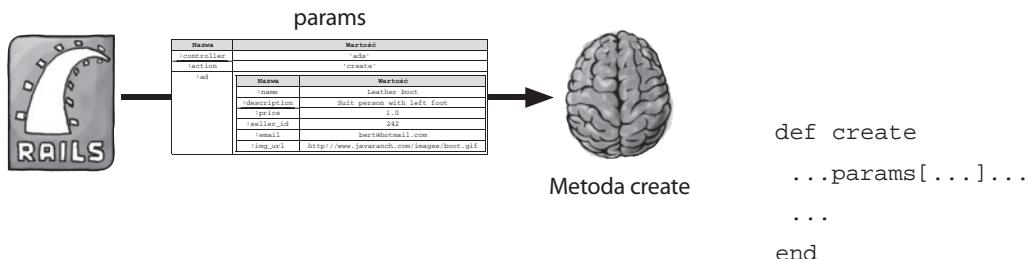
A tak się składa, że dane formularza znajdują się akurat w obiekcie tablicy asocjacyjnej:

```
@ad = Ad.new(params[:ad])
```

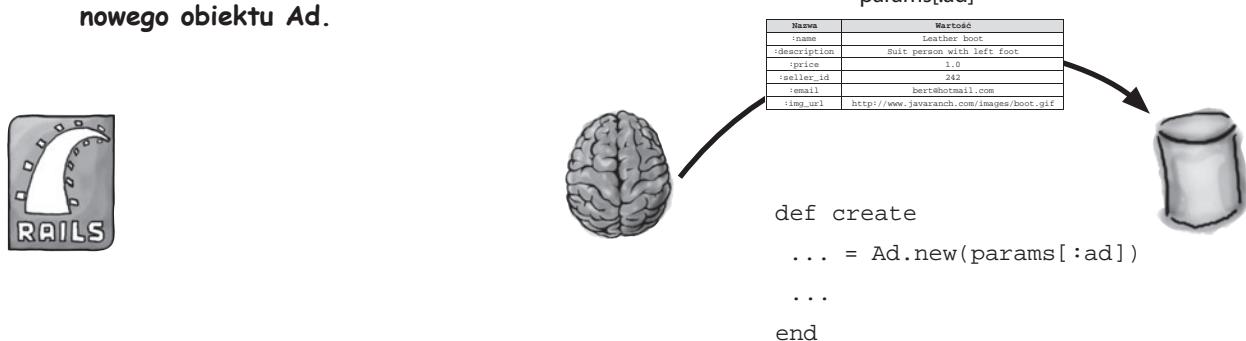


Metoda create kontrolera krok po kroku

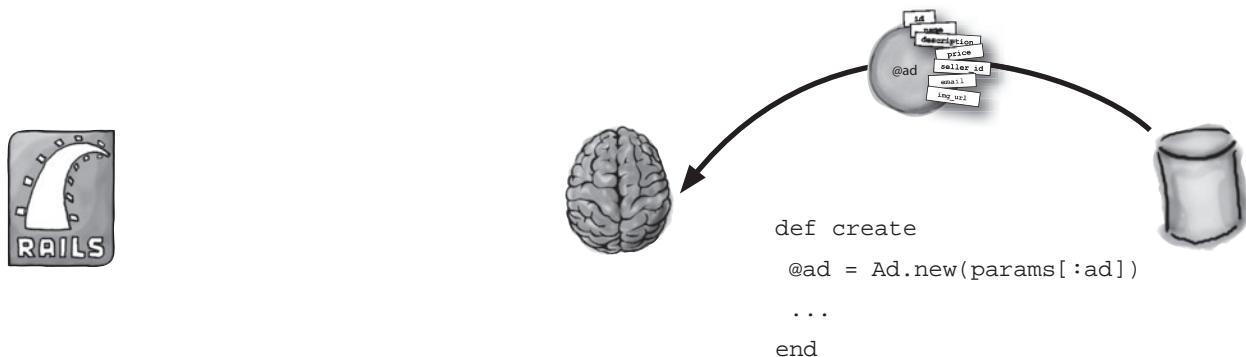
- 1** Rails przesyła dane z formularza do kontrolera za pomocą tablicy asocjacyjnej params[...].



- 2** Kontroler może odczytać surowe dane formularza, zaglądając do params[:ad]. Może następnie przespać tę wartość do metody Ad.new(...) w celu skonstruowania nowego obiektu Ad.



- 3** Obiekt Ad zwracany przez metodę Ad.new(...) ma atrybuty odpowiadające wartościom pól formularza.



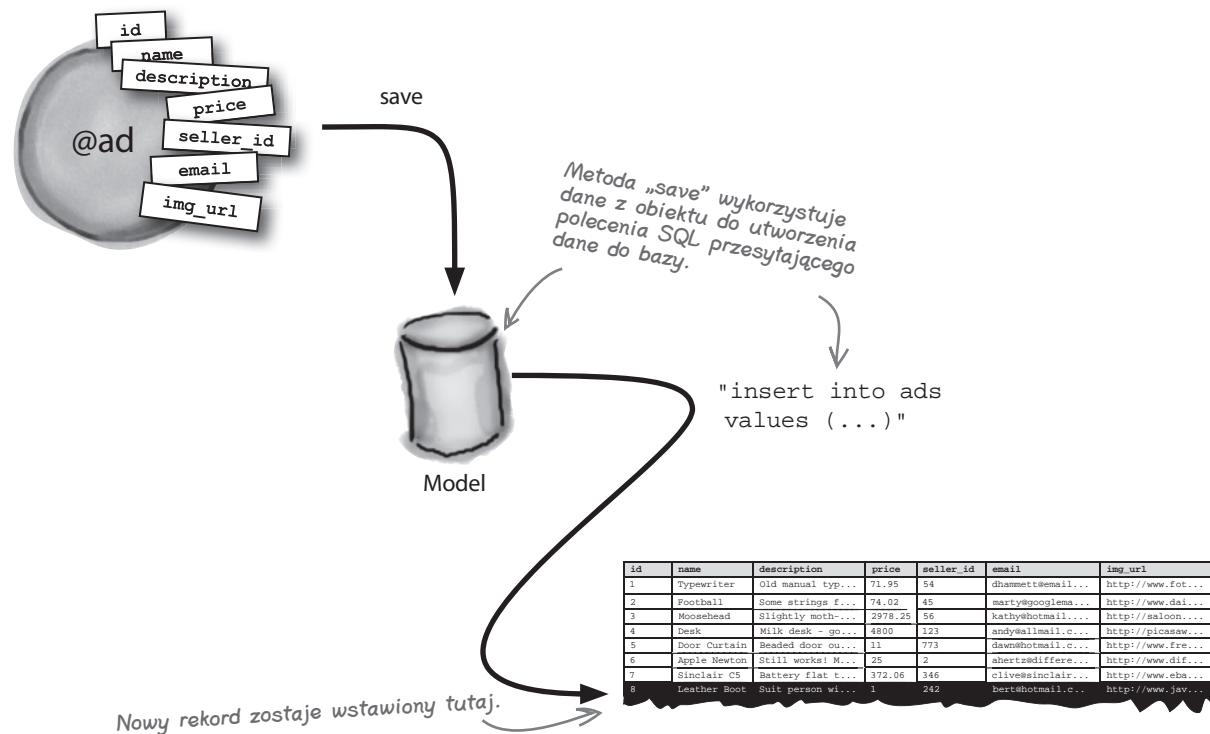
Kontroler musi zapisać rekord

Przyczyną konwersji danych formularza na obiekt była możliwość ich zapisania.

Jak można tego dokonać? Za pomocą

```
@ad.save
```

Kiedy na obiekcie modelu wywołujemy metodę `save`, Rails bada atrybuty i generuje instrukcję insert języka SQL, która aktualnia bazę danych:



Po wstawieniu metody `save`, metoda `create` kontrolera jest gotowa:

```
def create
  @ad=Ad.new(params[:ad])
  @ad.save
end
```



Jazda próbna

Uaktualnij metodę `create` kontrolera i przetestuj zmodyfikowany formularz tworzący ogłoszenie. Po kliknięciu przycisku *Create* zwracany jest komunikat o błędzie informujący o brakującym szablonie:

The screenshot shows a web browser window with the following details:

- Address bar: http://localhost:3000/ads/create
- Title bar: Action Controller: Exception caught
- Content area:
 - Template is missing**
 - Missing template ads/create.html.erb in view path /Users/davidg/mebay/app/views

Zaostrz ołówek



1. Czy utworzony rekord został zapisany?

.....

2. Co należy zrobić w celu naprawienia nowego błędu?

.....

.....

.....

Każde żądanie zasługuje na odpowiedź

Zaostrz ołówek



Rozwiążanie

1. Czy utworzony rekord został zapisany?

Dane zostały zapisane, zatem nowe ogłoszenie zostało utworzone.

2. Czy utworzony rekord został zapisany?

Niedostępny był szablon generujący odpowiedź potwierdzającą

fakt zapisania rekordu — konieczne jest zatem utworzenie

szablonu strony `create.html.erb`.

Rails narzeka, ponieważ nie ma możliwości wygenerowania ODPOWIEDZI na Twoje żądanie

Protokół HTTP działa w oparciu o *parę* żądań i odpowiedzi.

Dla każdego żądania musi istnieć odpowiedź.

Kiedy metoda `create` kontrolera zakończyła działanie, nowy rekord został utworzony. Platforma Rails powinna następnie wygenerować stronę odpowiedzi, dlatego szukała szablonu pasującego do bieżącej akcji. Bieżącą akcją było `create`, dlatego szukała pliku `create.html.erb`. Ale taki szablon nie istnieje!



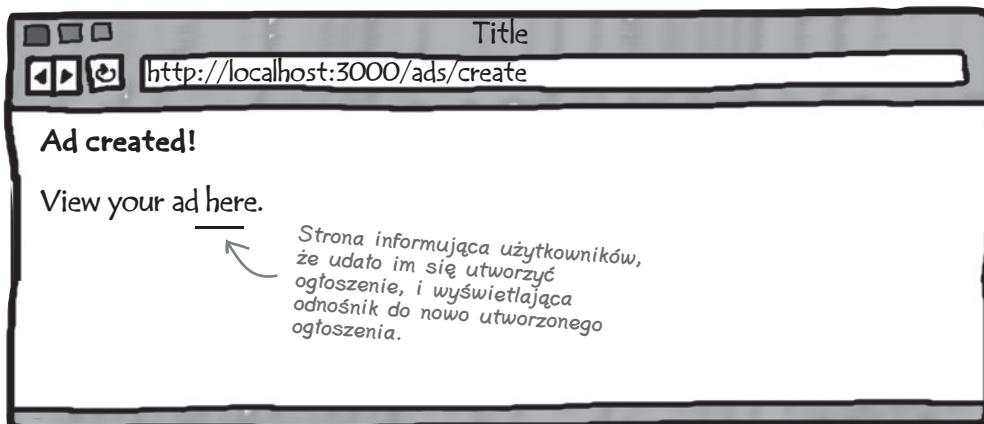
Musimy zatem napisać szablon strony `create.html.erb`.

Co jednak powinien zawierać ten szablon?



Ćwiczenie

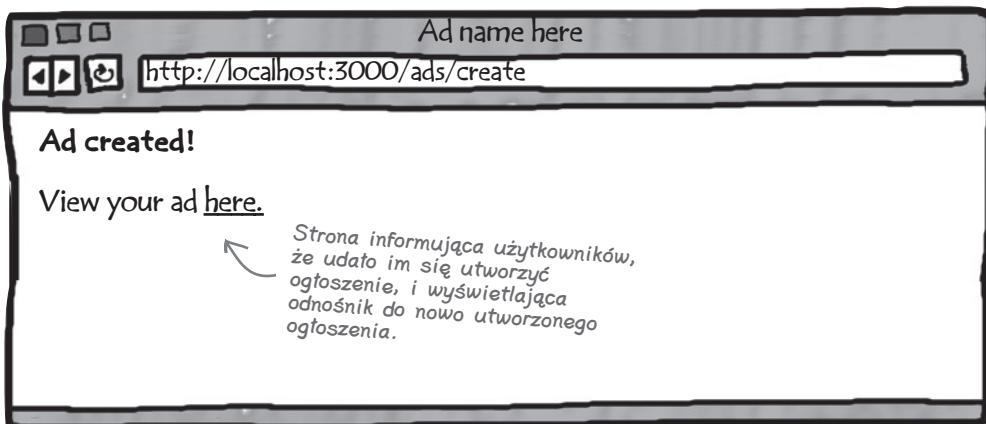
Utwórz plik `create.html.erb` informujący użytkowników, że rekord został utworzony, i udostępniający odnośnik do tego nowego rekordu.





Ćwiczenie
Rozwiązanie

Utwórz plik `create.html.erb` informujący użytkowników, że rekord został utworzony, i udostępniający odnośnik do tego nowego rekordu.



Twój kod HTML może wyglądać nieco inaczej od tego. To dobrze, o ile zawiera wyrażenia podające tytuł strony oraz odnośnik do nowego ogłoszenia.

`<h1>Ad created!</h1>`

`View your ad <a href="/ads/<%= @ad.id %>">here`

Kod jest boleśnie prosty, ponieważ większość formatowania i pozostatego kodu mieści się w naszym szablonie nadziednym.

Zmienna „@ad” kieruje do nowo wstawionego obiektu.

Ścieżka do nowo utworzonego ogłoszenia.



Jazda próbna

Teraz, gdy utworzyłeś już szablon strony `create.html.erb`, czas wypróbować aplikację.

Wprowadź jakieś dane, naciśnij przycisk „Create” i otrzymasz...

Ad created!

... stronę z potwierdzeniem wygenerowaną przez szablon `create.html.erb`.

Name: Leather boot
Description: Suit person with left foot
Price: 1.0
Seller Id: 242
Email: bert@hotmail.com

Strona z potwierdzeniem zawiera odnośnik do strony z nowo utworzonym ogłoszeniem.

Wkrótce widać, że użytkownicy dodali dziesiątki nowych ogłoszeń. Choć witryna jest naprawdę popularna, niektóre osoby mają jedno drobne zastrzeżenie...



Po co witryna pokazuje mi stronę z potwierdzeniem i odnośnikiem do mojego nowego ogłoszenia? Czemu nie przechodzi od razu do ogłoszenia?

WYSIL SZARE KOMÓRKI

Czy wyświetlenie odnośnika do nowego ogłoszenia na stronie `create` jest jakimś problemem? Jeśli tak — dlaczego?

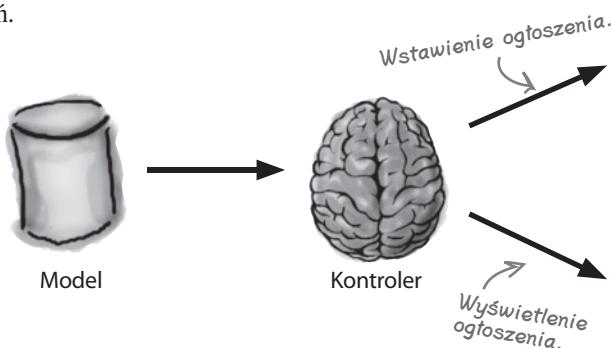
Ponowne wykorzystanie istniejących szablonów

Nie twórz nowej strony, użyj istniejącej

Użytkownicy nie chcą widzieć pośredniej strony z potwierdzeniem wygenerowanej przez szablon `create.html.erb`. Chcą po prostu od razu zobaczyć swoje ogłoszenie. Co zatem robimy?

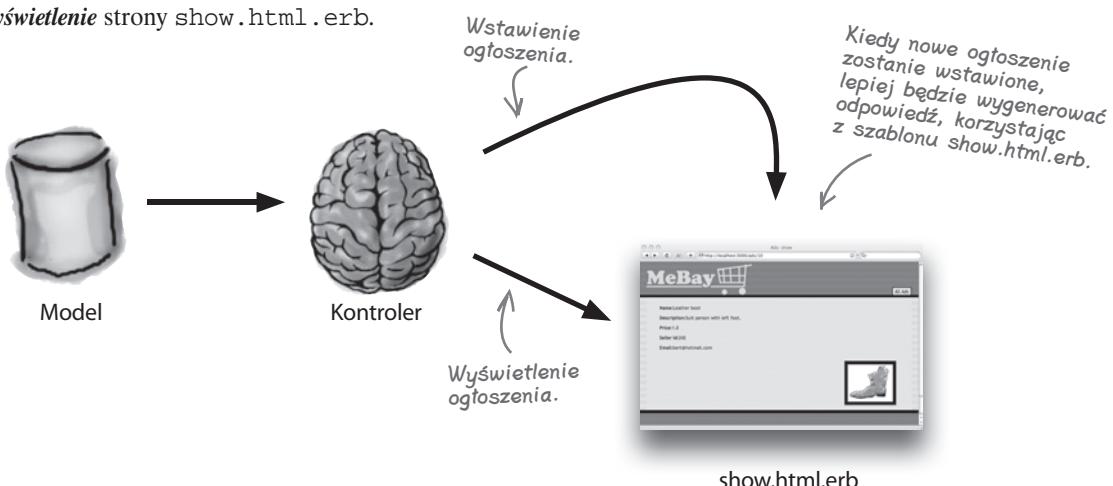
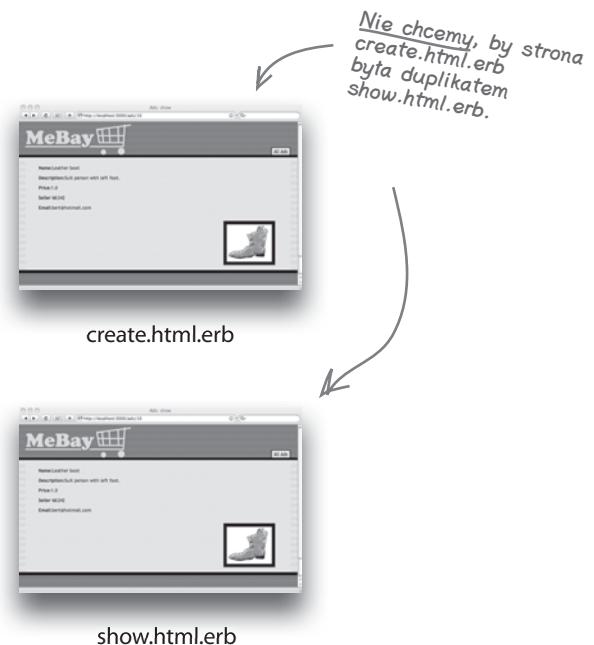
Moglibyśmy zmodyfikować szablon strony `create.html.erb` w taki sposób, by wyświetlał wszystkie szczegóły nowego ogłoszenia, prawda? W końcu zmienna `@ad` jest widoczna z szablonu strony i zawiera wszystkie szczegóły nowego ogłoszenia.

Byłoby to jednak złym pomysłem. Dlaczego? Ponieważ oznaczałoby to, że strona `create.html.erb` byłaby dokładnie taka sama jak szablon strony `show.html.erb` wykorzystywany do wyświetlania każdego z ogłoszeń.



Pamiętaj o zasadzie DRY:
„Nie powtarzaj się”

Byłoby to **duplikacja**, co oznaczałoby również konieczność utrzymywania większej ilości kodu w przyszłości. O wiele lepiej byłoby, gdyby akcja `create` w kontrolerze mogła wybrać **wyświetlenie** strony `show.html.erb`.



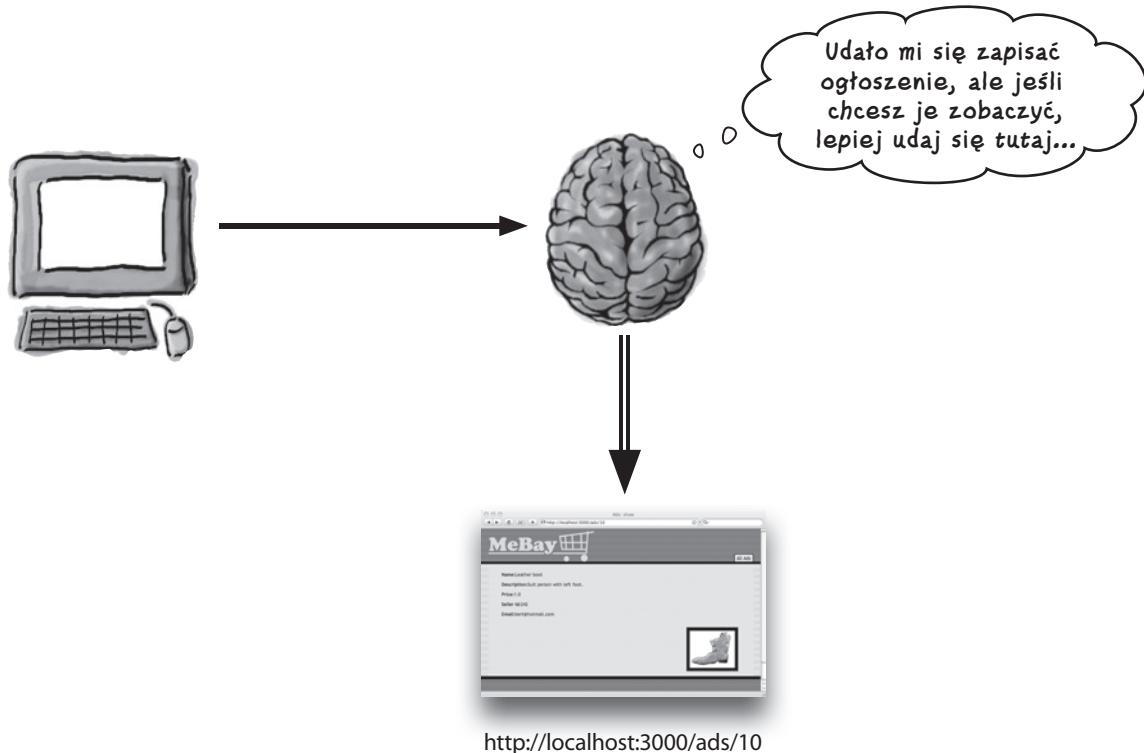
Jak jednak akcja kontrolera może wyświetlać stronę INNEJ akcji?

Metoda kontrolera współdziała z szablonem, by razem utworzyć akcję. We wszystkich pokazanych dotychczas przykładach metoda kontrolera oraz szablon strony były wykorzystywane wyłącznie w jednej akcji.

Z tego powodu metody kontrolera oraz szablony stron zawierały w sobie nazwę akcji. Kiedy wykonywaliśmy akcję show, używaliśmy metody show w kontrolerze ads oraz szablonu strony `show.html.erb`.

Nadal w celu wykonania akcji chcemy użyć metody kontrolera oraz szablonu strony, jednak wolelibyśmy *wybrać*, który szablon strony zostanie wywołany z kontrolerem.

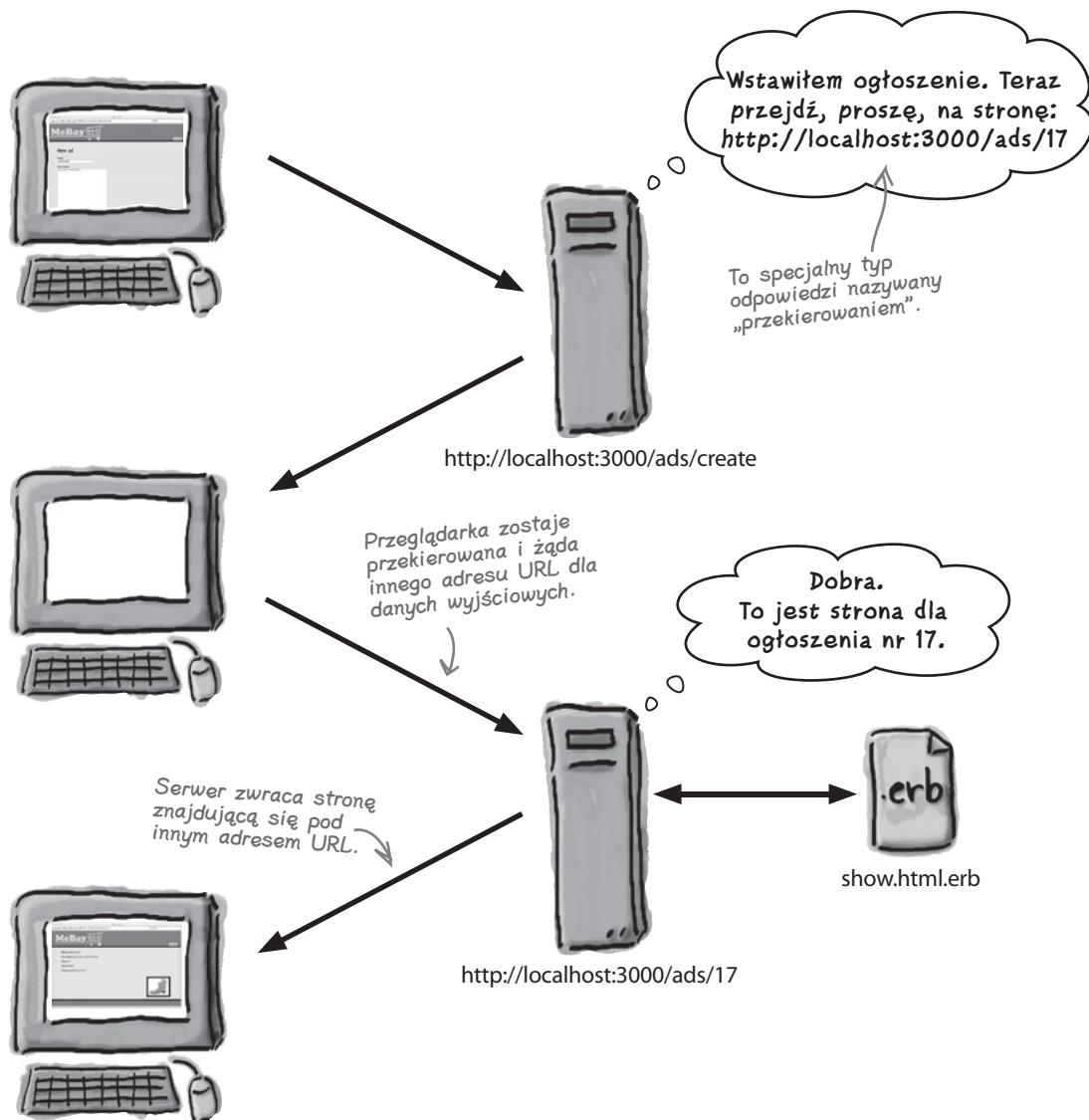
Akcja kontrolera to metoda kontrolera oraz szablon strony.



Potrzebny nam sposób, dzięki któremu kontroler poinformuje nas, że dane wyjściowe znajdują się pod innym adresem URL.

Przekierowania pozwalają kontrolerowi określić, który widok zostanie wyświetlony

Przekierowanie to specjalny rodzaj odpowiedzi kierowanej przez aplikację Rails do przeglądarki. Przekierowanie informuje przeglądarkę, że po dane wyjściowe ma się udać pod *inną* adres URL. Zatem nawet jeśli przeglądarka przesyłała dane do /ads/create, przekierowanie odsyła przeglądarkę /ads/17 (przykładowo — jeśli 17 to identyfikator nowego ogłoszenia).



Zaostrz ołówek



Polecenie `przekierowania (redirect_to)` przekierowuje przeglądarkę do adresu URL nowego ogłoszenia. Uzupełnij adres URL:

```
def create
  @ad = Ad.new(params[ :ad ])
  @ad.save
  redirect_to "/_____ /#{.....}"
end
```

Nie istniejąca grupa pytań

P: Piszcie, że przekierowanie jest specjalnym typem odpowiedzi. Czy tak naprawdę jest to odpowiedź?

O: Tak. Istnieje kilka typów odpowiedzi. Zazwyczaj odpowiedź zawiera informacje, które przeglądarka ma wyświetlić, jednak inne typy odpowiedzi — takie jak przekierowania — zawierają specjalne instrukcje dla przeglądarki. Przekierowanie to specjalna instrukcja informująca przeglądarkę, że ma przejść pod inny adres URL.

P: Jeśli zatem przeglądarka zostanie przekierowana, czy zmienia się adres w jej pasku adresu?

O: Dokładnie tak. Nawet jeśli przeglądarka zażądała określonego adresu URL, pasek adresu zostanie uaktualniony, tak by pokazać adres, na który nastąpiło przekierowanie.

P: Czy jeśli popełnię błąd w kodzie, mogę przekierować przeglądarkę w jakiś rodzaj nieskończonej pętli przekierowań?

O: Nie, ponieważ przeglądarki mają limit przekierowań, jakie wykonają. Jeśli Twój kod wysłał powtarzające się przekierowania, przeglądarka wkrótce się znudzi, przestanie nimi podążać i wyświetli komunikat o błędzie.

P: Czy jeśli ustanowię zmienną `@ad` na obiekt, a następnie przekieruję przeglądarkę do innego adresu URL, to ten nowy URL będzie widział mój obiekt `@ad`?

O: Dobre pytanie! Nie, nie będzie. Po przekierowaniu do innego adresu żadne zmienne przypisane przez kontroler nie będą widoczne pod nowym adresem.

P: Czy mogę przekierować przeglądarkę do adresu spoza mojej aplikacji?

O: Jasne. Przekierowanie to po prostu polecenie przeglądarce, by udało się pod określony adres URL. Jeśli chcesz przekierować przeglądarkę do jakieś witryny zewnętrznej, możesz to zrobić.

P: W jakich okolicznościach wykonuje się przekierowania?

O: Przekierowanie można wykonać w celu ponownego wykorzystania strony wyświetlającej informacje. Z tego powodu używamy przekierowania w aplikacji MeBay. Dobrym pomysłem jest również użycie przekierowania po wprowadzeniu uaktualnienia do bazy danych.

P: Dlaczego?

O: Dobrym pomysłem jest podzielenie akcji na dwie kategorie: akcje uaktualniające oraz akcje wyświetlające. Akcja uaktualniająca zmienia to, co znajduje się w bazie danych, a następnie przekierowuje do akcji wyświetlającej. W ten sposób, kiedy ktoś wprowadzi nowy rekord, a następnie w przeglądarce kliknie *Odśwież* na kolejnej stronie, odświeży tylko stronę wyświetlającą, a nie wprowadzi rekord raz jeszcze.

P: Co w łańcuchu przekierowania oznaczają znaki `#{} ?`

O: łańcuchy znaków języka Ruby mogą zawierać wyrażenia tego języka — takie jak nazwy Ruby. Dzięki umieszczeniu ich pomiędzy znakami `#{}` nakazujesz językowi Ruby zastąpienie wyrażenia jego wartością.

P: `params[...]` wygląda trochę jak tablica. Czy nią jest?

O: `params` zaprojektowano w ten sposób, by działała jak „tablica asocjacyjna” (ang. *associative array, hash*). Tablica asocjacyjna to specjalny typ tablicy, która może być indeksowana za pomocą elementów innych od liczb.

Utwórz, odczytaj, a teraz uaktualnij

Zaostrz ołówek



Rozwiążanie

Polecenie przekierowania (`redirect_to`) przekierowuje przeglądarkę do adresu URL nowego ogłoszenia. Uzupełnij adres URL:

```
def create
  @ad = Ad.new(params[:ad])
  @ad.save
  redirect_to "/ads /#{...@ad.id...}"
end
```

Symbol "#" oraz nawiasy "{}" wstawiają wartość zmiennej do łańcucha znaków.



Jazda próbna

Czas uaktualnić kod kontrolera. Później należy sprawdzić, czy nasze przekierowanie odsyła użytkowników do ich nowo utworzonych ogłoszeń:

Adres w pasku przeglądarki zmienia się na „/ads/40”, pomimo że formularz został przestany do „ads/create”.

New ad

Name: Leather boot

Description: Suit person with left foot

Price: 1.0

Seller: 242

Email: bert@hotmail.com

Img url: http://www.javaranch.com/images/boot.gif

Create

Ads: show

Name: Leather boot

Description: Suit person with left foot

Price: 1.0

Seller Id: 242

Email: bert@hotmail.com

Kiedy nowe ogłoszenie zostaje utworzone, przeglądarka automatycznie przechodzi do nowej strony.

Ale co się dzieje, kiedy ogłoszenie należy po opublikowaniu poprawić?

Niektórzy użytkownicy popełniają błędy przy tworzeniu swoich ogłoszeń i mogą chcieć wprowadzić do nich zmiany. Teraz chcą zatem czegoś więcej niż formularze wyświetlające i tworzące. Użytkownicy chcą mieć możliwość **edycji** swoich ogłoszeń.

The screenshot shows a web browser window with the title "Ads: show" and the URL "http://localhost:3000/ads/40". The page is titled "MeBay" with a shopping cart icon. On the right, there is a link "All Ads". The main content area displays a listing for a "Leather boot". The listing includes:

- Name: Leather boot
- Description: ~~Suit person with left foot~~ High quality kid-leather footwear. Suitable for person with suitable appendage
- Price: ~~10~~ 35.75
- Seller Id: 242
- Email: bert@hotmail.com

Annotations on the page include:

- An arrow points from the word "Description" to the text "High quality kid-leather footwear. Suitable for person with suitable appendage".
- An annotation text states: "Użytkownicy serwisu MeBay muszą mieć jakąś możliwość zmiany danych w ogłoszeniach."
- A small image of a leather boot is shown in a frame.

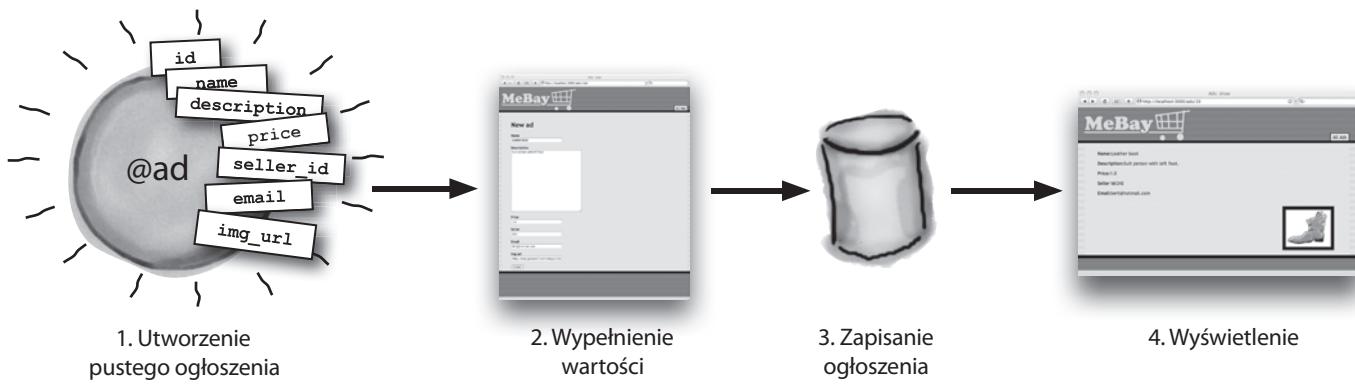
Oznacza to, że system musi pozwalać nie tylko na wstawianie danych, ale i uaktualnianie.
Czy będzie to trudne do zrobienia?

Uaktualnienie ogłoszenia przypomina utworzenie go... tylko jest trochę inne

Choć obecny system nie pozwala na edycję ogłoszeń, czy dodanie takiej możliwości wymagać będzie dużo pracy?

Zastanów się przez chwilę nad sekwencją czynności wykonywaną przez system w celu wstawienia ogłoszenia:

- 1 Nowy, pusty obiekt ogłoszenia zostaje utworzony i wykorzystany jest do wygenerowania formularza służącego do dodania ogłoszenia.
- 2 Formularz przesyłany jest do użytkownika, który uaktualnia wartości pól i odsyła go do aplikacji.
- 3 Pola z danymi przekształcane są z powrotem na obiekt Ad, który zapisywany jest w bazie danych.
- 4 Użytkownikowi przesyłana jest strona wyświetlająca nowe ogłoszenie.

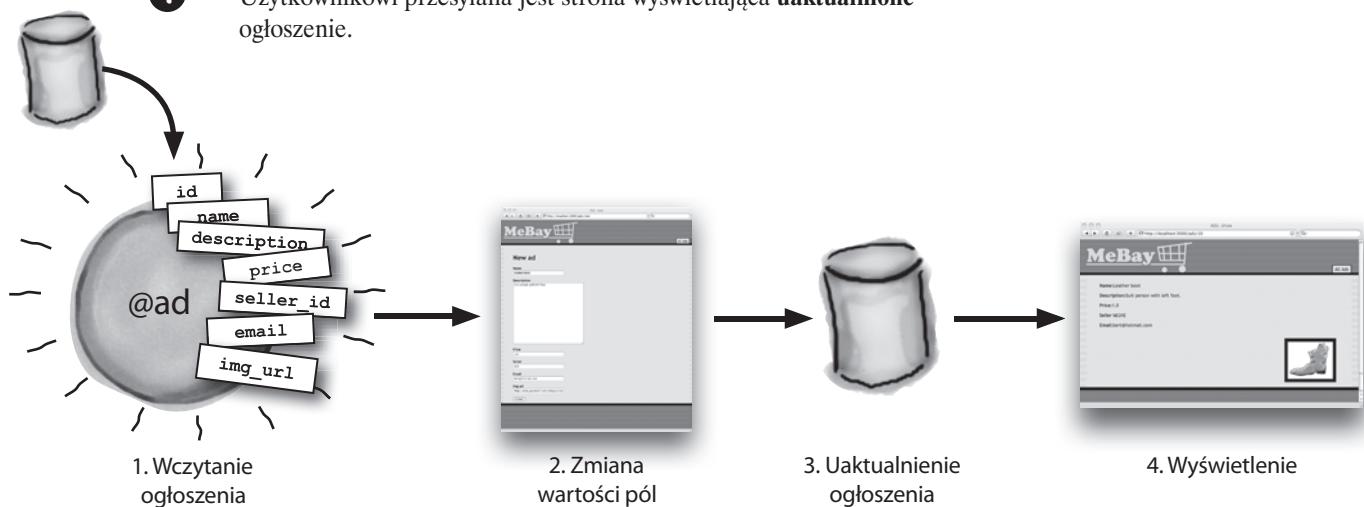


Załóżmy, że teraz chcesz zmienić stronę. Co spodziewałbyś się zobaczyć? Może formularz ze szczegółami ogłoszenia, który można przesłać ponownie, zapisując zmiany — co w zasadzie jest tą samą sekwencją czynności, jakiej użyłeś do utworzenia ogłoszenia? Przyjrzyjmy się bliżej sekwencji zmiany...

Zamiast tworzyć ogłoszenie, musimy je odnaleźć; zamiast je zapisać, musimy je uaktualnić

Kiedy ktoś edytuje ogłoszenie, wykorzystuje taki sam formularz jak wcześniej. Użytkownik zmienia szczegóły, a dane ogłoszenia zostają zapisane. Jak bardzo podobne są do siebie sekwencja zmiany i sekwencja tworzenia ogłoszenia?

- 1 Istniejące ogłoszenie **wczytywane** jest z bazy danych i zostanie wykorzystane do wygenerowania formularza **zmiany**.
- 2 Formularz przesyłany jest do użytkownika, który uaktualnia wartości pól i odsyła go do aplikacji.
- 3 Pola z danymi przekształcane są z powrotem na obiekt Ad, który zostaje wykorzystany do **uaktualnienia** bazy danych.
- 4 Użytkownikowi przesyłana jest strona wyświetlająca **uaktualnione** ogłoszenie.



Jak widać, sekwencje tych dwóch operacji prawie się od siebie nie różnią. W sekwencji **tworzenia nowy** obiekt ogłoszenia zostaje **utworzony** i **zapisany** w bazie danych. W sekwencji **zmiany istniejące** ogłoszenie zostaje **wczytane**, **uaktualnione** oraz **zapisane** w bazie danych.

Musimy się upewnić, że weźmiemy pod uwagę różnice pomiędzy tymi dwoma operacjami. Konieczne będzie zachowanie w sekwencji zmiany elementów takich, jak numer identyfikatora.

Czy sądzisz, że umiałbyś dodać opcję edytowania do obecnej wersji aplikacji?



Ćwiczenie (nieco dłuższe)

Dodaj na każdej stronie `show` odnośnik do edycji (na przykład `/ads/17/edit`) wywołujący nową akcję `edit`, która ma wyszukać ogłoszenie i wyświetlić je w formularzu.

```
<p>
  <b>Name:</b><%= @ad.name %>
</p>

<p>
  <b>Description:</b><%= @ad.description %>
</p>

<p>
  <b>Price:</b><%= @ad.price %>
</p>

<p>
  <b>Seller Id:</b><%= @ad.seller_id %>
</p>

<p>
  <b>Email:</b><%= @ad.email %>
</p>

<p>
  
</p>
```

Tutaj dodaj odnośnik
do akcji „edit”.

Wypisz tutaj
obie trasy.

Formularz prześle dane do akcji o nazwie `update` znajdującej się pod adresem URL takim, jak `/ads/17/update`. Utwórz trasy łączące `/ads/17/edit` ze stroną edycji, a `/ads/17/update` z akcją `update`, upewniając się, że w każdym przypadku numer identyfikatora zostanie przechowany w parametrze żądania o nazwie `:id`.

Teraz utwórz nowy szablon strony `edit.html.erb`, który pozwoli użytkownikowi zmodyfikować szczegóły ogłoszenia. Będzie on podobny do szablonu `new.html.erb`, jednak powinien wyświetlać nazwę ogłoszenia w nagłówku strony i będzie korzystał z akcji `update` zamiast `create`.

Napisz tutaj kod szablonu
`edit.html.erb`.



W kontrolerze `ads` będziesz potrzebował dwóch metod akcji, które udostępniają dane formularzowi edycji, a także uaktualniają ogłoszenie w bazie danych. Metoda `edit` udostępnii dane formularzowi edycji, natomiast metoda `update` zajmie się uaktualnieniem bazy danych.
Biorąc pod uwagę, że:

`@ad.update_attributes(_____)`

Wpisz tutaj strukturę danych tablicy asocjacyjnej zawierającej wartości pól przestane z formularza.

uaktualnia obiekt `@ad` w bazie danych, napisz poniżej kod metod `edit` oraz `update` kontrolera `ads`:

Napisz tutaj kod metody „edit”.



.....
.....
.....
.....
.....
.....

Tutaj napisz kod metody „update”.



Czasy się zmieniają...



Ćwiczenie
(nieco dłuższe)

Rozwiążanie

Twoje zadanie polegało na dodaniu na każdej stronie *show* odnośnika do edycji (na przykład */ads/17/edit*) wywołującego nową akcję **edit**, która ma wyszukać ogłoszenie i wyświetlić je w formularzu.

```
<p>
  <b>Name:</b><%= @ad.name %>
</p>

<p>
  <b>Description:</b><%= @ad.description %>
</p>

<p>
  <b>Price:</b><%= @ad.price %>
</p>

<p>
  <b>Seller Id:</b><%= @ad.seller_id %>
</p>

<p>
  <b>Email:</b><%= @ad.email %>
</p>

<p>
  
</p>

<a href="/ads/<%= @ad.id %>/edit">Edit</a>
```

Tutaj dodaj odnośnik
do akcji „edit”.

Wpisz tutaj
obie trasy.

Formularz przesyła dane do akcji o nazwie *update* znajdującej się pod adresem URL takim, jak */ads/17/update*. Powinieneś był zatem utworzyć trasy łączące */ads/17/edit* ze stroną edycji, a */ads/17/update* z akcją *update*, upewniając się, że w każdym przypadku numer identyfikatora zostanie przechowany w parametrze żądania o nazwie :id.

```
map.connect '/ads/:id/edit', :controller=>'ads', :action=> 'edit'
```

```
map.connect '/ads/:id/update', :controller=>'ads', :action=> 'update'
```

Upewnij się, że wstawisz te trasy „nad” innymi trasami w pliku config/routes.rb.

Twoje zadanie polegało na utworzeniu nowego szablonu strony `edit.html.erb`, który pozwoli użytkownikowi zmodyfikować szczegóły ogłoszenia. Jest on podobny do szablonu `new.html.erb`, jednak wyświetla nazwę ogłoszenia w nagłówku strony i korzysta z akcji `update` zamiast `create`:

```
<h1>Editing <%= @ad.name %></h1>

<% form_for(@ad,:url=>{:action=>'update'}) do |f| %>

<p><b>Name</b><br /><%= f.text_field :name %></p>

<p><b>Description</b><br /><%= f.text_area :description %></p>

<p><b>Price</b><br /><%= f.text_field :price %></p>

<p><b>Seller</b><br /><%= f.text_field :seller_id %></p>

<p><b>Email</b><br /><%= f.text_field :email %></p>

<p><b>Img url</b><br /><%= f.text_field :img_url %></p>

<p><%= f.submit "Update" %></p>

<% end %>
```

Nie martw się, jeśli nie wszystko w tym ćwiczeniu udało Ci się zrobić poprawnie. To było naprawdę trudne zadanie. Przeczytaj jednak odpowiedzi i sprawdź, czy rozumiesz, co się dzieje.

W kontrolerze `ads` potrzebujesz dwóch metod akcji, które udostępniają dane formularzowi edycji, a także uaktualniają ogłoszenie w bazie danych. Metoda `edit` udostępnia dane formularzowi edycji, natomiast metoda `update` zajmuje się uaktualnieniem bazy danych. Biorąc zatem pod uwagę, że

```
@ad.update_attributes( params[:ad] )
```

uaktualnia obiekt `@ad` w bazie danych, napisałś poniższy kod metod `edit` oraz `update` kontrolera `ads`:

```
def edit
  @ad = Ad.find(params[:id])
end
```

```
def update
  @ad = Ad.find(params[:id])
  @ad.update_attributes(params[:ad])
  redirect_to "/ads/#{@ad.id}"
end
```



Jazda próbna

Szablon `edit.html.erb` jest gotowy, podobnie jak trasy oraz dodatkowe metody kontrolera `ads`. Czas przetestować nową możliwość edycji ogłoszenia:

Kiedy ogłoszenie zostanie wyświetlone, sprzedający może kliknąć odnośnik „Edit” i otworzyć stronę edycji wygenerowaną przez szablon `edit.html.erb`.

The diagram illustrates the process of editing an advertisement on the MeBay website, showing four main screens:

- Ads: index**: A list of all ads. One ad is highlighted with the text: "Sprzedający wybiera swoje ogłoszenie z listy." An arrow points from this screen to the "Edit" link in the next screen.
- Ads: show**: A detailed view of a specific ad for a "Leather boot". It shows fields for Name, Description, Price, Seller ID, Email, and an "Edit" link. To the right is a thumbnail image of a leather boot.
- Ads: edit**: A form for editing the "Leather boot" ad. Fields include Name (Leather boot), Description (High quality kid leather footwear. Suitable for person with requisite appendage.), Price (35.75), Seller (242), Email (bert@hotmail.com), and Img url (http://www.javaranch.com/images/boot). An "Update" button is at the bottom. An arrow points from this screen back to the "Edit" link in the "Ads: show" screen.
- Ads: show**: The final state after updating, showing the edited ad details and the updated thumbnail image of the leather boot.

Annotations in Polish provide additional context:

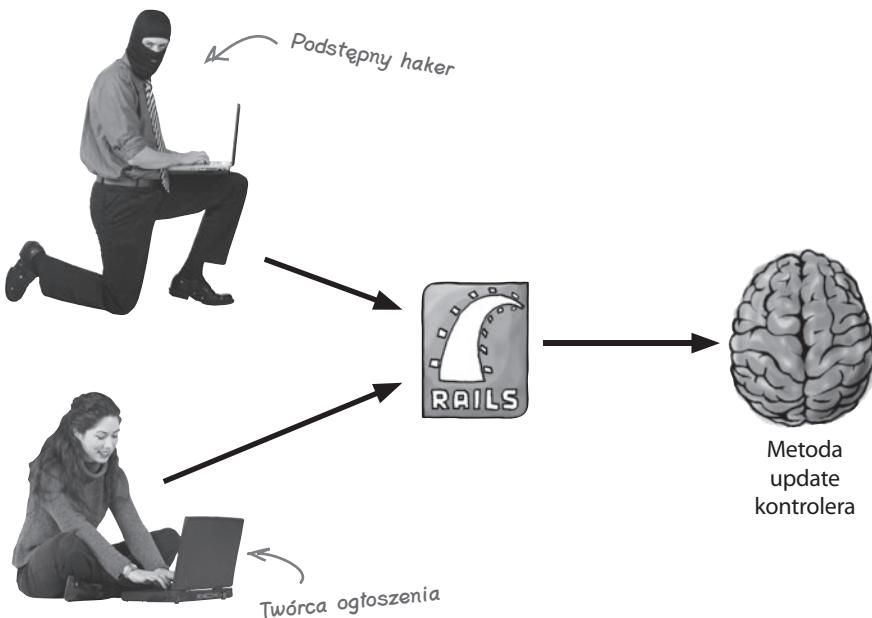
- "Po przeniesieniu na stronę edycji sprzedający może poprawić szczegóły ogłoszenia, a następnie kliknąć przycisk „Update”."
- "Ogłoszenie sprzedającego zostaje uaktualnione i wyświetlane."
- "Kod jest gotowy, a opcja edycji działa. Wszystko jest zatem w porządku, prawda?"



Ograniczanie dostępu do funkcji

Aplikacja może teraz tworzyć i aktualniać dane. Oznacza to jednak, że każdy, kto utworzył ogłoszenie, może zmodyfikować **wszystkie** ogłoszenia. A to jest już problematyczne.

Właściciele serwisu MeBay chcieli, aby każdy mógł tworzyć ogłoszenia, jednak nie chcieli, by *dowolna osoba* mogła modyfikować ogłoszenie po jego opublikowaniu.



Jednym ze sposobów zapobieżenia sytuacji, w której każdy może zmieniać ogłoszenia, jest zabezpieczenie funkcji `update` nazwą użytkownika oraz hasłem.

Pracownicy firmy MeBay zdecydowali, że jedynie *administratorzy systemu* będą mogli modyfikować ogłoszenia. Chcą zatem, by nowa opcja aktualniania była zabezpieczona **nazwą użytkownika** oraz **hasłem** administratora.

Na szczęście Rails bardzo ułatwia wprowadzanie zabezpieczeń. Skorzystamy ze specjalnego typu zabezpieczeń internetowych, zwanego **uwierzytelnieniem HTTP** (ang. *HTTP authentication*). To ten rodzaj zabezpieczenia, który wyświetla okno dialogowe i prosi o podanie **nazwy użytkownika** oraz **hasła**, kiedy ktoś próbuje wejść do zabezpieczonego obszaru strony internetowej.



Gotowy do podania Kod do logowania

By dodać do aplikacji zabezpieczenie oparte na logowaniu, konieczne będzie dodanie do kontrolera ads dwóch fragmentów kodu: **metody logowania** sprawdzającej nazwę użytkownika i hasło oraz **filtra** wywołującego metodę logowania zawsze, gdy wywoływane są pewne metody kontrolera:

```
class AdsController < ApplicationController
  before_filter :check_logged_in, :only => [:edit, :update]
  def new
    @ad = Ad.new
  end
  def create
    @ad = Ad.new(params[:ad])
    @ad.save
    redirect_to "/ads/#{@ad.id}"
  end
  def edit
    @ad = Ad.find(params[:id])
  end
  def update
    @ad = Ad.find(params[:id])
    @ad.update_attributes(params[:ad])
    redirect_to "/ads/#{@ad.id}"
  end
  def show
    @ad = Ad.find(params[:id])
  end
  def index
    @ads = Ad.find(:all)
  end
  private
  def check_logged_in
    authenticate_or_request_with_http_basic("Ads") do |username, password|
      username == "admin" && password == "t4k3th3r3dpill"
    end
  end
end
```

To jest kontroler z dodanym kodem do logowania.

To jest filtr.

Użytkownicy będą proszeni o zalogowanie się tylko wtedy, gdy będą próbowali wywołać metody „edit” bądź „update”.

Filtr wywołuje metodę „check_logged_in”, jeśli ktoś próbuje wywołać metody „edit” bądź „update”.

To nazwa zabezpieczonego obszaru strony internetowej – „domena”.

Nazwa użytkownika

Hast

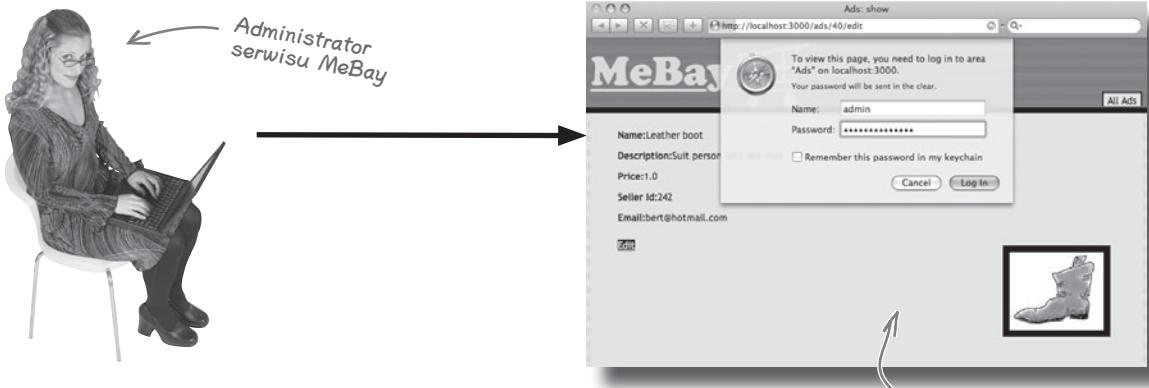


Jazda próbna

Kod zabezpieczający jest gotowy — czas otworzyć przeglądarkę i spróbować zmodyfikować ogłoszenie.



Po kliknięciu odnośnika „Edit” pojawia się okno dialogowe pytające o nazwę użytkownika i hasło.



Dostęp mają jedynie osoby z poprawną nazwą użytkownika oraz hasłem.

Teraz tylko osoby znające nazwę użytkownika administratora oraz hasło mogą modyfikować ogłoszenia na stronie. Nikt inny nie może się dostać na stronę edycji ogłoszenia bądź do funkcji update.

Czy zabezpieczenie zarówno strony, jak i funkcji update ma znaczenie? Strona edycji ogłoszenia powinna być niedostępna, by użytkownicy nie weszli na nią przypadkowo i nie tracili czasu, wprowadzając dane. Natomiast funkcja update (kod wykonujący samo uaktualnienie bazy danych) musi być zabezpieczona, na wypadek gdyby haker próbował się do niej dostać bezpośrednio, z pominięciem formularza edycji.

Ręcznie utworzyłeś system, który może tworzyć, odczytywać i uaktualniać ogłoszenia... a na dodatek jest bezpieczny!

...teraz jednak stare ogłoszenia trzeba usunąć

Witryna działa, wszystko idzie świetnie, jednak po niedługim czasie pojawia się problem: nawet po transakcji sprzedaży ogłoszenia nadal pozostają opublikowane.



Zespół MeBay mógłby oczywiście usuwać ogłoszenia za pomocą własnego systemu wprowadzania danych, jednak tak duże wrażenie zrobiło na nim to, jak proste było dodanie opcji edycji, że teraz chciałby, żebyś wprowadził również opcję usuwania.

Opcja ta będzie dostępna jedynie dla administratorów serwisu MeBay, dlatego konieczne jest zastosowanie tych samych zabezpieczeń co w przypadku edycji ogłoszeń. A ponieważ na stronie pojawiło się sporo spamu i ogłoszeń-żartów, firma MeBay chce, by funkcja usuwania była łatwo dostępna ze strony indeksującej. W ten sposób niewłaściwe ogłoszenia będzie można usuwać za pomocą jednego kliknięcia.

Sprawdźmy, co musimy zrobić...

Utwórz, odczytaj, uaktualnij i usuń



Ćwiczenie (nieco dłuższe)

Uaktualnij stronę `index.html.erb`, tak by dodać do niej odnośnik do usunięcia obok każdego ogłoszenia. Jeśli ogłoszenie ma `id = 17`, odnośnik powinien prowadzić do `/ads/17/delete`.

```
h1>All Ads</h1>
<ul>
<% for ad in @ads %>
  <li><a href="/ads/<%= ad.id %>"><%= ad.name %></a>
    [<a href="....." .....>Delete</a>]</li>
<% end %>
</ul>
```

↑
Wstaw tutaj odnośnik.

Utwórz trasę łączącą ścieżkę taką jak `/ads/17/delete` z akcją kontrolera ads o nazwie `destroy`. Pamiętaj, by zapisać numer identyfikatora jako parametr żądania.

.....
↑
Wpisz tutaj trasę.

Uzupełnij kod kontrolera ads, tak by możliwe było usunięcie ogłoszenia i odesłanie przeglądarki użytkownika z powrotem na stronę indeksującą pozostałe ogłoszenia.

By tego dokonać, będziesz musiał użyć metody, której jeszcze nie widzieliśmy — metody `destroy`. Usunie ona obiekt ogłoszenia z bazy danych.

```
class AdsController < ApplicationController
  before_filter :check_logged_in, :only => [:edit, :update, .....]
  def new
    @ad = Ad.new
  end
  def create
    @ad = Ad.new(params[:ad])
    @ad.save
    redirect_to "/ads/#{@ad.id}"
  end
  def edit
    @ad = Ad.find(params[:id])
  end
  def update
    @ad = Ad.find(params[:id])
    @ad.update_attributes(params[:ad])
    redirect_to "/ads/#{@ad.id}"
  end
  def show
    @ad = Ad.find(params[:id])
  end
  def index
    @ads = Ad.find(:all)
  end
  .....
  .....
  @ad.destroy
  .....
  .....
  private
  def check_logged_in
    authenticate_or_request_with_http_basic("Ads") do |username, password|
      username == "admin" && password == "t4k3th3r3dp1ll"
    end
  end
end
```



Usunie ona obiekt „@ad” z bazy danych.

To jest już trudniejsze



Ćwiczenie
(nieco dłuższe)

Rozwiązanie

Powinieneś był uaktualnić stronę `index.html.erb`, tak by dodać do niej odnośnik do usunięcia obok każdego ogłoszenia.

```
<h1>All Ads</h1>
<ul>
<% for ad in @ads %>
  <li><a href="/ads/<%= ad.id %>"><%= ad.name %></a>
    [<a href="/ads/<%= ad.id %>/delete" >Delete</a>]</li>
<% end %>
</ul>
```

Pętla „for” oznacza, że ogłoszenie jest zapisane pod zmienną o nazwie „ad”.

Powinieneś był również utworzyć trasę łączącą ścieżkę taką jak `/ads/17/delete` z akcją kontrolera ads o nazwie `destroy`. Pamiętaj, by zapisać numer identyfikatora jako parametr żądania.

Pamiętaj o wstawieniu tej trasy ponad pozostałymi trasami znajdującymi się w pliku `config/routes.rb`, by nie pomieszała się ona z trasą `"/ads/:id"`.

Dzięki temu identyfikator zostanie przechowany w parametrze żądania o nazwie „:id”.

To oznacza, że metoda kontrolera będzie musiała nosić nazwę „destroy”.

Twoje zadanie polegało na uzupełnieniu kodu kontrolera ads, tak by możliwe było usunięcie ogłoszenia i odesłanie przeglądarki użytkownika z powrotem na stronę indeksującą pozostałe ogłoszenia.

```
class AdsController < ApplicationController
  before_filter :check_logged_in, :only => [:edit, :update, .....]
  def new
    @ad = Ad.new
  end
  def create
    @ad = Ad.new(params[:ad])
    @ad.save
    redirect_to "/ads/#{@ad.id}"
  end
  def edit
    @ad = Ad.find(params[:id])
  end
  def update
    @ad = Ad.find(params[:id])
    @ad.update_attributes(params[:ad])
    redirect_to "/ads/#{@ad.id}"
  end
  def show
    @ad = Ad.find(params[:id])
  end
  def index
    @ads = Ad.find(:all)
  end
  def destroy
    @ad = Ad.find(params[:id])
    @ad.destroy
    redirect_to '/ads/'
  end
  private
  def check_logged_in
    authenticate_or_request_with_http_basic("Ads") do |username, password|
      username == "admin" && password == "t4k3th3r3dp11"
    end
  end
end
```

Dzięki temu użytkownik będzie musiał podać nazwę użytkownika oraz hasło, jeśli będzie próbował usunąć ogłoszenie.

Konieczne jest wczytanie obiektu ogłoszenia z bazy danych za pomocą identyfikatora podanego w ścieżce do akcji...

... a następnie należy przekierować przeglądarkę z powrotem do strony indeksującej.



Jazda próbna

Dodanie akcji `destroy` oznacza, że użytkownicy mogą usuwać ogłoszenia ze strony za pomocą jednego kliknięcia. Wykorzystuje ona te same zabezpieczenia co opcja edycji, dlatego przed usunięciem ogłoszenia użytkownik musi najpierw udowodnić, że jest administratorem.

Ads

MeBay

All Ads

- Typewriter [Delete]
- Football [Delete]
- Moosehead [Delete]
- Desk [Delete]
- Door curtain [Delete]
- Apple Newton [Delete]
- Sinclair C5 [Delete]
- Edsel [Delete]
- Diamond [Delete]
- Leather boot [Delete]
- "Days of Our Lives" [Create Edition] [Delete]

Usuniemy ten rekord.

Ads: index

To view this page, you need to log in to area "Ads" on localhost:3000.
Your password will be sent in the clear.

Name: admin
Password: [REDACTED]
 Remember this password in my keychain

Cancel Log In

Ads: index

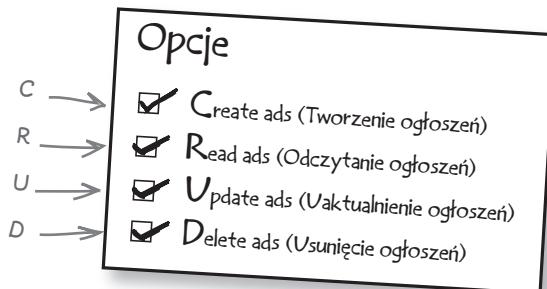
MeBay

All Ads

- Typewriter [Delete]
- Football [Delete]
- Moosehead [Delete]
- Desk [Delete]
- Door curtain [Delete]
- Apple Newton [Delete]
- Sinclair C5 [Delete]
- Edsel [Delete]
- Diamond [Delete]
- Leather boot [Delete]
- "Days of Our Lives" [Create Edition] [Delete]

Rekord zostat usuniety.

Skoro system potrafi usuwać ogłoszenia, aplikacja wykonuje wszystkie podstawowe operacje CRUD:



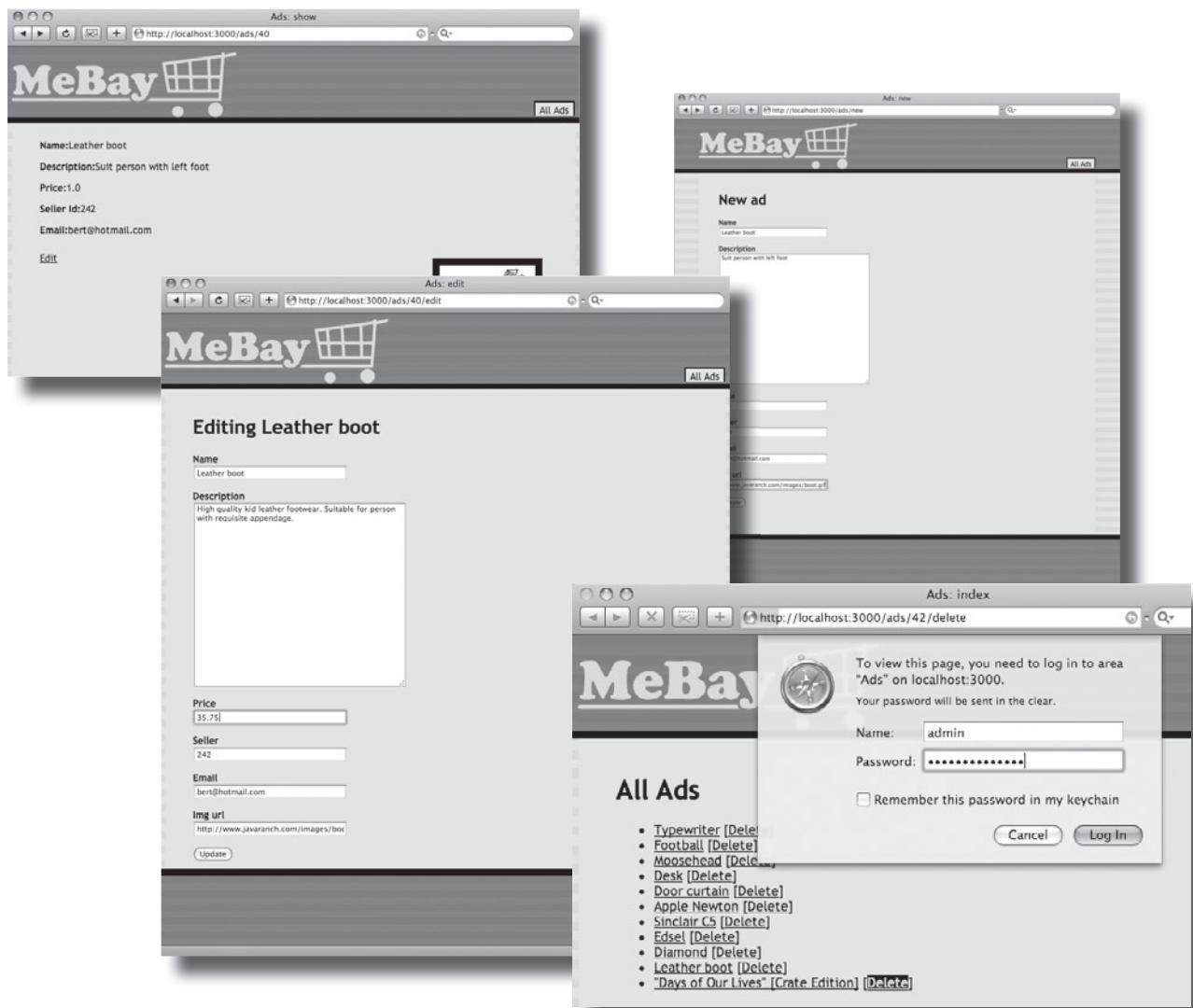
Ale w końcu to samo robiło rusztowanie...
po co więc pisać własny kod?

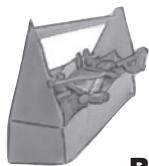
Wykonanie tego samodzielnie dało Ci możliwość zrobienia więcej, niż potrafi rusztowanie

Możesz **wybrać**, które funkcje są dostępne.

Możesz **dodawać** dodatkowe opcje, takie jak zabezpieczenia.

I **rozumiesz** teraz, jak tworzyć kod wstawiający, uaktualniający oraz usuwający dane, dzięki czemu będziesz w stanie poprawić kod wygenerowany przez rusztowanie.





Niezbędnik programisty Rails

Masz za sobą rozdział 3. i teraz do swojego niezbędnika programisty Rails możesz dodać umiejętność ręcznego tworzenia aplikacji Rails wstawiających, aktualniających oraz usuwających dane.

Narzędzia Rails

`@ad.save` zapisuje obiekty modelu.

`@ad.update_attributes` aktualnia obiekt modelu.

`redirect_to` pozwala kontrolerowi odesłać przeglądarkę pod inny adres URL.

`http_authentication` ekspresowo wprowadza zabezpieczenia.

Narzędzia języka Ruby

`params[...]` to *tablica asocjacyjna*, czyli tablica indeksowana za pomocą *nazwy*.

`nil` to specjalny obiekt domyślny oznaczający „brak wartości”.

Wewnątrz Rails wywoływanie metod na obiekcie „`nil`” powoduje błędy.

Znaki „`{`” oraz „`}`” wstawiają wyrażenia do łańcuchów znaków; przykład: `1 + 1 = #{1+1}`.

4. Wyszukiwanie w bazie danych

Prawda czy konsekwencje?



Mówiłem Ci,
że starucha nie
przestanie wsuwać tej
słodkiej babki...

Każda decyzja ma swoje konsekwencje. W Rails wiedza o tym, jak podejmować **dobre decyzje**, może zaoszczędzić Ci zarówno czasu, jak i wysiłku. W tym rozdziale przyjrzymy się, jak **wymagania użytkownika** wpływają na wybory, jakich dokonujesz, już **od samego początku** tworzenia Twojej aplikacji. Czy powinieneś użyć rusztowania, czy lepiej zmodyfikować wygenerowany kod? Czy powinieneś tworzyć wszystko od nowa? Bez względu na wybór, kiedy nadziejdzia pora dalszego dostosowania aplikacji do własnych potrzeb, będziesz musiał nauczyć się obsługi **wyszukiwania w bazie danych — dostępu do danych** w sposób, który ma sens zarówno z Twojego punktu widzenia, jak i z punktu widzenia **potrzeb Twoich użytkowników**.

Wymagania użytkownika

Dbaj o siebie z Rubyville Health Club

Klub Rubyville Health Club szczyci się umiejętnością dobrania odpowiedniego typu zajęć dla każdego, a ostatnio wprowadzono nawet nową usługę osobistego trenera. Popyt na tę usługę jest ogromny... tak duży, że trenerom z trudem przychodzi zapamiętywanie wszystkich danych każdego z klientów. Trenerzy potrzebują, by ktoś szybko stworzył dla nich odpowiednią aplikację.

Interes świetnie się kręci,
ale mamy kłopot z prześledzeniem
wszystkich prywatnych zajęć fitness
naszych klientów. Myślisz, że dasz
radę pomóc?

Osobistym trenerom potrzebna jest aplikacja internetowa, która będzie pozwalała szybko i łatwo zarządzać ćwiczeniami każdego z klientów. Na początek potrzebują strony, która będzie wymieniała wszystkie podstawowe szczegóły dotyczące programu treningowego każdego klienta, a także pozwalała dodawać, uaktualniać oraz usuwać rekordy. Poniżej znajduje się szkic strony głównej:

The screenshot shows a web browser window with the title "ClientWorkouts: find" and the URL "http://localhost:3000/client_workouts/find/". The page displays a table titled "Listing client workouts for Lenny Goldberg". The table has columns: Trainer, Duration mins, Date of workout, Paid amount, and three actions: Show, Edit, and Destroy. The data in the table is as follows:

Trainer	Duration mins	Date of workout	Paid amount	Action
Clint	30	2009-07-14 09:14:00 UTC	25.0	Show Edit Destroy
Brad	30	2009-07-19 09:13:00 UTC	25.0	Show Edit Destroy
Sven	90	2009-08-02 09:13:00 UTC	75.0	Show Edit Destroy
Marshall	15	2009-09-29 13:15:00 UTC	15.0	Show Edit Destroy
Clint	30	2009-10-01 09:11:00 UTC	25.0	Show Edit Destroy
Sara	30	2009-10-05 19:00:00 UTC	25.0	Show Edit Destroy

Annotations on the right side of the table:

- An arrow points from the text "Lenny jest jednym z klientów korzystających z usługi osobistego trenera." to the first row of the table.
- An arrow points from the text "To są wszystkie jego zajęcia." to the last row of the table.

At the bottom left of the page is a link labeled "New client workout".





Ćwiczenie

Zacznijmy aplikację od utworzenia zbioru stron przy pomocy rusztowania dla tego modelu:

client_workouts

Kolumna	Typ
client_name (dane klienta)	string
trainer (trener)	string
duration_mins (czas trwania treningu w minutach)	integer
date_of_workout (data treningu)	date
paid_amount (zapłacona kwota)	decimal

Jak powinno w takiej sytuacji wyglądać polecenie scaffold? Napisz swoją odpowiedź tutaj. Wykonaj także to polecenie:

.....
.....

Napisz swoją odpowiedź tutaj.

Teraz spójrz raz jeszcze na to, co trenerzy chcieliby robić za pomocą aplikacji, i wypisz poniżej różnice pomiędzy tym, co aplikacja **ma robić**, a tym, co **robi obecnie** w oparciu o wersję z rusztowaniem, jaką właśnie utworzyłeś.



.....
.....
.....
.....
.....
.....
.....

Z rusztowaniem czy bez?



Ćwiczenie
Rozwiązanie

Pamiętaj, że przed wykonaniem tego polecenia musiszbyś utworzyć nową aplikację Rails...

Zacznijmy aplikację od utworzenia rusztowania zbioru stron dla tego modelu:

client_workouts

Kolumna	Typ
client_name (dane klienta)	string
trainer (trener)	string
duration_mins (czas trwania treningu w minutach)	integer
date_of_workout (data treningu)	date
paid_amount (zapłacona kwota)	decimal

Jak powinno w takiej sytuacji wyglądać polecenie scaffold? Napisz swoją odpowiedź tutaj.

ruby script/generate scaffold client_workout client_name:string trainer:string

duration_mins:integer date_of_workout:date paid_amount:decimal

... a potem
musiszbyś wykonać
„rake db:migrate”
w celu utworzenia
tabeli.

Twoje zadanie polegało na wypisaniu różnic pomiędzy tym, co aplikacja **powinna robić**, a tym, co **robi** jej wersja oparta na rusztowaniu.

Trenerzy chcą przeglądać sesje treningowe zapisane dla każdego z klientów. Aplikacja wygenerowana za pomocą rusztowania pokazuje jedynie sesje treningowe pojedynczego klienta, a nie wszystkich klientów razem.

Rusztowanie jest niepoprawne — ale czy lepiej napisać własny kod, czy też poprawić rusztowanie?

Aplikacja oparta na rusztowaniu nie robi dokładnie tego, co chcemy.

Wcześniej widzieliśmy, że **proste** aplikacje łatwiej jest tworzyć ręcznie, bez korzystania z rusztowania. Innym rozwiązaniem jest utworzenie aplikacji opartej na rusztowaniu, a później **modyfikacja** czy **rozwiniecie** kodu wygenerowanego przez Rails.

Co powinniśmy zrobić w tej sytuacji?



Jazda próbna

Jeśli jeszcze tego nie zrobiłeś, użyj polecenia scaffold do wygenerowania aplikacji. Będziemy mogli porównać ją z tym, czego oczekują trenerzy, i sprawdzimy, ile nam do tego brakuje.

Aplikacja w zasadzie wygląda dość podobnie...

Jedna część wygenerowanego kodu wygląda w *pewnym sensie* podobnie do strony, której potrzebują trenerzy. Strona indeksująca wyświetla zbiór danych, który wygląda prawie jak ten, o jaki prosili trenerzy:

The screenshot shows two browser windows side-by-side. Both windows have a header bar with icons and a URL bar showing `http://localhost:3000/client_workouts/find/`.

Left Window (ClientWorkouts: find):

Listing client workouts for Lenny Goldberg

Trainer	Duration mins	Date of workout	Paid amount
Clint	30	2009-07-14	
Brad	30	2009-07-19	
Sven	90	2009-08-02	
Marshall	15	2009-09-29	
Clint	30	2009-10-01	
Sara	30	2009-10-05	

[New client workout](#)

Right Window (ClientWorkouts: index):

Listing client_workouts

Client name	Trainer	Duration mins	Date of workout	Paid amount	Show	Edit
Kirk Stigwood	Clint	60	2009-10-05	50.0	Show	Edit
Lenny Goldberg	Clint	30	2009-07-14	25.0	Show	Edit
Lenny Goldberg	Brad	30	2009-07-19	25.0	Show	Edit
Lenny Goldberg	Sven	90	2009-08-02	75.0	Show	Edit
Lenny Goldberg	Marshall	15	2009-09-29	15.0	Show	Edit
Lenny Goldberg	Clint	30	2009-10-01	25.0	Show	Edit
Lenny Goldberg	Sara	30	2009-10-05	25.0	Show	Edit

[New client workout](#)

Annotations in the image:

- An arrow points from the text "...a to jest strona indeksująca aplikacji opartej na rusztowaniu." to the left window.
- An annotation above the right window says "To chcą zobaczyć trenerzy..." with an arrow pointing to it.
- An annotation below the left window says "... a to jest strona indeksująca aplikacji opartej na rusztowaniu." with an arrow pointing to it.
- An annotation below the right window says "... czy lepiej poprawić rusztowanie?" with an arrow pointing to a woman holding an apple.
- An annotation below the right window says "Czy powinniśmy zacząć od początku..." with an arrow pointing to the woman holding an apple.

Wygenerowana strona nie tylko wygląda podobnie do tego, czego potrzebujemy; wiemy też, że aplikacja wygenerowana za pomocą rusztowania da nam dostęp do standardowych operacji wykonywanych na danych klientów. Innymi słowy, aplikacja wygenerowana za pomocą rusztowania domyślnie będzie pozwalała nam tworzyć, odczytywać, uaktualniać oraz usuwać rekordy.

Czy w tym przypadku lepiej będzie **poprawić rusztowanie** i wprowadzić pożądane zmiany, czy **zaczynać od początku**, tak jak robiliśmy to z serwisem MeBay?

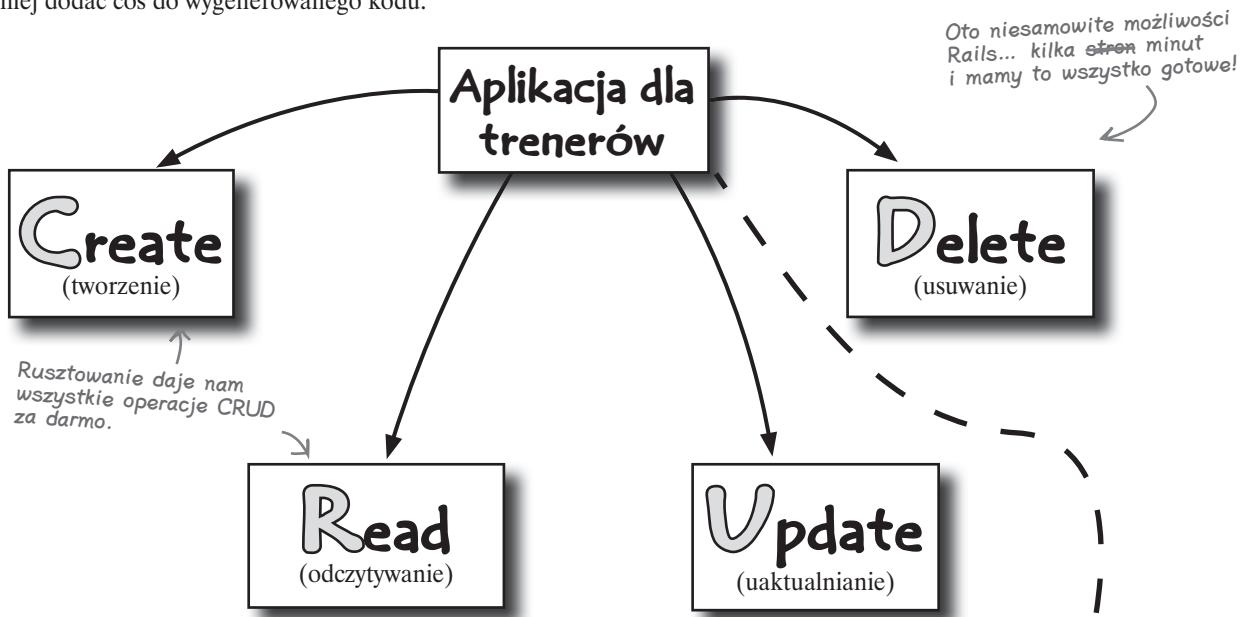


Poprawimy rusztowanie

Kiedy tworzyliśmy aplikację MeBay, zdecydowaliśmy się nie korzystać z rusztowania. Powodem było to, że klient na początku chciał czegoś tak prostego, że łatwiej było zacząć aplikację od podstaw. Klient chciał o wiele mniejszą liczbę funkcji niż udostępniane przez rusztowanie.

Tym razem potrzebny jest nam dostęp do *wszystkich* operacji CRUD oraz możliwość odszukania sesji treningowych *konkretnego klienta*.

A ponieważ potrzebna jest nam większa liczba funkcji, możemy użyć rusztowania jako podstawy aplikacji, co załatwi nam większość pracy, a później dodać coś do wygenerowanego kodu.

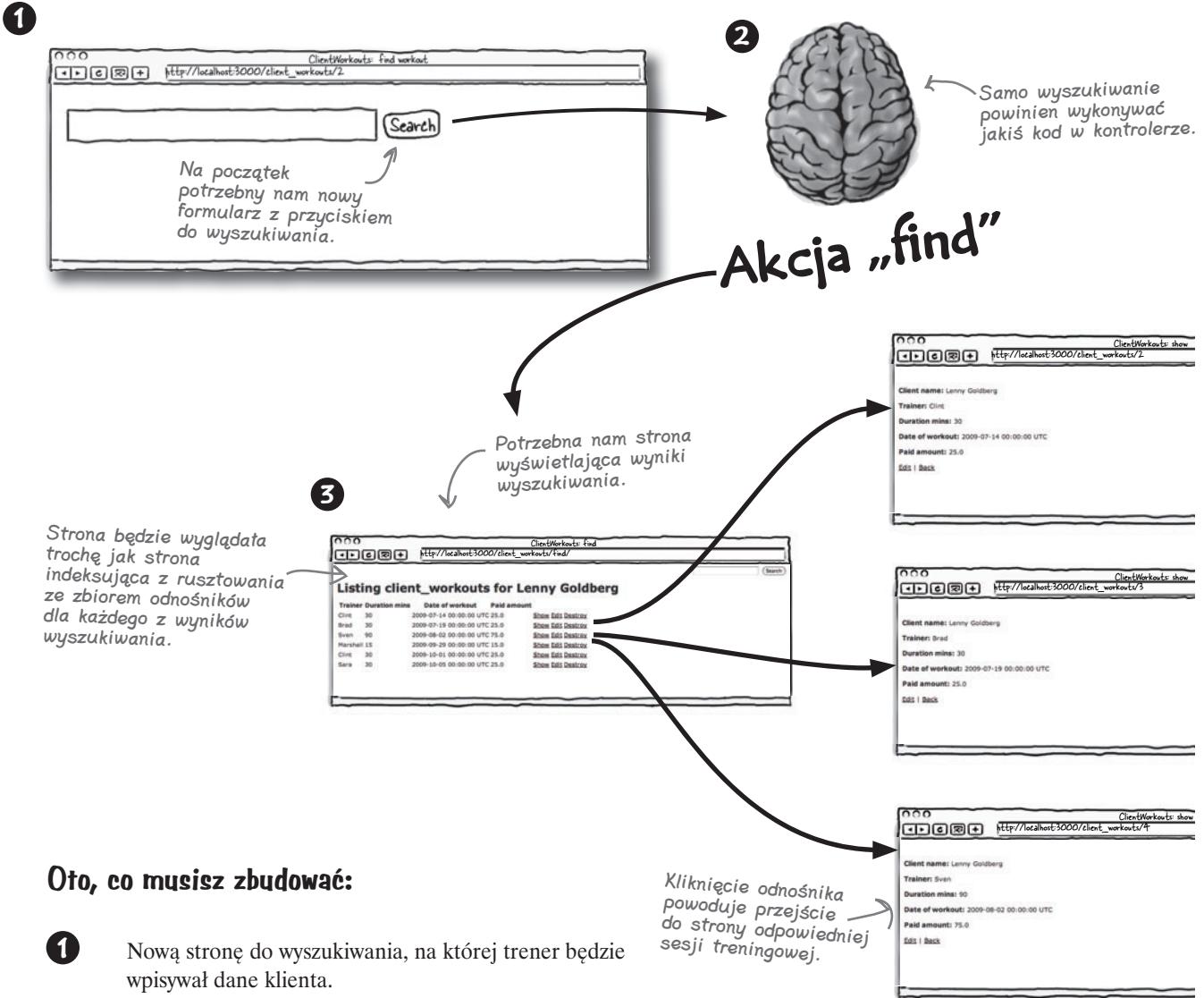


Jak możemy zatem odszukać i wyświetlić sesje treningowe dla jednego klienta?



Zaprojektowanie opcji wyszukiwania

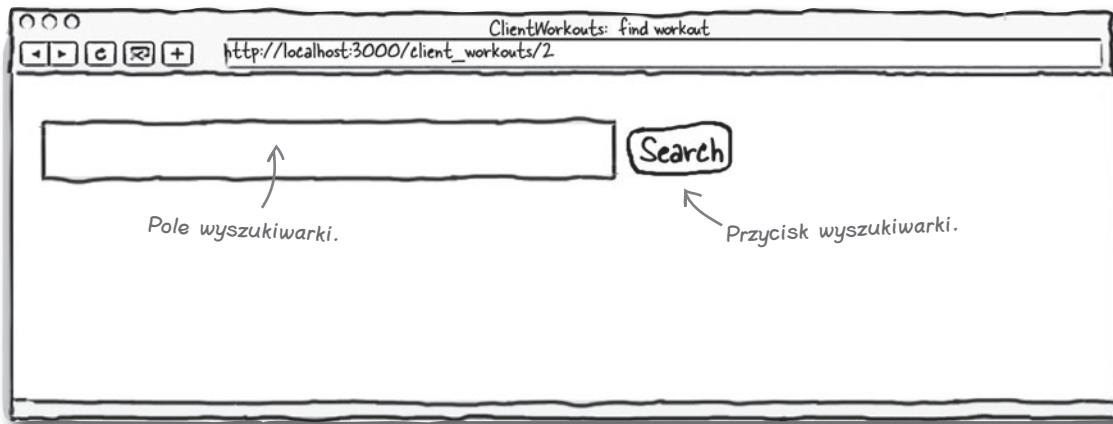
Oto jak powinna wyglądać funkcja wyszukiwania:



Od czego powinniśmy zacząć?

Zacznijmy od utworzenia formularza

Musimy utworzyć kilka nowych komponentów, zacznijmy więc od interfejsu użytkownika. W ten sposób szybko otrzymamy informacje zwrotne od trenerów. Tak, według trenerów, powiniem wyglądać formularz do wyszukiwania klientów:



Tworzyłeś już wcześniej strony z formularzami. Czy ta jest w jakiś sposób inna od poprzednich?

Przyjrzyj się innym formularzom wygenerowanym dla aplikacji, na przykład służącym do tworzenia i edycji danych. Zawierają one pola odpowiadające polem obiektów modelu ClientWorkout. Różnica polega na tym, że tym razem nie mamy obiektu modelu odpowiadającego formularzowi wyszukiwania. W jaki sposób tworzymy formularz, kiedy nie mamy modelu, na którym można by go oprzeć?

Wyszukiwarka będzie wymagała nowego typu formularza

Musimy utworzyć formularz *bez* korzystania z obiektu modelu, jednak metoda pomocnicza `form_for`, z której korzystaliśmy wcześniej, wymaga do działania obiektu modelu. Co zatem możemy zrobić?

Na szczęście istnieje inny znaczek pomocniczy tworzący formularze bez modelu — czyli dokładnie to, co potrzebne jest nam w tej sytuacji.



Magnesiki z kodem — formularz wyszukiwania

Będziesz musiał utworzyć szablon strony wyszukiwania. Problem polega na tym, że po tym, jak udało nam się poskładać najważniejszą część kodu na drzwiach lodówki, ktoś za mocno trzasnął drzwiami i wszystkie magnesiki odpadły!

Na szczęście kod dla formularza bez modelu jest dość podobny do tego opartego na modelu.

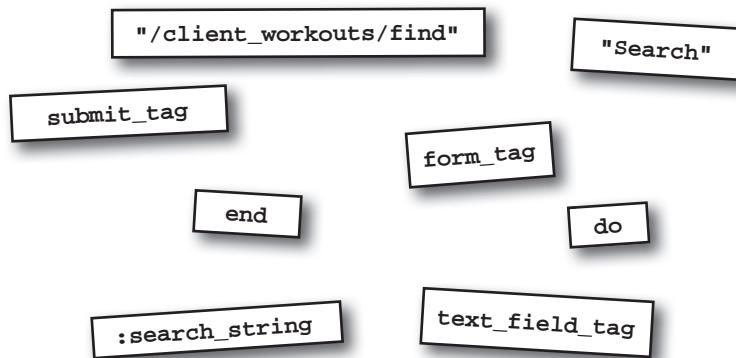
Czy potrafisz odgadnąć, jak powinien wyglądać kod?

<% %>

<%= %>

<%= %>

<% %>

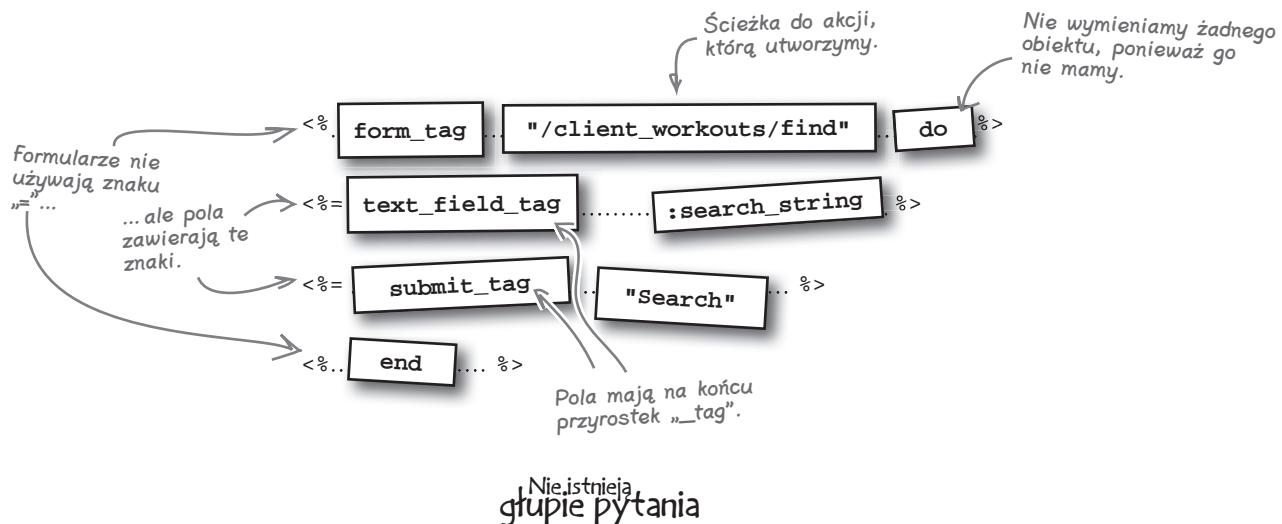




Magnesiki z kodem — formularz wyszukiwania: Rozwiążanie

Twoje zadanie polegało na zebraniu magnesików i ułożeniu ich w kod tworzący formularz wyszukiwania niewymagający odpowiadającego mu modelu.

Czy udało Ci się odgadnąć, jak powinien wyglądać kod?



P: Nie rozumiem, dlaczego nie możemy w pełni skorzystać z rusztowania. Czy rusztowanie nie daje nam wszystkiego?

O: Rusztowanie udostępnia jedynie podstawowe operacje CRUD (tworzenia, odczytania, uaktualniania oraz usuwania). Większość aplikacji wymaga funkcji wykraczających poza podstawowe operacje CRUD.

P: Dlaczego nie użyliśmy rusztowania w aplikacji MeBay?

O: W aplikacji MeBay nie użyliśmy rusztowania, ponieważ na początku potrzebna nam była jedynie prosta aplikacja do odczytu. W przypadku naprawdę prostych aplikacji łatwiej i wydajniej jest tworzyć aplikację ręcznie.

P: A czy nie lepiej byłoby wygenerować aplikację za pomocą rusztowania, po czym usunąć niechciane operacje?

O: Owszem, mogliśmy to zrobić. Prawdopodobnie większość aplikacji będziesz zaczynać od rusztowania. Jedynie w przypadkach, gdy potrzebujesz bardziej ograniczonej funkcjonalności lub gdy ta funkcjonalność różni się mocno od rusztowania, będziesz tworzył aplikację ręcznie.

P: Czyli czasami lepiej jest zacząć od początku, a czasami bardziej opłaca się zmodyfikować rusztowanie. Kiedy powiniem wybrać określone rozwiązanie? Jak mogę dokonać najlepszego wyboru?

O: Użyj rusztowania, jeśli masz zamiar korzystać z operacji CRUD, czyli tworzenia,

odczytywania, uaktualniania oraz usuwania. Zadaj sobie pytanie: co będzie szybsze — ręczne tworzenie kodu czy usunięcie niewykorzystanego kodu rusztowania?

P: Czym jest formularz oparty na modelu?

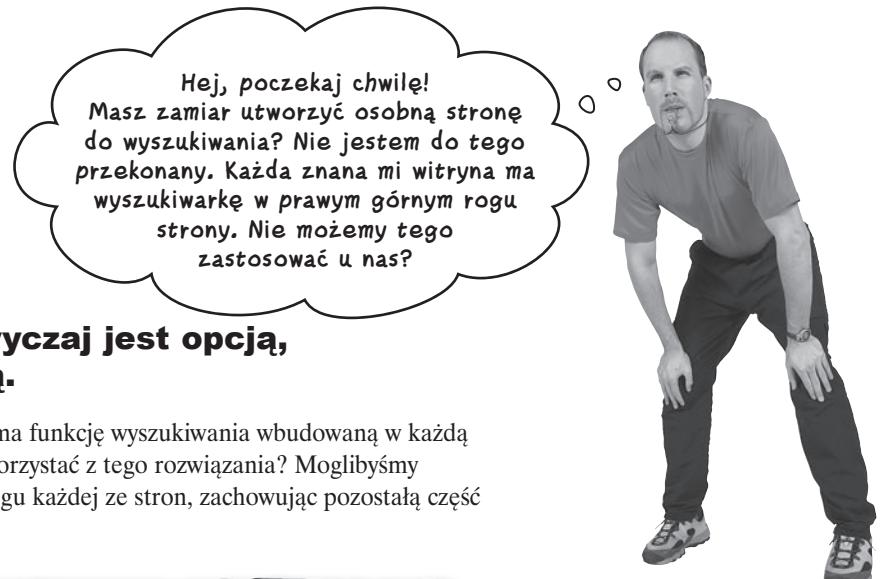
O: Formularz ten jest powiązany z obiektem modelu. Kiedy formularz tego typu jest wyświetlany, wartości pól pochodzą z atrybutów obiektu modelu.

P: A formularz, który nie jest oparty na modelu?

O: To formularz, który nie jest powiązany z obiektem modelu. Formularz tego typu wykorzystywany jest w połączeniu ze zbiorem samodzielnych wartości pól i najczęściej używany jest do wyszukiwania czy innego typu danych, które nie zostają zapisane do bazy danych.

Dodanie wyszukiwania do interfejsu

Kiedy mamy już kod formularza wyszukiwania, powinniśmy pewnie utworzyć dla niego nowy szablon strony.



Wyszukiwanie zazwyczaj jest opcją, a nie osobną stroną.

Większość witryn internetowych ma funkcję wyszukiwania wbudowaną w każdą stronę, więc może powinniśmy skorzystać z tego rozwiązania? Moglibyśmy dodać wyszukiwarkę w górnym rogu każdej ze stron, zachowując pozostałą część poszczególnych stron bez zmian.

Moglibyśmy umieścić pole wyszukiwarki w tym miejscu każdej strony...

... a pozostałą część strony pozostawić bez zmian.

Trainer	Duration mins	Date of workout	Paid amount	
Clint	30	2009-07-14 09:14:00 UTC	25.0	Show Edit Destroy
Brad	30	2009-07-19 09:13:00 UTC	25.0	Show Edit Destroy
Sven	90	2009-08-02 09:13:00 UTC	75.0	Show Edit Destroy
Marshall	15	2009-09-29 13:15:00 UTC	15.0	Show Edit Destroy
Clint	30	2009-10-01 09:11:00 UTC	25.0	Show Edit Destroy
Sara	30	2009-10-05 19:00:00 UTC	25.0	Show Edit Destroy

New client_workout

Dodanie kodu do każdej strony oznacza, że będziemy musieli utrzymywać mnóstwo powtarzającego się kodu. Co jednak, gdyby udało nam się dodać nowy kod wyszukiwania tylko do jednego pliku?

Zaostrz ołówek



Jest taki plik, do którego moglibyśmy dodać formularz wyszukiwarki, tak by pojawiał się on na każdej z podstron aplikacji. Jak nazywa się ten plik? Napisz swoją odpowiedź tutaj.

.....

Układy stron są współdzielone

Zaostrz ołówek



Rozwiążanie

Czy pamiętasz układy stron?
Zawierają one kod, który
znajdzie się w każdym
szablonie strony.

Jest taki plik, do którego moglibyśmy dodać formularz wyszukiwarki, tak
by pojawiał się on na każdej z podstron aplikacji. Jak nazywa się ten plik?
Napisz swoją odpowiedź tutaj.

.....
app/views/layouts/client_workouts.html.erb



Gotowy do podania Kod

Ten kod należy zapisać w pliku układu strony

app/views/layouts/client_workouts.html.erb

To plik układu strony,
który znajduje się w katalogu
app/views/layouts.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
  
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">  
<head>  
  <meta http-equiv="content-type" content="text/html; charset=UTF-8" />  
  <title>ClientWorkouts: <%= controller.action_name %></title>  
  <%= stylesheet_link_tag 'scaffold' %>  
</head>  
<body>  
  
<span style="text-align: right">  
  <% form_tag "/client_workouts/find" do %>  
    <%= text_field_tag :search_string %>  
    <%= submit_tag "Search" %>  
  <% end %>  
</span>  
  
<p style="color: green"><%= flash[:notice] %></p>  
  
<%= yield %>  
  
</body>  
</html>
```

Ten kod dodaje arkuszów stylów
public/stylesheets/scaffold.css.

To nasz nowy kod formularza
wyszukiwania.

Ten kod można wykorzystać
do przesypania wiadomości
do strony. Na razie się tym
nie przejmuj.



Jazda próbna

Zmodyfikuj układ strony aplikacji w taki sposób, by zawierał on funkcję wyszukiwania. Jeśli odświeżysz każdą z podstron aplikacji, pole wyszukiwarki powinno się pojawić w prawym górnym rogu.

Listing client_workouts

Client name: Kirk Stigwood
Lenny Goldb...
Lenny Goldb...
Client name: Kirk Stigwood
Trainer: Clint
Duration mins: 60
Date of workout: 2009-10-05
Paid amount: 50.0

New client_workout

Client name:
Trainer:
Duration mins:
Date of workout: 2008 December 10
Paid amount:
Create **Back**

Editing client_workout

Client name: Kirk Stigwood
Trainer: Clint
Duration mins: 60
Date of workout: 2009 October 5
Paid amount: 50
Update **Show | Back**

W jaki sposób możemy uzyskać dostęp do zawartości pola wyszukiwania? Poniżej znajduje się kod HTML wygenerowany na każdej z podstron w celu utworzenia formularza wyszukiwania:

```
<form action="/client_workouts/find" method="post">
  <input id="search_string" name="search_string" type="text" />
  <input name="commit" type="submit" value="Search" />
</form>
```

Zaostrz ołówek



Zakładając, że wiesz, jak będzie wyglądał kod HTML formularza, jak sądzisz, jakiego wyrażenia będzie można użyć wewnątrz kodu kontrolera w celu uzyskania dostępu do zawartości pola wyszukiwania? Napisz swoją odpowiedź tutaj.

Użycie tablicy params

Zaostrz ołówek



Rozwiążanie

Zakładając, że wiesz, jak będzie wyglądał kod HTML formularza, jak sądzisz, jakiego wyrażenia będzie można użyć wewnątrz kodu kontrolera w celu uzyskania dostępu do zawartości pola wyszukiwania? Napisz swoją odpowiedź tutaj.

.....
params[:search_string]

.....
params[“search_string”] też
by zadebiutować, jednak lepiej
jest użyć symbolu.

Czy parametry formularza są inaczej ustrukturyzowane?

Metoda pomocnicza **form_for**, z której korzystaliśmy w poprzednim rozdziale, tworzy **formularz oparty na modelu** — czyli formularz HTML bazujący na atrybutach obiektu modelu. Po przesłaniu formularza opartego na modelu Rails wie, że najprawdopodobniej będziesz chciał zmienić wartości pól z powrotem w obiekcie modelu. Przykładowo po przesłaniu wygenerowanego za pomocą rusztowania formularza edycji (utworzonego za pomocą **form_for**) parametry formularza zostają ustrukturyzowane następująco:

Oto zawartość tablicy „params[...]” (parametrów żądania) przestana przez formularz treningów klientów.

Wszystkie dane formularza przechowane są w „params[...]” pod nazwą „:client_workout”.

Name	Value												
:controller	‘client_workouts’												
:action	‘update’												
:client_workout	<table border="1"><thead><tr><th>Name</th><th>Value</th></tr></thead><tbody><tr><td>:client_name</td><td>Kirk Stigwood</td></tr><tr><td>:trainer</td><td>Clint</td></tr><tr><td>:paid_amount</td><td>50</td></tr><tr><td>duration_mins</td><td>60</td></tr><tr><td>...</td><td></td></tr></tbody></table>	Name	Value	:client_name	Kirk Stigwood	:trainer	Clint	:paid_amount	50	duration_mins	60	...	
Name	Value												
:client_name	Kirk Stigwood												
:trainer	Clint												
:paid_amount	50												
duration_mins	60												
...													

To są parametry z formularza opartego na modelu przechowane w sposób odpowiadający strukturze modelu.

„params[:client_workout]” zwraca te tablice asocjacyjną wartości formularza.

W przypadku formularza opartego na modelu tablicę asocjacyjną wszystkich wartości pól można otrzymać za pomocą prostego wyrażenia, takiego jak `params[:client_workout]`.

To są parametry żądania przestane przez formularz wyszukiwania.

Jak jednak wygląda to w przypadku metody pomocniczej **form_tag**? Metoda **form_tag** tworzy **formularz, który nie jest oparty na modelu**. Jest to formularz wykorzystywany do edycji zbioru **samodzielnego wartości pól**. Z tego powodu formularz wyszukiwania (tworzony za pomocą **form_tag**) tworzy parametry żądania ustrukturyzowane w następujący sposób:

Name	Value
:controller	client_workouts
:action	find
:search_string	Kirk Stigwood

„params[:search_string]” daje nam wartość samego pola „search_string”.



Ćwiczenie

Skoro już wiesz, jak pobrać łańcuch znaków wyszukiwania z formularza, czas utworzyć po stronie serwera kod kontrolera wykonujący wyszukiwanie. Na początek wyświetlimy wartość pola wyszukiwania w konsoli. W ten sposób możemy sprawdzić, czy formularz działa poprawnie.

Formularz zostanie przesłany do ścieżki o nazwie `/client_workouts/find`. Domyślne trasy z pliku `config/routes.rb` będą w stanie odwzorować dla nas tę trasę. Wybierz, która z tras domyślnych zostanie użyta:

```
map.connect ':controller/:action/:id'  
map.connect ':controller/:action/:id.:format'
```

Następnie utwórz metodę `find` kontrolera wyświetlającą zawartość pola wyszukiwania. Wskazówka: łańcuch znaków możesz wyświetlić w oknie konsoli za pomocą polecenia:

```
puts <łańcuch znaków>
```



Polecenie to wyświetli łańcuch znaków w oknie, w którym uruchomiony jest serwer WWW.

Zastosowanie polecenia puts w testowaniu



Ćwiczenie

Rozwiązanie

Skoro już wiesz, jak pobrać łańcuch znaków wyszukiwania z formularza, czas utworzyć po stronie serwera kod kontrolera wykonujący wyszukiwanie. Na początek wyświetlimy wartość pola wyszukiwania w konsoli. W ten sposób możemy sprawdzić, czy formularz działa poprawnie.

Formularz zostanie przesłany do ścieżki o nazwie `/client_workouts/find`. Domyślne trasy z pliku `config/routes.rb` będą w stanie odwzorować dla nas tę trasę. Wybierz, która z tras domyślnych zostanie użyta:

Nie musimy tworzyć
trasy — zostanie użyta
ta trasa domyślna.

```
map.connect ':controller/:action/:id'  
map.connect ':controller/:action/:id.:format'
```

Następnie utwórz metodę `find` kontrolera wyświetlającą zawartość pola wyszukiwania. Wskazówka: łańcuch znaków możesz wyświetlić w oknie konsoli za pomocą polecenia:

```
puts <string>
```

Polecenie to wyświetli łańcuch znaków w oknie,
w którym uruchomiony jest serwer WWW.

Wystarczy tylko wypisać
poprawną nazwę
parametru.

```
def find
```

```
    puts params[:search_string]
```

```
end
```



Jazda próbna

Wypróbujmy działanie kodu. W dowolnej ze stron aplikacji w polu wyszukiwania wpisz jakiś tekst i kliknij przycisk *Search*.

Takie coś wyświetlane jest po naciśnięciu przycisku wyszukiwania.

Nie martw się, jeśli po naciśnięciu przycisku otrzymujesz błąd. Dzieje się tak, ponieważ nie utworzyliśmy jeszcze szablonu strony z wynikami wyszukiwania. Najciekawsze dane wyjściowe znajdziesz w konsoli, w której uruchomiłeś serwer WWW. Gdzieś pośród błędów dotyczących brakującego szablonu z wynikami powinieneś zobaczyć takie coś:

Łańcuch znaków wyszukiwania →

```
Plik Edycja Okno Pomoc
Rendering client_workouts/show
Completed in 9ms (View: 4, DB: 0) | 200 OK [http://localhost/
client_workouts/1]

Lenny Goldberg

Processing ClientWorkoutsController#find (for 127.0.0.1 at
2008-10-13 22:00:40) [POST]
ActionView::MissingTemplate (Missing template client_workouts/
```

Udało nam się zatem utworzyć formularz wyszukiwania na każdej stronie, a po stronie serwera mamy kod potrafiący odczytywać łańcuch znaków, jakiego szuka użytkownik.

Teraz musimy wykonać samo wyszukiwanie.



CELNE SPOSTRZEŻENIA

- Aplikacje często muszą robić coś więcej niż tylko tworzyć, odczytywać, uaktualniać i usuwać rekordy.
- Czasami będziesz musiał zaprojektować własną sekwencję stron. Najłatwiej jest rozpocząć od punktu widzenia użytkownika i zastanowienia się, jak będzie on korzystał z aplikacji.
- Jeśli potrzebny Ci formularz, który nie odpowiada obiektowi modelu, użyj `form_tag` zamiast `form_for`.
- W formularzu `form_tag` konieczne jest używanie pól `_tag`.
- `params[:nazwa_pola]` da Ci wartość pola o nazwie `:nazwa_pola` w formularzu, który nie jest oparty na modelu.
- Polecenie `puts "łańcuch znaków"` zwraca łańcuch znaków do konsoli.

Nie, istnieją głupie pytania

P: Kiedy muszę zastosować `form_tag` zamiast `form_for`?

O: Metody `form_tag` należy użyć, kiedy formularz będzie edytował dane nieprzechowywane w obiekcie modelu. Wykorzystaliśmy `form_tag` w formularzu wyszukiwania, ponieważ nie istnieje obiekt modelu z pojedynczym atrybutem `search_string`.

P: Dlaczego metody pomocnicze `form_for` oraz `form_tag` otacza się znakami `<% ... %>` zamiast `<%= ... %>`?

O: Metody pomocnicze formularzy wykorzystywane są w połączeniu ze scriptletami (`<% ... %>`), a nie wyrażeniami (`<%= ... %>`), ponieważ robią coś więcej niż tylko generowanie kodu HTML.

Pamiętasz, że w pętlach `for` także wykorzystywaliśmy scriptlety? Robiliśmy tak, ponieważ pętla `for` kontroluje zawartość kodu w ciele pętli. W podobny sposób formularz „kontroluje” generowanie kodu HTML dla pól, jakie zawiera.

P: Brzmi to na dość skomplikowane...

O: Nie przejmuj się — nie musisz teraz wszystkiego rozumieć. Dopóki będziesz pamiętać, by używać `form_for` oraz

`form_tag` w połączeniu ze scriptletami, wszystko będzie w porządku.

P: Czy nadal mogę odczytywać pojedyncze pola z formularza modelu `form_for`?

O: Tak. Pola formularza dla formularzy `client_workouts` można odczytać za pomocą `params[:client_workout]` — jest to po prostu kolejna tablica asocjacyjna. By otrzymać wartość pola `trainer`, należy użyć `params[:client_workout][:trainer]`.

P: Dla akcji `find` wykorzystaliśmy trasę domyślną. Czy nie mogliśmy skorzystać z tego wcześniej?

O: Nie. Trasy wykorzystywane wcześniej nie pasowały odpowiednio blisko do tras domyślnych.

P: Skąd mam wiedzieć, kiedy powiniensem utworzyć własną trasę?

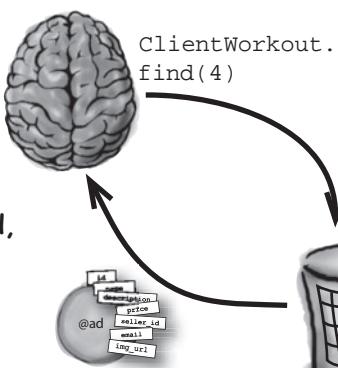
O: Jeśli wpiszesz `rake routes` w wierszu poleceń, zobaczysz wszystkie trasy domyślne w aplikacji. Jeśli żadna z nich nie pasuje albo jeśli jest tam trasa pasująca do niewłaściwej akcji, będziesz musiał dodać własną trasę.

Jak możemy znaleźć rekordy klientów?

Czy zatem mamy jakiś problem z odczytaniem rekordów dla określonego klienta? Dotychczas kiedy odczytywaliśmy rekordy, robiliśmy to albo poprzez zwrócenie pojedynczego rekordu, albo odnalezienie wszystkich rekordów w tabeli. Co tym razem się zmieniło?

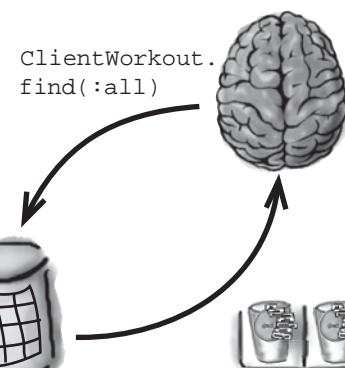
1 Odczytanie pojedynczego rekordu

Pojedynczy rekord możemy odczytać za pomocą wartości z kolumny id. Wiemy, że ta technika zwróci tylko jeden rekord, ponieważ numer identyfikatora jest unikalny dla każdego rekordu.



2 Odczytanie wszystkich rekordów

Jeśli zamiast przekazywać numer identyfikatora, przekażemy specjalny symbol :all, model zwróci tablicę zawierającą wszystkie rekordy powiązanej tabeli.

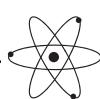


3 Odczytanie rekordów pasujących do określonych kryteriów

Tym razem chcemy otrzymać tylko rekordy pasujące do określonych kryteriów wyszukiwania. Być może będziemy chcieli zwrócić więcej niż jeden rekord, dlatego potrzebujemy czegoś, co zwróci nam tablicę obiektów modelu. Nie chcemy jednak obiektu modelu dla każdego rekordu, a jedynie te odpowiadające kryteriom wyszukiwania.



Po przekazaniu „:all” model zwraca tablicę ze wszystkimi rekordami tabeli. Tym razem chcemy otrzymać tylko niektóre rekordy, pasujące do określonych kryteriów wyszukiwania.



WYSIL
SZARE KOMÓRKI

Zastanów się nad danymi znajdującymi się w powiązanej tabeli bazy danych. Co oznacza dla rekordu w tabeli pasowanie do kryteriów wyszukiwania? Czy istnieje coś, co byłoby prawdziwe dla pasujących rekordów, a nieprawdziwe dla całej reszty?

Potrzebne nam jedynie te rekordy, gdzie client_name = łańcuch wyszukiwania

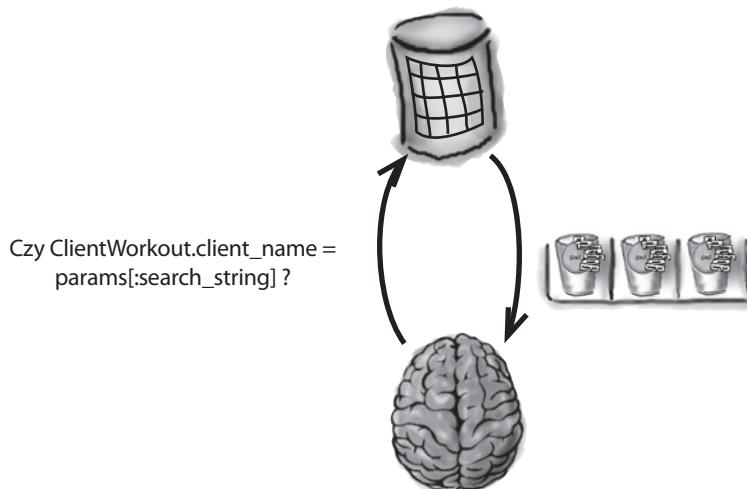
Trenerzy chcą wyszukać wszystkie sesje treningowe dla określonego klienta.
Model będzie potrzebował prostego testu, który będzie prawdziwy dla pasujących rekordów, a fałszywy dla całej reszty. Potrzebne będzie coś takiego:

Czy ClientWorkout.client_name = params[:search_string] ?
To to, co zostało wprowadzone do pola wyszukiwania.

Jeśli model może zastosować ten test do każdego z rekordów tabeli,
będzie w stanie odnaleźć pasujące rekordy:

Id	client_name	trainer	duration_mins	date_of_workout	paid_amount	created_at	updated_at
1	Kirk Stigwood	Clint	60	2009-10-05	50	2008-10-05 20:...	2008-10-05 20:...
2	Lenny Goldberg	Clint	30	2009-07-14	25	2008-10-06 09:...	2008-10-06 09:...
3	Lenny Goldberg	Brad	30	2009-07-19	25	2008-10-06 09:...	2008-10-06 09:...
4	Lenny Goldberg	Sven	90	2009-08-02	75	2008-10-06 09:...	2008-10-06 09:...
5	Lenny Goldberg	Marshall	15	2009-09-29	15	2008-10-06 09:...	2008-10-06 09:...
6	Lenny Goldberg	Clint	30	2009-10-01	25	2008-10-06 09:...	2008-10-06 09:...
7	Lenny Goldberg	Sara	30	2009-10-05	25	2008-10-05 20:...	2008-10-05 20:...

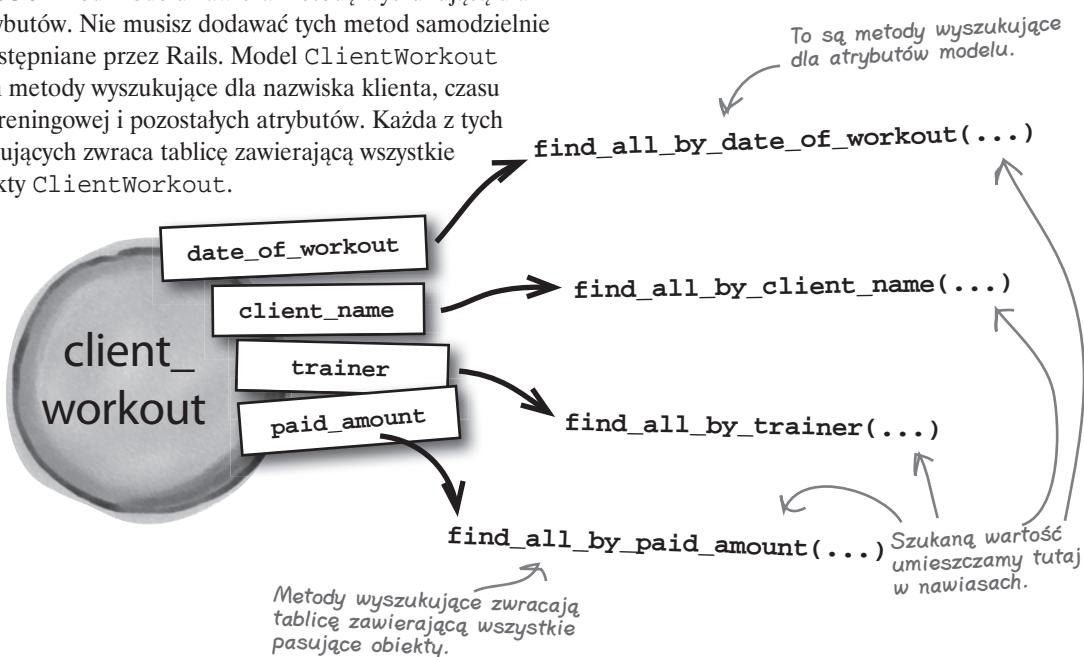
Potrzebna jest nam zatem funkcja wyszukująca, która potrafi odnaleźć wszystkie rekordy zawierające określoną wartość w określonej kolumnie tabeli.



Dla każdego atrybutu istnieje metoda wyszukująca

Wiele aplikacji musi wyszukiwać wszystkie rekordy z określoną wartością w kolumnie bazy danych, a Rails bardzo to ułatwia.

W jaki sposób? Kod modelu zawiera metodę wyszukującą dla każdego z atrybutów. Nie musisz dodawać tych metod samodzielnie — są one udostępniane przez Rails. Model ClientWorkout zawiera zatem metody wyszukujące dla nazwiska klienta, czasu trwania sesji treningowej i pozostałych atrybutów. Każda z tych metod wyszukujących zwraca tablicę zawierającą wszystkie pasujące obiekty ClientWorkout.



Pamiętaj, że atrybut obiektu modelu odzworowany jest na kolumnę bazy danych w powiązanej tabeli. Każdą z metod wyszukujących można wykorzystać do odszukania wszystkich rekordów zawierających pewną wartość w określonej kolumnie.

Zaostrz ołówek



Uzupełnij kod metody `find`.

```
def find
  @client_workouts = .....
end
```



Zaostrz ołówek

Rozwiążanie

Uzupełnij kod metody `find`.

Szukamy nazwisk klientów, dlatego skorzystamy z tej metody wyszukującej.

```
def find
  @client_workouts = ClientWorkout.find_all_by_client_name(params[:search_string])
end
```

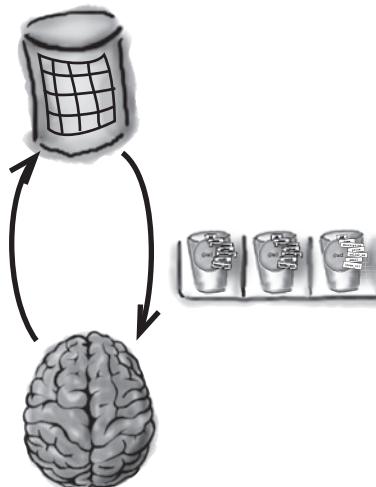
To są dane, które użytkownik wprowadził do pola wyszukiwania.

Co dalej?

Kiedy mamy już kod, który odnajdzie wszystkie rekordy pasujące do wyszukiwania, musimy wyświetlić znalezione wyniki użytkownikowi. Jak to jednak zrobić?

Musimy utworzyć szablon strony `find.html.erb` wyświetlający wyniki wyszukiwania.

Czy `ClientWorkout.client_name = params[:search_string]` ?



Niezbędny jest nam szablon strony wyświetlającej zwracane dane.

A screenshot of a web browser window titled 'ClientWorkouts: Find' with the URL 'http://localhost:3000/client_workouts/find'. The page displays a table titled 'Listing client_workouts for Lenny Goldberg' with the following data:

Trainer	Duration	Date of workout	Paid amount	Action
Clint	30	2009-07-14	25.0	Show Edit Delete
Bred	30	2009-07-19	25.0	Show Edit Delete
Sven	90	2009-08-02	75.0	Show Edit Delete
Marshall	15	2009-09-29	15.0	Show Edit Delete
Clint	30	2009-10-01	25.0	Show Edit Delete
Sara	30	2009-10-05	25.0	Show Edit Delete



Ćwiczenie

Utwórz szablon strony dla metody `find`, który wyświetli listę sesji treningowych z uwzględnieniem danych trenera, czasu trwania sesji, daty jej przeprowadzenia oraz zapłaconej kwoty:

<Trainer name> <Workout duration> <Date of the workout> <Amount paid>

Wskazówka: strona indeksująca wygenerowana przez rusztowanie jest podobna do tej, którą masz utworzyć.

Wyświetlenie klientów



Ćwiczenie
Rozwiążanie

Utwórz szablon strony dla metody `find`, który wyświetli listę sesji treningowych z uwzględnieniem danych trenera, czasu trwania sesji, daty jej przeprowadzenia oraz zapłaconej kwoty:

<Trainer name> <Workout duration> <Date of the workout> <Amount paid>

Wskazówka: strona indeksująca wygenerowana przez rusztowanie jest podobna do tej, którą masz utworzyć.

Nie martw się, jeśli Twoja odpowiedź nieco się od tego różni.

<h1>Listing client_workouts for <%= params[:search_string] %></h1>

```
<table>
  <tr>
    <th>Trainer</th>
    <th>Duration mins</th>
    <th>Date of workout</th>
    <th>Paid amount</th>
  </tr>

  <% for client_workout in @client_workouts %>
  <tr>
    <td><%= h client_workout.trainer %></td>
    <td><%= h client_workout.duration_mins %></td>
    <td><%= h client_workout.date_of_workout %></td>
    <td><%= h client_workout.paid_amount %></td>
    <td><%= link_to 'Show', client_workout %></td>
    <td><%= link_to 'Edit', edit_client_workout_path(client_workout) %></td>
    <td><%= link_to 'Destroy', client_workout, :confirm => 'Are you sure?', :method => :delete %></td>
  </tr>
  <% end %>
</table>
```

Nie istnieją
grupie pytania

P: Czy mogliśmy w tej sytuacji wykorzystać ponownie plik `index.html.erb`?

O: W tym szablonie nie ma danych klienta, więc nieco się on różni od szablonu `index.html.erb`. Ale zawsze ponowne wykorzystanie kodu tam, gdzie jest to możliwe, jest dobrym pomysłem. Tyle tylko, że w tej sytuacji by to nie zadziałało.



Jazda próbna

Funkcja wyszukiwania powinna teraz doskonale działać. Sprawdźmy to...

ClientWorkouts: index

http://localhost:3000/client_workout: ^ Q~

Lenny Goldberg

client_workouts

Duration mins	Date of workout	Paid amount	
50	2009-10-05	50.0	Show Edit Destroy
30	2009-07-14	25.0	Show Edit Destroy

ClientWorkouts: find

http://localhost:3000/client_workouts/find

Listing client_workouts for Lenny Goldberg

Trainer	Duration mins	Date of workout	Paid amount	
Clint	30	2009-07-14	25.0	Show Edit Destroy
Brad	30	2009-07-19	25.0	Show Edit Destroy
Sven	90	2009-08-02	75.0	Show Edit Destroy
Marshall	15	2009-09-29	15.0	Show Edit Destroy
Clint	30	2009-10-01	25.0	Show Edit Destroy
Sara	30	2009-10-05	25.0	Show Edit Destroy

Aplikacja naprawdę mi się podoba,
ale chciałbym móc przeszukiwać sesje
treningowe także pod kątem nazwiska trenera.
Nie mówiłem o tym wcześniej?

Sesje treningowe
Lenny'ego Goldberg'a.



 **WYSIL**
SZARE KOMÓRKI

Jak zmieni to kryteria wyszukiwania?

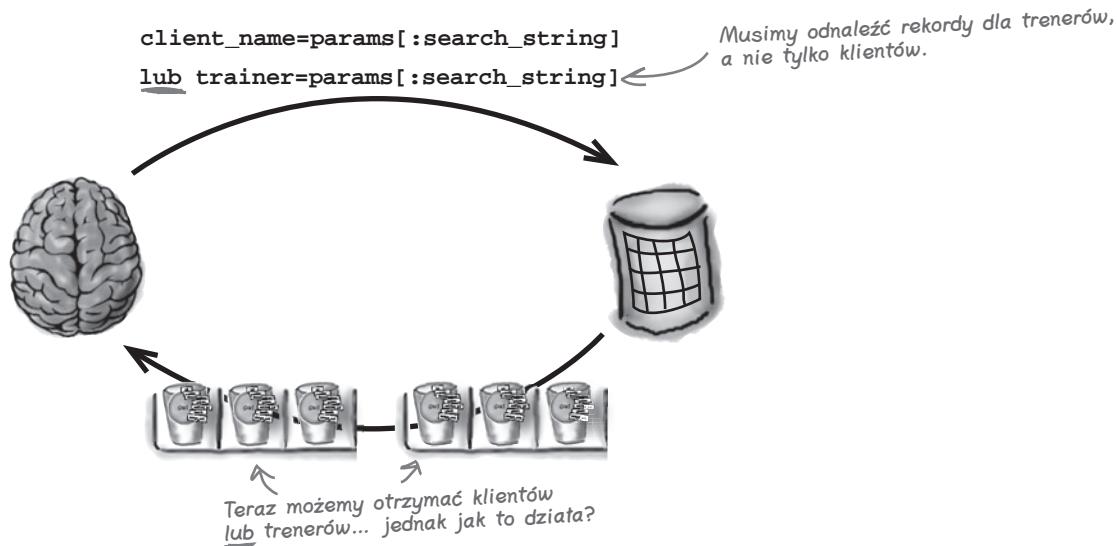
Dopasuj to LUB tamto...

Musimy dopasować albo nazwisko klienta, albo trenera

Wyszukiwanie działa poprzez odnalezienie wszystkich rekordów zawierających określone dane klienta. Jeśli jednak powinno być w stanie wyszukiwać także po nazwisku trenera, test logiczny zastosowany na każdym rekordzie musi być nieco bardziej skomplikowany. Zamiast

```
client_name = params[:search_string]
```

kryteria wyszukiwania muszą teraz być następujące:



Czy widzisz tu jakiś problem?

Obiekt modelu zawiera metodę wyszukującą dla każdego z atrybutów. Każda z metod wyszukujących zawiera prosty test, który zostaje zastosowany do rekordów bazy danych, dzięki czemu sprawdza się jedną kolumnę bazy pod kątem określonej wartości. Test stał się jednak teraz bardziej złożony. Czy istnieje zatem jakiś sposób wybrania testu, jaki metoda wyszukująca zastosuje na rekordach bazy danych?



Metody wyszukujące z bliska

Metody wyszukujące piszą zapytania do bazy danych

Jak tak naprawdę działa metoda wyszukująca? Co się dzieje, kiedy się ją wykonuje? Zadanie metody wyszukującej polega na komunikacji z bazą danych w naszym imieniu. Pamiętaj, że po wywołaniu metoda wyszukująca generuje zapytanie do bazy danych w języku nazywanym SQL (Structured Query Language).

[Uwaga od działu marketingu: pamiętać o dodaniu reklamy książki Head First SQL.]



```
ClientWorkout.find_all_by_client_name('Lenny Goldberg')
```

Zapytanie SQL

```
SELECT *
FROM client_workouts
WHERE client_name= 'Lenny Goldberg/'
```

Kryteria wyszukiwania



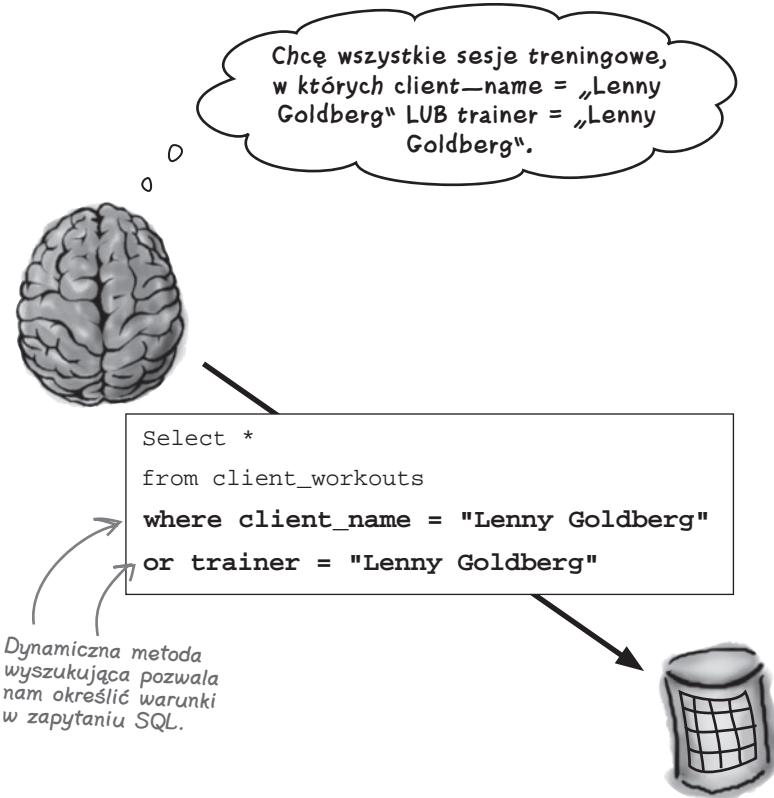
#	client_name	trainer	duration_min	date_of_workout	paid_amount	created_at	updated_at
1	Kris Elligwood	Client	30	2009-07-18	25	2009-10-06 09:11...	2009-10-06 09:11...
2	Lenny Goldberg	Client	30	2009-07-19	25	2009-10-06 09:11...	2009-10-06 09:11...
3	Lenny Goldberg	Brad	30	2009-07-19	25	2009-10-06 09:11...	2009-10-06 09:11...
4	Lenny Goldberg	Sven	90	2009-09-02	75	2009-10-06 09:11...	2009-10-06 09:11...
5	Lenny Goldberg	Marshall	15	2009-09-29	15	2009-10-06 09:11...	2009-10-06 09:11...
6	Lenny Goldberg	Client	30	2009-10-03	25	2009-10-06 09:11...	2009-10-06 09:11...
7	Lenny Goldberg	Sara	30	2009-10-05	25	2009-10-06 09:11...	2009-10-06 09:11...

Zapytanie zostaje wykorzystane przez bazę danych do odnalezienia wszystkich pasujących wierszy tabeli. Model przekształca te wiersze na obiekty modelu i zwraca je w tablicy do kontrolera.

Jeśli tak działają metody wyszukujące, co trzeba zmienić, jeżeli chcemy wyszukać klienta lub trenera?

Musimy być w stanie zmodyfikować warunki wykorzystane w zapytaniu SQL

Potrzebny jest nam jakiś sposób przekazania modelowi, że ma wygenerować zapytanie SQL wyglądające mniej więcej tak:



Warunki w zapytaniu SQL są jednak generowane przez metodę wyszukującą. Udało nam się przekazać metodzie wyszukującej łańcuch znaków (na przykład „Lenny Goldberg”), ale dotychczas nie zrobiliśmy jeszcze nic, co zmodyfikowałoby samą strukturę warunków, które stają się częścią zapytania SQL przesyłanego do bazy danych.

Czy możliwość modyfikacji parametrów zapytania SQL jest naprawdę aż tak istotna? No cóż — tak właśnie jest. Metody wyszukujące szukające pasujących wartości w określonych atrybutach są przydatne — jednak określenie warunków SQL pozwala Ci wykonać o wiele więcej. Pozwala zastąpić domyślne zachowanie metody wyszukującej i przejąć całkowitą kontrolę nad danymi, do których dostęp odbywa się przez model. Właśnie tego teraz potrzebujemy — więcej kontroli nad zapytaniem SQL.

W jaki sposób możemy zmodyfikować te warunki?

Kod SQL podaje się za pomocą :conditions

Metody wyszukujące generowane dla każdego z atrybutów są proste i łatwe w użyciu, jednak problem polega na tym, że nie są zbyt elastyczne. Często musimy wykonywać o wiele bardziej skomplikowane zapytania do bazy danych.

Z tego powodu wszystkie metody wyszukujące pozwalają na przekazanie parametru o nazwie `:conditions`, który zawiera dodatkowe warunki, jakie mają być dodane do kodu SQL generowanego przez metodę.

Oto jedno z rozwiązań, jakie można zastosować w przypadku wyszukiwania trenerów oraz klientów:

```
@client_workouts = ClientWorkout.find(:all,
:conditions=>["client_name = 'Lenny Goldberg' OR trainer = 'Lenny Goldberg'"])
↑
Parametr „:conditions” ustawiony jest na tablicę.
```

Ta wersja metody wyszukującej zwraca wszystkie rekordy, w których trener bądź klient nazywają się „Lenny Goldberg”, ale czy widzisz, co będzie tutaj problemem? Co dzieje się, kiedy chcemy wyszukać *kogoś innego* od Lenny'ego? Tak naprawdę chcemy możliwości wyszukania czegokolwiek, co zapisane jest w `params[:search_string]`. Jak to jednak zrobić?

Na szczęście Rails udostępnia sposób wykonania tego. Pozwala na użycie w warunkach parametrów, jak poniżej:

```
@client_workouts = ClientWorkout.find(:all,
:conditions=>["client_name = ? OR trainer = ?", params[:search_string], params[:search_string]])
```

Znaki `?` w pierwszym łańcuchu znaków w tablicy `:conditions` zostają zastąpione po kolei kolejnymi wartościami. Oznacza to, że metoda wyszukująca będzie teraz w stanie generować poprawną instrukcję SQL dla czegokolwiek, co znajdzie się w parametrze wyszukiwania. Właściwe rekordy zostaną zwrocone dla dowolnych danych wyszukiwanych przez trenera.

Zamiast wyszukiwać Lenny'ego Goldberg'a, możemy wyszukać cokolwiek, co znajduje się w `params[:search_string]`. Znak `?` zostaje zastąpiony tą wartością.

Ponieważ Rails wstawia te wartości parametrów do SQL za nas, zrobi to w sposób bezpieczny, unikając ataku znanego jako „SQL Injection” („wstawianie SQL”).

Czy to wszystko działa?

Znajdź mnie, proszę



Jazda próbna

Uaktualnij swój kod metody wyszukującej i przeładowuj aplikację.



To wyszukiwanie
naprawdę mi się
podoba!

ClientWorkouts: find
http://localhost:3000/client_workouts/find

Listing client_workouts for Lenny Goldberg

Trainer	Duration	mins	Date of workout	Paid amount	Action
Clint	30		2009-07-14	25.0	Show Edit Destroy
Brad	30		2009-07-19	25.0	Show Edit Destroy
Sven	90		2009-08-02	75.0	Show Edit Destroy
Marshall	15		2009-09-29	15.0	Show Edit Destroy
Clint	30		2009-10-01	25.0	Show Edit Destroy
Sara	30		2009-10-05	25.0	Show Edit Destroy

Kiedy wyszukuję
klienta, otrzymuję jego
sesje treningowe.



ClientWorkouts: find
http://localhost:3000/client_workouts/find

Listing client_workouts for Clint

Trainer	Duration	mins	Date of workout	Paid amount	Action
Clint	60		2009-10-05	50.0	Show Edit Destroy
Clint	30		2009-07-14	25.0	Show Edit Destroy
Clint	30		2009-10-01	25.0	Show Edit Destroy

Kiedy wyszukuję
samego siebie,
otrzymuję prowadzone
przez siebie sesje.



JAKI JEST MÓJ CEL?

Klub zaczął zapisywać wszystkie gry rozgrywane na zewnętrznym boisku baseballowym. Dopasuj metody wyszukujące bazy danych do tego, jak mogłyby one być użyte.

Metoda wyszukująca

```
BaseballGame.find(:all,  
:conditions=>[  
'month_no > ? and month_no < ?',  
9, 3])
```

Cel

Gry rozgrywane poza sezonem.

```
BaseballGame.find(:all,  
:conditions=>[  
'month_no > ? or month_no < ?',  
3, 9])
```

To zapytanie tak naprawdę nigdy nie zwróci, więc nie zostanie użyte.

```
BaseballGame.find(:all,  
:conditions=>[  
'month_no > ? and month_no < ?',  
3, 9])
```

Gry rozgrywane w sezonie.

```
BaseballGame.find(:all,  
:conditions=>[  
'month_no > ? or month_no < ?',  
9, 3])
```

To zapytanie zwróci wszystko, więc również nie będzie używane.

JAKI JEST MÓJ CEL?

ROZWIĄZANIE

Twoim zadaniem było dopasowanie metod wyszukujących do sposobu ich użycia.

Metoda wyszukująca

```
BaseballGame.find(:all,  
:conditions=>[  
'month_no > ? and month_no < ?',  
9, 3])
```

Cel

Gry rozgrywane poza sezonem.

```
BaseballGame.find(:all,  
:conditions=>[  
'month_no > ? or month_no < ?',  
3, 9])
```

To zapytanie tak naprawdę nigdy nic nie zwróci, więc nie zostanie użyte.

```
BaseballGame.find(:all,  
:conditions=>[  
'month_no > ? and month_no < ?',  
3, 9])
```

Gry rozgrywane w sezonie.

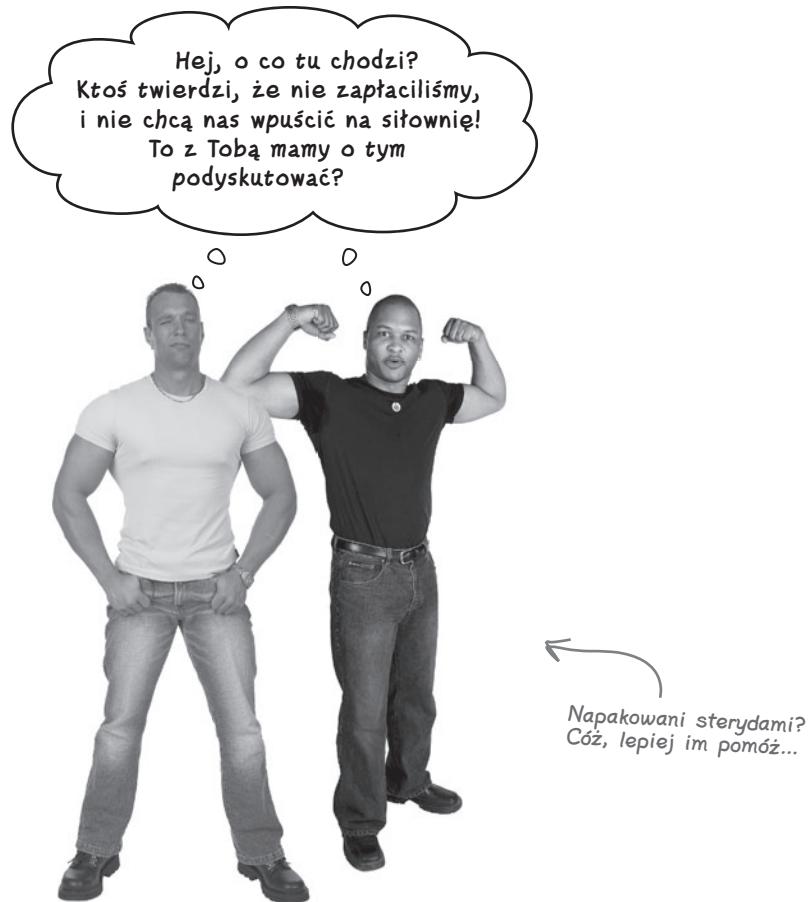
```
BaseballGame.find(:all,  
:conditions=>[  
'month_no > ? or month_no < ?',  
9, 3])
```

To zapytanie zwróci wszystko, więc również nie będzie używane.

Rozlega się pukanie do drzwi...

Akurat gdy demonstrujesz system, rozlega się pukanie do drzwi.

To kilku dorodnych panów z siłowni.



Wydaje się, że jest jakiś problem z danymi wprowadzonymi do systemu... Przygotuj się na kolejny rozdział, w którym zagłębimy się w problem z siłowni.



Niezbędnik programisty Rails

Masz za sobą rozdział 4. i teraz
do swojego niezbędnika programisty
**Rails możesz dodać umiejętność
wyboru, czy korzystać z rusztowania
oraz sprytnego wybierania
odpowiednich danych dla aplikacji.**

Narzędzia Rails

`find(:all, :conditions=<...>)` pozwala określić kod
SQL wykorzystywany do wybierania rekordów
z bazy danych.

`form_tag` generuje proste formularze, które nie
są powiązane z obiektami modelu.

Narzędzia języka Ruby

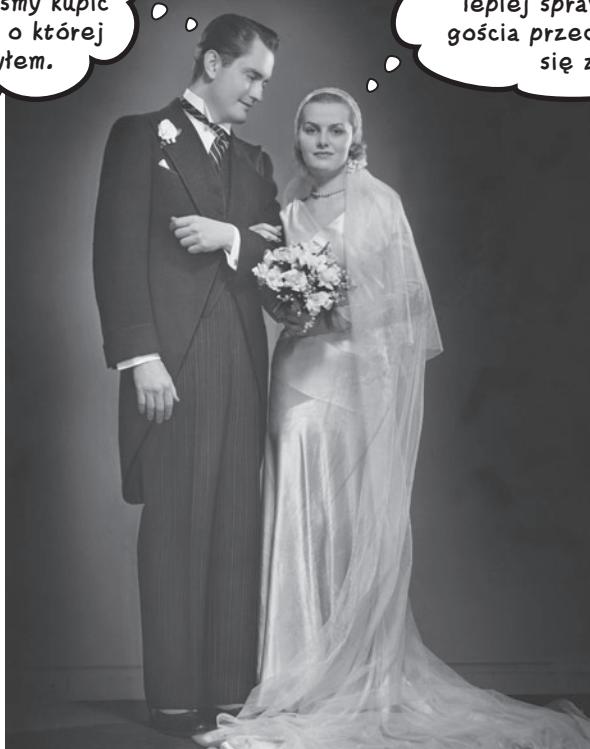
`puts <łańcuch znaków>` wyświetla łańcuch
znaków w konsoli (w której uruchomiony
jest serwer WWW).

5. Sprawdzanie poprawności danych

Zapobieganie błędom

Kochanie, gdybyśmy sprzedali wszystkie ślubne prezenty, moglibyśmy kupić tę farmę pijawek, o której zawsze marzyłem.

Powinnam była lepiej sprawdzić tego gościa przed związaniem się z nim.

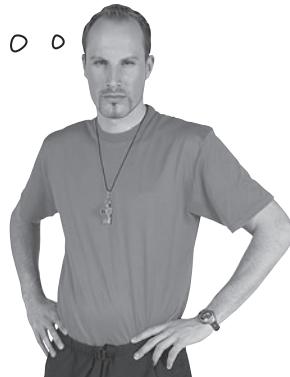


Każdy popełnia błędy... ale wielu z nich można zapobiec! Nawet przy najlepszych chęciach użytkownicy nadal będą wprowadzać niepoprawne dane do Twojej aplikacji internetowej i to Ty będziesz musiał poradzić sobie z konsekwencjami. Wyobraź sobie, co by było, gdyby istniała jakaś metoda zapobiegania występowaniu błędów. Do tego właśnie służą **validatory**. Czytaj dalej, a pokażemy Ci, jak można dodać **sprytne sprawdzanie błędów w Rails** do Twojej aplikacji internetowej, tak byś mógł **przejąć kontrolę** nad tym, jakie dane są dozwolone, a jakich należy się wystrzegać.

Uwaga — pojawiły się niepoprawne dane

W aplikacji dla trenerów osobistych wszystko wydawało się działać tak dobrze — przynajmniej do momentu, gdy pojawiło się kilku klientów siłowni. Twierdzą oni, że zapłacili należne opłaty i mają potwierdzenia tych transakcji, jednak ich płatności nie pokazują się w systemie.

Nie rozumiem. Wpisałem zapłaconą kwotę 50 dolarów, kliknąłem przycisk, ale płatność pokazuje się jako „0.0”.



Co zatem poszło nie tak?

Kliknięcie przycisku zatwierdzającego powinno było spowodować zapisanie danych w bazie, jednak coś poszło nie tak. Zamiast zapisać zapłaconą kwotę jako **\$50**, w systemie mamy **0.0**. Ale dlaczego? Przyjrzyjmy się łańcuchowi zdarzeń, które do tego doprowadziły.

- 1 Trener wpisuje „\$50” w polu płatności w widoku, a następnie kliką przycisk zatwierdzający. Do kontrolera przekazywana jest wartość „\$50”.

Paid amount
\$50
Create
Back

- 2 Kontroler otrzymuje wartość „\$50” jako kwotę transakcji i przekazuje ją do modelu.

Mówisz, że „paid-amount” ma wartość „\$50”? Przekażę to modelowi.

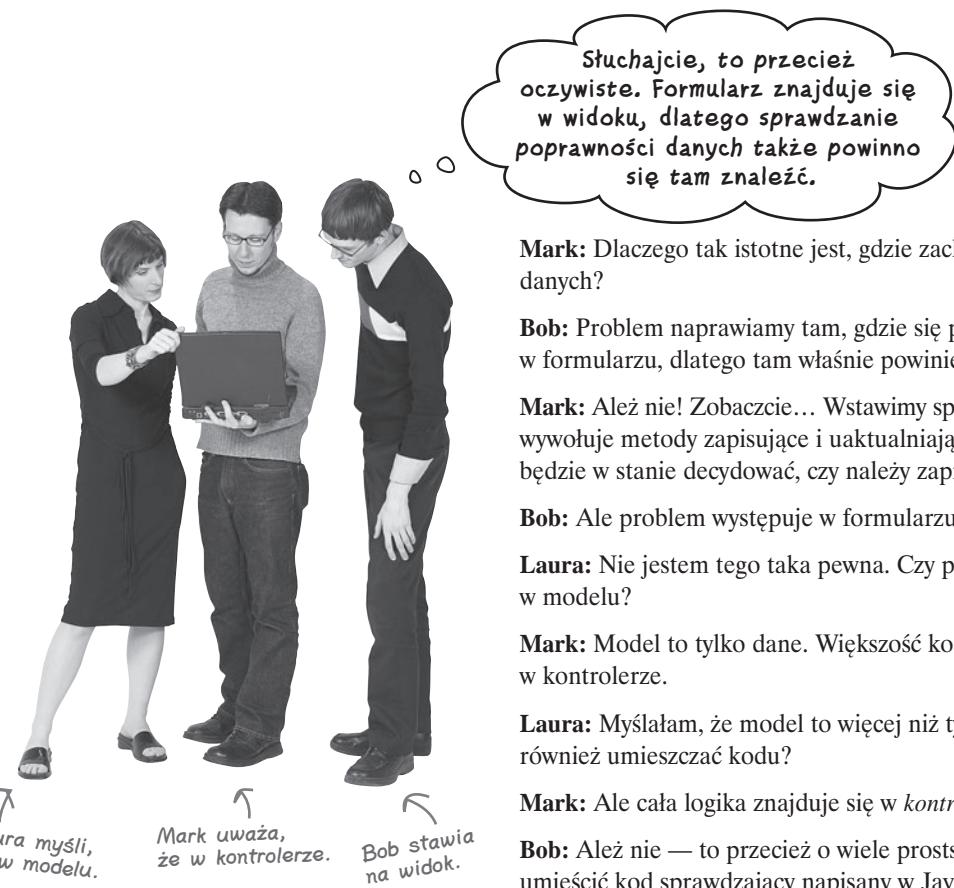


- 3 Model otrzymuje wartość „\$50”, ale pojawia się problem. Ponieważ wartość zawiera symbol „\$”, model nie jest w stanie przekształcić jej na liczbę. Zamiast tego w bazie danych zapisuje „0.0”.

„\$50” nie jest liczbą, zamiast tego zapiszę „0.0”.



Problem spowodowany został przez wprowadzenie przez trenera na stronie internetowej niewłaściwego typu danych i musimy jakoś zapobiec powtórzeniu się tej sytuacji. Musimy napisać kod sprawdzający dane formularza przed zapisaniem ich do bazy danych — *gdzie jednak powinien trafić kod tego typu?*



Mark: Dlaczego tak istotne jest, gdzie zachodzi sprawdzanie poprawności danych?

Bob: Problem naprawiamy tam, gdzie się pojawia. Błąd występuje w formularzu, dlatego tam właśnie powinien być naprawiony.

Mark: Ależ nie! Zobaczcie... Wstawimy sprawdzanie do kontrolera. Kontroler wywołuje metody zapisujące i aktualizujące dane. Sprawimy, że kontroler będzie w stanie decydować, czy należy zapisać obiekt, czy też nie.

Bob: Ale problem występuje w formularzu.

Laura: Nie jestem tego taka pewna. Czy problem tak naprawdę nie leży w modelu?

Mark: Model to tylko dane. Większość kodu, jaki piszemy, znajduje się w kontrolerze.

Laura: Myślałam, że model to więcej niż tylko dane. Nie możemy tam również umieszczać kodu?

Mark: Ale cała logika znajduje się w kontrolerze.

Bob: Ależ nie — to przecież o wiele prostsze. W kodzie formularza należy umieścić kod sprawdzający napisany w JavaScriptie.

Mark: A co jeśli użytkownik wyłączył obsługę JavaScriptu w przeglądarce?

Bob: Daj spokój, *nikt* już dzisiaj nie wyłącza JavaScriptu.

Laura: Ale czy można na tym polegać? Szczególnie gdy oczywiste jest, gdzie powinien trafić kod sprawdzający.

Mark: Do kontrolera.

Laura: Do modelu.

Zaostrz ołówek



Gdzie, Twoim zdaniem, powinno się odbywać sprawdzanie poprawności danych? Dlaczego? Napisz swoją odpowiedź tutaj.

Sprawdzanie poprawności w modelu

Zaostrz ołówek



Rozwiążanie

Gdzie, Twoim zdaniem, powinno się odbywać sprawdzanie poprawności danych? Dlaczego? Napisz swoją odpowiedź tutaj.

Dane powinny być sprawdzane w modelu, na wypadek gdyby były one zapisywane przez różne części kontrolera lub widoku.

Kod sprawdzający poprawność danych przynależy do MODELU

Problem z umieszczeniem kodu sprawdzającego poprawność danych w widoku bądź kontrolerze polega na tym, że dwa odrębne fragmenty kodu mogą próbować zapisać wartości w bazie danych. Jeśli na przykład w kontrolerze mamy metody wstawiające oraz modyfikujące dane, obie będą potrzebowaly sprawdzania poprawności tych danych. Jeśli sprawdzanie poprawności danych będzie skupione w modelu, nie ma znaczenia, w jaki sposób dane trafią do bazy — i tak zostaną one przedtem sprawdzone.

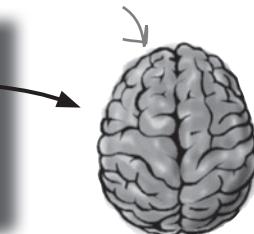
Wstawianie niepoprawnych danych

Użytkownik w formularzu do dodawania danych wprowadza niepoprawne dane.

Paid amount
\$50
Create
Back

Plik new.html.erb

W kontrolerze wywoływana jest akcja wstawiania.



Hej, myślisz, że przyjmę takie coś? To nie jest liczba!



Edycja niepoprawnych danych

Użytkownik w formularzu do edycji danych wprowadza niepoprawne dane.

Paid amount
\$50
Update
Back

Plik edit.html.erb

W kontrolerze wywoływana jest akcja edycji.



Bez względu na to, która akcja kontrolera próbuje dodać dane, model jest w stanie wychwycić błąd.

Z tego powodu na ogólny dobrym pomysłem jest *sprawdzanie poprawności danych w modelu*. W końcu jest to jeden z powodów, dla których w ogóle *mamy do dyspozycji* warstwę modelu. Model to nie tylko dane. Powodem opakowania bazy danych warstwą kodu jest to, by można było dodać sprytne sztuczki, takie jak sprawdzanie poprawności, których baza sama nie udostępnia. W jaki sposób możemy tak naprawdę dodać sprawdzanie poprawności danych do modelu?

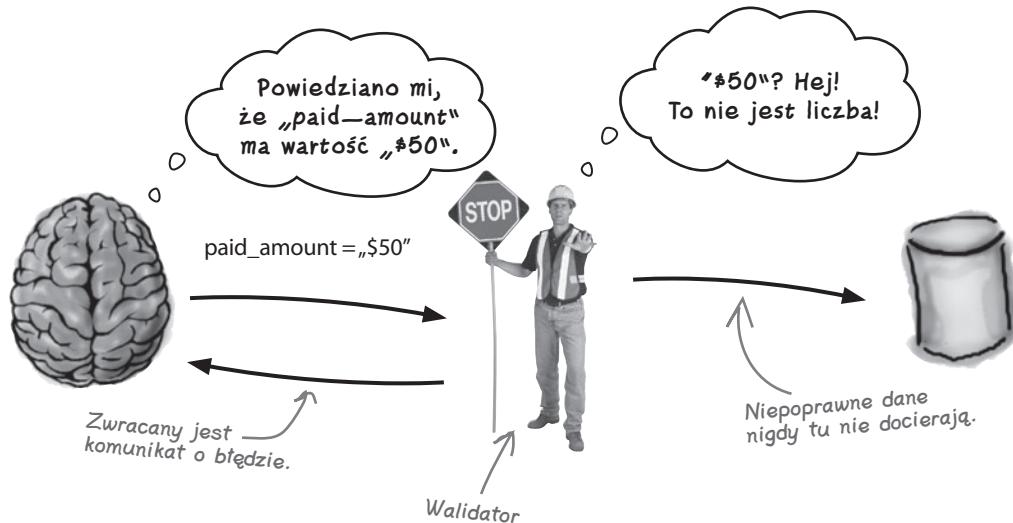
Na potrzeby prostego sprawdzania poprawności danych Rails wykorzystuje validatory

Każdy system musi wykonywać jakiś rodzaj sprawdzania danych, które są do niego wprowadzane; czasami kod sprawdzający może stać się długi i skomplikowany.

Co może zrobić Rails, by nam w tym pomóc? W końcu potrzebne nam sprawdzanie poprawności danych jest mocno dostosowane do konkretnych potrzeb, prawda?

No cóż — i tak, i nie. Zbiór reguł poprawności dla **Twoich** danych będzie najprawdopodobniej unikalny dla Twojego systemu. Ale same reguły będą raczej sprawdzać niewielki zbiór typowych błędów, takich jak **brakujące dane**, dane w **zlym formacie** czy dane **niewłaściwego typu**.

Z tego powodu Rails zawiera zbiór wbudowanych, standardowych narzędzi sprawdzających zwanych **validatorami** (ang. *validator*). Validator jest obiektem języka Ruby, który przegląda dane wprowadzane przez użytkowników i przeprowadza na nich proste sprawdzanie. Kiedy wykonywane jest to sprawdzanie? Zawsze gdy ktoś próbuje zapisać lub uaktualnić dane w bazie.



Validatory to szybki i efektywny sposób poprawy jakości danych. Pomagają filtrować to, co jest dozwolone w Twojej bazie danych, a co nie jest. W przypadkach, gdy dane są niepoprawne, validatory udostępniają nawet zbiór komunikatów o błędach, które pomogą użytkownikowi zrozumieć, co poszło nie tak.

Jak jednak działają validatory?

Jak działają validatorzy?

Przyjrzyjmy się aplikacji ClientWorkout, kiedy przechodzi sekwencję sprawdzania poprawności danych.

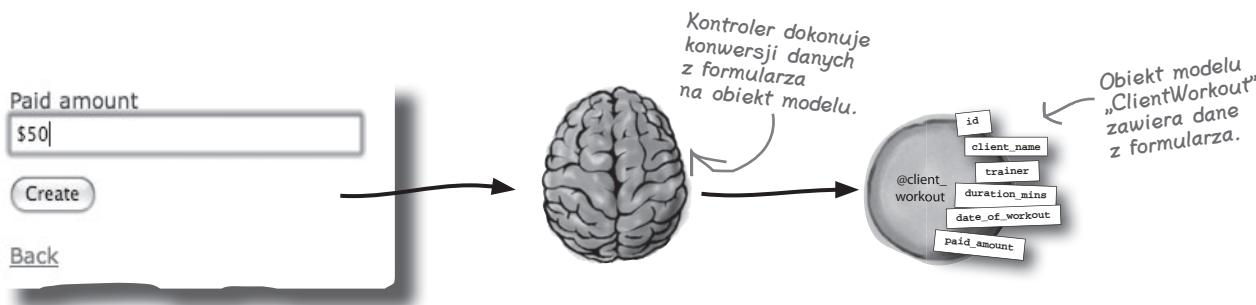
1 Użytkownik przesyła szczegóły sesji treningowej klienta.

Problem polega na tym, że pole paid_amount zawiera „\$50” zamiast po prostu „50”, a „\$50” nie da się przekonwertować na wartość liczbową.

Paid amount
\$50
Create
Back

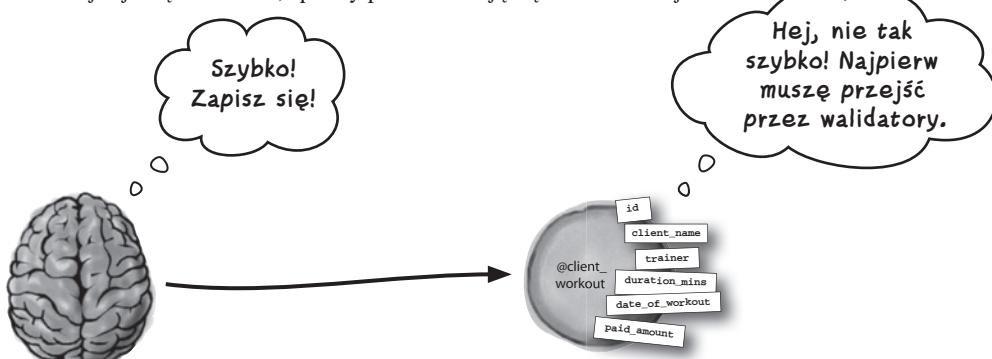
2 Kontroler dokonuje konwersji danych z formularza na obiekt modelu ClientWorkout.

Obiekt modelu przechowuje kopię danych formularza i wykorzystuje ją do wygenerowania wartości swoich atrybutów. Jeśli zapytasz obiekt o wartość atrybutu paid_amount, spróbuje on przekształcić „\$50” na liczbę, jednak ponieważ jest to niemożliwe, kontroler odpowie, że paid_amount ma wartość 0 . 0.



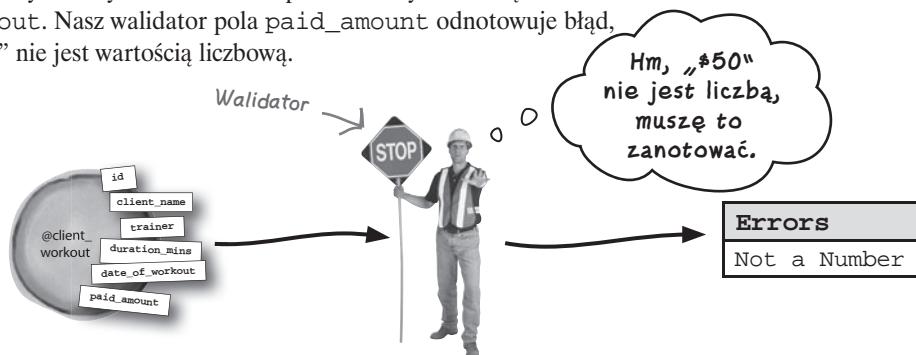
3 Kontroler próbuje zapisać obiekt.

Kontroler prosi obiekt modelu, by sam się zapisał. W normalnych warunkach obiekt zapisałby rekord samego siebie do bazy danych z paid_amount o wartości 0 . 0. Jeśli jednak w modelu znajduje się validator, sprawy przedstawiają się nieco inaczej...



4 Obiekt modelu wykonuje validator(y).

Kiedy obiekt modelu proszony jest o wstawienie czy uaktualnienie rekordu bazy danych, najpierw wykonuje swoje validatory. Validatory sprawdzają wartość powiązanych danych z formularza przechowywanych wewnątrz obiektu ClientWorkout. Nasz validator pola paid_amount odnotowuje błąd, ponieważ „\$50” nie jest wartością liczbową.



5 Obiekt modelu decyduje, czy zapisanie rekordu może się odbyć.

Dopiero po wykonaniu validatora obiekt modelu decyduje, czy można zapisać rekord w bazie danych. W jaki sposób podejmuje tę decyzję? Sprawdza, czy zostały zwrócone jakieś błędy. Validator pola paid_amount zwrócił błąd, dlatego model pomija zapisanie rekordu i informuje kontroler, że coś poszło nie tak.



6 Kontroler odsyła użytkownika do formularza.

Kod w kontrolerze wie, że coś poszło nie tak, odsyła zatem użytkownika na stronę z formularzem, by mógł on poprawić błędy.



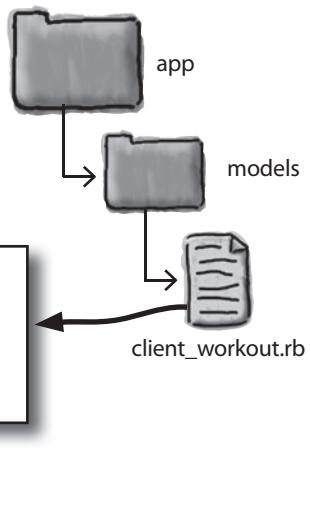
Sprawdźmy, czy coś jest liczbą

Sprawdzimy pole `:paid_amount` za pomocą validatora o nazwie `validates_numericality_of`. Validator przynależy do obiektu modelu, dlatego musimy dodać go do kodu modelu w pliku `client_workout.rb`.

Oto validator.
Sprawdza, czy
dane pole ma
wartość liczbową.

```
class ClientWorkout < ActiveRecord::Base
  validates_numericality_of :paid_amount
end
```

To pole, którego
poprawność
sprawdzamy.



Ten kod tworzy *instancję* validatora dla każdego obiektu `ClientWorkout`.

Validator potrzebuje nazwy atrybutu, jaki ma sprawdzać. Jak w przypadku prawie wszystkich nazw w języku Ruby, nazwa atrybutu podana jest w postaci *symbolu* — `:paid_amount`.

Pamiętaj, że *symbol* nieco przypomina *łańcuch znaków*. Symbole zawsze zaczynają się od znaku dwukropka (`:`) i zazwyczaj wykorzystuje się je do odniesienia się do nazw elementów, takich jak pola czy atrybuty.

Jak będzie to działało w naszej aplikacji? Powiedzmy, że kontroler ma obiekt modelu `ClientWorkout` o nazwie `@client_workout`.

Za każdym razem, gdy kontroler wywołuje `@client_workout.save` lub `@client_workout.update_attributes`, obiekt modelu wykona validator `validates_numericality_of`.

Jeśli oryginalne dane formularza wewnętrz obiektu modelu mają zapisane „\$50” w polu `:paid_amount`, validator wygeneruje błąd. Rails zauważyci błąd i zarzuci uaktualnienie bazy danych.

Tyle teorii. Zobaczmy, jak to działa.



Jazda próbna

Skoro mamy już validator, wypróbujmy działanie strony. Oto, co dzieje się, kiedy próbujemy dodać rekord z niepoprawnymi danymi w polu paid_amount:

Oto strona z nową sesją treningową, w której do pola „paid_amount” wprowadzamy wartość „\$50”.

Wartość „\$50” nie jest liczbą, dlatego wygenerowany zostaje komunikat o błędzie, sesja treningowa nie może być zapisana w bazie danych, a użytkownik odesłany zostaje do formularza w celu poprawienia błędu.

Jak to wygląda na stronie z edycją sesji? Czy validator działa również tam? Powinno tak być, ponieważ model wywołuje dokładnie ten sam validator, kiedy ktoś próbuje uaktualnić rekord w bazie danych.

Tym razem spróbujmy ze stroną do edycji sesji treningowych.

To rozwiązało problem z płatnościami klientów, ale mam także inne kłopoty...



I ktoś tu mówi o mozo wymagających klientach...

Użytkownicy pomijają niektóre pola formularzy

Niektóre osoby pozostawiają pola formularza puste. Przykładowo jeden z trenerów zapomniał wprowadzić własne nazwisko przy niektórych z dodanych sesji treningowych. Później, kiedy próbował wyszukać wszystkie prowadzone przez siebie sesje, nie był w stanie odnaleźć tych, w których zapomniał podać nazwiska.

Kiedy wyszuka się sesje treningowe Steve'a, otrzymuje się taką stronę... nie ma na niej nowej sesji z Davidem Ferrie.

The left screenshot shows a 'New client_workout' form. It has fields for 'Client name' (David Ferrie), 'Trainer' (empty), 'Duration mins' (45), 'Date of workout' (2009-11-10), and 'Paid amount' (75). A note says 'Steve dodaje sesję treningową dla Davida Ferrie...' pointing to the Trainer field. Another note says 'Steve ciągle zapomina wpisać swoje dane.' pointing to the Trainer field. The right screenshot shows a 'Listing client_workouts for Steve' page with two rows of data:

Trainer	Duration mins	Date of workout	Paid amount	Action
Steve	60	2009-11-08	100.0	Show Edit Destroy
Steve	30	2009-11-22	50.0	Show Edit Destroy

A note says '...jednak teraz ta sesja zniknęła.' pointing to the second row.

Wypełnione muszą zostać pola z danymi trenera, ale także danymi klienta. Lenny Goldberg odbył kilka sesji treningowych, w których jego nazwisko nie zostało odnotowane. Lenny normalnie uiszcza rachunek na koniec miesiąca, więc kiedy obsługa próbowała wyszukać wszystkie jego niezapłacone sesje treningowe, nie była w stanie znaleźć niektórych z nich. Trenerzy nie mogą sobie na to pozwolić!

Czy validatory mogłyby pomóc?

Dotychczas validator wykorzystaliśmy do sprawdzenia, czy wprowadzane dane są wartością liczbową. Istnieje jednak cała rodzina validatorów, które mogą robić inne rzeczy, od sprawdzania, czy wartość znajduje się w liście, do upewniania się, czy jest unikalna dla określonej kolumny tabeli.

Jak sprawdzamy obowiązkowe pola?

W Rails dostępny jest validator sprawdzający wartości w polach, których wypełnienie jest obowiązkowe. Ten validator to `validates_presence_of`:

```
validates_presence_of :nazwa_pola
```

Tutaj trafia nazwa obowiązkowego pola.

Oto kod, do którego dodane zostały te validatory:

Teraz dodaj ten kod do swojej wersji aplikacji.

```
class ClientWorkout < ActiveRecord::Base
  validates_numericality_of :paid_amount
  validates_presence_of :trainer
  validates_presence_of :client_name
end
```

Dzięki temu dane trenera oraz klienta zawsze zostaną wypełnione.

Nie istniejąca grupa pytań

P: Czy jeśli validator zwróci błąd, obiekt modelu i tak wykona pozostałe validatory?

O: Dobre pytanie — ale tak. Nawet jeśli obiekt modelu już po pierwszym niepowodzeniu będzie wiedział, że można anulować operację zapisywania, nadal wykona pozostałe validatory przed poinformowaniem kontrolera o błędach.

P: Dlaczego tak jest?

O: Wyobraź sobie, że w formularzu popełniłeś kilka błędów. Dzięki wykonaniu wszystkich validatorów obiekt modelu zapewnia to, że zobaczysz *wszystkie* komunikaty o błędach, dzięki czemu będziesz mógł je wszystkie naprawić przed ponownym przesłaniem formularza. Gdybyś widział tylko pierwszy błąd, być może musiałbyś przesyłać formularz kilka razy.

P: Dlaczego obiekt modelu przechowuje kopię danych formularza? Dlaczego nie przechowuje po prostu wartości z formularza w normalnych atrybutach?

O: Obiekt modelu musi wykonywać validatory na oryginalnych łańcuchach znaków przesłanych przez formularz. Jeśli na przykład formularz przechował wartość `paid_amount` w atrybutie liczbowym, musiałby przekształcić dane na coś w stylu `0 . 0`. To ukryłoby fakt wystąpienia problemu.

P: Kiedy zatem proszę obiekt `ClientWorkout` o `@client_workout.paid_amount`, obiekt ten nie zwraca właściwej wartości atrybutu?

O: Obiekt `@client_workout` szuka wartości przesłanej z pola formularza (`,$50"`), a następnie zwraca jej wersję liczbową. Robi to za każdym razem, gdy prosiš o `@client_workout.paid_amount`.

P: Czy właśnie dlatego zapisuje „\$50” do bazy danych jako „0.0”?

O: Tak. Jeśli nie ma validatorów, model konstruuje instrukcję języka SQL `INSERT` bądź `UPDATE` z parą atrybut-wartość obiektu modelu. Kiedy model patrzy

na wartość `@client_workout.paid_amount`, widzi `0 . 0` — i to właśnie zostaje przesłane do bazy danych.

P: Czy mogę poprosić obiekt modelu o pominięcie sprawdzania poprawności danych?

O: Tak. Jeśli wywołasz `@client_workout.save(false)`, obiekt zostanie zapisany bez wykonywania validatorów.

P: Czy mogę zmieniać komunikaty o błędach?

O: Tak — możesz dodać swoje własne komunikaty o błędach w postaci łańcucha znaków, na przykład:

```
validates_presence_of :trainer, "Where's your name?"
```

P: W jaki sposób wyświetlane są komunikaty o błędach?

O: W formularzu znajduje się znacznik o nazwie `f.error_messages`. Więcej informacji na temat tego procesu pojawi się później.



Jazda próbna

Czas uruchomić przeglądarkę i spróbować wprowadzić jakieś niepoprawne dane w formularzu do dodawania nowych sesji treningowych. Miejmy nadzieję, że nasz nowy kod validatora będzie w stanie przechwycić wszelkie problemy.

New client_workout

Client name

Pozostaw te pola puste...

Trainer

Duration mins

Date of workout

Paid amount

... a tutaj wpisz wartość nieliczbową.

Create

Back

New client_workout

3 errors prohibited this client workout from being saved

There were problems with the following fields:

- Client name can't be blank
- Paid amount is not a number
- Trainer name can't be blank

Świetnie,
wygląda na to, że udało
Ci się rozwiązać nasze
problemy z niepoprawnymi
danymi! Dziękujemy!



*JAKI JEST MÓJ CEL?

W tym systemie musieliszy sprawdzić jedynie kilka elementów, zobaczymy jednak, jakie inne validatory są dostępne. Sprawdź, czy potrafisz odgadnąć, które validatory służą do jakich celów:

`validates_length_of :pole1,
:maximum=>32`

Sprawdza, czy numer karty kredytowej wygląda jak numer karty kredytowej.

`validates_format_of :pole1,
:with=>/wyrażenie regularne/`

Sprawdza, czy masowa korespondencja nie trafia dwa razy do tej samej osoby.

`validates_uniqueness_of :pole1`

Czy grupa mięśni została poprawnie zapisana?

`validates_inclusion_of :pole1,
:in=>[wartość1, wartość2, ..., wartośćn]`

Sprawdza, czy nazwa użytkownika zmieści się w kolumnie bazy danych.

*JAKI JEST MÓJ CEL?

ROZWIĄZANIE

W tym systemie musieliszy sprawdzić jedynie kilka elementów, zobaczymy jednak, jakie inne validatory są dostępne. Sprawdź, czy potrafisz odgadnąć, które validatory służą do jakich celów:

`validates_length_of :poleł,
:maximum=>32`

Sprawdza, czy numer karty kredytowej wygląda jak numer karty kredytowej.

`validates_format_of :poleł,
:with=>/wyrażenie regularne/`

Sprawdza, czy masowa korespondencja nie trafia dwa razy do tej samej osoby.

`validates_uniqueness_of :poleł`

Czy grupa mięśni została poprawnie zapisana?

`validates_inclusion_of :poleł,
:in=>[wartość1, wartość2, ..., wartośćn]`

Sprawdza, czy nazwa użytkownika zmieści się w kolumnie bazy danych.

Validatorzy są proste i działają dobrze

Jakość danych znacznie się teraz poprawiła, a dane sesji treningowych przestały znikać w tajemniczy sposób. Księgowa jest zadowolona, ponieważ ma wgląd do danych wszystkich klientów, którzy nie uregulowali płatności, a pakerzy z siłowni nie muszą się już martwić o to, że ich płatności zaginęły.

Załatwiliśmy sprawę z klientami siłowni, a ponieważ udało się to zrobić tak szybko, uniknęliśmy nowych problemów.
Dobra robota!





CELNE SPOSTRZEŻENIA

- Walidatory definiowane są w **modelu**.
- Walidatory sprawdzają dane formularza przechowywane w **obiekcie modelu**.
- Walidatory wykonywane są **przed** operacjami wstawienia danych do bazy lub uaktualnienia ich.
- Obiekt modelu może zawierać **kilka** validatorów.
- Obiekt modelu za każdym razem wykona **wszystkie walidatory**.
- **Kontroler** powinien sprawdzać, czy operacja zapisu lub uaktualnienia powiodła się.
- Jeśli pojawią się **błędy**, użytkownik powinien zostać odesłany do formularza.
- Dostępnych jest **wiele** validatorów.

Wszystko szło tak dobrze, dopóki...



W MeBay wydarzyło się coś dziwnego

Ekipa z firmy MeBay słyszała o Twojej pracy z validatorami, więc próbowała dodać je do kodu, który dla nich napisałeś.

Nie uzyskali jednak dobrych wyników...



Cóż... chyba można powiedzieć,
że to niezadowolony klient?

Zespół z firmy MeBay dodał jedynie walidatory sprawdzające, czy wszystkie pola dla nowych ogłoszeń zostały wypełnione, a także czy pola liczbowe oraz zawierające adresy e-mail są poprawnie sformatowane.

Przyjrzyjmy się z bliska temu, co się dzieje...

Validatorzy sprawdzają, jednak nie wyświetlają błędów

Ktoś wprowadził ogłoszenie z pustą pozycją w miejscu ceny, by sprawdzić, co jest nie tak z validatorami.

To model „Ad” w aplikacji MeBay, z nowymi validatorami.

- 1 Tworzone ogłoszenie nie zawiera ceny.

The screenshot shows the 'New ad' form. The 'Name' field contains 'Toby Ziegler', 'Description' is 'Slightly damaged softball. Some fragments of glass. The result of a very long night's work.', and 'Seller' is '172'. The 'Email' field contains 'ziegler@ww.com'. The 'Price' field is empty. A callout bubble points to it with the text 'Brakuje tu ceny.'

```
class Ad < ActiveRecord::Base  
  validates_presence_of :price  
  validates_presence_of :name  
end
```

- 3 Ponieważ pojawiły się błędy, rekord nie zostaje zapisany.



- 4 Zamiast odestać użytkownika do formularza, system odsyła go do strony indeksującej znajdującej się pod adresem /ads/.

- 2 Validator validates_presence_of :price nie pozwala na utworzenie ogłoszenia.



The screenshot shows the 'All Ads' index page. It lists various items with their delete options: Typewriter [Delete], Football [Delete], Moosehead [Delete], Desk [Delete], Dog [Delete], Apple Newton [Delete], Sinclair C5 [Delete], Edsel [Delete], and Leather boot [Delete]. The new entry from step 1 is missing from the list.

Co zatem poszło nie tak?

Zaostrz ołówek



- Jeśli ogłoszenie zostaje wprowadzone poprawnie, kolejną stroną widzaną przez użytkownika jest strona nowego ogłoszenia. Dlaczego Rails wyświetla stronę indeksującą znajdująca się pod adresem `/ads/` zamiast strony nowego ogłoszenia z błędami?

2. Walidatory działały dobrze w aplikacji klubu fitness. Jak sądzisz, dlaczego nie zadziaływały w aplikacji MeBay?

Zaostrz ołówek

Rozwiązywanie

- Jeśli ogłoszenie zostaje wprowadzone poprawnie, kolejną stroną widzaną przez użytkownika jest strona nowego ogłoszenia. Dlaczego Rails wyświetla stronę indeksującą znajdująca się pod adresem /ads/?

Identyfikator rekordu jest pusty, dopóki nie zostanie zapisany.

Normalnie aplikacja przekierowuje użytkownika do:

/ads/:id

Ponieważ ogłoszenie nie zostało zapisane, użytkownik zostanie przekierowany do:

/ads/:puste

- Validatory działały dobrze w aplikacji klubu fitness. Jak sądzisz, dlaczego nie zadziałały w aplikacji MeBay?

Aplikacja klubu utworzona została za pomocą rusztowania, natomiast aplikacja MeBay stworzona została ręcznie. Kod rusztowania sprawdza błędy i je wyświetla, natomiast nasz własny kod tego nie robi!

ClientWorkouts: index					
Client name	Trainer	Duration	mins	Date of workout	Paid amount
Kirk Sogwood	Clint	60		2009-10-05	50.0
Lenny Goldberg	Clint	30		2009-07-14	25.0
Lenny Goldberg	Brad	30		2009-07-19	25.0
Lenny Goldberg	Mark	60		2009-09-29	75.0
Lenny Goldberg	MarkHall	15		2009-09-29	15.0
Lenny Goldberg	Clint	30		2009-10-01	25.0
Lenny Goldberg	Sara	30		2009-10-05	25.0
Jack Ruby	Steve	60		2009-11-08	100.0
Clem Bertrand	Steve	30		2009-11-22	50.0



Ads: indexAll Ads	
MeBay	
All Ads	
• Zajęcia (Deleted)	
• Konsultacje (Deleted)	
• Rezerwacje (Deleted)	
• Wyciągi (Deleted)	
• Wykup rezerwacji (Deleted)	
• Wykonanie rezerwacji (Deleted)	
• Wykonanie wyciągu (Deleted)	
• Wykonanie zajęć (Deleted)	
• Wykonanie konsultacji (Deleted)	

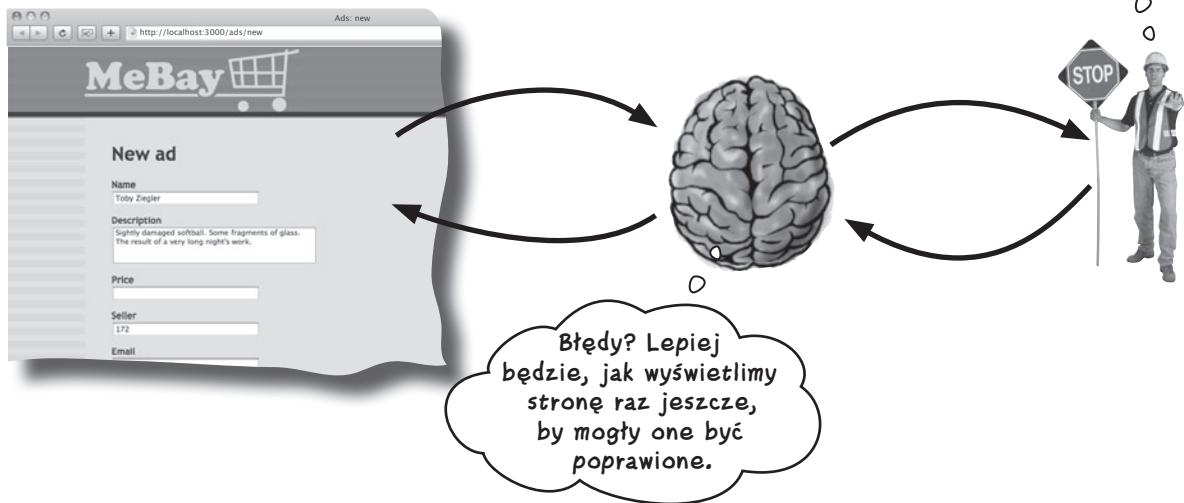


Jeśli tworzysz własne strony, musisz także pisać własny kod komunikatów o błędach

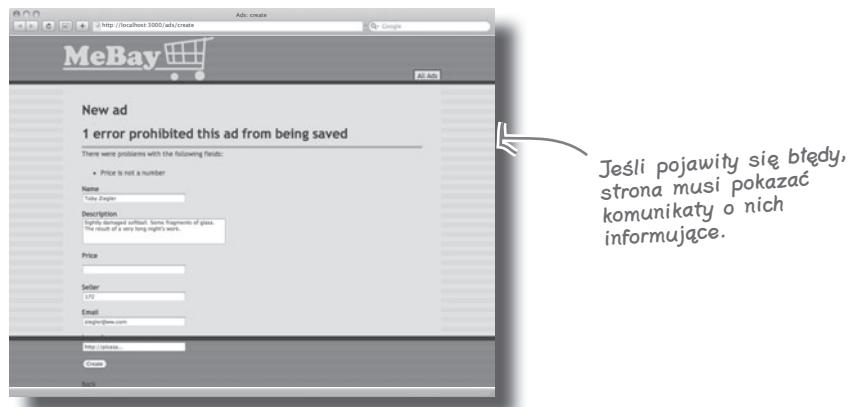
Kiedy tworzysz część aplikacji za pomocą rusztowania, Rails generuje kod potrzebny do obsługi błędów. Jeśli jednak tworzysz ten kod ręcznie, jesteś w dużej mierze zdany na samego siebie. Musimy zatem zmodyfikować kod aplikacji MeBay tak, by obsługiwał on błędy.

Kod ten będzie musiał robić dwie rzeczy:

- 1 Jeśli pojawi się błąd, system będzie musiał ponownie wyświetlić stronę, na której pojawił się błąd.



- 2 Strona z formularzem będzie musiała wyświetlać wszystkie błędy wygenerowane przez walidatory.



Co jest pierwszą rzeczą, jaką musi robić aplikacja?

Kontroler musi wiedzieć, czy wystąpił błąd

Jeśli użytkownik wprowadzi do formularza niewłaściwe dane, Rails musi odesłać użytkownika z powrotem do formularza z błędem. Takie przechodzenie między stronami obsługiwane jest przez kontroler. Pamiętaj, że kontroler odpowiedzialny jest za to, jakie dane są wczytywane oraz zapisywane, a także które strony są wyświetlane.

Co musi zrobić kod kontrolera, by móc obsługiwać błędy w aplikacji MeBay?

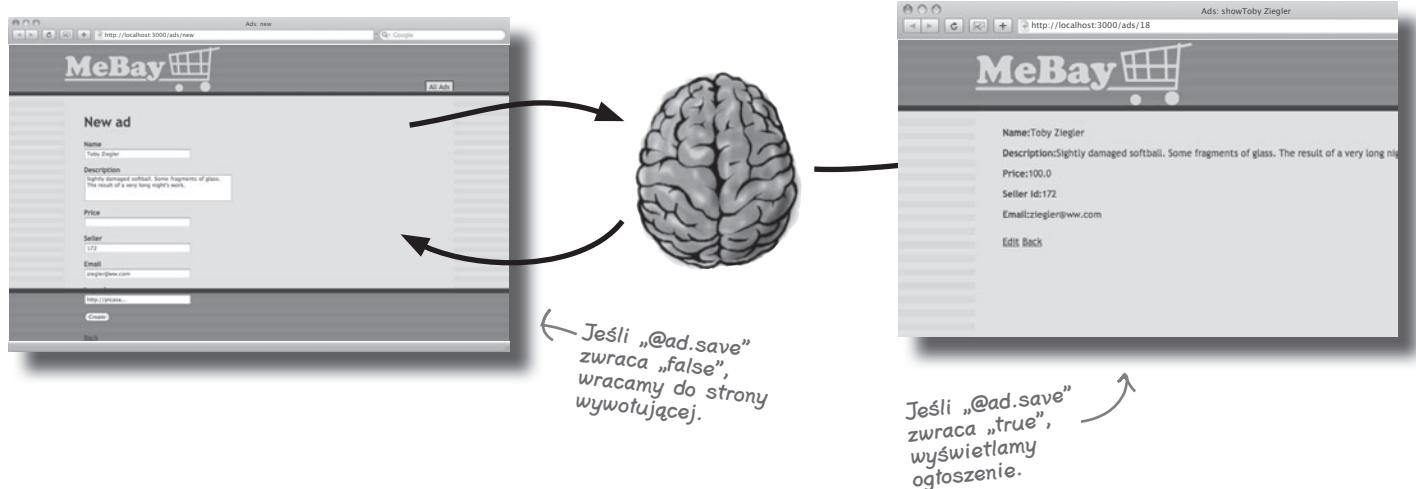
Oto, co robi obecnie aplikacja, kiedy przesyłane jest nowe ogłoszenie:

```
def create
  @ad = Ad.new(params[:ad])
  @ad.save
  redirect_to "/ads/#{@ad.id}"
end
```

Jeśli wystąpi błąd,
nie chcemy tego
przekierowania.

Kod zawsze robi to samo — próbuje zapisać ogłoszenie w bazie danych, a następnie przejść na stronę wyświetlającą dane. Nie ma obecnie znaczenia, czy zapis *powiedzie się*... i tu właśnie leży problem.

Skąd mamy wiedzieć, czy metoda zapisu się powiodła? Cóż, w Ruby każde polecenie **zwraca pewną wartość**. Jeśli pojawił się problem w zapisie ogłoszenia, polecenie `@ad.save` zwróci wartość **false**. Wartość zwracaną z polecenia `@ad.save` wykorzystujemy do ustalenia, czy powinniśmy **ponownie wyświetlić stronę**, czy też wyświetlić zapisane ogłoszenie.



Żeby to jednak zrobić, musimy poznać nieco lepiej język Ruby...

Łamigłówka



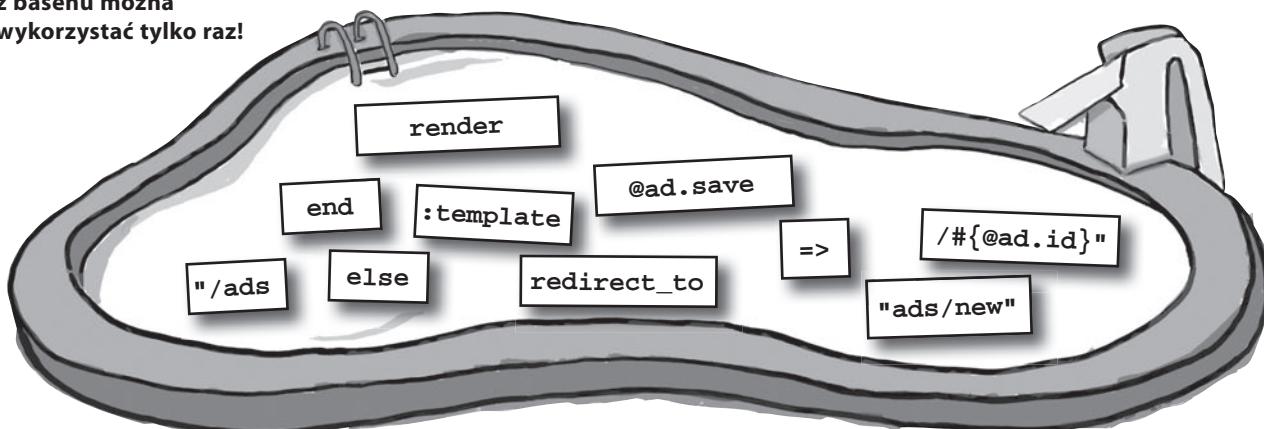
Kod potrzebny do poprawienia przechodzenia między stronami podany jest poniżej. Wykorzystuje element języka Ruby, z którym jeszcze się nie spotkaliśmy — polecenie `if`. Twoje zadanie polega na użyciu fragmentów kodu z basenu i utworzeniu polecenia niezbędnego do poprawienia przechodzenia między stronami.

`if`

Jeśli zapis powiedzie się, przekieruj do wyświetlenia nowego ogłoszenia.

Jeśli się nie powiedzie, wyświetl ponownie szablon „ads/new”.

Uwaga: każdy element z basenu można wykorzystać tylko raz!



Łamigłówka: Rozwiążanie



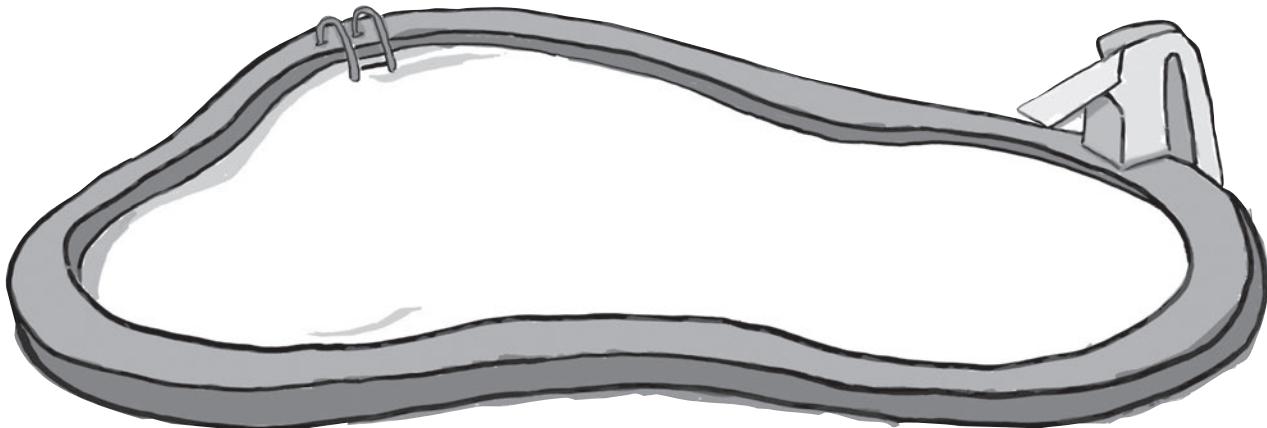
Kod potrzebny do poprawienia przechodzenia między stronami podany jest poniżej. Wykorzystuje element języka Ruby, z którą jeszcze się nie spotkaliśmy — polecenie `!f`. Twoje zadanie polega na użyciu fragmentów kodu z basenu i utworzeniu polecenia niezbędnego do poprawienia przechodzenia między stronami.

```
if @ad.save
  redirect_to "/ads/#{@ad.id}"
else
  render :template => "ads/new"
end
```

Jeśli zapis powiedzie się, przekieruj do wyświetlenia nowego ogłoszenia.

Rails pozwala wywołać metodę „render” z parametrem „:action”, dlatego można to przepisać jako: render :action=>'new'.

Jeśli się nie powiedzie, wyświetl ponownie szablon „ads/new”.





Jazda próbna

Uaktualnij kod aplikacji MeBay tak, by zawierała ona nasz własny kod sterujący przemieszczaniem się między stronami. Co się teraz dzieje, jeśli nie podamy wartości w polu z ceną?

Nieżytym pomysłem jest ponowne pobranie aplikacji MeBay z serwera FTP Helionu, by otrzymać wszystkie uaktualnienia.

Wróciliśmy do tego samego formularza, ale błędy nie zostały wyświetcone.

Ads: new

MeBay

New ad

Name: Toby Ziegler

Description:
Slightly damaged softball. Some fragments of glass.
The result of a very long night's work.

Price:

Seller: 172

Email: ziegler@www.com

<http://picasa...>

Create

Back

Ads: new

MeBay

New ad

Name: Toby Ziegler

Description:
Slightly damaged softball. Some fragments of glass.
The result of a very long night's work.

Price:

Seller: 172

Email: ziegler@www.com

<http://picasa...>

Create

Back

Kiedy walidatory zostają wykonane, kontroler zauważa, że pojawił się problem z danymi formularza, przez co ponownie wyświetla formularz, by użytkownik mógł naprawić błąd. Model nie zapisuje rekordu, a kontroler działa teraz poprawnie.

Tylko... Czy czegoś tutaj nie brakuje?

Nadal musimy wyświetlić komunikaty o błędach!

Świetnie, że aplikacja ponownie wyświetla formularz — ale **co potem?**

By naprawić problemy z formularzem, użytkownik musi wiedzieć, **co poszło nie tak.**

Niezbędne są mu *komunikaty o błędach*, pokazujące:

- ➊ W których **polach** wystąpił problem.
- ➋ Jaki konkretnie był to **problem**.

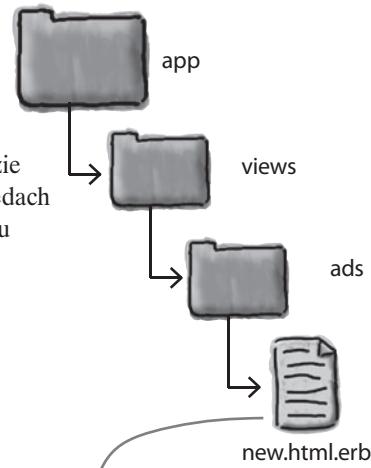
Za każdym razem, gdy jeden z validatorów *zwraca błąd*, komunikat o tym błędzie przechowany zostaje w **modelu**. My chcemy jednak wyświetlać komunikaty o błędach w **widoku**. Oznacza to, że komunikaty muszą zostać przetransferowane z modelu do widoku.

Jaka część **interfejsu** jest ściśle związana z obiektem modelu?

Formularz! A obiekt formularza ma specjalną metodę, która jest w stanie wygenerować blok błędu. Metoda ta nosi nazwę **error_messages**:

Komunikaty o błędach
generowane są przez
metodę obiektu
formularza o nazwie
„error_messages”.

```
<h1>New ad</h1>
<% form_for(@ad, :url=>{:action=>'create'}) do |f| %>
  <%= f.error_messages %>
  <p><b>Name</b><br /><%= f.text_field :name %></p>
  <p><b>Description</b><br /><%= f.text_area :description %></p>
  <p><b>Price</b><br /><%= f.text_field :price %></p>
  <p><b>Seller</b><br /><%= f.text_field :seller_id %></p>
  <p><b>Email</b><br /><%= f.text_field :email %></p>
  <p><b>Img url</b><br /><%= f.text_field :img_url %></p>
<% end %>
```





Jazda próbna

Zespół MeBay jest teraz o wiele bardziej zadowolony. Dzięki walidacji dane są poprawne, a użytkownicy od razu widzą, jakiego typu problemy napotkali i w jaki sposób można je naprawić.

Tym razem
wróciliśmy
do tego samego
formularza, ale
błędy zostaną
wyświetlone. ☺

New ad

Name

Description

Price

Seller

Email

Create

Back

Ads: new

MeBay

All Ads

New ad

1 error prohibited this ad from being saved

There were problems with the following fields:

- Price can't be blank

Name

Description

Czas opublikować nową wersję kodu użytkownikom i zobaczyć, co oni o niej sądzą.

System MeBay wygląda przepięknie

Teraz, gdy system poprawnie informuje o błędach, zespół MeBay dodaje coraz więcej validatorów. Kontroler sprawdza wystąpienie błędów i informuje o ewentualnych problemach. Wkrótce dane w systemie są naprawdę wysokiej jakości, a liczba błędów drastycznie spada.



Pozostaje tylko jedna rzecz do zrobienia. Validatory zapobiegają pojawiению się poważnych problemów z danymi, jednak błędy wyświetlane są jedynie wtedy, gdy publikowane jest **nowe ogłoszenie**.

Nie są jednak zgłaszane, kiedy ogłoszenia są edytowane . . .



Ćwiczenie

Ten kod wykonywany jest, gdy do serwera przesyłana jest strona z edycją ogłoszenia.

```
def update
    @ad = Ad.find(params[:id])
    @ad.update_attributes(params[:ad])
    redirect_to "/ads/#{@ad.id}"
end
```

Zwraca „true”, jeśli uaktualnienie powiodło się.

Popraw ten kod, tak by właściwie reagował na pojawienie się błędów.

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Podaj nazwę pliku, w którym trzeba dodać kod wyświetlający komunikaty o błędach.

.....

Obsługa błędów pojawiających się także w czasie edycji



Ćwiczenie
Rozwiązanie

Ten kod wykonywany jest, gdy do serwera przesyłana jest strona z edycją ogłoszenia.

```
def update
  @ad = Ad.find(params[:id])
  @ad.update_attributes(params[:ad])
  redirect_to "/ads/#{@ad.id}"
end
```

Zwraca „true”, jeśli uaktualnienie powiodło się.

Popraw ten kod, tak by właściwie reagował na pojawienie się błędów.

```
def update
  @ad = Ad.find(params[:id])
  if @ad.update_attributes(params[:ad])
    redirect_to "/ads/#{@ad.id}"
  else
    render :template=>"/ads/edit"
  end
end
```

Mogłeś także użyć:
render :action=>:edit

Podaj nazwę pliku, w którym trzeba dodać kod wyświetlający komunikaty o błędach.

app/views/ads/edit.html.erb

```
<h1>Editing <%= @ad.name %></h1>
<% form_for(@ad,:url=>{:action=>'update'}) do |f| %>
  <%= f.error_messages %>
  <p><b>Name</b><br /><%= f.text_field :name %></p>
```



Niezbędnik programisty Rails

Masz za sobą rozdział 5. i teraz
do swojego niezbędnika programisty
Rails możesz dodać umiejętność
wykorzystywania validatorów.

Narzędzia Rails

`validates_length_of :pole1, :maximum=>32` sprawdza, czy pole nie jest dłuższe niż 32 znaki.

`validates_format_of :pole1, :with=>/wyrażenie regularne/` sprawdza, czy pole pasuje do wyrażenia regularnego.

`validates_uniqueness_of :pole1` sprawdza, czy żaden inny rekord tabeli nie ma tej samej wartości dla pola „pole1”.

`validates_inclusion_of :pole1, :in=>[wartość1, wartość2, ..., wartośćn]` sprawdza, czy pole zawiera jedną z podanych wartości.

`f.error_messages` wyświetla błędy w formularzu.

Metody „save” oraz „update_attributes” wywołane na obiekcie modelu zwracają „true”, jeśli zadziałaty, i „false”, jeśli nie.

`render :template=>"a/template"` generuje dane wyjściowe za pomocą pliku `app/views/a/template.html.erb`.

`render :action=>'new'` generuje szablon dla akcji „new”.

6. Tworzenie połączeń

Łączenie wszystkiego razem

Weź kilka sprawdzonych składników, wymieszaj je ze sobą, a otrzymasz coś, co smakuje po prostu wyborne.



Niektóre rzeczy lepsze są razem niż osobno. Posmakowałeś zatem niektórych kluczowych składników Rails. Tworzyłeś całe aplikacje internetowe, a także brałeś to, co wygenerowała platforma Rails, i przystosowywałeś do swoich potrzeb. W rawdziwym świecie życie może jednak być bardziej skomplikowane. Czytaj dalej... czas zacząć budować wielofunkcyjne strony internetowe! I nie tylko to — czas zacząć sobie radzić ze skomplikowanymi powiązaniami między danymi, a także przejąć kontrolę nad danymi, pisząc własne walidatory.

Linie Coconut Airways potrzebują nowego systemu rezerwacji

Nie istnieje lepszy sposób podróżowania pomiędzy wyspami niż hydroplan, a linia Coconut Airways posiada całą flotę tych maszyn. Oferuje wycieczki krajoznawcze i inne wyprawy, a także wygodne kursy wahadłowe pomiędzy wszystkimi lokalnymi wyspami. Usługa ta jest popularna zarówno wśród turystów, jak i lokalnej ludności.



Zapotrzebowanie na loty tej linii jest bardzo duże, dlatego potrzebny jest internetowy system rezerwacyjny, który odciąży pracowników. System musi być w stanie zarządzać rezerwacjami lotów oraz miejsc. Oto dane, jakie trzeba będzie przechowywać:

Flight (Lot)	
id (identyfikator)	integer
departure (odlot)	datetime
arrival (przylot)	datetime
destination (kierunek)	string
baggage_allowance (limit bagażu)	decimal
capacity (liczba miejsc)	integer

Maksymalny limit bagażu w funtach.

Oto informacje o locie...

Pamiętaj: Rails automatycznie dodaje do każdej tabeli kolumnę o nazwie „id”.

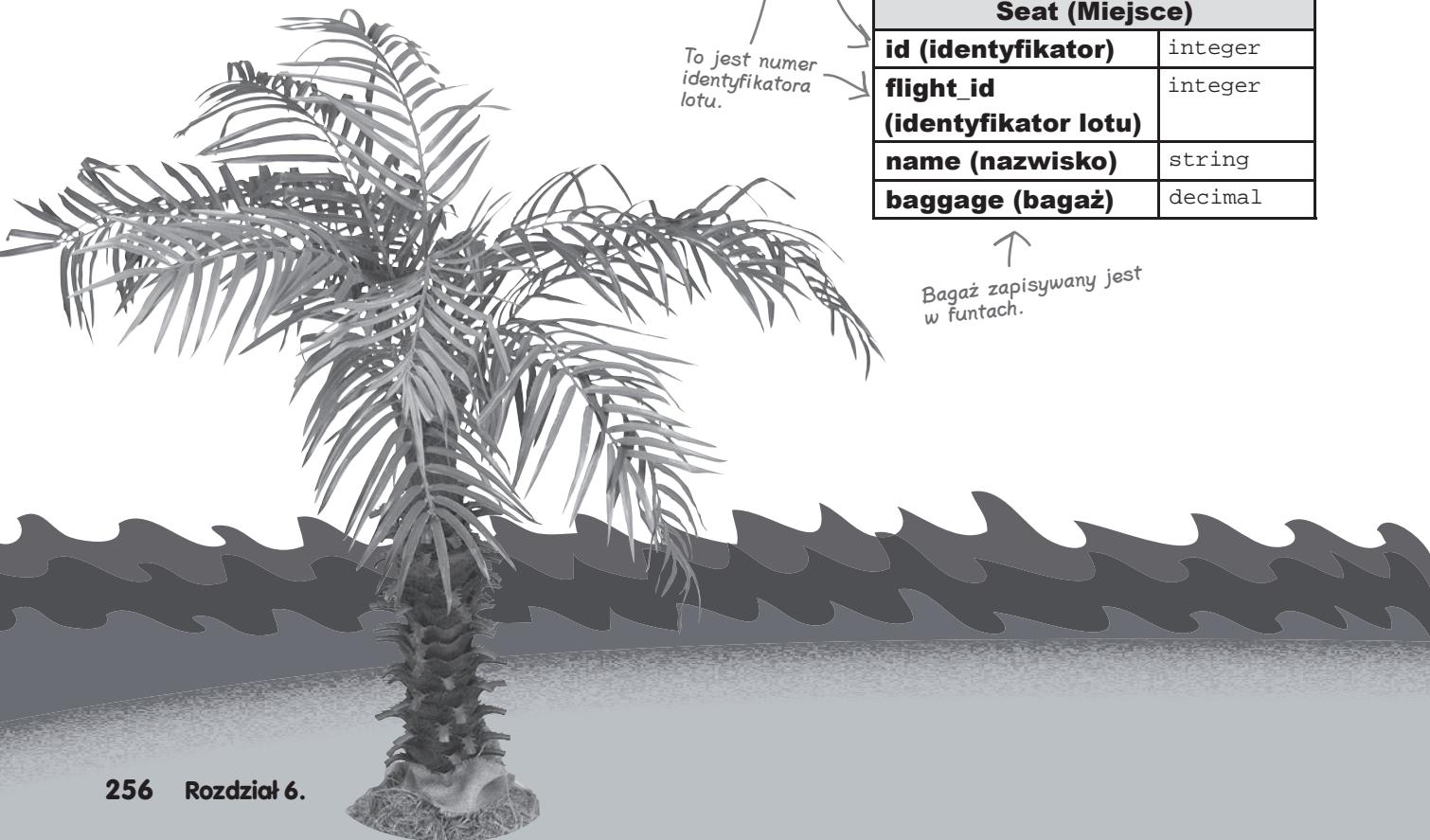
To jest identyfikator miejsca.

... a to rezerwacja miejsca.

Seat (Miejsce)	
id (identyfikator)	integer
flight_id (identyfikator lotu)	integer
name (nazwisko)	string
baggage (bagaż)	decimal

To jest numer identyfikatora lotu.

↑
Bagaż zapisywany jest w funtach.



Zaostrz ołówek



Jak będą brzmiały instrukcje służące do:

1. Utworzenia aplikacji o nazwie coconut?

.....
.....
.....

2. Utworzenia rusztowania danych lotu?

.....
.....
.....

3. Utworzenia rusztowania danych rezerwacji miejsca?

.....
.....
.....

4. W czym leży problem z utworzeniem danych lotów oraz miejsc za pomocą rusztowania?

.....
.....
.....

Połączenie tabel

Zaostrz ołówek



Rozwiążanie

Jak będą brzmiały instrukcje służące do:

1. Utworzenia aplikacji o nazwie coconut?

```
rails coconut
```

W rusztowaniu nie musisz wspominać o kolumnach "id". Zostaną one dodane automatycznie.

2. Utworzenia rusztowania danych lotu?

```
ruby script/generate scaffold flight departure:datetime arrival:datetime  
destination:string baggage_allowance:decimal capacity:integer
```

3. Utworzenia rusztowania danych rezerwacji miejsca?

```
ruby script/generate scaffold seat flight_id:integer name:string baggage:decimal
```

Pamiętaj: by utworzyć tabele, będziesz musiał użyć polecenia "rake db:migrate"!

4. W czym leży problem z utworzeniem danych lotów oraz miejsc za pomocą rusztowania?

Utworzenie rusztowania dla danych lotów oraz miejsc generuje jeden zestaw stron dla lotów i drugi dla miejsc.

Nie łączy ich ze sobą.

Chcemy widzieć loty i rezerwacje miejsc razem

Gdybyśmy po prostu utworzyli rusztowanie i nie dostosowali aplikacji do swoich potrzeb, trudno byłoby z niej korzystać. By zarezerwować miejsce, użytkownik musiałby odszukać identyfikator lotu z jego adresu URL:

Oto informacje o locie.

Departure: 2009-11-11 11:30:00 UTC
Arrival: 2009-11-11 12:15:00 UTC
Destination: St Cuthberts Island
Baggage allowance: 30.0
Capacity: 12
[Edit](#) | [Back](#)

New seat

Flight:
Name:
Baggage:
[Create](#) | [Back](#)

By zarezerwować miejsce, użytkownik musi odszukać identyfikator lotu.

Musimy wyświetlić lot łącznie z jego rezerwacją miejsc.

Zobaczmy, co daje nam rusztowanie dla miejsc

Strona lotu powinna wyglądać mniej więcej tak:

The screenshot shows a flight booking interface. At the top, it displays flight details: Departure: 2009-11-11 11:30:00 UTC, Arrival: 2009-11-11 12:16:00 UTC, Destination: St. Cuthbert's Island, Baggage allowance: 30.0, Capacity: 12. Below this, a section titled "Listing seats" shows a table of reserved seats:

Name	Baggage	Show	Edit	Destroy
Brad Sturgeon	22.0	Show	Edit	Destroy
Kirk Avery	15.0	Show	Edit	Destroy
Drew Burdine	18.0	Show	Edit	Destroy
James Blake	19.0	Show	Edit	Destroy
Giffon Brillat	25.0	Show	Edit	Destroy
Ted Brinkley	19.0	Show	Edit	Destroy
Jesse Carr	15.0	Show	Edit	Destroy
Jack Cahey	18.0	Show	Edit	Destroy

Below the table is a "New seat" form with fields for Name and Baggage, and buttons for Create, Back, Edit, and Back.

Annotations on the left side of the screenshot:

- "Oto lista rezerwacji miejsc." (Here is the list of reservations) points to the table.
- "To formularz do wprowadzenia nowej rezerwacji miejsca." (This is the form for entering a new seat reservation) points to the "New seat" form.

Zobaczmy, jak to wygląda w porównaniu ze stronami miejsc wygenerowanymi przez rusztowanie:

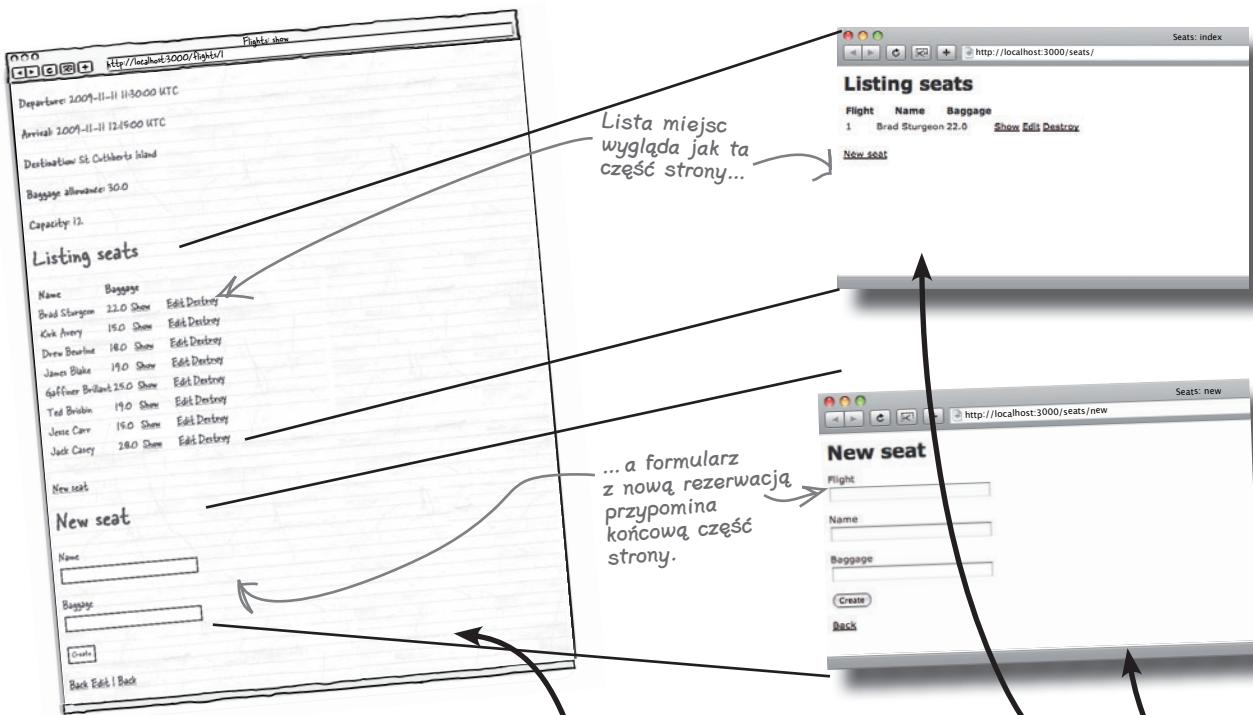
The diagram shows four browser windows illustrating generated views:

- Strona index.html.erb**: Shows a list of seats with columns for Flight, Name, and Baggage. A specific row for "Brad Sturgeon" has "Edit" and "Destroy" links.
- Strona show.html.erb**: Shows a detailed view for a single seat ("Flight: 1, Name: Brad Sturgeon, Baggage: 22.0") with "Edit" and "Back" links.
- Strona edit.html.erb**: Shows an edit form for an existing seat, pre-filled with "Flight: 1, Name: Sturgeon, Baggage: 22". It includes "Update", "Show", and "Back" buttons.
- Strona new.html.erb**: Shows a new seat creation form with fields for Flight, Name, and Baggage, and "Create" and "Back" buttons.

Czy któraś z tych stron może nam pomóc wygenerować stronę lotu?

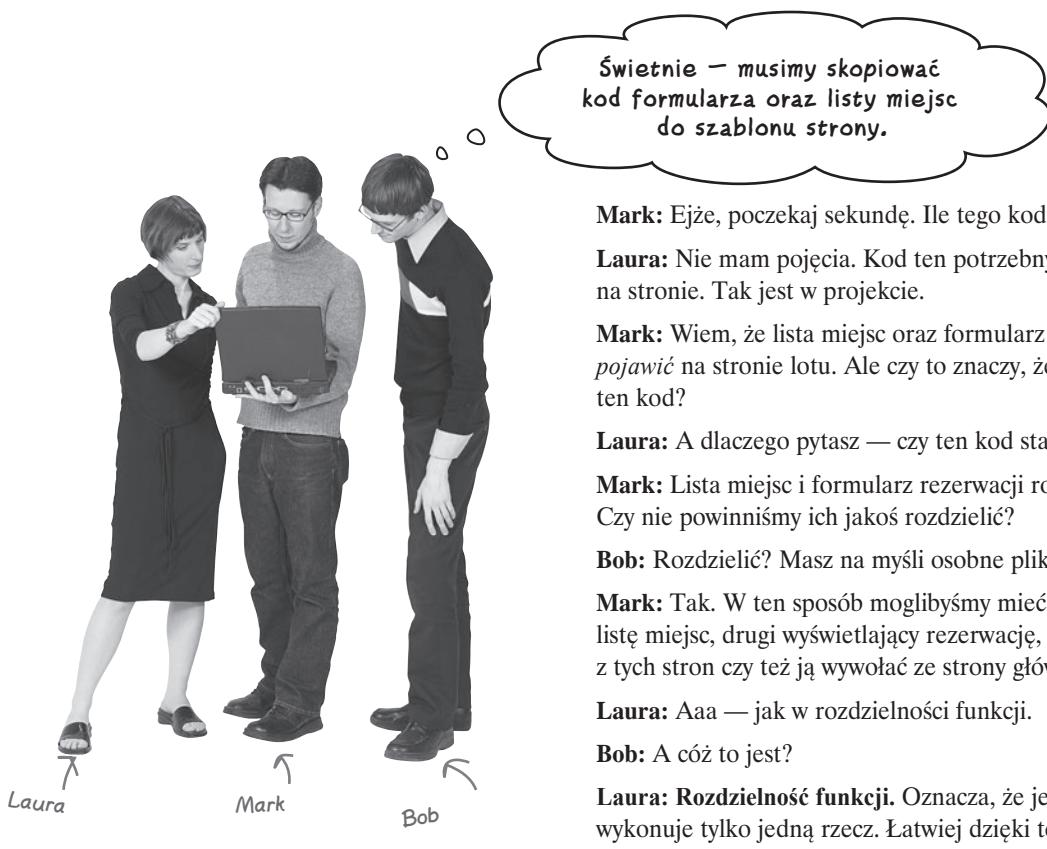
Na stronie lotu musi się znaleźć formularz rezerwacji oraz lista miejsc

Dwie z wygenerowanych stron wyglądają dość podobnie do tego, co chcielibyśmy mieć na stronie lotu — lista miejsc oraz formularz rezerwacji. Środkowa część strony lotu wygląda podobnie do listy miejsc, natomiast formularz rezerwacji przypomina końcową część strony:



Strona pokazująca lot musi zatem zawierać kod widoku przypominający stronę indeksującą miejsc, a także formularz nowej rezerwacji miejsca.

Czy powinniśmy po prostu skopiować kod z każdego formularza do strony lotu?



Mark: Ejże, poczekaj sekundę. Ile tego kodu jest?

Laura: Nie mam pojęcia. Kod ten potrzebny jest nam jednak na stronie. Tak jest w projekcie.

Mark: Wiem, że lista miejsc oraz formularz rezerwacji muszą się pojawić na stronie lotu. Ale czy to znaczy, że musimy tam mieć ten kod?

Laura: A dlaczego pytasz — czy ten kod stanowi jakiś problem?

Mark: Lista miejsc i formularz rezerwacji robią zupełnie inne rzeczy. Czy nie powinniśmy ich jakoś rozdzielić?

Bob: Rozdzielić? Masz na myśli osobne pliki?

Mark: Tak. W ten sposób moglibyśmy mieć jeden plik wyświetlający listę miejsc, drugi wyświetlający rezerwację, a potem dołączyć każdą z tych stron czy też ją wywołać ze strony głównej.

Laura: Aaa — jak w rozdzielności funkcji.

Bob: A cóż to jest?

Laura: **Rozdzielność funkcji.** Oznacza, że jeden fragment kodu wykonuje tylko jedną rzecz. Łatwiej dzięki temu śledzić błędy.

Bob: Pewnie, brzmi super... ale jak się to robi?

Jak możemy podzielić zawartość strony na odrębne pliki?

Jeśli podzielimy stronę na odrębne pliki, łatwiej będzie nią zarządzać. Jak jednak możemy to zrobić?

Rails pozwala na przechowywanie fragmentów stron w odrębnych plikach nazywanych **szablonami części strony** lub krócej — **szablonami częściowymi** (ang. *partial*). Szablon częściowy przypomina podprocedurę zwracającą jedynie niewielką część strony. W naszym przypadku możemy wykorzystać dwa szablony częściowe: jeden dla listy miejsc i drugi służący do dodawania nowej rezerwacji miejsca.

Szablony częściowe to po prostu pliki Embedded Ruby, podobnie jak zwykłe szablony stron. Jedyna różnica polega na tym, że w przeciwieństwie do zwykłych szablonów, nazwy szablonów częściowych zaczynają się od znaku „_”.

Możemy wykorzystać szablony częściowe we fragmentach z nową rezerwacją oraz listą miejsc.

Name	Baggage	Action
Brad Shurpen	22.0	Show
Kirt Avery	15.0	Show
Drew Beutline	18.0	Show
James Brake	19.0	Show
Troy Brinkman	22.0	Show
Ted Cebula	19.0	Show
Jesse Orry	15.0	Show
Jack Cauley	28.0	Show

New seat

Name:

Baggage:

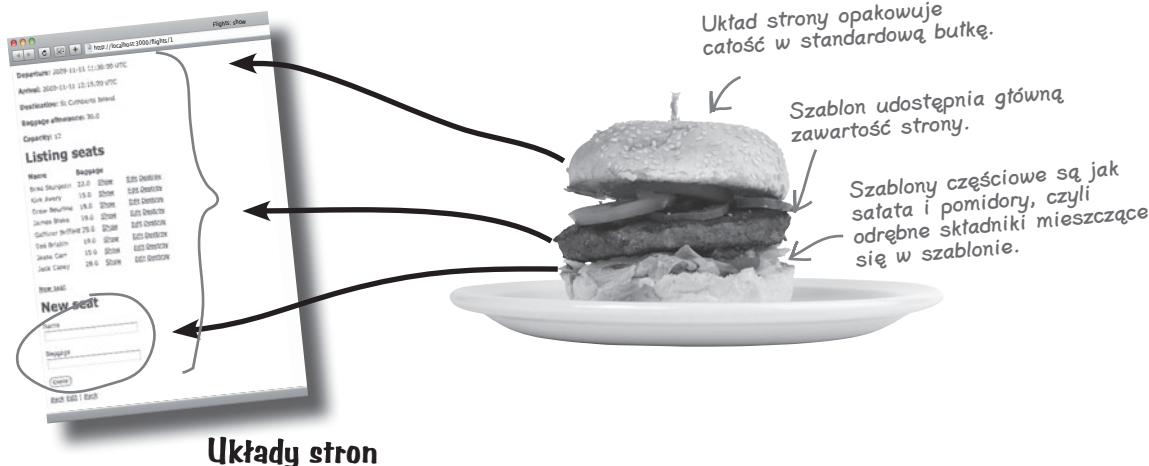
[Back](#) [Edits / Back](#)



Pliki ERb z bliska

Mamy teraz trzy rodzaje plików Embedded Ruby (ERb): **szablony**, **układy stron** oraz **szablony częściowe**. Jakie są między nimi różnice i w jaki sposób każdy z typów pliku ERb pasuje do pozostałych?

Stronę internetową można poskładać z plików ERB w podobny sposób, jak kilka składników wykorzystuje się do złożenia hamburgera.



Układy stron

Układ strony nadaje zbiorowi stron spójny wygląd, przede wszystkim dzięki udostępnieniu fragmentów standardowego kodu HTML, który znajduje się na górze i na dole każdej strony — podobnie jak bułka opakowująca hamburgera. Domyslnie wszystkie strony powiązane z określonym kontrolerem będą współdzieliły ten sam układ strony.

Szablony

Szablon stanowi główną zawartość strony internetowej, podobnie do nadzienia hamburgera. Szablon powiązany jest z akcją. Z tego powodu mamy jeden szablon pokazujący szczegóły lotu i kolejny służący do dodawania nowego lotu.

Szablony częściowe

Szablon może wywoływać odrębne **szablony częściowe**, które budują główną zawartość strony. Szablony częściowe są jak mniejsze składniki hamburgera, takie jak pomidor czy sałata. Szablony częściowe pozwalają rozbić złożony szablon strony na mniejsze części. Pozwalają również oddzielić powtarzające się treści, takie jak menu czy paski nawigacyjne. Szablony częściowe mogą być wykorzystywane przez zwykłe szablony, ale także przez układy stron.

Kilka plików ERb w pełnych kostiumach gra na przyjęciu w grę „Kim jestem?”. Każdy da Ci wskazówkę — a Ty będziesz musiał spróbować odgadnąć, czym są, na podstawie tego, co mówią. Załóż, że zawsze mówią o sobie prawdę. Wypełnij luki po prawej stronie, identyfikując uczestników zabawy.

Dzisiejsi uczestnicy gry:

Mogą się pojawić dowolne typy czarujących plików ERb, jakie dotychczas widziałeś!

Kim jestem?



Typ pliku ERb

Zawieram menu nawigacyjne.

Zawieram tytuł, który pojawia się w oknie przeglądarki.

Jeśli ktoś potrzebuje utworzyć nowy obiekt, wyświetlam formularz.

Wyświetlam dane kontaktowe oraz informację o prawach autorskich.

Nadaję zestawowi stron standardowo wyglądający pasek nawigacyjny.

Spotkanie z ERb

Kilka plików ERb w pełnych kostiumach gra na przyjęciu w grę „Kim jestem?”. Każdy da Ci wskazówkę — a Ty będziesz musiał spróbować odgadnąć, czym są, na podstawie tego, co mówią. Załóż, że zawsze mówią o sobie prawdę. Wypełnij luki po prawej stronie, identyfikując uczestników zabawy.

Dzisiaj uczestnicy gry:

Mogą się pojawić dowolne typy czarujących plików ERb, jakie dotychczas widziałeś!

Kim jestem?



To fragment strony, który można wykorzystać w różnych miejscach.

Zawieram menu nawigacyjne.

Ciąg cała część HTML <title> pochodzi z układu strony.

Zawieram tytuł, który pojawia się w oknie przeglądarki.

Typ pliku ERb

szablon częściowy

układ strony

szablon

szablon częściowy

układ strony

Nadaję zestawowi stron standardowo wyglądający pasek nawigacyjny.

Układ strony kontroluje wygląd kilku stron, choć najprawdopodobniej wywoła pasek nawigacyjny z odrębnego szablonu częściowego.

ERb SKŁADA nasze strony

Musimy utworzyć szablony częściowe dla formularza rezerwacji oraz listy miejsc, a następnie Embedded Ruby może przetworzyć stronę lotu i wywołać szablony częściowe za każdym razem, gdy dotrze do wyrażenia `render`.



To pozwala na rozdzielenie funkcji: mamy odrębne komponenty zajmujące się rezerwacjami oraz miejscami, które w miarę potrzeby są łączone dla użytkownika.

ERb posłada naszą stronę lotu z szablonu `show.html.erb`, szablonu częściowego formularza rezerwacji oraz szablonu częściowego listy miejsc.



Do zrobienia

- Utworzyć szablon częściowy z formularzem rezerwacji.
- Dodać formularz rezerwacji do strony.
- Utworzyć szablon częściowy z listą miejsc.
- Dodać listę miejsc do strony.

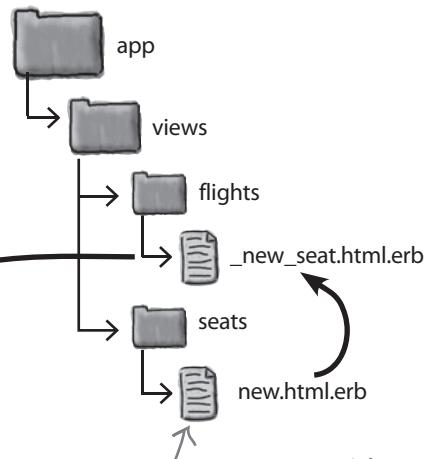
Kiedy Rails otrzyma żądanie udostępnienia informacji o locie, użycie szablonów częściowych, szablonów stron oraz układu strony w połączeniu z Embedded Ruby do wygenerowania jednej odpowiedzi HTML.

Zacznijmy od pierwszego elementu na liście — formularza rezerwacji.

Jak można utworzyć szablon częściowy formularza rezerwacji?

Szablony częściowe są po prostu kolejnym rodzajem pliku ERb, dlatego zawierają **te same typy znaczników**, jakie znajdują się w szablonach stron. Poniżej znajduje się zawartość naszego nowego szablonu częściowego `_new_seat.html.erb`. Zawiera on dokładnie ten sam kod co strona z nową rezerwacją miejsca, co oznacza, że wystarczy skopiować plik `app/views/seats/new.html.erb` i zapisać go jako `app/views/flights/_new_seat.html.erb`:

```
<h1>New seat</h1>
<% form_for(@seat) do |f| %>
  <%= f.error_messages %>
  <p>
    <%= f.label :flight_id %><br />
    <%= f.text_field :flight_id %>
  </p>
  <p>
    <%= f.label :name %><br />
    <%= f.text_field :name %>
  </p>
  <p>
    <%= f.label :baggage %><br />
    <%= f.text_field :baggage %>
  </p>
  <p>
    <%= f.submit "Create" %>
  </p>
<% end %>
<%- link_to 'Back', seats_path %>
```



By utworzyć szablon częściowy, skopiuj plik `app/views/seats/new.html.erb` i zapisz go jako `app/views/flights/_new_seat.html.erb`. Przedrostek „`_`” sprawia, że plik staje się szablonem częściowym.

Musimy usunąć ten odnośnik do listy wszystkich miejsc. Nie jest nam potrzebny.

Mogliśmy pozostawić szablon częściowy w folderze `seats`, jednak przesuniemy go do folderu `flights`, by nieco łatwiej było go wywołać. Naprawdę istotne jest, by szablon częściowy zaczynał się od znaku `_`. Znak `_` wykorzystywany jest przez Rails do odróżnienia szablonów częściowych od szablonów stron.

Teraz musimy dołączyć szablon częściowy do szablonu strony

Utworzenie szablonu częściowego to tylko połowa zadania. Teraz musimy zmodyfikować szablon `show.html.erb` lotu, tak by jego dane wyjściowe **zawierały** szablon częściowy. Szablony częściowe, podobnie jak szablony stron, są tak naprawdę fragmentami kodu w języku Ruby, zamaskowanego w taki sposób, by wyglądał jak HTML.

I w taki sam sposób jak jeden fragment kodu Ruby wywołuje inny, szablon strony może z łatwością **wywołać** szablon częściowy.

```
<p>
  <b>Departure:</b>
  <%=h @flight.departure %>
</p>

<p>
  <b>Arrival:</b>
  <%=h @flight.arrival %>
</p>

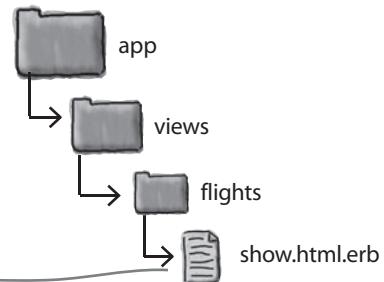
<p>
  <b>Destination:</b>
  <%=h @flight.destination %>
</p>

<p>
  <b>Baggage allowance:</b>
  <%=h @flight.baggage_allowance %>
</p>

<p>
  <b>Capacity:</b>
  <%=h @flight.capacity %>
</p>

<%= render :partial=>"new_seat" %>
<%= link_to 'Edit', edit_flight_path(@flight) %> |
<%= link_to 'Back', flights_path %>
```

„`new_seat`” odnosi się do szablonu częściowego `_new_seat.html.erb`.



W wywołaniu render nie używa się pełnej nazwy pliku.

Choć szablony częściowe zaczynają się od znaku „_”, a kończą „.html.erb”, obie te części należy pominąć przy wywoływaniu szablonu częściowego za pomocą polecenia render.

Wywołanie polecenia `render` mówi Embedded Ruby, że ma przetworzyć szablon częściowy i dołączyć jego dane wyjściowe w tym miejscu pliku.

Szablon częściowy powinien teraz być widoczny na stronie lotu.



Jazda próbna

Przyjrzyjmy się stronie `show.html.erb` i sprawdźmy, czy formularz rezerwacji jest poprawnie wyświetlany. Kiedy wprowadzimy do systemu kilka lotów, a następnie spróbujemy zobaczyć pierwszy z nich, przechodząc do strony:

`http://localhost:3000/flights/1`

widzimy następujący obraz:

To dane wyjściowe wygenerowane przez stronę lotu `show.html.erb`.

RuntimError in Flights#show

Showing app/views/flights/_new_seat.html.erb where line #3 raised:

Called id for nil, which would mistakenly be 4 -- if you really wanted the id of nil, use object_id

Extracted source (around line #3):

```
1: <h1>New seat</h1>
2:
3: <% form_for(@seat) do |f| %>
4:   <%= f.error_messages %>
5:
6:   <p>
```

Trace of template inclusion: app/views/flights/show.html.erb

RAILS_ROOT: /Users/davidg/data/writing/books/hfror/code/chap6-master_detail/coconut1

[Application Trace](#) | [Framework Trace](#) | [Full Trace](#)

```
vendor/rails/actionpack/lib/action_controller/record_identifier.rb:76:in `dom_id'
vendor/rails/actionpack/lib/action_view/helpers/html/html_options_helper.rb:10:in `dom_id'
```

Pojawił się dziwny błąd. Strona lotu działała przed wstawieniem szablonu częściowego, co więc poszło źle?

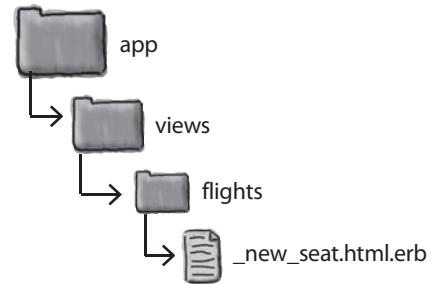


WYSIL

SZARE KOMÓRKI

Czy patrząc na błędy wygenerowane w jeździe próbnej oraz na kod szablonu częściowego, jesteś w stanie powiedzieć, co spowodowało, że aplikacja nie działa?

```
<h1>New seat</h1>
<% form_for(@seat) do |f| %>
  <%= f.error_messages %>
  <p>
    <%= f.label :flight_id %><br />
    <%= f.text_field :flight_id %>
  </p>
  <p>
    <%= f.label :name %><br />
    <%= f.text_field :name %>
  </p>
  <p>
    <%= f.label :baggage %><br />
    <%= f.text_field :baggage %>
  </p>
  <p>
    <%= f.submit "Create" %>
  </p>
<% end %>
```



Szablony częściowe także potrzebują danych

Musimy przekazać szablonowi częściowemu miejsce!

Problem spowodowało to, że kod ERb zawiera referencję do zmiennej `@seat`.

Dlaczego jest to problemem?

Plik ten był kiedyś szablonem strony powiązanym z kontrolerem `SeatsController`. `SeatsController` inicjalizował instancję zmiennej `@seat` w następujący sposób:

```
@seat = Seat.new
```

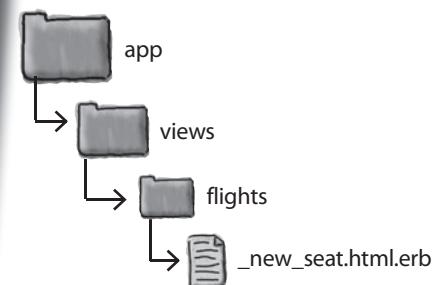
Teraz plik stał się szablonem częściowym wykorzystywany przez kontroler `FlightsController`, a ten kontroler nie zawiera instancji zmiennej `@seat`. Musimy zmienić `@seat` w zmienną lokalną o nazwie `seat`:

Zamiast korzystać ze zmiennej „`@seat`”, możemy użyć zmiennej lokalnej „`seat`”.

```
<h1>New seat</h1>
<% form_for(seat) do |f| %>
  <%= f.error_messages %>
  <p>
    <%= f.label :flight_id %><br />
    <%= f.text_field :flight_id %>
  </p>
  <p>
    <%= f.label :name %><br />
    <%= f.text_field :name %>
  </p>
  <p>
    <%= f.label :baggage %><br />
    <%= f.text_field :baggage %>
  </p>
  <p>
    <%= f.submit "Create" %>
  </p>
<% end %>
```

```
<h1>New seat</h1>
<% form_for(@seat) do |f| %>
```

Problem spowodowało to odwołanie do zmiennej „`@seat`”.



Zmienna `seat` nazywana jest **zmienną lokalną**, ponieważ żaden element znajdujący się poza szablonem częściowym nie może ani jej odczytywać, ani nic do niej zapisywać. Jeśli tak jednak jest, w jaki sposób możemy przekazać szablonowi częściowemu wartość zmiennej `seat`?

Zmienne lokalne można przekazywać do szablonu częściowego

Zwykłe szablony oraz szablony częściowe w swoim działaniu przypominają metody lub funkcje języka Ruby. Kiedy szablon wyświetla szablon częściowy, przypomina to nieco sytuację, w której jedna funkcja wywołuje drugą.

A ponieważ szablon częściowy faktycznie jest jak funkcja, w następujący sposób można do niego przekazywać parametry:

```
<%= render :partial=>"new_seat", :locals=>{ :seat=>_____ } %>
```

Metoda `render` może przyjmować tablicę asocjacyjną o nazwie `locals`. Wewnątrz tablicy asocjacyjnej można umieścić zbiór wartości indeksowanych za pomocą nazwy zmiennej. Jak w prawie każdym innym miejscu w Rails, nazwy wyrażane są za pomocą *symboli*.

Jaką wartość powinniśmy jednak przekazać dla `seat`? Przyjrzyjmy się wartości wykorzystywanej przez oryginalny kontroler `SeatsController`:

```
def new
  @seat = Seat.new
```

Ponieważ do zainicjalizowania `seat` wykorzystany zostaje formularz, musimy jedynie przekazać do niego świeżo utworzony obiekt `Seat`:

```
<%= render :partial=>"new_seat", :locals=>{ :seat=>Seat.new } %>
```

Czy naprawiło to problem pojawiający się na stronie lotu?

Nie istniejąca grupa pytań

P: Czy muszę sprawić, by szablon częściowy korzystał ze zmiennej lokalnej?

O: Nie. Szablony częściowe mogą widzieć te same zmienne instancji (zmienne zaczynające się od znaku @) co zwykłe szablony. Używanie zmiennych lokalnych w szablonach częściowych jest jednak dobrą praktyką.

P: Dlaczego?

O: Dzięki temu szablon częściowy jest w mniejszym stopniu uzależniony od reszty kodu. Szablony stron są i tak ściśle uzależnione od kontrolera, dlatego dla nich korzystanie ze zmiennych instancji kontrolera jest zupełnie uzasadnione. Szablony częściowe *nie są* jednak tak ściśle powiązane z kontrolerami. Wiele aplikacji wykorzystuje *współdzielone* szablony częściowe, które używane są przez *więcej niż jeden* kontroler. Jeśli szablony częściowe używają jedynie zmiennych lokalnych, łatwiej jest nimi zarządzać.



Jazda próbna

Kiedy obiekt `seat` zostanie poprawnie zainicjalizowany, powinniśmy uniknąć poprzedniego błędu. Spróbujmy odświeżyć stronę lotu:

Flights: show
http://localhost:3000/flights/1

Departure: 2009-11-11 11:30:00 UTC
Arrival: 2009-11-11 12:15:00 UTC
Destination: St Cuthberts Island
Baggage allowance: 30.0
Capacity: 12

New seat

Flight: 1
Name: Brad Sturgeon
Baggage: 22.0
Create Edit | Back

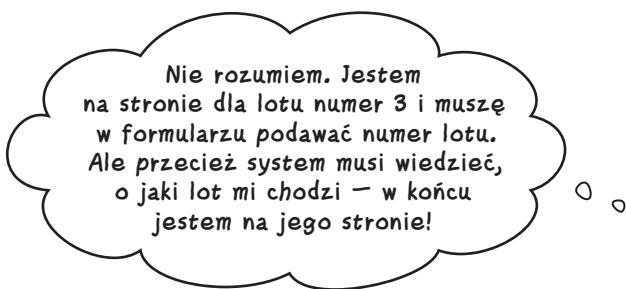
Seats: show
http://localhost:3000/seats/1

Flight: 1
Name: Brad Sturgeon
Baggage: 22.0
Edit | Back

Do zrobienia

- Utworzyć szablon częściowy z formularzem rezerwacji.
- Dodać formularz rezerwacji do strony.
- Utworzyć szablon częściowy z listą miejsc.
- Dodać listę miejsc do strony.

Tym razem formularz wyświetlany jest poprawnie. Mamy zmienną lokalną „`seat`”, zatem koniec problemów z wyświetlaniem.



Numer identyfikatora lotu można przekazać do szablonu częściowego z formularzem rezerwacji.

Jak jednak formularz może użyć identyfikatora? I jak może wpisać domyślną wartość tego pola *bez* pytania o to użytkownika?

Wydaje się, że wcale nie zakończyliśmy tego kroku – ciągle jeszcze pozostało sporo do zrobienia...

Do zrobienia

- Utworzyć szablon częściowy z formularzem rezerwacji.
- Dodać formularz rezerwacji do strony.
- Utworzyć szablon częściowy z listą miejsc.
- Dodać listę miejsc do strony.

Zaostrz ołówek



Numer lotu można podać przy tworzeniu obiektu Seat. Dodaj niezbędny kod do pliku *flights/show.html.erb*:

```
<%= render :partial=>"new_seat", :locals=>{:seat=>Seat.new( _____ )} %>
```



Zaostrz ołówek

Rozwiążanie

Numer lotu można podać przy tworzeniu obiektu Seat. Dodaj niezbędny kod do pliku `flights/show.html.erb`:

To „`flight.id`” z kropką w środku.

```
<%= render :partial=>"new_seat", :locals=>{ :seat=>Seat.new(:flight_id=>@flight.id) } %>
```

Ten kod pochodzi z pliku
`app/views/flights/show.html.erb`.

To „`flight_id`” ze znakiem
„_” w środku.

Mogliśmy przekazać w tablicy
asocjacyjnej wartości ustawiające
początkowe wartości obiektu modelu.

```
<p>
  <b>Departure:</b>
  <%=h @flight.departure %>
</p>

<p>
  <b>Arrival:</b>
  <%=h @flight.arrival %>
</p>

<p>
  <b>Destination:</b>
  <%=h @flight.destination %>
</p>

<p>
  <b>Baggage allowance:</b>
  <%=h @flight.baggage_allowance %>
</p>

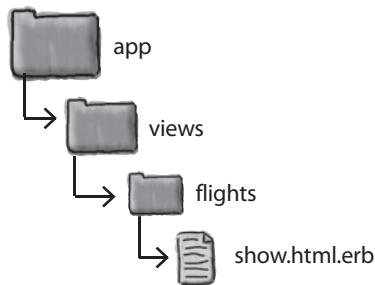
<p>
  <b>Capacity:</b>
  <%=h @flight.capacity %>
</p>

<%= render :partial=>"new_seat", :locals=>{ :seat=>Seat.new(:flight_id=>@flight.id) } %>

<%= link_to 'Edit', edit_flight_path(@flight) %> |
<%= link_to 'Back', flights_path %>
```



Twój szablon
`show.html.erb` powinien
teraz wyglądać tak:



Łamigłówka



Użytkownicy nie powinni być zobowiązani do podawania numeru lotu, dlatego musimy ten numer przechować za pomocą ukrytego pola. Czy potrafisz złożyć kod za to odpowiedzialny?

Usuneliśmy stare pole „flight_id”.

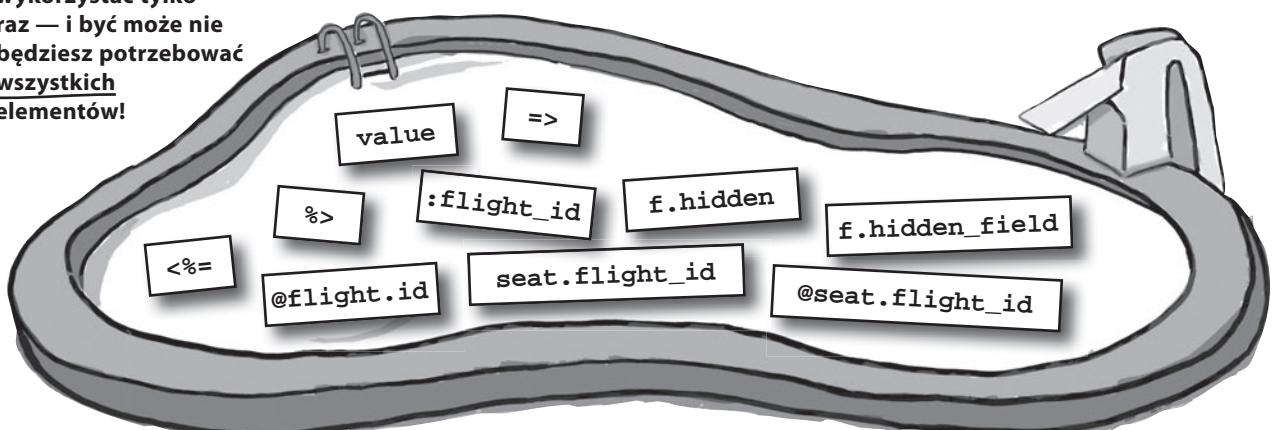
```
<h1>New seat</h1>
<% form_for(seat) do |f| %>
  <%= f.error_messages %>
```

Kod ukrytego pola musi trafić w końcu tutaj.

```
.....>
<p>
  <%= f.label :name %><br />
  <%= f.text_field :name %>
</p>
<p>
  <%= f.label :baggage %><br />
  <%= f.text_field :baggage %>
</p>
<p>
  <%= f.submit "Create" %>
</p>
<% end %>
```

To plik
app/views/flights/_new_seat.html.erb.

Uwaga: każdy element z basenu można wykorzystać tylko raz — i być może nie będziesz potrzebować wszystkich elementów!



Łamigłówka: Rozwiążanie



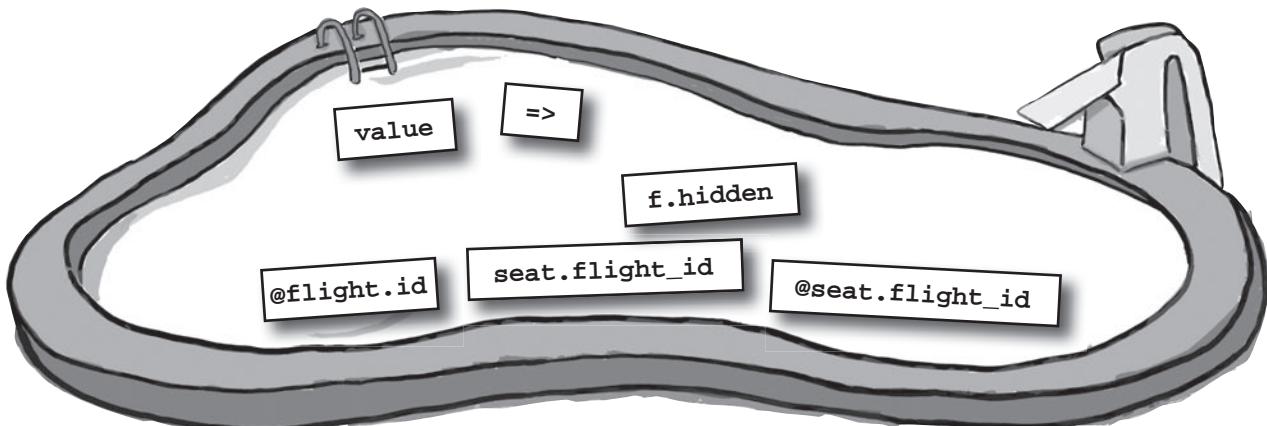
Nie trzeba już podawać numeru lotu, dlatego musimy ten numer przechować za pomocą ukrytego pola.
Czy potrafisz złożyć kod za to odpowiedzialny?

Powiązane pola
zawsze kończą się
na „_field”.

```
<h1>New seat</h1>
<% form_for(seat) do |f| %>
  <%= f.error_messages %>
  <%= f.hidden_field :flight_id %>
  <p>
    <%= f.label :name %><br />
    <%= f.text_field :name %>
  </p>
  <p>
    <%= f.label :baggage %><br />
    <%= f.text_field :baggage %>
  </p>
  <p>
    <%= f.submit "Create" %>
  </p>
<% end %>
```

Obiekt „seat” zawiera już identyfikator lotu, zatem nie musimy go przekazywać.

To plik
app/views/flights/_new_seat.html.erb.





Jazda próbna

Gdy teraz udamy się na stronę lotu, okaże się, że pole z numerem lotu zniknęło z formularza — dokładnie tak, jak tego chcieliśmy.

Flights: show
http://localhost:3000/flights/1

Departure: 2009-11-11 11:30:00 UTC
Arrival: 2009-11-11 12:15:00 UTC
Destination: St Cuthberts Island
Baggage allowance: 30.0
Capacity: 12

New seat

Name: Brad Sturgeon
Baggage: 22
[Create](#)

[Edit](#) | [Back](#)

Nie trzeba już podawać numeru identyfikatora lotu.

Seats: show
http://localhost:3000/seats/1

Flight: 1
Name: Brad Sturgeon
Baggage: 22.0
[Edit](#) | [Back](#)

Nowy system rezerwacji miejsc sam wybrat właściwy numer lotu.

Formularz działa!

Numer lotu jest teraz automatycznie pobierany z obiektu lotu. Co dalej?

<input checked="" type="checkbox"/>	Utworzyć szablon częściowy z formularzem rezerwacji.
<input checked="" type="checkbox"/>	Dodać formularz rezerwacji do strony.
<input type="checkbox"/>	Utworzyć szablon częściowy z listą miejsc.
<input type="checkbox"/>	Dodać listę miejsc do strony.

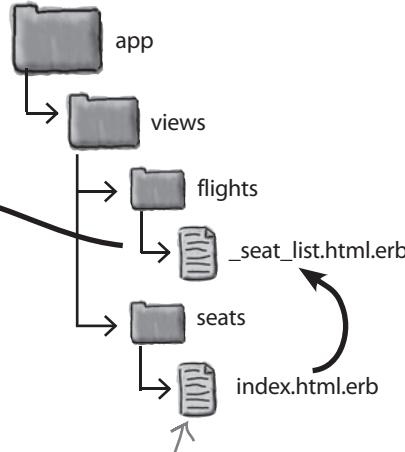
Teraz musimy utworzyć szablon częściowy z listą miejsc.

Niezbędny jest nam szablon częściowy dla listy miejsc

Listę indeksującą miejsc moźemy przekształcić mniej więcej tak samo, jak zrobiliśmy to w przypadku formularza rezerwacji — kopiując oryginalny szablon miejsca do pliku szablonu częściowego. Nazwijmy ten nowy szablon częściowy `_seat_list.html.erb`:

To jest dolna część pliku — powyżej niej znajdują się nagłówki tabeli oraz tytuł. Usuń nagłówek dla kolumny Flight

```
<% for seat in @seats %>
  <tr>
    <td><=%= h seat.flight_id %></td> ← Będziesz musiał zmienić nazwę zmiennej instancji „@seats” na „seats”.
    <td><=%= h seat.name %></td> ← Nie musisz wyświetlać identyfikatora lotu, gdyż jest on częścią strony lotu.
    <td><=%= h seat.baggage %></td>
    <td><=%= link_to 'Show', seat %></td>
    <td><=%= link_to 'Edit', edit_seat_path(seat) %></td>
    <td><=%= link_to 'Destroy', seat, :confirm => 'Are
      <td><=%= link_to 'Destroy', seat, :method => :delete %></td>
      you sure?', :method => :delete %></td>
      Usuń odnośnik do strony „New seat”, ponieważ nie jest nam on potrzebny.
    </tr>
<% end %>
<=%= link_to 'New seat', new_seat_path %>
```



By utworzyć szablon częściowy, skopiuj plik /seats/index.html.erb i zapisz jako /flights/_seat_list.html.erb.

Ale przecież szablon częściowy z listą miejsc potrzebuje tablicy miejsc...

Strona indeksująca miejsc wyświetlała zawartość zmiennej instancji kontrolera SeatsController o nazwie `@seats`. Kontroler SeatsController tworzył zmienną instancję tuż przed wyświetleniem strony `index.html.erb`. A jak jest teraz? Skopiowaliśmy szablon `index.html.erb` do szablonu częściowego, który zostanie wyświetlony po wykonaniu kontrolera FlightsController... nie istnieje zatem zmienna instancji `@seats` zawierająca tablicę miejsc.

Oznacza to, że musimy dostarczyć nowemu szablonowi częściowemu `_seat_list.html.erb` tablicę miejsc. Jakie wartości musimy udostępnić na potrzeby tablicy miejsc? Kontroler SeatsController zainicjalizował instance zmiennej `@seat` w następujący sposób:

```
def index
  @seats = Seat.find(:all)
```

Na razie wywołajmy listę miejsc w ten sposób i sprawdźmy, jak to działa:

```
<%= render :partial=>"seat_list", :locals=>{:seats=>Seat.find(:all)} %>
```

Dodamy to wywołanie do pliku app/views/flights/show.html.erb.

Prekażemy to jako wartość dla tablicy miejsc.

Czy to zadziała? Zobaczmy...



Jazda próbna

Wprowadź te wszystkie zmiany, dodaj nowy szablon częściowy i sprawdź działanie aplikacji.

Wszystkie części strony są teraz na swoich miejscach.

Name	Baggage		
Brad Sturgeon	22.0	Show	Edit Destroy
Kirk Avery	15.0	Show	Edit Destroy
Drew Beurline	18.0	Show	Edit Destroy
James Blake	19.0	Show	Edit Destroy
Geffiner Brilliant	25.0	Show	Edit Destroy
Ted Brisbin	19.0	Show	Edit Destroy
Jesse Carr	15.0	Show	Edit Destroy
Jack Cosey	28.0	Show	Edit Destroy
Brent Chase	19.0	Show	Edit Destroy
Tom Christie	15.0	Show	Edit Destroy
Ryan Cleary	19.0	Show	Edit Destroy
Julien Collard	16.0	Show	Edit Destroy
Charlie Collins	19.0	Show	Edit Destroy

Do zrobienia

- Utworzyć szablon częściowy z formularzem rezerwacji.
- Dodać formularz rezerwacji do strony.
- Utworzyć szablon częściowy z listą miejsc.
- Dodać listę miejsc do strony.

Formularz wygląda, jakby działał. Zobaczmy, co sądzą użytkownicy.

Ludzie trafiają na niewłaściwe loty

Każdy uważa, że system wygląda świetnie, dlatego zostaje on udostępniony użytkownikom. Niestety, wkrótce ktoś dostrzega problem...



Stary... Zarezerwowałem lot na imprezę na plaży, ale wyładowałem na historycznej wyprawie do starej kolonii trędowatych!

Co się stało?

Strona lotu wyświetla *wszystkie* rezerwacje miejsc na *wszystkie* loty.

Departure: 2009-11-11 11:30:00 UTC
Arrival: 2009-11-11 12:15:00 UTC
Destination: St Cuthberts Island
Baggage allowance: 30.0
Capacity: 12

Listing seats

Name	Baggage	Action
Brad Sturgeon	22.0	Show Edit Destroy
Kirk Avery	15.0	Show Edit Destroy
Drew Beurline	18.0	Show Edit Destroy
James Blake	19.0	Show Edit Destroy
Gaffiner Brilliant	25.0	Show Edit Destroy
Ted Brisbin	19.0	Show Edit Destroy
Jesse Carr	15.0	Show Edit Destroy
Jack Casey	28.0	Show Edit Destroy
Brent Chase	19.0	Show Edit Destroy
Tom Christie	15.0	Show Edit Destroy
Ryan Cleary	19.0	Show Edit Destroy
Julien Collard	16.0	Show Edit Destroy
Charlie Collins	19.0	Show Edit Destroy

New seat

To są strony różnych lotów!
Każdy lot w systemie pokazuje tę samą listę rezerwacji miejsc.

Departure: 2009-11-11 13:30:00 UTC
Arrival: 2009-11-11 14:30:00 UTC
Destination: Titchmarsh Island
Baggage allowance: 25.0
Capacity: 22

Listing seats

Name	Baggage	Action
Brad Sturgeon	22.0	Show Edit Destroy
Kirk Avery	15.0	Show Edit Destroy
Drew Beurline	18.0	Show Edit Destroy
James Blake	19.0	Show Edit Destroy
Gaffiner Brilliant	25.0	Show Edit Destroy
Ted Brisbin	19.0	Show Edit Destroy
Jesse Carr	15.0	Show Edit Destroy
Jack Casey	28.0	Show Edit Destroy
Brent Chase	19.0	Show Edit Destroy
Tom Christie	15.0	Show Edit Destroy
Ryan Cleary	19.0	Show Edit Destroy
Julien Collard	16.0	Show Edit Destroy
Charlie Collins	19.0	Show Edit Destroy

New seat

Co się dzieje? Problem spowodował polecenie render wywołujące szablon częściowy listy miejsc. Jak sobie przypominasz, szablon częściowy wywołaliśmy w następujący sposób:

```
<%= render :partial=>"seat_list",  
:locals=>{ :seats=>Seat.find(:all)} %>
```

Kod ten wyświetla listę wszystkich miejsc z bazy danych. Takie rozwiązywanie było dobre, dopóki lista miejsc była stroną indeksującą dla danych miejsc... ale skoro teraz wyświetlamy te dane dla określonego lotu, musimy ograniczyć wyświetlane miejsca jedynie do tych należących do bieżącego lotu.

Moglibyśmy naprawić metodę wyszukującą... ale lepiej jest utworzyć *powiązanie*.

Powiązanie łączy ze sobą modele

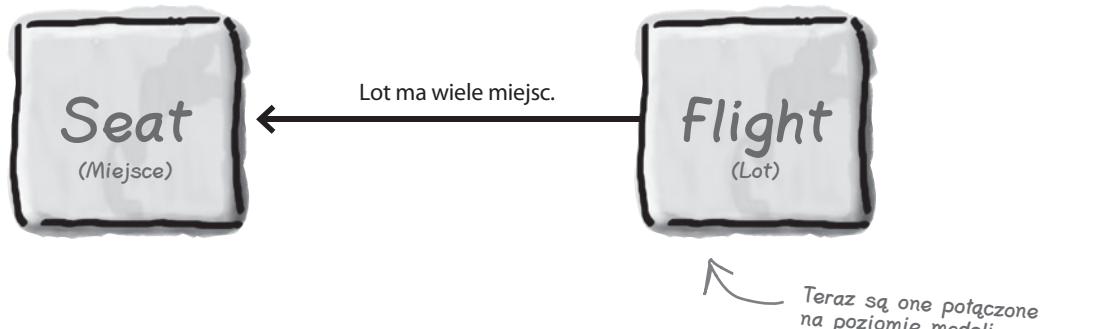
Często spotkasz się z sytuacją, w której określone modele są zazwyczaj wykorzystywane razem, tak jak w przypadku lotów oraz rezerwacji miejsc. Być może będziesz musiał użyć danych z jednego typu (na przykład identyfikatora lotu) do odnalezienia powiązanych obiektów w drugim typie (jak w przypadku miejsc zarezerwowanych na określony lot).

Moglibyś użyć metody wyszukującej, która odczytałaby powiązane obiekty. Przykładowo gdybyś miał obiekt lotu o nazwie `@flight`, moglibyś odnaleźć powiązane obiekty miejsc w następujący sposób:

```
Seat.find_all_by_flight_id(@flight.id)
```

Zwraca tablicę obiektów miejsc.

Tak naprawdę łatwiej jest *połączyć* ze sobą oba modele za pomocą **powiązania**:



Powiązanie sprawia, że obiekty jednego typu wydają się *atrybutami* obiektów drugiego typu. Przykładowo jeśli utworzymy w modelu lotu powiązanie łączące go z modelem miejsc, możemy odnosić się do miejsc powiązanych z lotem w następujący sposób:

```
@flight.seats
```

Kod ten powoduje zwrócenie dokładnie tego samego co pokazana wyżej metoda wyszukująca, jednak zdefiniowanie powiązania między dwoma modelami powoduje uproszczenie kodu i zredukowanie szansy popełnienia błędu wynikającego z powtarzającego się definiowania metod wyszukujących przeskakujących między jednym modelem a drugim. Sprawi też, że kod będzie o wiele łatwiejszy do odczytania.

Brzmi nieźle. Jak zatem działają powiązania?

Powiązania wyglądają jak atrybuty



- Powiązania z bliska -

Powiązania łączą dane z tabel miejsc oraz lotów, dopasowując dane z kolumn seat.flight_id oraz flight.id.



Nazwy są
ISTOTNE
w Rails.
seat.flight_id
odpowiada
flight.id

ID	departure	arrival	destination	baggage_allowa	capacity	created_at	updated_at
1	2009-11-11 11:...	2009-11-11 12:...	St Cuthberts I...	30	12	2008-11-11 11:...	2008-11-11 11:...
2	2009-11-11 13:...	2009-11-11 14:...	Titchmarsh Isl...	25	22	2008-11-11 12:...	2008-11-11 12:...
3	2009-11-11 08:...	2009-11-11 09:...	St Augustine L...	8	4	2008-11-11 12:...	2008-11-11 12:...

ID	flight_id	name	baggage	created_at	updated_at
1	1	Brad Sturgeon	22	2008-11-11 11:...	2008-11-11 11:...
2	1	Kirk Avery	15	2008-11-11 12:...	2008-11-11 12:...
3	1	Drew Beurline	18	2008-11-11 12:...	2008-11-11 12:...
4	1	James Blake	19	2008-11-11 12:...	2008-11-11 12:...
5	1	Gaffiner Brill...	25	2008-11-11 12:...	2008-11-11 12:...
6	1	Ted Brisbin	19	2008-11-11 12:...	2008-11-11 12:...
7	1	Jesse Carr	15	2008-11-11 12:...	2008-11-11 12:...
8	1	Jack Casey	28	2008-11-11 12:...	2008-11-11 12:...
9	2	Brent Chase	19	2008-11-11 12:...	2008-11-11 12:...
10	2	Tom Christie	15	2008-11-11 12:...	2008-11-11 12:...
11	2	Ryan Cleary	19	2008-11-11 12:...	2008-11-11 12:...
12	2	Julien Collard	16	2008-11-11 12:...	2008-11-11 12:...
13	2	Charlie Collins	19	2008-11-11 12:...	2008-11-11 12:...

By powiązanie działało, pole w tabeli seats musi nosić nazwę flight_id, a także **musi** być liczbą całkowitą.

Kiedy mamy już powiązanie, oznacza to, że gdy Rails widzi:

```
@flights.seats
```

To wygląda jak atrybut, ale tak naprawdę jest powiązaniem pomiędzy dwoma tabelami.

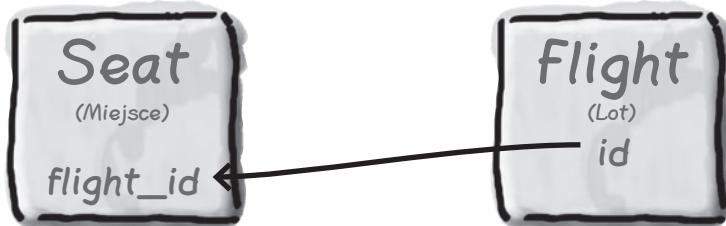
potraktuje to jak:

```
Seat.find_all_by_flight_id(@flight.id)
```

To wymaganie wynika z tego, że kolumna „id” w odpowiadającej tabeli „flights” musi być z nim powiązana.

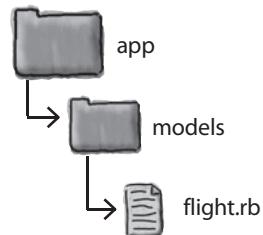
Jak jednak definiujemy powiązanie?

Do modelu Flight dodajemy specjalny atrybut o nazwie seats, stąd logiczne jest, że powiązanie definiujemy w kodzie modelu Flight.



```
class Flight < ActiveRecord::Base
  has_many :seats
end
```

To jest powiązanie. Lot ma wiele („has many”) miejsc.



Polecenie has_many przyjmuje nazwę powiązanego modelu, a ponieważ będzie wykorzystywane do odnajdywania tablic powiązanych miejsc, nazwa modelu jest w liczbie mnogiej. Parametrem polecenia has_many jest zatem :seats, a nie :seat (bez „s” na końcu). Kiedy powiązanie jest gotowe, można użyć nowego atrybutu w następujący sposób:

```
@flight.seats
```

Atrybut seats zwraca tablicę obiektów miejsc powiązanych z lotem:

To atrybut „seats”, który jest tak naprawdę wynikiem zwracanym przez metodę wyszukującą dla pasujących miejsc.

Zaostrz ołówek



Popraw poniższy wiersz z szablonu *app/views/flights/show.html.erb*, tak by wykorzystywał on nowe powiązanie:

```
<%= render :partial=>"seat_list", :locals=>{:seats=> _____} %>
```

Zaostrz ołówek



Rozwiążanie

Popraw poniższy wiersz z szablonu `app/views/flights/show.html.erb`, tak by wykorzystywał on nowe powiązanie:

```
<%= render :partial=>"seat_list", :locals=>{:seats=> @flight.seats } %>
```



Jazda próbna

Wprowadź wszystkie te zmiany i przeładuj stronę! Strony lotów pokazują teraz jedynie miejsca przydzielone dla określonego lotu. Kiedy spojrzymy na strony lotów numer 1 oraz 3, zobaczymy, że mają one różne listy rezerwacji miejsc:

The screenshot shows two browser windows side-by-side, each displaying a "Flights: show" page for a specific flight.

Left Window (Lot 1):

- Flight Details:** Departure: 2009-11-11 11:30:00 UTC, Arrival: 2009-11-11 12:15:00 UTC, Destination: St Cuthberts Island, Baggage allowance: 30.0, Capacity: 12.
- Listing seats:** A table showing reserved seats:

Name	Baggage	
Brad Sturgeon	22.0	Show Edit Destroy
Kirk Avery	15.0	Show Edit Destroy
Drew Beurline	18.0	Show Edit Destroy
James Blake	19.0	Show Edit Destroy
Gaffiner Brilliant	25.0	Show Edit Destroy
Ted Brisbin	19.0	Show Edit Destroy
Jesse Carr	15.0	Show Edit Destroy
Jack Casey	28.0	Show Edit Destroy
- New seat:** Form fields for Name and Baggage, with a "Create" button.
- Bottom:** "Edit | Back" links.

Right Window (Lot 3):

- Flight Details:** Departure: 2009-11-11 13:30:00 UTC, Arrival: 2009-11-11 14:30:00 UTC, Destination: Titchmarsh Island, Baggage allowance: 25.0, Capacity: 22.
- Listing seats:** A table showing reserved seats:

Name	Baggage	
Brent Chase	19.0	Show Edit Destroy
Tom Christie	15.0	Show Edit Destroy
Ryan Cleary	19.0	Show Edit Destroy
Julien Collard	16.0	Show Edit Destroy
Charlie Collins	19.0	Show Edit Destroy
- New seat:** Form fields for Name and Baggage, with a "Create" button.
- Bottom:** "Edit | Back" links.

A central text annotation between the two windows reads: "Różne loty mają teraz różne rezerwacje miejsc." (Different flights now have different seat reservations.)

Niektóre osoby mają jednak za duży bagaż

Teraz pojawił się problem z bagażem. Niektóre osoby pojawiają się na lotnisku ze zbyt ciężkim bagażem — znacznie przekraczającym limit dla danego lotu. Dane lotu zapisują maksymalny dozwolony limit bagażu, jednak wielu pasażerów jest niezadowolonych, ponieważ przekazali oni linii lotniczej, ile bagażu zabierają ze sobą, kiedy wprowadzali rezerwację miejsca, a system przeciwko temu nie protestował. System należy zatem zmodyfikować, tak by nie pozwalał rezerwować miejsc ludziom ze zbyt ciężkim bagażem, jeszcze zanim pojawią się z nim na lotnisku.

Flights: show

Departure: 2009-11-11 13:30:00 UTC
Arrival: 2009-11-11 14:30:00 UTC
Destination: Titchmarsh Island
Baggage allowance: 25.0
Capacity: 22

Listing seats

Name	Baggage	Action
Brent Chase	19.0	Show Edit Destroy
Tom Christie	15.0	Show Edit Destroy
Ryan Cleary	19.0	Show Edit Destroy
Julien Collard	16.0	Show Edit Destroy
Charlie Collins	19.0	Show Edit Destroy

New seat

Name: Sam Seaborn
Baggage: 110

[Create](#) [Edit](#) [Back](#)

Ktoś próbuje podróżować z o wiele cięższym bagażem, niż jest to dozwolone.



Zaostrz ołówek



W Rails dane sprawdzamy za pomocą validatora. Jak sądzisz, który z poniższych validatorów zapobiegnie rezerwacji miejsc przez użytkowników ze zbyt ciężkim bagażem?

- Żaden. Będziemy musieli napisać własny.
- validates_length_of
- validates_format_of
- validates_uniqueness_of
- validates_inclusion_of



Zaostrz ołówek

Rozwiążanie

W Rails dane sprawdzamy za pomocą validatora. Jak sądzisz, który z poniższych validatorów zapobiegnie rezerwacji miejsc przez użytkowników ze zbyt ciężkim bagażem?

- Żaden. Będziemy musieli napisać własny.
- validates_length_of
- validates_format_of
- validates_uniqueness_of
- validates_inclusion_of

Musimy napisać WŁASNY validator

Rails zawiera zbiór wbudowanych validatorów, które mogą wykonywać sporo podstawowych testów, takich jak na przykład to, czy dane są wprowadzane lub czy są poprawnie sformatowane. Czasami jednak konieczne jest sprawdzenie czegoś, czego podstawowe validatory nie obejmują.

W przypadku bagażu Rails nie zawiera specjalnego validatora validates_too_much_baggage. Nie istnieje również validator maksymalnej wartości. Musimy zatem napisać własny validator.

Jeśli w kodzie modelu Seat utworzymy metodę o nazwie validate, metoda ta zawsze będzie wywoływana przez obiekt modelu tuż przed zapisem bądź uaktualnieniem danych w bazie:

```
class Seat < ActiveRecord::Base
  def validate
    if name == flight_id
      errors.add_to_base("Your name is the same as your flight number")
    end
  end
end
```



To kolejny przypadek, w którym nazwy są w Rails niezwykle istotne. Dzięki użyciu konkretnej nazwy („validate”) Rails wie, co zrobić z tą metodą.

Polecenie errors.add_to_base(...) wstawia komunikat do listy błędów. Jeśli utworzony zostanie komunikat o błędzie, operacja zapisu lub uaktualnienia zostaje przerwana, a użytkownik powinien zostać odesłany do formularza w celu naprawienia błędu.



Ćwiczenie

Napisz własny validator sprawdzający, czy rezerwacja miejsca obejmuje bagaż mieszczący się w limicie dla danego lotu.

Metoda wyszukującą czy powiązanie?



Ćwiczenie
Rozwiążanie

Napisz własny validator sprawdzający, czy rezerwacja miejsca obejmuje bagaż mieszczący się w limicie dla danego lotu.

```
class Seat < ActiveRecord::Base
  def validate
    if baggage > Flight.find(flight_id).baggage_allowance
      errors.add_to_base("You have too much baggage")
    end
  end
```

Limit bagażu odczytujemy z obiektu lotu. Obiekt lotu możemy odczytać za pomocą metody wyszukującej oraz identyfikatora lotu.

By sprawdzić obiekt lotu z obiektu miejsca, wykorzystujemy metodę wyszukującą. Czy można w jakiś sposób wykorzystać tutaj powiązania?



Powiązania są lepsze od ręcznych metod wyszukujących.

Zamiast wykorzystywać metody wyszukujące do sprawdzania powiązanego obiektu lotu, można zdefiniować **powiązanie** między miejscami a lotami. Pytanie brzmi: jakiego rodzaju powiązanie jest nam potrzebne?

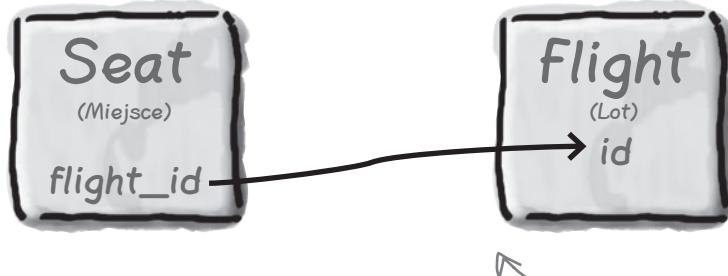
Gdy wcześniej tworzyliśmy powiązanie, nadaliśmy modelowi Flight nowy atrybut o nazwie seats:

```
@flight.seats
```

Ale co potrzebne jest nam teraz? Wcześniej mieliśmy obiekt Flight i chcieliśmy wiedzieć, jakie są powiązane z nim miejsca. Różnica polega na tym, że teraz sprawdzamy obiekt miejsca, a żeby to zrobić, musimy wiedzieć o powiązanym z nim lotem. Jakiego rodzaju powiązanie jest nam potrzebne tym razem?

Potrzebne nam jest ODWROTNÉ powiązanie

Tym razem potrzebujemy powiązań, które działa w *odwrotną stronę* niż to wykorzystane poprzednio. Mając określony obiekt miejsca, chcemy otrzymać powiązany z nim lot:



Chcemy w rezerwacjach miejsca mieć atrybut taki jak poniższy:

`@seat.flight`

Tym razem idziemy od miejsca do lotu... i potrzebny nam jest tylko jeden rekord — lot dla określonej rezerwacji miejsca.

Chcemy wiedzieć, do którego lotu przynależy rezerwacja miejsca.

A każda rezerwacja miejsca związana jest tylko z jednym lotem.

Jak sądzisz, jak będzie w tym przypadku wyglądał kod?



Magnesiki z kodem

By z obiektu miejsca otrzymać powiązany lot, konieczne jest dodanie kodu do modelu. Jednak do którego modelu i jakiego kodu? Użyj poniższych magnesików do uzupełnienia luk.

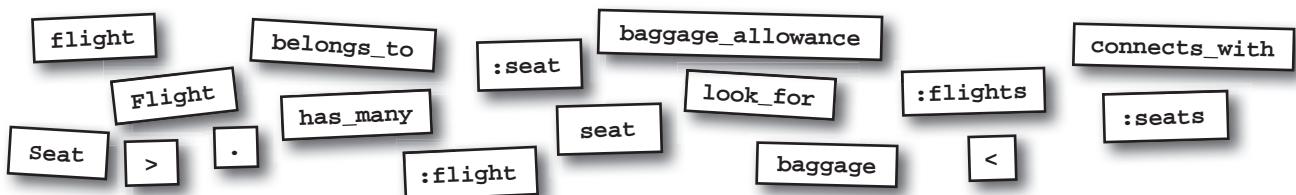
Powiązanie zostanie zdefiniowane w modelu

i będzie stanowiło polecenie wyglądające w następujący sposób:

.....

Wykorzystujący powiązanie warunek `if` w wyżej wymienionym modelu będzie wyglądał następująco:

`if`





Magnesiki z kodem: Rozwiążanie

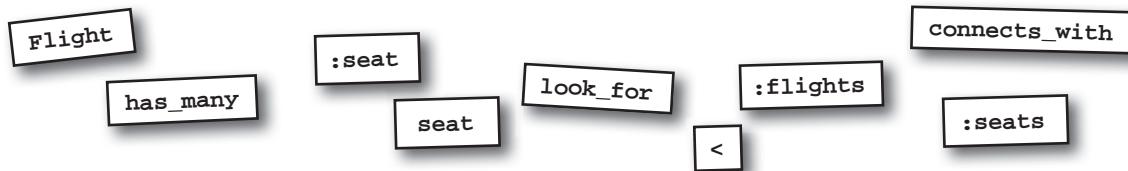
By z obiektu miejsca otrzymać powiązany lot, konieczne jest dodanie kodu do modelu. Jednak do którego modelu i jakiego kodu? Użyj poniższych magnesików do uzupełnienia luk.

Powiązanie zostanie zdefiniowane w modelu **seat** i będzie stanowiło polecenie wyglądające w następujący sposób:



Wykorzystując powiązanie warunek **if** w wyżej wymienionym modelu będzie wyglądał następująco:

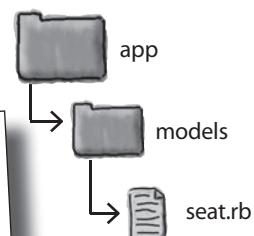
if **baggage** **>** **flight** **.** **baggage_allowance**



Jak wygląda teraz model Seat?

Wprowadźmy zmiany do modelu Seat:

```
class Seat < ActiveRecord::Base
  belongs_to :flight
  def validate
    if baggage > flight.baggage_allowance
      errors.add_to_base("You have too much baggage")
    end
  end
end
```



↑
Uaktualnij swoją wersję modelu „Seat”, tak by pasowała ona do tej.



Jazda próbna

Co się dzieje, kiedy teraz próbujesz zarezerwować miejsce z bagażem, którego ciężar przekracza limit dozwolony na lot? Sprawdź i sam się przekonaj...

Departure: 2009-11-11 13:30:00 UTC
Arrival: 2009-11-11 14:30:00 UTC
Destination: Titchmarsh Island
Baggage allowance: 25.0
Capacity: 22

Name	Baggage	Action
Brent Chase	19.0	Show
Brent Chase	19.0	Edit
Brent Chase	19.0	Destroy
Tom Christie	15.0	Show
Tom Christie	15.0	Edit
Tom Christie	15.0	Destroy
Ryan Cleary	19.0	Show
Ryan Cleary	19.0	Edit
Ryan Cleary	19.0	Destroy
Julien Collard	16.0	Show
Julien Collard	16.0	Edit
Julien Collard	16.0	Destroy
Charlie Collins	19.0	Show

New seat

Name: Sam Seaborn
Baggage: 110

Validator i powiązanie razem zajmują się problemem bagażu.

Seats: create

New seat

1 error prohibited this seat from being saved
There were problems with the following fields:

- You have too much baggage

Flight: 2
Name: Sam Seaborn
Baggage: 110



CELNE SPOSTRZEŻENIA



- Podział strony na **szablony częściowe** sprawi, że będzie ona łatwiejsza do utrzymania.
- Szablony częściowe, szablony stron oraz układy stron to trzy typy plików **Embedded Ruby**.
- Szablony częściowe wykorzystywane są do generowania **fragmentów** stron.
- Szablony stron tworzą **główną zawartość** strony.
- Układy stron wykorzystywane są do tworzenia standardowego **kodu HTML opakowującego strony**.
- Szablony częściowe mogą być **wywoływane przez** szablony stron lub układy stron.
- Szablonom częściowym można nadać **zmienne lokalne**.
- Szablony częściowe muszą się zaczynać od znaku „_”, a kończyć się rozszerzeniem `.html.erb`.
- Szablon częściowy wywołujemy za pomocą funkcji `render`.
- **Połączania** sprawiają, że łatwiej jest znajdująć w innych modelach połączone dane.
- Połączania działają jak **metody wyszukujące**.
- Atrybuty `has_many` zwracają tablice.
- Atrybuty `belongs_to` zwracają pojedyncze obiekty.
- Własne walidatory tworzy się, dodając do modelu metodę o nazwie `validate`.

Nie istniejąca glupie pytania

P: Czy naprawdę muszę dzielić swoją stronę na szablony częściowe?

O: Nie musisz, jednak zazwyczaj łatwiej jest utrzymać większą liczbę mniejszych plików.

P: Dlaczego tak jest?

O: Jeśli gdzieś pojawi się błąd, łatwiej będzie odnaleźć niepoprawny kod w większej liczbie mniejszych plików.

P: Jaki może być inny powód użycia szablonów częściowych?

O: Możliwość ponownego użycia. Jeśli masz standardowe menu czy część z informacjami kontaktowymi, możesz ich

użyć ponownie w różnych szablonach oraz układach stron.

P: W jaki sposób można wywołać szablon częściowy z układu strony?

O: Za pomocą metody `render`, tak jak w przypadku szablonu strony.

P: Czy zatem połączania działają dzięki łączeniu tabel za pomocą pól z kluczami?

O: Tak. Domyślnie połączania działają dzięki połączeniu pola `id` jednej tabeli z innym polem kończącym się na `_id` w innej tabeli. Dlatego właśnie `id` w tabeli lotów łączy się z `flight_id` w tabeli rezerwacji miejsc.

P: Zatem to, że kolumna tabeli rezerwacji miejsc nosi nazwę `flight_id`, ma znaczenie?

O: Tak. Gdybyś nie użył tej nazwy, platforma Rails nie wiedziałaby, jak zbudować połączanie.

P: Czy miało znaczenie, jaki typ danych reprezentuje `flight_id`?

O: Dobre pytanie. Musi to być liczba całkowita, ponieważ tego typu Rails używa w polach identyfikatorów.



Ćwiczenie

Rozbuduj swój validator w taki sposób, by sprawdzał również, czy liczba rezerwacji miejsc na lot nie przekracza liczby dostępnych miejsc.

[Wskazówka: wszystkie tablice mają metodę o nazwie `size` zwracającą liczbę elementów tablicy.]

```
class Seat < ActiveRecord::Base
  belongs_to :flight
  def validate
    if baggage > flight.baggage_allowance
      errors.add_to_base("You have too much baggage")
    end
    .....
    .....
    ...
  end
end
```

Metoda size zwraca rozmiar tablicy



Ćwiczenie

Rozbuduj swój validator w taki sposób, by sprawdzał również, czy liczba rezerwacji miejsc na lot nie przekracza liczby dostępnych miejsc.

Rozwiązanie

[Wskazówka: wszystkie tablice mają metodę o nazwie `size` zwracającą liczbę elementów tablicy.]

```
class Seat < ActiveRecord::Base
  belongs_to :flight
  def validate
    if baggage > flight.baggage_allowance
      errors.add_to_base("You have too much baggage")
    end
    if flight.seats.size >= flight.capacity . . .
      errors.add_to_base("The flight is fully booked") . . .
    end
  end
end
```

Nie istniejąca
grupie pytania

P: Hej, zaczekaj chwilę... Skąd warunek `>=?`? Czy nie sprawdzamy, kiedy jest więcej rezerwacji miejsc, niż jest to dozwolone?

O: Tak właśnie jest, ale pamiętaj — powiązanie działa jak metoda wyszukująca. Kiedy analizujemy `flight.seats`, odczytujemy rezerwacje miejsc z bazy danych. Sprawdzenie poprawności danych wykonywane jest *przed* zapisaniem w bazie danych nowej rezerwacji, dlatego nie będzie obejmowało właśnie dokonywanej rezerwacji. Z tego powodu potrzebujesz właśnie `>=`.



Jazda próbna

Wprowadź wszystkie zmiany omówione na poprzednich stronach i sprawdź raz jeszcze działanie aplikacji.

Name	Baggage
Brad Sturgeon	22.0
Show	
Kirk Avery	15.0
Show	
Drew Beurline	18.0
Show	
James Blake	19.0
Show	
Gaffiner Brilliant	25.0
Show	
Ted Brabin	19.0
Show	
Jesse Carr	15.0
Show	
Jack Casey	28.0
Show	
Tom Halpin	15.0
Show	
Jack Hampson	18.0
Show	
Stew Harris	17.0
Show	
Alex Hunt	19.0
Show	

New seat

Name:

Baggage:

[Edit](#) | [Back](#)

Strona wyświetla teraz poprawną liczbę rezerwacji dla lotu. Co się jednak stanie, kiedy ktoś będzie próbował zarezerwować miejsce na wyprzedany lot?

Seats: create

1 error prohibited this seat from being saved

There were problems with the following fields:

- The flight is fully booked

Flight:

Name:

Baggage:

Coconut Airways radzi sobie świetnie

System wystartował w Coconut Airways

Naszej linii lotniczej wiedzie się świetnie. Turyści oraz lokalni mieszkańców z łatwością korzystają z systemu. Samoloty nie są przeciążane zbyt dużym bagażem, a liczba rezerwacji nie przekracza puli dostępnych miejsc. Tak naprawdę pracownicy mogą teraz bardziej produktywnie spędzać czas, który zaoszczędzili...





Niezbędnik programisty Rails

Masz za sobą rozdział 6. i teraz
do swojego niezbędnika programisty
Rails możesz dodać umiejętność
wykorzystywania połączeń.

Narzędzia Rails

`render :partial=>„nazwa”` wyświetla `_nazwa.html.erb`.

Zmienną przekazuje się do szablonu częściowego
za pomocą:

`render :partial=>„nazwa”, :locals=>{:zmienna1=>„wartość1”}`

Własne walidatory tworzy się, dodając
do modelu metodę o nazwie „validate”.

`errors.add_to_base(...)` tworzy komunikat o błędzie.

`belongs_to` definiuje powiązanie z obiektem
do jego obiektu nadzawanego.

`has_many` to powiązanie odwrotne.

7. Ajax

Ograniczanie ruchu

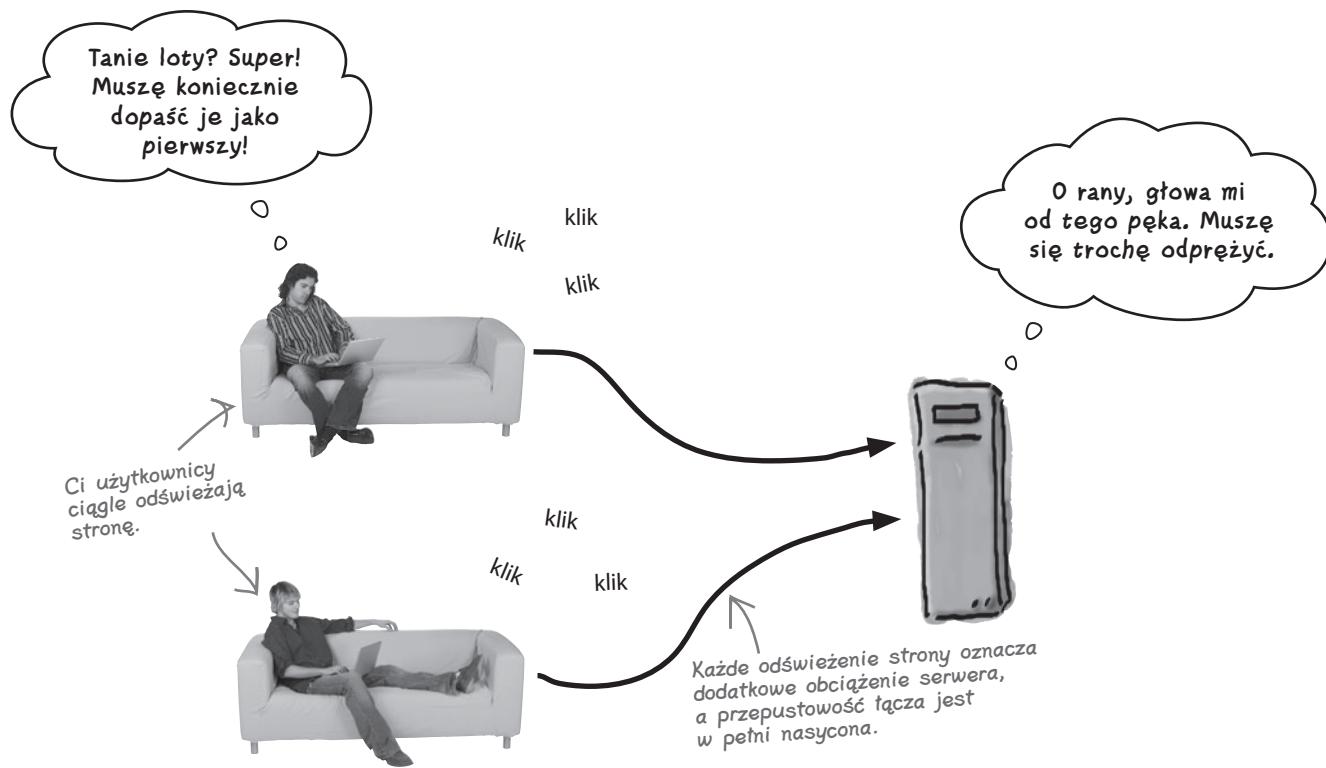


Każdy chce uzyskać z życia jak najwięcej... podobnie z aplikacją. Bez względu na to, jak jesteś dobry w obsłudze Rails, czasami tradycyjne aplikacje internetowe sobie nie radzą. Bywa, że użytkownicy pragną czegoś bardziej **dynamicznego**, czegoś, co odpowiada na wszystkie ich kaprysy. Ajax pozwala tworzyć **szylkie aplikacje internetowe z doskonałym czasem reakcji**, zaprojektowane tak, by użytkownik mógł **czerpać z Internetu jak najwięcej**. Rails ma wbudowany własny zestaw bibliotek Ajaks, które tylko czekają na to, aż ich użyjesz! Pora **szykko i łatwo dodać do aplikacji fantastyczne możliwości oferowane przez Ajax** i zachwycić jeszcze większą liczbę użytkowników.

Linie Coconut Airways mają nową ofertę

Linie Coconut Airways wprowadziły nową, promocyjną ofertę: ostatnie trzy miejsca w każdym locie sprzedawane są za pół ceny!

Pojawił się jednak pewien problem. Oczywiście każdy chce dostać te trzy ostatnie miejsca, dlatego ostatnią godzinę czy dwie przed czasem zamknięcia odprawy klienci powtarzają naciskanie przycisku *Odśwież* w przeglądarce w nadzieję trafienia na tani lot. Niestety, wzrost ruchu na stronie powoduje ogromne obciążenie serwera linii Coconut Airways.



Dodatkowe żądania powodują spowolnienie strony Coconut Airways. Tyle osób przesyła żądania dotyczące informacji lotów, które niedługo mają wystartować, że inni użytkownicy nie są w stanie przedrwać się przez stronę w celu zarezerwowania biletów na swoje loty. Firma Coconut Airways chce, byś ponownie zajął się aplikacją i sprawdził, czy da się zmniejszyć obciążenie serwera WWW.

Które części strony najbardziej się zmieniają?

Większość ruchu sieciowego trafia na stronę ze szczegółami lotu — tam w końcu są pokazane rezerwacje miejsc. Jest to strona wygenerowana za pomocą szablonu `app/views/flights/show.html.erb` oraz szablonów częściowych: `_seat_list.html.erb` i `_new_seat.html.erb`. Na stronie znajdują się trzy główne części:

Informacje o locie

Ta część zawiera limit bagażu dla lotu oraz maksymalną liczbę dostępnych miejsc.

Lista rezerwacji miejsc

Tutaj pokazywane są wszystkie aktualnie zarezerwowane miejsca.

Formularz rezerwacji miejsca

Użytkownicy wykorzystują go do zarezerwowania miejsca.

Za każdym razem gdy użytkownik naciśnie w przeglądarce przycisk *Odśwież*, wysyła do serwera żądanie całej strony. Oznacza to, że serwer musi ponownie wygenerować stronę z szablonu oraz szablonów częściowych, a całość opakować w układ strony lotu. Jeśli w tym samym czasie otrzymuje jedno lub dwa żądania, tak naprawdę nie jest to problemem, jednak w tej chwili serwer jest przeciążony ilością przetwarzania, jakie musi wykonać.

Czy możemy w jakiś sposób zredukować obciążenie serwera?

Zaostrz ołówek



Oto jak strony składane są przez serwer WWW. Zaznacz plik Embedded Ruby, który, Twoim zdaniem, generuje interesujące użytkownika uaktualnione informacje.



Szablon częściowy

Plik `_new_seat.html.erb`

Szablon częściowy

Plik `_seat_list.html.erb`

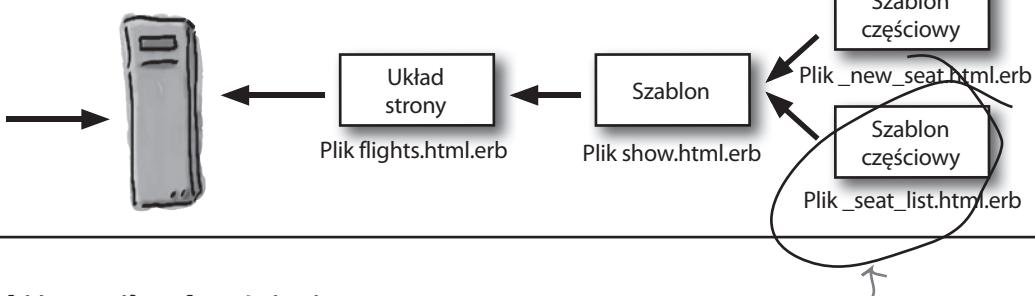
Uaktualnienie tego, co się zmienia

Zaostrz ołówek



Rozwiążanie

Oto jak strony składane są przez serwer WWW. Zaznacz plik Embedded Ruby, który, Twoim zdaniem, generuje interesujące użytkownika uaktualnione informacje.



Potrzebny nam jakiś sposób uaktualnienia jedynie listy zarezerwowanych miejsc

Kiedy użytkownicy odświeżają stronę, większa jej część się nie zmienia. Jedyną częścią, która może kiedykolwiek się zmienić, jest część wyświetlająca rezerwacje miejsc.

Co tak naprawdę dzieje się, kiedy użytkownik kliknie przycisk *Odśwież* przeglądarki? Przycisk ten mówi przeglądarce, że ma ponownie zażądać całej strony, ponieważ cała strona to jedyne, co jest dostępne. Aplikacja nie pozwala w tej chwili przeglądarce na zażądanie czegokolwiek mniejszego. Być może jest tak, że jedyną interesującą częścią strony jest lista zarezerwowanych miejsc, jednak przeglądarka może tę listę otrzymać, *jedynie* żądając całej strony.

Odświeżenie strony powoduje zażądanie jej w całości, ale tak naprawdę wystarczyłoby nam zażądanie szablonu częściowego listy rezerwacji miejsc.

Pierwsze, co musimy zrobić, to zmodyfikowanie aplikacji w taki sposób, by interesująca nas część strony — lista miejsc — była dostępna za pomocą osobnego żądania. Musimy pozwolić, by przeglądarka mogła zażądać określonego adresu URL, który wygeneruje *jedynie* listę zarezerwowanych miejsc.

Zaostrz ołówek



Jak zdefiniowałbyś w pliku `routes.rb` trasę odpowiadającą żądaniu `/flights/:flight_id/seats` dla akcji o nazwie `flight_seats` w kontrolerze `seats`?

.....
.....



Magnesiki z kodem

Uzupełnij metodę `flight_seats` w kontrolerze `seats`:

```
def flight_seats
  .... = .....(params[.....])
  .... => ..... , :locals=>{:seats=>.....}
  ....
end
```

`@flight`

`:partial`

`Flight.find`

`"flights/seat_list"`

`:flight_id`

`@flight.seats`

`render`



Zaostrz ołówek

Rozwiążanie

Jak zdefiniowałbyś w pliku `routes.rb` trasę odpowiadającą żądaniu `/flights/:flight_id/seats` dla akcji o nazwie `flight_seats` w kontrolerze `seats`?

```
map.connect '/flights/:flight_id/seats', :action=> 'flight_seats', :controller=>'seats'
```

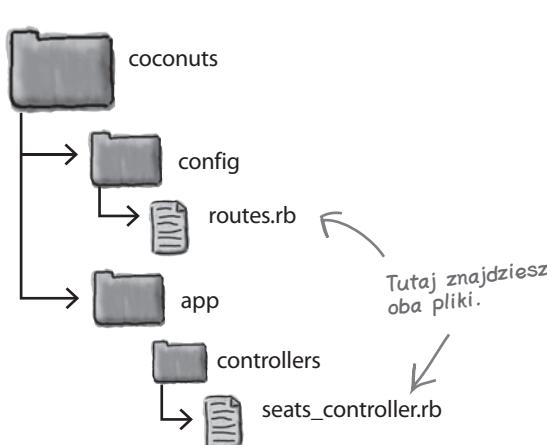


Magnesiki z kodem: Rozwiążanie

Uzupełnij metodę `flight_seats` w kontrolerze `seats`:

Pamiętaj o dodaniu tej trasy w pliku `config/routes.rb` "ponad" istniejącymi trasami.

```
def flight_seats
  @flight = Flight.find(params[:flight_id])
  render :partial => "flights/seat_list", :locals=>{ :seats=>@flight.seats }
end
```





Jazda próbna

Wyobraź sobie, że dla lotu o identyfikatorze 2 miejsca są już zarezerwowane.

Jeśli spróbujemy udać się pod adres:

`http://localhost:3000/flights/2/seats`

co powinniśmy zobaczyć?

Name	Baggage		
Brad Sturgeon	22.0	Show	Edit Destroy
Kirk Avery	15.0	Show	Edit Destroy
Drew Beurline	18.0	Show	Edit Destroy
James Blake	19.0	Show	Edit Destroy
Gafiner Brillant	25.0	Show	Edit Destroy
Ted Brisbin	19.0	Show	Edit Destroy
Jesse Carr	15.0	Show	Edit Destroy
Jack Casey	28.0	Show	Edit Destroy
Tom Halpin	15.0	Show	Edit Destroy
Jack Hampson	18.0	Show	Edit Destroy
Stew Harris	17.0	Show	Edit Destroy

Utworzona trasa powinna odwzorować `/flights/2/seats` na akcję `flight_seats` oraz kontroler `seats`, a także utworzyć nowy parametr żądania `flight_id = 2`. Kontroler wyszukuje lot numer 2 w bazie danych, a następnie generuje kod HTML z szablonu częściowego `_seat_list.html.erb` i zwraca go do przeglądarki.

Przyjrzyj się wygenerowanemu przez kontroler kodowi HTML widocznemu z prawej strony. Co widzisz?

Zwracany kod HTML nie jest tak naprawdę pełną stroną internetową, a jedynie jej **fragmentem**. Co z nim jednak zrobimy? Nie możemy przecież żądać, by użytkownicy patrzyli na tę stronę, zamiast udać się na stronę lotu — nie będzie to wyglądało zbyt dobrze. W końcu w którymś momencie użytkownicy będą chcieli zarezerwować miejsce, dlatego chcemy, by pozostały na stronie lotu.

Musimy w jakiś sposób zmusić przeglądarkę do zażądania tego fragmentu strony, a następnie użycia go do aktualnienia listy rezerwacji miejsc na stronie.

Ale jak możemy to zrobić?

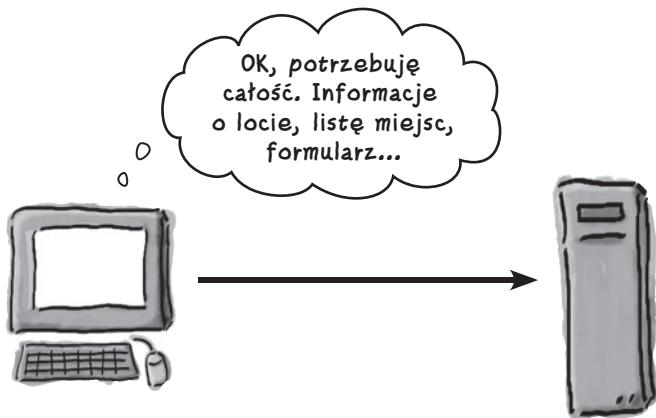
```
<h1>Listing seats</h1>
<table>
  <tr>
    <th>Name</th>
    <th>Baggage</th>
  </tr>
  <tr>
    <td>Brent Chase</td>
    <td>19.0</td>
  </tr>
```

Oto fragment strony wygenerowany przez kontroler z szablonu częściowego `_seat_list.html.erb`.

Przeglądarki uaktualniają całe strony

Czy przeglądarka nie uaktualnia zawsze całą strony?

W momencie gdy użytkownik naciśnie przycisk *Odwieź*, przeglądarka żąda *całej* strony internetowej.



Zła wiadomość jest taka, że **przeglądarka zawsze zrobi tylko to**. Żądanie całej strony jest wbudowane w mózg przeglądarki. Przycisk *Odwieź* oznacza „odśwież całą stronę” — i właśnie to się stanie, bez względu na okoliczności.

Ale dlaczego tak jest?

Pod maską przeglądarki działają jedynie na całych stronach internetowych. W języku HTML nie ma nic, co pozwoliłoby przeglądarce zażądać jedynie części strony — zawsze jest to „wszystko albo nic”. Nie ma znaczenia, że mamy teraz ogólnodostępny fragment strony internetowej. Nie istnieje możliwość, by przeglądarka sama żądała fragmentu strony i go wykorzystała.

W jaki sposób możemy zatem obejść ten problem?

Na szczęście dla nas istnieje sztuczka, którą możemy wykorzystać w celu zmuszenia przeglądarki do uaktualnienia jedynie części strony. Sztuczka ta jest następująca:

Zmusimy coś INNEGO od przeglądarki do wykonania żądania.

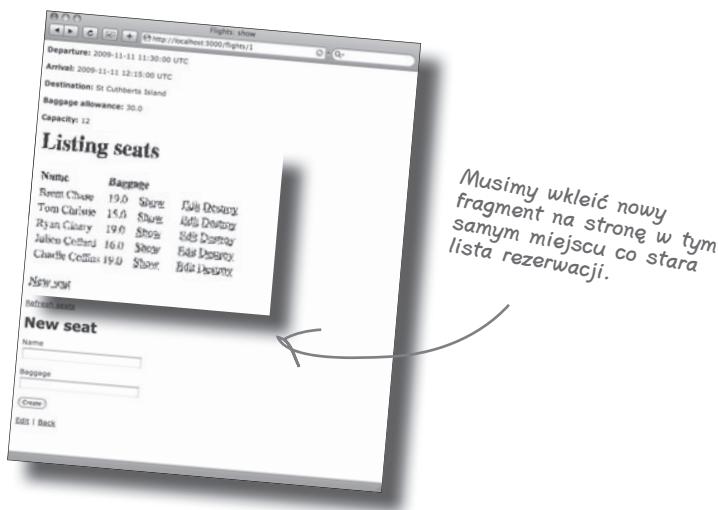
Co INNEGO może wykonać żądanie?

Wewnątrz każdej przeglądarki znajduje się **moduł obsługi języka JavaScript**. JavaScript pozwala na modyfikację normalnego działania przeglądarki. Jest w stanie dynamicznie zmieniać wygląd strony internetowej, uaktualniać zawartość wyświetlanego kodu HTML, a także odpowiadać na różne zdarzenia wewnętrz strony, takie jak naciskanie przycisków. Co ważniejsze, JavaScript potrafi również **wykonywać żądania** niezależnie od przeglądarki.

Co jednak tak naprawdę oznacza tu słowo „**niezależnie**”? To prawda, że JavaScript może nakazać przeglądarce przejście do innej strony, jednak jest także w stanie zrobić coś o wiele bardziej subtelnego.

W tle JavaScript może po cichu wykonywać żądania do serwera WWW i odczytywać zawartość tego, co odsyła serwer. Wszystko to może się dziać *bez przeniesienia przeglądarki do innego adresu URL*. JavaScript może w tle wykonać dziesiątki, a nawet setki żądań, a Ty nic nie zauważysz. Przeglądarka będzie wyglądała tak, jakby po prostu wyświetlała stronę.

Jest to takie istotne, ponieważ JavaScript może wykonać **żądanie w tle**, inaczej **żądanie asynchroniczne**, prosząc o najświeższą wersję listy rezerwacji miejsc. Po zwróceniu fragmentu strony JavaScript może wykorzystać ten fragment do uaktualnienia części strony wyświetlającej listę zarezerwowanych miejsc.



Do wykonania żądania możemy zamiast przeglądarki wykorzystać JavaScript. W ten sposób nie będziemy musieli odświeżać całej strony, a sama witryna internetowa będzie reagować o wiele szybciej.

Użycie JavaScriptu do uaktualnienia bieżącej strony znane jest pod nazwą **Ajax**. Platforma Rails ma wbudowaną obsługę tej technologii. Jak jednak możemy z niej korzystać?

Najpierw musimy dołączyć biblioteki Ajaksa...

Jak jednak możemy zmusić JavaScript przeglądarki do wykonania asynchronicznych żądań? Ten typ przetwarzania może być dość skomplikowany.

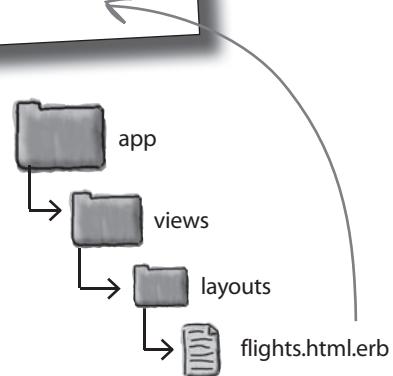
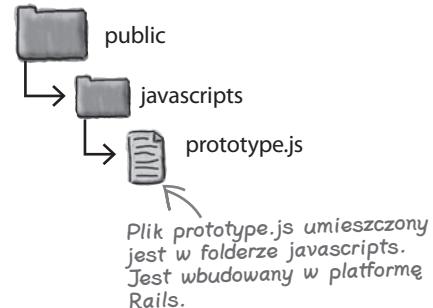
Tak naprawdę w celu wykonania wewnętrz przeglądarki żądań opartych na Ajaksie należy wykonać sporą ilość kodu w JavaScriptcie. Kod ten nie tylko obsługuje wszystkie szczegóły przetworzenia żądania, ale będzie to musiał zrobić w sposób zgodny z wszystkimi najważniejszymi przeglądarkami. Utworzenie i debugowanie takiego kodu byłoby koszmarem, dlatego większość aplikacji opartych na Ajaksie dla uproszczenia korzysta ze standardowych bibliotek JavaScriptu. W Rails wbudowana jest taka właśnie biblioteka o nazwie *Prototype*.

Biblioteka Prototype znajduje się w pliku o nazwie *prototype.js* umieszczonym w folderze *javascripts*. Choć jednak biblioteka znajduje się w kodzie aplikacji, nie zostanie automatycznie dołączona do stron internetowych generowanych przez aplikację. By udostępnić *Prototype* przeglądarce, konieczne jest odwołanie się do tej biblioteki w układach strony aplikacji:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <title>Flights: <%= controller.action_name %></title>
    <%= stylesheet_link_tag 'scaffold' %>
    <%= javascript_include_tag 'prototype' %> ← Ten wiersz zapewnia
                                                udostępnienie biblioteki
                                                Prototype przeglądarce.
</head>
<body>
<p style="color: green"><%= flash[:notice] %></p>
<%= yield %>
</body>
</html>
```

Metoda pomocnicza *javascript_include_tag* umożliwia pobranie przez przeglądarkę biblioteki Prototype z właściwego adresu URL.

Po zainstalowaniu biblioteki JavaScriptu na stronach internetowych będziesz gotowy do tworzenia własnego kodu Ajaksa.



...a następnie dodać odnośnik „Odśwież” oparty na Ajaksie

Biblioteka Ajaksa ułatwia wykonywanie asynchronicznych żądań do serwera, jednak *nie napisze* za Ciebie niezbędnego Ci własnego kodu. Jakiego typu własny kod jest nam potrzebny?

Problem z przeciążeniem serwera spowodowany został przez użytkowników naciskających raz za razem przycisk *Odśwież* w swoich przeglądarkach, co sprawia, że system działa wolniej nie tylko dla nich, ale i dla pozostałych użytkowników. Możemy ten problem obejść, udostępniając użytkownikom na stronie odnośnik podpisany „Odśwież”. Odnośnik ten aktualni jedynie listę rezerwacji miejsc na stronie, a ponieważ powoduje pobranie mniejszej ilości kodu HTML, będzie działał szybciej od przycisku *Odśwież* przeglądarki. Zmniejszy również obciążenie serwera, co ułatwi życie pozostałym klientom.

Jak będzie zatem działał odnośnik „Odśwież”? Ajax działa w pełni za pomocą JavaScriptu, dlatego odnośnik będzie musiał generować zdarzenie JavaScriptu. Zdarzenie z odnośnika wywoła bibliotekę Prototype i nakaże jej wykonanie żądania otrzymania najświeższej części `seat_list` ze strony. Kiedy kod HTML zostanie zwrócony z przeglądarki, JavaScript dynamicznie zastąpi listę zarezerwowanych miejsc na stronie za pomocą nowego kodu.

Jak powinien wyglądać taki kod?

Zaostrz ołówek



Oto kod dodający odnośnik w JavaScriptie do szablonu `show.html.erb` lotu. Napisz, co, Twoim zdaniem, robi każda z części kodu.

```
<div id="seats"> .....  
<%= render :partial=>"seat_list", :locals=>{:seats=>@flight.seats} %>  
</div>  
  
<%= link_to_remote( .....  
  "Refresh seats", .....  
  :url=>"/flights/#{@flight.id}/seats", .....  
  :method=>"get", .....  
  :update=>"seats") %> .....  
  
<%= render :partial=>"new_seat", :locals=>{:seat=>  
  Seat.new(:flight_id=>@flight.id)} %>
```



Zaostrz ołówek

Rozwiążanie

Oto kod dodający odnośnik w JavaScriptie do szablonu `show.html.erb` lotu. Napisz, co Twoim zdaniem robi każda z części kodu.

```
<div id="seats"><!-- Nazywamy część strony, która się zmienia.-->
<%= render :partial=>"seat_list", :locals=>{:seats=>@flight.seats} %>
</div>

<%= link_to_remote( -- Tworzy odnośnik z JavaScriptem uaktualniającym miejsca.
  "Refresh seats", -- To jest podpis odnośnika.
  :url=>"/flights/#{@flight.id}/seats", -- To adres URL, z którego będzie pochodzić nowa lista miejsc.
  :method=>"get", -- To oznacza, że jedynie wczytujemy, a nie uaktualniamy dane.
  :update=>"seats" ) -- To identyfikator części strony, którą uaktualniamy.

<%= render :partial=>"new_seat", :locals=>{:seat=>
  Seat.new(:flight_id=>@flight.id)} %>
```

Kiedy Embedded Ruby przetwarza szablon `show.html.erb`, generuje odnośnik HTML wywołujący po kliknięciu biblioteki Ajаксa:

Odnosnik wywołuje bibliotekę Ajaksą (Prototype) w momencie wystąpienia zdarzenia kliknięcia.

Ta metoda pomocnicza generuje ten kod HTML.

```
</div>
<a href="#" onklik="new Ajax.Updater('seats', '/flights/1/
seats', {asynchronous:true, evalScripts:true, method:'get',
parameters:'authenticity_token=' + encodeURIComponent('7cb578
0328778ef35ee9d26689784bba0d562170')); return false;">Refresh
seats</a>
```

```
<h1>New seat</h1>
```

Nieistniejąca grupa pytań

P: Czym jest żądanie asynchronousne?

O: Żądanie asynchronousne to żądanie, które wykonywane jest w tle. Żądania asynchronousne generowane są przez JavaScript.

P: Czym różnią się one od normalnych żądań?

O: Normalne żądania generowane są wtedy, gdy użytkownik kliknie odnośnik lub wpisze adres URL. Żądania asynchronousne generowane są przez JavaScript w odpowiedzi na zdarzenie.

P: Czy odświeżenie strony naprawdę jest aż takim obciążeniem dla serwera i łącza sieciowego?

O: Może tak być, jeśli pozostała część strony wymaga dużych ilości kodu HTML. Przeglądarka może też próbować przeładować obrazki strony, które są dużym obciążeniem dla łącza. Utworzenie niektórych części strony może także wymagać dużej ilości przetwarzania. Ajax pozwala pozostawić te części strony bez zmian, co obniża obciążenie serwera.

P: Czy żeby pisać kod oparty na Ajaksie, muszę znać język JavaScript?

O: Rails wygeneruje za Ciebie kod oparty na Ajaksie, dlatego nie musisz się uczyć języka JavaScript. Jeśli jednak znasz JavaScript, będziesz miał większą kontrolę nad tym, jak wykonywane są wywołania Ajaksa, a także będziesz mógł lepiej zrozumieć, jak działa aplikacja.

P: Wygenerowany JavaScript tworzy parametr o nazwie authenticity_token. Do czego on służy?

O: Token uwierzytelniający wykorzystywany jest przez Rails do zapewnienia, że żądanie pochodzi ze strony wygenerowanej przez Rails. Bez niego Rails odrzuci żądanie.

P: Jak działa taki token?

O: To wartość wygenerowana przez Rails. Obecność tej wartości w żądaniu pokazuje, że żądanie pochodzi ze strony utworzonej przez Rails, a nie z jakiejś aplikacji zewnętrznej, która próbuje uzyskać dostęp do naszego systemu.

P: Twierdzicie, że żądania oparte na Ajaksie wysyłane są przez JavaScript, a nie przeglądarkę, jednak czy JavaScript nie jest po prostu częścią przeglądarki?

O: Tak, jednak moduł JavaScriptu jest w stanie wykonywać żądania, które nie są częścią normalnej sekwencji korzystania z przeglądarki — o to właśnie chodzi. Żądania oparte na Ajaksie pozwalają uaktualniać części strony bez wykonywania żądań pełnej strony i bez modyfikowania historii stron przeglądarki.

P: Dlaczego skorzystaliśmy z metody pomocniczej javascript_include_tag, zamiast po prostu wprowadzić kod HTML ładujący JavaScript?

O: Gdybyś chciał samodzielnie pisać kod HTML, zwykły kod HTML zadziałałby, jednak programiści Rails zazwyczaj starają się korzystać z metod pomocniczych wszędzie tam, gdzie jest to możliwe. Metody pomocnicze są zazwyczaj nieco krótsze od wpisanego ręcznie kodu HTML, a ponadto uzupełniają za Ciebie pewne konfiguracje specyficzne dla aplikacji. Przykładowo metoda javascript_include_tag uzupełni standardową ścieżkę do skryptów: /javascripts/....

P: Nie brzmi to szczególnie pomocnie.

O: Metoda ta dodaje również losową liczbę na końcu lokalizacji JavaScriptu.

P: Do czego to służy?

O: Oznacza to, że jeśli ktoś odświeży stronę w przeglądarce, przeglądarka pobierze również nową kopię biblioteki JavaScriptu. W ten sposób jeśli zmienisz cokolwiek w bibliotece, przeglądarka zawsze zażąda najnowszej wersji.

Jazda próbna



Jazda próbna

Kiedy mamy już odnośnik oparty na JavaScriptie, sprawdźmy, jak wygląda aplikacja. Przeładuj aplikację i wypróbowuj jej działanie.

1

Pierwszy użytkownik udaje się na stronę lotu w celu zarezerwowania miejsca.

Widzi szczegóły lotu, a także listę miejsc już zarezerwowanych oraz formularz rezerwacji. Pomiędzy listą miejsc a formularzem rezerwacji znajduje się nowy odnośnik oparty na Ajaksie.

Name	Baggage	Show	Edit	Destroy
Brent Chase	19.0	Show	Edit	Destroy
Tom Christie	15.0	Show	Edit	Destroy
Ryan Cleary	19.0	Show	Edit	Destroy
Julien Collard	16.0	Show	Edit	Destroy
Charlie Collins	19.0	Show	Edit	Destroy

[Refresh seats](#)

New seat

Name

Baggage

[Create](#)

[Edit](#) | [Back](#)



2

Drugi użytkownik odwiedza stronę i rezerwuje miejsce.

Po przesłaniu formularza strona zostaje u tego użytkownika odświeżona i widzi on nową rezerwację. Tak samo jak każdy, kto teraz otworzy tę stronę. Co jednak z pierwszym użytkownikiem?

Name	Baggage	Show	Edit	Destroy
Brent Chase	19.0	Show	Edit	Destroy
Tom Christie	15.0	Show	Edit	Destroy
Ryan Cleary	19.0	Show	Edit	Destroy
Julien Collard	16.0	Show	Edit	Destroy
Charlie Collins	19.0	Show	Edit	Destroy
Jesse James Garrett	12	Show		

[Refresh seats](#)

New seat

Name

Baggage

[Create](#)

[Edit](#) | [Back](#)





Flights: show
http://localhost:3000/flights/2

Name	Baggage	
Brent Chase	19.0	Show Edit Destroy
Tom Christie	15.0	Show Edit Destroy
Ryan Cleary	19.0	Show Edit Destroy
Julien Collard	16.0	Show Edit Destroy
Charlie Collins	19.0	Show Edit Destroy
Jesse James Garrett	12.0	Show Edit Destroy

[Refresh seats](#)

New seat

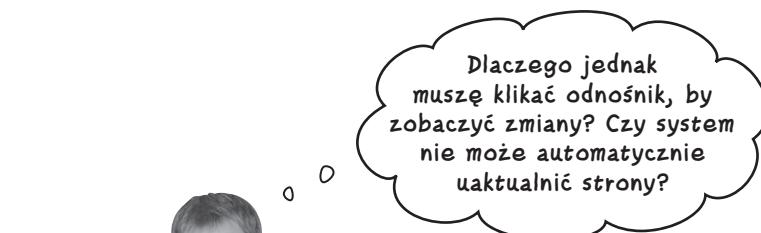
Name

Baggage

3

Pierwszy użytkownik może zobaczyć nową rezerwację dzięki kliknięciu odnośnika odświeżającego listę.

Odnośnik ten wyzwala zdarzenie JavaScriptu wywołujące bibliotekę Ajaxa i odświeża listę rezerwacji, pokazując nową rezerwację.



System działa doskonale, jednak niektórzy użytkownicy zastanawiają się, czemu muszą siedzieć przy komputerze i ciągle klikać odnośnik tylko po to, by przekonać się, czy są jakieś nowe rezerwacje. O wiele wygodniej byłoby, gdyby strona mogła w jakiś sposób automatycznie odkryć, że pojawiły się nowe rezerwacje.

Czy jest to jednak możliwe?

Przeglądarka musi prosić o uaktualnienie

Automatyczne uaktualnienie strony jest jednak problematyczne ze względu na sposób funkcjonowania Internetu.

W idealnym świecie aplikacja internetowa byłaby w stanie poinformować użytkownika o każdej zmianie w liście zarezerwowanych miejsc. Niestety, serwery WWW nie działają w ten sposób. Przemawiają jedynie wtedy, kiedy ktoś do nich przemówi.



Serwer odeśle **odpowiedź** jedynie wtedy, gdy sam otrzyma **żądanie**. Nawet jeśli serwer ma nowe informacje, o których chciałby poinformować przeglądarkę, i tak nie może nic zrobić. Musi czekać, aż przeglądarka go o te nowe informacje *poprosi*.

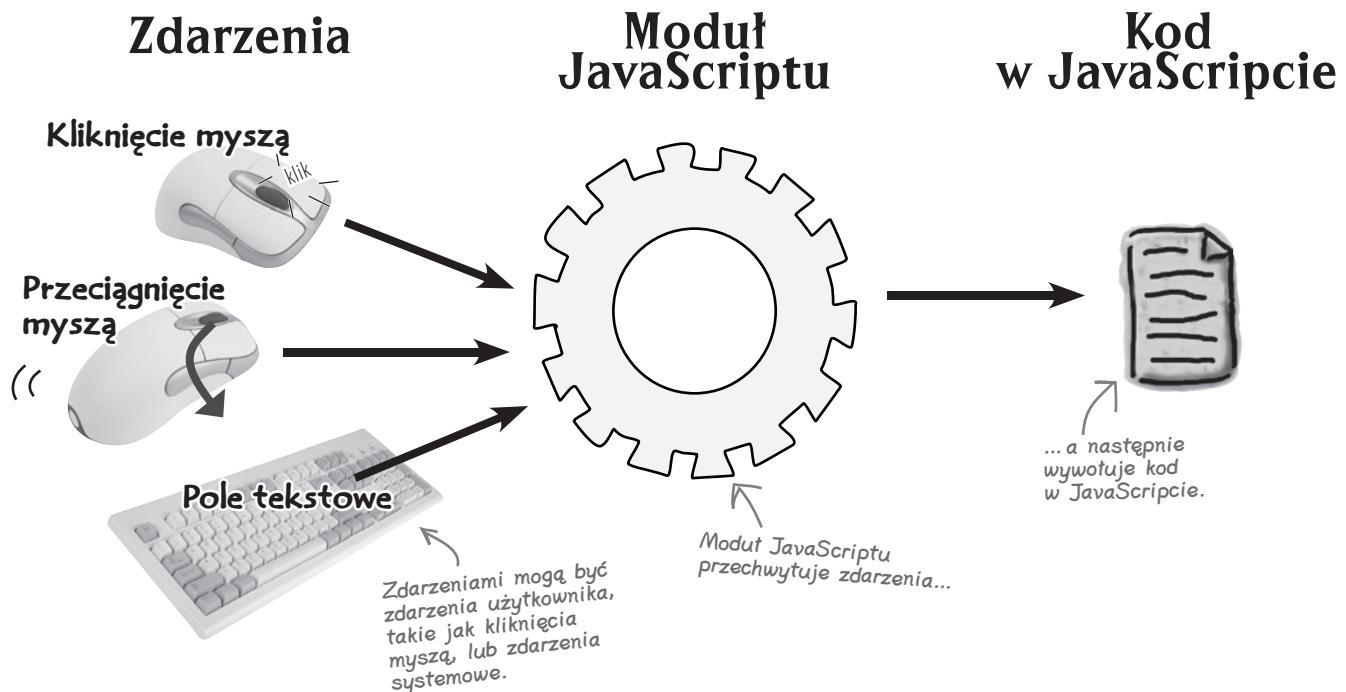
Oznacza to, że jeśli chcielibyśmy, by przeglądarka była automatycznie informowana o każdej zmianie listy miejsc, zawiedziemy się. Zamiast tego musimy sprawić, by przeglądarka ciągle prosiła. I prosiła. I prosiła...



Czy jednak POWINNIŚMY nakazywać przeglądarkę nieustanne proszenie?

Zastanów się jeszcze raz nad sposobem działania odnośnika odświeżającego listę innego na Ajaksie. Kiedy ktoś go kliknie, odnośnik generuje zdarzenie JavaScriptu, co z kolei wywołuje bibliotekę Prototype, prosząc o pobranie nowej wersji listy zarezerwowanych miejsc.

Kluczowym elementem jest to, że wszystko zaczyna się od *zdarzenia* — czegoś, co dzieje się poza JavaScriptem.



Fragment kodu w JavaScriptcie może zostać zarejestrowany wraz ze zdarzeniem, co oznacza, że kiedy zdarzenie wystąpi, kod w JavaScriptcie zostaje wykonany.

W naszej sytuacji musimy wykonywać ten sam kod w JavaScriptcie raz za razem na nowo. Jakiego rodzaju zdarzenie może tego dokonać? Z pewnością nie będzie to zdarzenie wygenerowane przez działalność użytkownika. Zamiast tego musimy zarejestrować kod w JavaScriptcie ze zdarzeniem **licznika**.

Licznik to zdarzenie systemowe, które pojawia się w regularnych interwałach, zazwyczaj co kilka sekund. Musimy utworzyć licznik, a następnie zarejestrować w połączeniu z nim kod w JavaScriptcie odświeżający listę zarezerwowanych miejsc.

Na szczęście Rails może nam w tym pomóc.

Licznik obsługuje się podobnie jak przycisk czy odnośnik

Jedyna różnica pomiędzy wykonaniem fragmentu kodu opartego na Ajaksie w momencie naciśnięcia przycisku bądź kliknięcia odnośnika a wykonywaniem go raz za razem co kilka sekund sprowadza się tak naprawdę do typu oczekiwanej zdarzenia.

Z tego powodu kod w języku Ruby umieszczany w szablonie strony przypomina właściwie kod użyty do utworzenia odnośnika w JavaScriptie:

```
<%= periodically_call_remote(  
  :url=>"_____", ↗ To adres URL, z którego będzie pochodzić nowa lista miejsc.  
  :method=>"get", ↗ To oznacza, że jedynie wczytujemy, a nie aktualniemy dane.  
  :update=>"_____", ↗ To identyfikator części strony, którą aktualniemy.  
  :frequency=>"__" %> ↗ Liczba sekund pomiędzy zdarzeniami licznika.
```

Powyższy kod w języku Ruby utworzy kod w JavaScriptie wykonujący żądanie nowej listy miejsc co kilka sekund. Następnie aktualnia on określoną część strony za pomocą kodu HTML zwróconego przez serwer. Jedyne rzeczywiste różnice pomiędzy tą metodą pomocniczą a kodem tworzącym w JavaScriptie odnośnik są następujące:

- 1 Przycisk bądź odnośnik potrzebuje tekstu podpisu.
- 2 Licznikowi należy podać częstotliwość żądania.

Zaostrz ołówek



Strona lotu powinna nadal zawierać odnośnik odświeżający listę rezerwacji, jednak musi również zawierać kod licznika. Napisz kod licznika, który dodamy do szablonu `app/views/flights/show.html.erb` w celu uaktualnienia listy miejsc trzy razy na minute:

Nie istnieja
głupie pytania

P: Czy naprawdę nie istnieje żaden sposób, by serwer mógł się skontaktować z przeglądarką?

U: Przeglądarka mogłaby utrzymywać otwarte połączenie z serwerem, jednak wymagałoby to ogromnej liczby połączeń dla nawet marginalnie popularnych aplikacji. O wiele częściej stosowanym rozwiążaniem jest odpytywanie serwera.

P: Czy częstotliwość zawsze podawana jest w sekundach?

U: Tak, częstotliwość zawsze podaje się w sekundach. To, że nazywamy to „częstotliwością”, może się wydawać dziwne, ponieważ tak naprawdę wcale nie podajemy częstotliwości (czyli tego, ile razy na minutę coś się dzieje). W rzeczywistości podajemy „okres”, czyli ilość czasu pomiędzy kolejnymi aktywnościami.

P: Jaka jest częstotliwość domyślna?

C: Domyślnie częstotliwość ma wartość dziesięciu sekund

P: Skąd pochodzi identyfikator części strony?

O: Każdy znacznik w kodzie HTML może otrzymać identyfikator. Jest to unikalne odwołanie oznaczające określoną część strony. Zazwyczaj aplikacje oparte na Ajaksie opakowują jakąś część strony w znacznik `<div>` z identyfikatorem. Pozwala to na nadanie identyfikatora jednemu znacznikowi lub całej grupie znaczników HTML za jednym razem.



Zaostrz ołówek

Rozwiążanie

Strona lotu powinna nadal zawierać odnośnik odświeżający listę rezerwacji, jednak musi również zawierać kod licznika. Napisz kod licznika, który dodamy do szablonu `app/views/flights/show.html.erb` w celu uaktualnienia listy miejsc trzy razy na minutę:

```
<%= periodically_call_remote(
  :url=>"/flights/#{@flight.id}/seats",
  :method=>"get",
  :update=>"seats",
  :frequency=>"20") %>
```

Twój szablon `show.html.erb` powinien teraz zawierać poniższy kod:

```
<%=h @flight.baggage_allowance %>
</p>
<p>
  <b>Capacity:</b>
  <%=h @flight.capacity %>
</p>
<div id="seats">
<%= render :partial=>"seat_list", :locals=>{ :seats=>@flight.seats} %>
</div>
<%= link_to_remote
  "Refresh seats", :url=>"/flights/#{@flight.id}/seats",
  :method=>"get", :update=>"seats" %>
<%= periodically_call_remote(
  :url=>"/flights/#{@flight.id}/seats",
  :method=>"get", :update=>"seats", :frequency=>"20") %>
<%= render :partial=>"new_seat", :locals=>{ :seat=>Seat.new(:flight_id=>@flight.id)} %>
```

To jest dolna część pliku show.html.erb.



Jazda próbna

Po dodaniu kodu licznika system powinien automatycznie uaktualniać listę rezerwacji miejsc bez konieczności odświeżania strony przez użytkownika.

- 1** Pierwszy użytkownik udaje się na stronę lotu w celu zarezerwowania miejsca.
Widzi szczegóły lotu.

Flights: show
http://localhost:3000/flights/2

Name	Baggage	
Brent Chase	19.0	Show Edit Destroy
Tom Christie	15.0	Show Edit Destroy
Ryan Cleary	19.0	Show Edit Destroy
Julien Collard	16.0	Show Edit Destroy
Charlie Collins	19.0	Show Edit Destroy

[Refresh seats](#)

New seat

Name

Baggage

[Edit](#) | [Back](#)



- 2** W czasie gdy rezerwuje swoje miejsce, strona odwiedza drugi użytkownik.
Szybko rezerwuje miejsce i przesyła swoje dane.

New seat

Name

Baggage

[Edit](#)



- 3** Pierwszy użytkownik automatycznie widzi nową rezerwację miejsca.
Choć pierwszy użytkownik nie dotknął klawiatury, strona automatycznie uaktualnia listę zarezerwowanych miejsc w ciągu 20 sekund.

Nowa rezerwacja pokazuje się automatycznie.

Flights: show
http://localhost:3000/flights/2

Name	Baggage	
Brent Chase	19.0	Show Edit Destroy
Tom Christie	15.0	Show Edit Destroy
Ryan Cleary	19.0	Show Edit Destroy
Julien Collard	16.0	Show Edit Destroy
Charlie Collins	19.0	Show Edit Destroy
Jesse James Garrett	12.0	Show Edit Destroy

[Refresh seats](#)

New seat





Cała prawda o Ajaksie

Wywiad tygodnia:
Różne typy średnich

Head First: Cześć, Ajax — witamy serdecznie. Tak miło z twojej strony...

Ajax: Cała przyjemność po mojej stronie.

Head First: ...że zgodołeś się z nami dziś porozmawiać.

Ajax: Oj, przerwałem ci.

Head First: Można tak...

Ajax: Bardzo często to robię. Strasznie mi przykro. Bywam nieco, jak by to powiedzieć, hiperaktywny.

Head First: Jesteś chyba bardzo pracowitą technologią?

Ajax: Widzisz to? Właśnie uaktualniłem tabelę z danymi! Co — technologią? Nie jestem żadną technologią. Jestem sposobem życia, maleńka! A przynajmniej sposobem pisania aplikacji internetowych.

Head First: Co masz na myśli?

Ajax: No cóż, Rails, JavaScript, Prototype — to wszystko jest oprogramowanie. Oczywiście nic w tym złego. Są super, ale ja jestem zdecydowanie ponad to. Prototype jest po prostu wspomagającą mnie biblioteką.

Head First: Czym zatem jesteś?

Ajax: Jestem techniką projektowania. Jeśli wykonujesz asynchroniczne żądania w JavaScrpicie w celu uaktualnienia strony internetowej, wykorzystujesz właśnie mnie.

Head First: Asynchroniczne? Oznacza to, że twoje żądania...

Ajax: ...przerywają normalną pracę przeglądarki, właśnie tak. Żądania odbywają się w tle, w czasie gdy użytkownik znajduje się na stronie.

Head First: A żądanie oparte na Ajaksie może być wywołane przez cokolwiek?

Ajax: No właściwie. XHR może być wygenerowany przez prawie wszystko — dowolny rodzaj zdarzenia JavaScriptu.

Head First: Najmocniej przepraszać — ale XH...?

Ajax: XHR. To moi mali kumple od żądań opartych na Ajaksie. Poprawna nazwa to „XML HTTP Requests”, czyli „żądania XML HTTP”.

Head First: Mówisz, że nie jesteś oprogramowaniem, ale przecież użytkownicy instalują biblioteki do Ajaksa, czyż nie?

Ajax: Możesz też samodzielnie napisać cały kod od podstaw, ale jasne, większość osób korzysta z gotowych bibliotek do Ajaksa, takich jak Prototype. Biblioteki obsługują tworzenie żądań i zajmują się wszystkim, co jest zwracane.

Head First: A jaki typ danych zwracany jest przez żądanie oparte na Ajaksie?

Ajax: Cokolwiek, czego sobie zażyczysz, maleńka. Fragmenty stron napisane w języku HTML. Dane w formacie XML czy JavaScriptu. A nawet sam kod w JavaScrpicie.

Head First: Rozumiem. Powiedz mi jeszcze: JavaScript...

Ajax: E tam, czym jest ten mikrus? Hej, przepraszać cię, ale muszę lecieć.

Head First: Słucham?

Ajax: Ktoś właśnie kliknął przycisk JavaScriptu. Otrzymałem zdarzenie kliknięcia z moją nazwą. Skontaktuję się z tobą później...

Head First: Ajax, dziękujemy...

Ajax: Nie ma o czym mówić.

Ktoś ma kłopot ze swoim wieczorem kawalerskim

Muszę zarezerwować 19 miejsc na mój wieczór kawalerski. Żeby to zrobić, muszę ciągle naciskać przycisk „Wstecz”, by powrócić do strony lotu!

W tej chwili, kiedy rezerujesz miejsce, przeglądarka przesyła formularz do serwera, a serwer zwraca do przeglądarki stronę wyświetlającą zarezerwowane miejsce. Co jednak, gdy ktoś chce zarezerwować więcej miejsc? W takiej sytuacji musi nacisnąć przycisk *Wstecz* w przeglądarce, by wrócić do strony lotu, zarezerwować kolejne miejsce... otrzymać kolejne potwierdzenie, znowu użyć przycisku *Wstecz*...

Dotychczas napisaliśmy kod uaktualniający listę miejsc bez przechodzenia do nowej strony. Czy możemy zrobić coś podobnego, kiedy miejsce jest zarezerwowane? Gdyby formularz mógł w jakiś sposób przesłać rezerwację do serwera, a następnie uaktualnić listę miejsc, użytkownik mógłby pozostać na tej samej stronie. Gdyby musiał zarezerwować większą liczbę miejsc, byłby od razu na właściwej stronie, która by mu na to pozwoliła.



Kiedy rezerwacja zostanie wykonana, powinna się natychmiast pokazać.

Lista miejsc oraz formularz rezerwacji pokazane są na stronie lotu.

Name	Baggage	Edit
Bruce Sturgess	22.0	Show
Kirk Avery	15.0	Show
Drew Beattie	18.0	Show
James Blake	19.0	Show
Guffman	25.0	Show
Ted Bratton	18.0	Show
Jerry	14.0	Show
Ike Casy	28.0	Show
Tom Hagen	15.0	Show
Jack Hargreen	18.0	Show
Stan Harris	17.0	Show

New seat

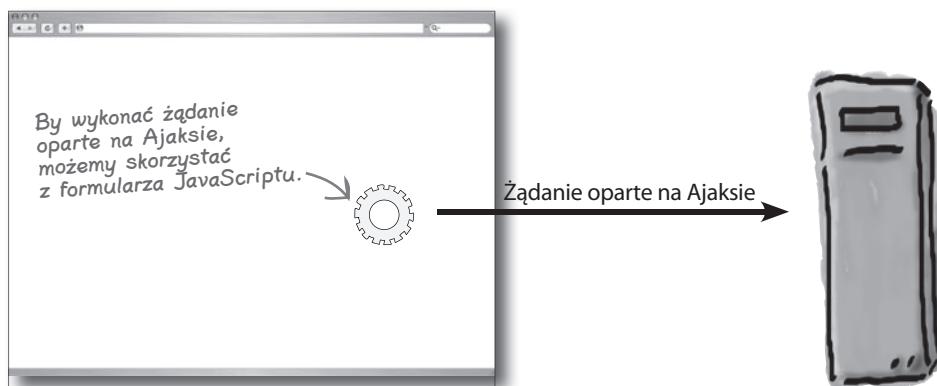
Name:
Baggage:

[Edit](#) | [Back](#)

Formularz musi wykonać żądanie oparte na Ajaksie

Jeśli pozwolimy przeglądarce przesłać formularz, wiemy, że zostaniemy odesłani na inną stronę. To tak jak z problemem, który mieliśmy, gdy użytkownik korzystał z przycisku *Odśwież* — jest to wbudowana funkcja przeglądarki, której nie możemy zmodyfikować.

Co zatem możemy zrobić? Musimy użyć formularza innego typu. Zamiast korzystać ze standardowego formularza HTTP, musimy użyć formularza w JavaScriptie i wykonać żądanie za jego pomocą.



Zamiast po prostu prosić przeglądarkę o przesłanie danych formularza, przycisk musi generować zdarzenie JavaScriptu, które prześle dane z formularza za pomocą żądania opartego na Ajaksie. Dlaczego jest to tak istotne? Oznacza to, że rezerwacja miejsca *nie będzie* powodowała wysłania przeglądarki do innej strony.

Formularz musi pozostawać pod KONTROLĄ JavaScriptu

Musimy przekształcić formularz z prostego formularza HTTP na generujący zdarzenia JavaScriptu i dynamicznie aktualniający bieżącą stronę zamiast odsyłania przeglądarki do innego adresu URL. Oto zawartość szablonu częściowego formularza rezerwacji:

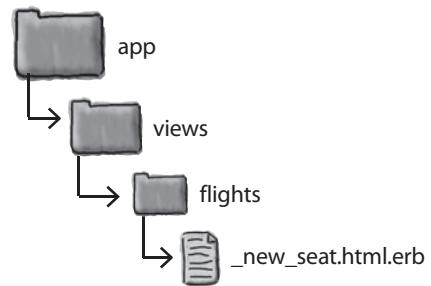
```
<% form_for(seat) do |f| %>
  <%= f.error_messages %>

  <%= f.hidden_field :flight_id %>

  <p>
    <%= f.label :name %><br />
    <%= f.text_field :name %>
  </p>

  <p>
    <%= f.label :baggage %><br />
    <%= f.text_field :baggage %>
  </p>

  <p>
    <%= f.submit "Create" %>
  </p>
<% end %>
```



Jak jednak możemy sprawić, by formularz działał w zupełnie inny sposób?

Musimy zmienić ten fragment:

```
<% form_for(seat) do |f| %>
```

na ten:

```
<% remote_form_for(seat, :update=>'seats' ) do |f| %>
```

To stosunkowo niewielka zmiana, jednak w rzeczywistości formularz będzie działał zupełnie inaczej...

Pamiętasz o kontrolerze?



Jazda próbna

Kiedy użytkownik trafi na stronę lotu (<http://localhost:3000/flights/2>), formularz rezerwacji miejsca wygląda dokładnie tak jak wcześniej:

The screenshot shows a table of flight records with columns for Name, Baggage, and actions (Show, Edit, Destroy). Below the table is a 'New seat' form with fields for Name (Jesse James Garrett) and Baggage (12). A 'Create' button is present. At the bottom are 'Edit | Back' links.

Jednak tak naprawdę kod HTML jest zupełnie inny.
Co zatem dzieje się, kiedy miejsce zostaje zarezerwowane?

The screenshot shows the results of a seat reservation. It displays the departure time (2009-11-11 13:30:00 UTC), arrival time (2009-11-11 14:30:00 UTC), destination (Titchmarsh Island), baggage allowance (25.0), and capacity (22). A message states "Seat was successfully created." Below this, it shows the flight details (Flight: 2, Name: Jesse James Garrett, Baggage: 12.0) and a 'Refresh seats' link. At the bottom is another 'New seat' form with the same fields and 'Create' button.

Gdzie trafiły wszystkie nasze
rezerwacje miejsc???????

Coś jest nie tak. Spójrzmy do bazy danych...
miejsce zostało poprawnie zarezerwowane, jednak
strona lotu wygląda niepoprawnie. Dlaczego?

Zmieniliśmy kod widoku, jednak kod kontrolera — kod znajdujący się na serwerze — nadal robi to samo co wcześniej: odsyła kod HTML ze szczegółami nowo zarezerwowanego miejsca.
Potrzebna jest nam teraz nowa wersja listy miejsc.

Naprawmy kod kontrolera.

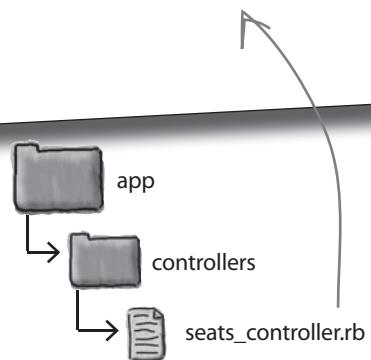
Musimy zastąpić metodę create

W tej chwili metoda create kontrolera seats wygląda następująco:

```
def create
  @seat = Seat.new(params[:seat]) ← Nie martw się, jeśli nie rozumiesz, co robi cały kod; i tak za chwilę zastąpimy go czymś innym.
  respond_to do |format|
    if @seat.save
      flash[:notice] = 'Seat was successfully created.'
      format.html { redirect_to(@seat) }
      format.xml { render :xml => @seat, :status => :created,
                   :location => @seat }

    else
      format.html { render :action => "new" }
      format.xml { render :xml => @seat.errors,
                   :status => :unprocessable_entity }
    end
  end
end
```

Musimy zastąpić to kodem tworzącym obiekt Seat, zapisującym obiekt w bazie danych, a następnie generującym nową kopię listy miejsc. Jak jednak powinien wyglądać taki kod?



Zaostrz ołówek



Napisz nową metodę `create`, która zawsze będzie tworzyła obiekt `Seat` w oparciu o dane formularza, prosiła nowy obiekt o zapisanie się w bazie danych i wyświetlała zawartość szablonu częściowego `_seat_list.html.erb`.

.....
.....
.....
.....
.....



Zaostrz ołówek

Rozwiążanie

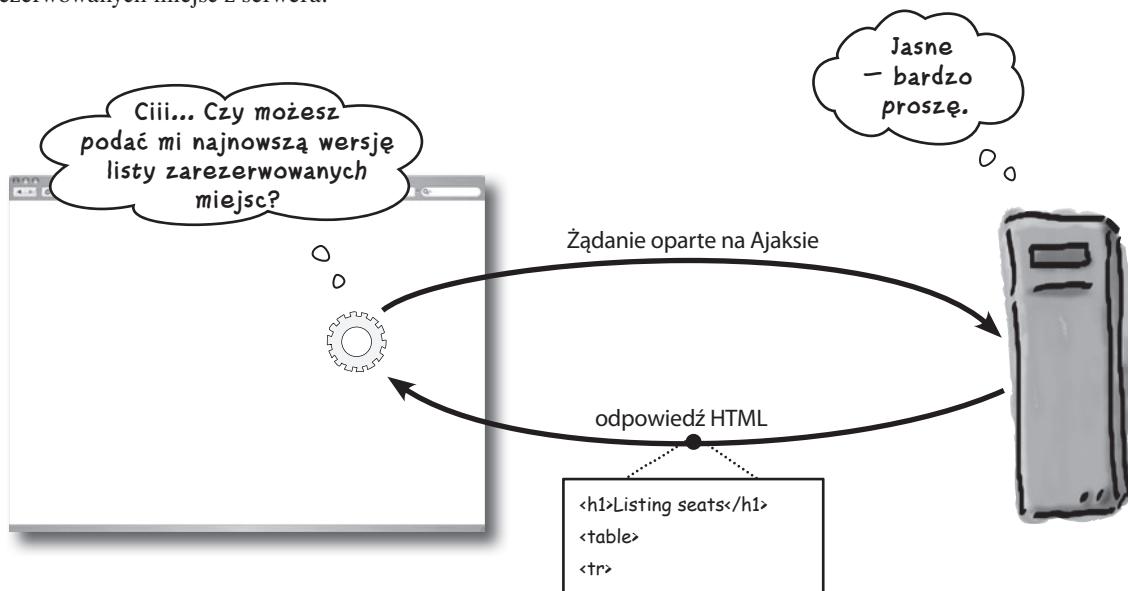
Utwórz obiekt
miejscia dokładnie
tak jak wcześniej.

```
def create
  @seat = Seat.new(params[:seat])
  @seat.save
  render :partial=>'flights/seat_list', :locals=>{:seats=>@seat.flight.seats}
end
```

To pozwala na
wyświetlenie listy
miejsc dla wszystkich
miejsc tego lotu.

Jaki efekt ma ten kod?

Nowa metoda `create` oznacza, że kiedy formularz oparty na Ajaksie przesyła nową rezerwację, powinien on otrzymać nową kopię listy zarezerwowanych miejsc z serwera:





Jazda próbna

Załóżmy, że użytkownik udaje się na stronę lotu i przesyła nowe żądanie rezerwacji:

Name	Baggage	Show	Edit	Destroy
Brent Chase	19.0	Show	Edit	Destroy
Tom Christie	15.0	Show	Edit	Destroy
Ryan Cleary	19.0	Show	Edit	Destroy
Julien Collard	16.0	Show	Edit	Destroy
Charlie Collins	19.0	Show	Edit	Destroy
Jesse James Garrett	12.0	Show	Edit	Destroy

System działa! Kiedy teraz utworzona zostaje rezerwacja, przeglądarka pozostaje na tej samej stronie, a nowy rekord rezerwacji pojawia się natychmiast.

Fantastycznie!
Teraz mogę zabrać się do zarezerwowania pozostałych 18 potrzebnych mi miejsc.





CELNE SPOSTRZEŻENIA

- JavaScript może wykonywać **żądania w tle** do serwera.
- JavaScript może wykorzystać zwrócony kod HTML do uaktualnienia jedynie **części strony**.
- Uaktualnienie strony za pomocą żądań wykonywanych w tle to **Ajax**.
- Żądania nazywane są XHR (**żądaniami XML HTTP**).
- Kod w JavaScriptie można wykonać w momencie wystąpienia **zdarzenia**.
- Zdarzenia mogą być wynikiem **działan użytkownika** (na przykład kliknięcia myszą) lub **zdarzeń systemowych** (jak liczniki).
- Jeśli nie chcesz, by formularz odesłał przeglądarkę do nowej strony, musisz przekształcić go na **formularz oparty na Ajaksie**.
- By z formularza zrobić taki oparty na Ajaksie, musisz zmienić `form_for` na `remote_form_for`.
- **Kod kontrolera** obsługujący żądanie formularza musi odesłać kod HTML uaktualniający stronę.
- Jeśli formularz otrzyma parametr `:update`, będzie wiedział, w którym miejscu strony należy umieścić zwrócony kod HTML.

Nie istniejąca głupie pytania

P: Jak to się dzieje, że wystarczy poprawić tylko metody pomocnicze formularzy, a nie wszystkie pola formularza?

O: Pola formularza pozostają bez zmian, ponieważ zawierają te same pola danych co wcześniej. Jedyną różnicą pomiędzy formularzem opartym na Ajaksie a „normalnym” formularzem HTML jest to, że zdarzenie `onsubmit` dla formularza opartego na Ajaksie wywołuje bibliotekę Prototype, zamiast po prostu przesyłać formularz. Wszystkie pozostałe elementy pozostają bez zmian.

P: Widziałem gdzieś, że formularze oparte na Ajaksie generowane są za pomocą `form_remote_for`. Na czym polega różnica?

O: Na niczym — `form_remote_for` to po prostu alias metody `remote_form_for`. Obie metody robią dokładnie to samo.

P: A co, jeśli muszę przekształcić formularz niepowiązany z obiektem modelu i utworzony za pomocą `form_tag`?

O: Dostępna jest metoda `form_remote_tag`, która może zostać użyta w tej sytuacji.

P: Nie rozumiem. *Formularz* może zastąpić kod HTML na *stronie*?

O: Niezupełnie. By wykonać żądanie oparte na Ajaksie, formularz wywołuje funkcję języka JavaScript. To funkcja języka JavaScript zastępuje kod HTML strony.

P: Czy kiedy serwer otrzymuje żądanie formularza, nadal będzie ono wyglądało tak samo?

O: Żądanie będzie takie samo, tak jakby zostało przesłane przez formularz HTML. Biblioteka Prototype skonstruuje żądanie tak, by wyglądało ono jak normalne żądanie HTTP.

P: Z jakiej metody HTTP korzysta formularz oparty na Ajaksie?

O: Podobnie jak formularz HTML, domyślnie korzysta on z metody POST.

P: Ale mogę zmienić metodę, prawda?

O: Metodę HTTP możesz zmienić, wstawiając do metody pomocniczej parametr `:method=>`.

Kim jestem?



Kilku członków Klubu Ajaksa w pełnych kostiumach gra na przyjęciu w grę „Kim jestem?”. Każdy da Ci wskazówkę — a Ty będziesz musiał spróbować odgadnąć, czym są, na podstawie tego, co mówią. Załóż, że zawsze mówią o sobie prawdę. Wypełnij luki po prawej stronie, identyfikując uczestników zabawy.

Dzisiaj uczestnicy gry:

Mogą się pojawić dowolne typy czarujących elementów Ajaksa, jakie dotychczas widziałeś!

Nazwa

Jestem biblioteką wykorzystywana przez Rails do generowania w przeglądarce żądań opartych na Ajaksie.

Jestem językiem działającym wewnętrz przeglądarki.

Jestem żądaniem wykorzystywanym w aplikacjach opartych na Ajaksie, a moi przyjaciele nazywają mnie „XHR”.

Jestem zdarzeniem, ale nie zdarzeniem użytkownika.

Jestem wykorzystywany do wygenerowania formularza opartego na Ajaksie i powiązanego z obiektem modelu.

Mogę wywołać zarejestrowany ze mną kod przeglądarki.

Kilku członków Klubu Ajaksa w pełnych kostiumach gra na przyjęciu w grę „Kim jestem?”. Każdy da Ci wskazówkę — a Ty będziesz musiał spróbować odgadnąć, czym są, na podstawie tego, co mówią. Załóż, że zawsze mówią o sobie prawdę. Wypełnij luki po prawej stronie, identyfikując uczestników zabawy.

Dzisiajscy uczestnicy gry:

Mogą się pojawić dowolne typy czarujących elementów Ajaksu, jakie dotychczas widziałeś!

Kim jestem?



Nazwa

Jestem biblioteką wykorzystywaną przez Rails do generowania w przeglądarce żądań opartych na Ajaksie.

Prototype

Jestem językiem działającym wewnątrz przeglądarki.

JavaScript

Jestem żądaniem wykorzystywanym w aplikacjach opartych na Ajaksie, a moi przyjaciele nazywają mnie „XHR”.

Żądanie XML HTTP

Jestem zdarzeniem, ale nie zdarzeniem użytkownika.

Zdarzenie systemowe

Jestem wykorzystywany do wygenerowania formularza opartego na Ajaksie i powiązanego z obiektem modelu.

remote_form_for

Mogę wywołać zarejestrowany ze mną kod przeglądarki.

Zdarzenie

Teraz pojawił się problem z rezerwacjami lotów

Organizator wieczoru kawalerskiego rezerwował właśnie miejsca na wybrany lot, kiedy napotkał problem. Gdy zaczynał wykonywać rezerwacje, lot miał wiele wolnych miejsc, ale później...

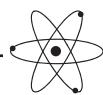
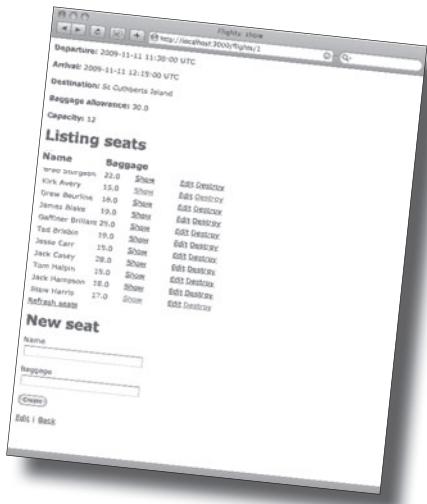
Hej, gdzie się podziały moje cztery ostatnie rezerwacje? Wpisuję szczegółowe informacje, ale rezerwacje gdzieś zginęły.



Ktoś inny rezerwował miejsca w tym samym czasie.

Choć Ajax jest w stanie zarezerwować miejsca, uproszczony kod kontrolera nie sprawdza, czy pojawił się błąd i czy wszystkie miejsca nie zostały już zarezerwowane.

Co zatem powinniśmy zrobić? Poza wyświetleniem aktualnej wersji listy rezerwacji kod kontrolera powinien w jakiś sposób aktualniać na stronie część z powiadomieniami, by poinformować o tym, czy rezerwacja się powiodła.



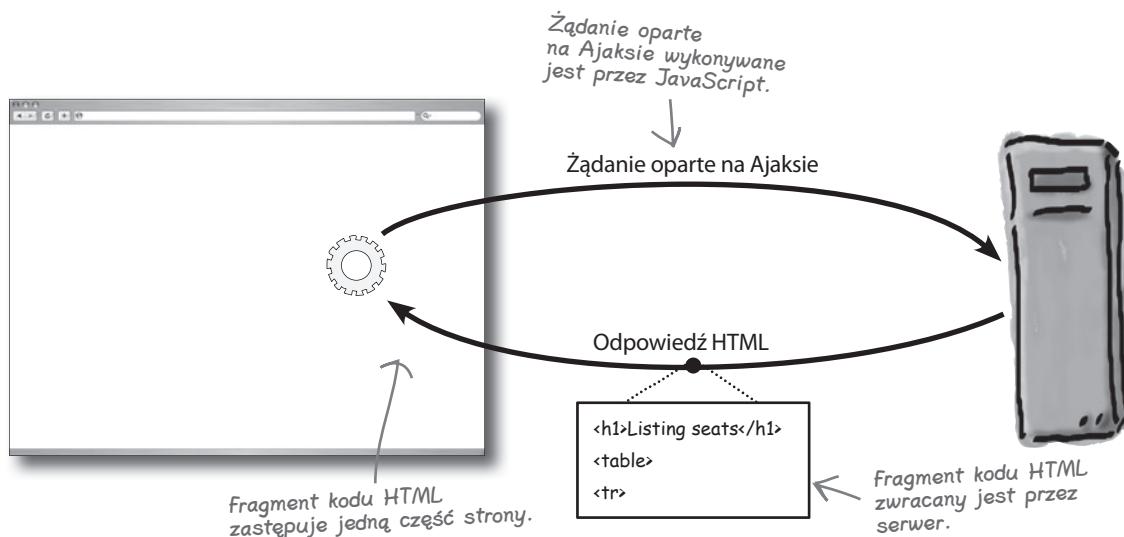
WYSIL
SZARE KOMÓRKI

Brzmi to rozsądnie, ale jaki widzisz w tym problem?

Synchroniczna asynchroniczność?

Potrafimy uaktualnić jedną część strony naraz

Dotychczas przy wykonywaniu żądań opartych na Ajaksie zawsze uaktualniałyśmy tylko jedną część strony za pomocą kodu HTML zwracanego przez serwer:



Co tym razem się zmieniło?

Różnica polega na tym, że tym razem musimy uaktualnić listę miejsc oraz część z powiadomieniami znajdująca się na górze strony. Wynika z tego, że konieczne jest zastąpienie dwóch całkowicie odrębnych fragmentów kodu HTML.

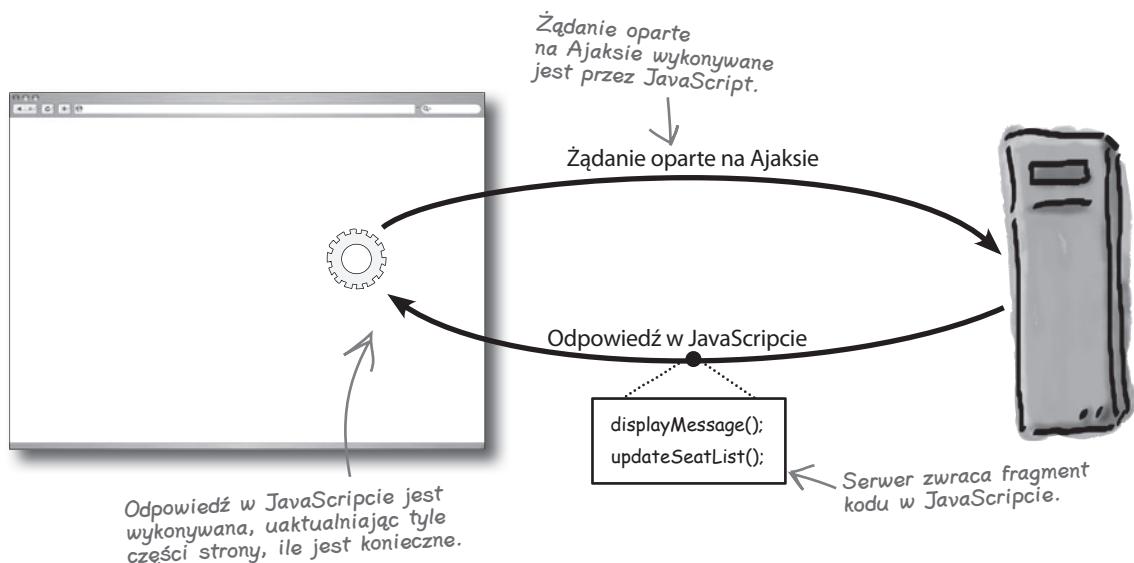
Jak możemy użyć jednej odpowiedzi z serwera do wprowadzenia większej liczby zmian na stronie? Czy powinniśmy wykonać kilka żądań? Czy też przesyłać kilka fragmentów kodu HTML?

Tak naprawdę istnieje o wiele bardziej elegancki sposób wykonania kilku operacji w wyniku jednego żądania.

Sztuczka polega na odesłaniu z powrotem czegoś innego niż kod HTML.

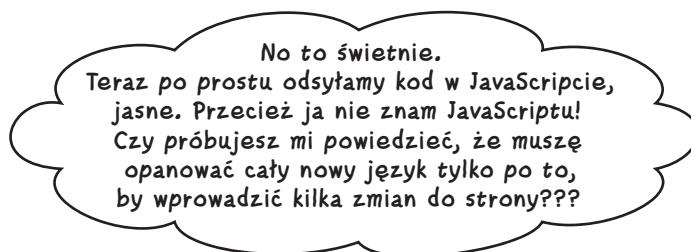
Kontroler musi zamiast HTML zwracać kod w JavaScriptie

Jeśli kontroler odsyła do przeglądarki dane HTML, JavaScript zazwyczaj zrobi z nimi coś prostego, na przykład zastąpi część strony. Jeśli jednak kontroler odsyła do przeglądarki kod w JavaScriptie, kod ten może robić tyle rzeczy, ile tylko potrzeba.



Jeśli zatem kontroler chce uaktualnić listę zarezerwowanych miejsc na stronie, później wyświetlić potwierdzenie, a następnie wykonać jakąś ciekawą animację odwracającą całą stronę do góry nogami, wystarczy tylko, że odeśle odpowiedni kod w języku JavaScript. Cokolwiek, co znajduje się w kodzie w języku JavaScript, zostanie wykonane.

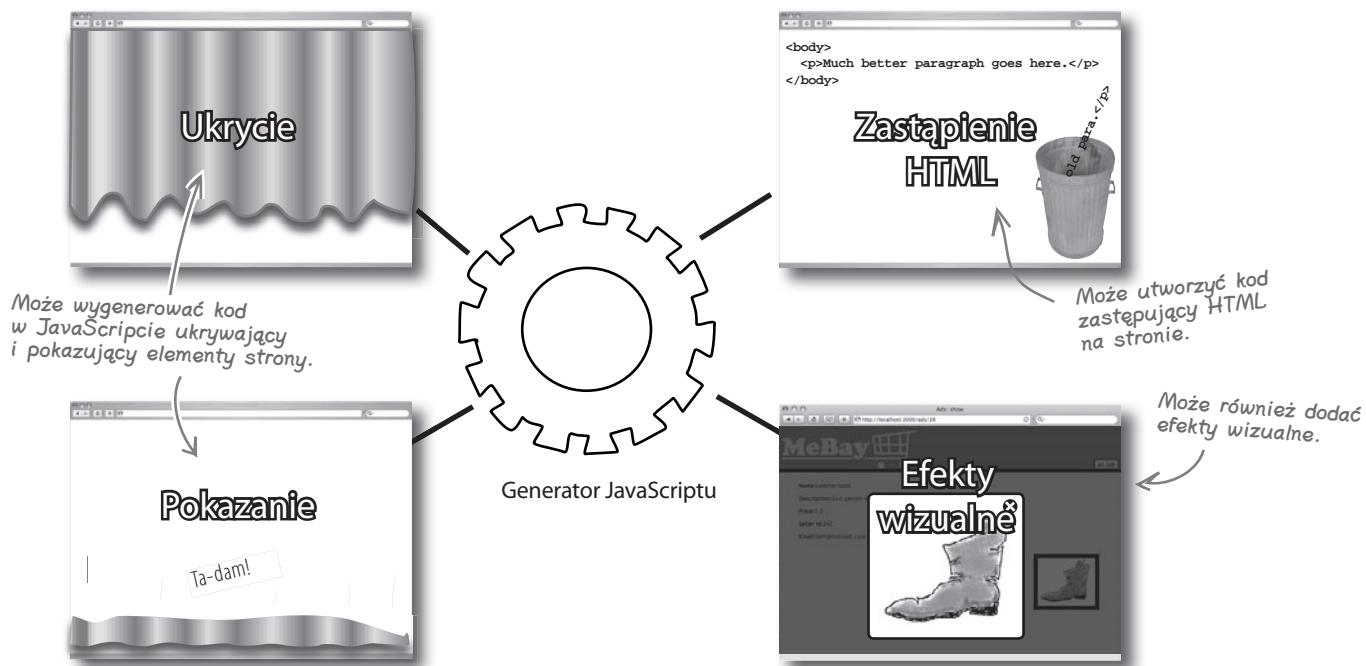
[Uwaga od Straży Dobrego Smaku:
nie chciałbyś tego naprawdę zrobić.]



Możesz pozwolić Rails na napisanie kodu w JavaScriptie za Ciebie.

Jeśli kontroler musi zamiast HTML odsyłać z powrotem kod w JavaScriptie, możesz oczekwać, że niezbędne będzie, byś nauczył się pisać kod w tym języku. Tak naprawdę wcale tak nie jest.

Rails udostępnia obiekt zwany *generatorem JavaScriptu*, który robi dokładnie to, co sugeruje jego nazwa — *generuje JavaScript*.



Choć tak naprawdę znajomość JavaScriptu może być korzystna, większość kodu w tym języku odsyłanego do przeglądarki robi stosunkowo standardowe rzeczy — na przykład zastępuje fragment kodu HTML, ukrywa część strony czy wywołuje inną funkcję JavaScriptu wykonującą animację. Tymczasem generator JavaScriptu potrafi napisać za Ciebie kod wykonujący każdą z tych czynności.

Wystarczy jedynie odpowiednio go wywołać.



Magnesiki z kodem

Uzupełnij kod kontrolera, tak by generował on kod w JavaScriptie, zastępujący kod HTML w elemencie `<div>` o identyfikatorze `notice` i informujący o zarezerwowaniu miejsca.

```
def create

  @seat = Seat.new(params[:seat])

  render :update do |page|

    if .....

      page. _____ , .....

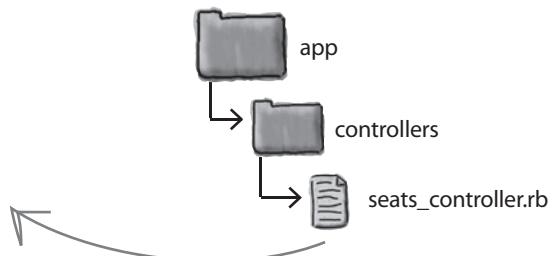
    else

      page. _____ , .....

    end

  end

end
```



'Sorry - the seat could not be booked'

replace_html

'notice'

@seat.save

'notice'

replace_html

'Seat was successfully booked'



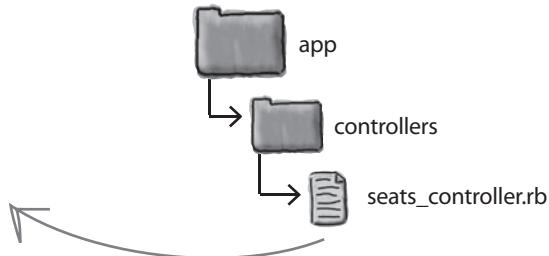
Magnesiki z kodem: Rozwiążanie

Uzupełnij kod kontrolera, tak by generował on kod w JavaScriptie, zastępujący kod HTML w elemencie `<div>` o identyfikatorze `notice` i informujący o zarezerwowaniu miejsca.

```
def create

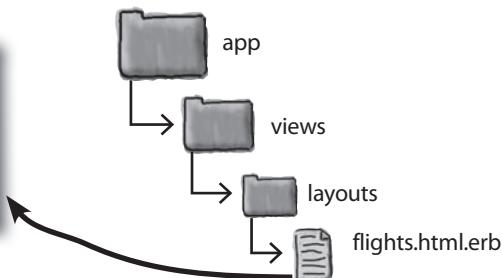
  @seat = Seat.new(params[:seat])

  render :update do |page|
    if ... @seat.save ...
      page. [replace_html] [ 'notice' ] , [ 'Seat was successfully booked' ]
    else
      page. [replace_html] [ 'notice' ] , [ 'Sorry - the seat could not be booked' ]
    end
  end
end
```



Kod kontrolera generuje JavaScript uaktualniający część strony internetowej o identyfikatorze `notice`. Która to będzie część strony? Układ strony dla strony lotu zawiera specjalny obszar na górze, służący do zamieszczania powiadomień. Musisz dokonać edycji układu `flights.html.erb` i dodać do elementu `<p>` odpowiedni identyfikator, jak poniżej:

```
<p style="color: green" id="notice">
  <%= flash[:notice] %>
</p>
```



Co generuje Rails?

Generator JavaScriptu tej strony tworzy poniższy kod w języku JavaScript:

"Nie ma szans, żebyś chciał
to pisać samodzielnie..."*

```
try {
  Element.update("notice", "Seat was successfully booked");

  Element.update("seats", "<h1>Listing seats</h1>\n\n<table>\n<tr>\n  <th>Name</th>\n  <th>Baggage</th>\n</tr>\n<tr>\n<td>Brad Sturgeon</td>\n  <td>22.0</td>\n  <td><a href=\"/seats/1\">Show</a></td>\n  <td><a href=\"/seats/1/edit\">Edit</a></td>\n  <td><a href=\"/seats/1\" onklik=\"if (confirm('Are you sure?')) { var f = document.createElement('form'); f.style.display = 'none'; this.parentNode.appendChild(f); f.method = 'POST'; f.action = this.href;var m = document.createElement('input'); m.setAttribute('type', 'hidden'); m.setAttribute('name', '_method'); m.setAttribute('value', 'delete'); f.appendChild(m);var s = document.createElement('input'); s.setAttribute('type', 'hidden'); s.setAttribute('name', 'authenticity_token'); s.setAttribute('value', 'aec87b235224924109e33b3207d464c64207e733'); f.appendChild(s);f.submit(); }>Destroy</a></td>\n</tr>\n<tr>\n<td>Kirk Avery</td>\n  <td>15.0</td>\n  <td><a href=\"/seats/2\">Show</a></td>\n  <td><a href=\"/seats/2/edit\">Edit</a></td>\n  <td><a href=\"/seats/2\" onklik=\"if (confirm('Are you sure?')) { var f = document.createElement('form'); f.style.display = 'none'; this.parentNode.appendChild(f); f.method = 'POST'; f.action = this.href;var m = document.createElement('input'); m.setAttribute('type', 'hidden'); m.setAttribute('name', '_method'); m.setAttribute('value', 'delete'); f.appendChild(m);var s = document.createElement('input'); s.setAttribute('type', 'hidden'); s.setAttribute('name', 'authenticity_token'); s.setAttribute('value',",
  
```

Kod ten zostanie zwrócony do przeglądarki, kiedy formularz rezerwacji oparty na Ajaxie przesłany zostanie do kontrolera. Poprzednio przeglądarka musiała pobrać zawartość odpowiedzi kontrolera i wykorzystać ją do zastąpienia jakiejś części strony. Teraz chcemy, by przeglądarka *wykonała* odpowiedź. Chcemy, by wykonała nasz wygenerowany kod w JavaScriptie.

Jak jednak możemy nakazać formularzowi wykonanie odpowiedzi w JavaScriptie?

Czy wszystko gotowe?

**Jeśli nie powiesz, gdzie umieścić odpowiedź,
zostanie ona wykonana**

Przyjrzyjmy się kodowi Embedded Ruby generującemu formularz oparty na Ajaksie:

Plik `_new_seat.html.erb`

```
<% remote_form_for(seat, :update=>'seats') do |f| %>
  <%= f.error_messages %>

  <%= f.hidden_field :flight_id %>

  <p>
    <%= f.label :name %><br />
    <%= f.text_field :name %>
  </p>

  <p>
    <%= f.label :baggage %><br />
    <%= f.text_field :baggage %>
  </p>

  <p>
    <%= f.submit "Create" %>
  </p>
<% end %>
```

Kod ten tworzy cały JavaScript niezbędny do uruchomienia żądania opartego na Ajaksie w momencie naciśnięcia przycisku *Create*. Formularz pobiera następnie cokolwiek, co zwracane jest przez serwer, i wykorzystuje do zastąpienia części strony opisanej identyfikatorem `seats`.

Takie rozwiązanie było dobre, dopóki serwer zwracał do przeglądarki kod HTML. Teraz jednak odsyła on kod w JavaScriptie i nie chcemy, by formularz umieścił go w przypadkowym miejscu. Chcemy go wykonać, a to już zupełnie inna sprawa.

Zmiana, którą musimy wprowadzić do szablonu strony, jest bardzo niewielka. By formularz wykonał kod, wystarczy tylko usunąć parametr `update`:

```
<% remote_form_for(seat) do |f| %>
```

Parametr „`update`”
został usunięty.



Jazda próbna

Gdy teraz rezerwowane jest miejsce, formularz wyświetla komunikat o powodzeniu lub niepowodzeniu tej operacji.

The screenshot illustrates a two-step booking process:

- Step 1 (Left Window):** Shows a list of seats with names like Brent Chase, Tom Christie, Ryan Cleary, Julien Collard, and Charlie Collins. Each seat has a "Show" link and "Edit Destroy" buttons. A "New seat" button is visible.
- Step 2 (Right Window):** Shows the details of a new booking for "Jesse James Garrett" with baggage "12". A "Create" button is present. The right window also displays a confirmation message: "Seat was successfully booked" followed by flight details: Departure 2009-11-11 13:30:00 UTC, Arrival: 2009-11-11 14:30:00 UTC, Destination: Titchmarsh Island, Baggage allowance: 25.0, Capacity: 22.

Annotations in the right window explain the behavior:

- "Pojawia się potwierdzenie." (Confirmation appears) points to the success message.
- "Nowa rezerwacja nie jest jednak widoczna na liście..." (The new reservation is not yet visible in the list...) points to the list of seats where the new booking is not yet listed.

Strona nie uaktualnia jednak listy zarezerwowanych miejsc. Musimy wygenerować dodatkowy kod w JavaScriptie, uaktualniający listę miejsc.

Zaostrz ołówek



Musisz napisać dodatkowe wywołanie uaktualniające listę zarezerwowanych miejsc za pomocą zawartości odpowiedniego szablonu częściowego.

```
page.replace_html ' _____ ', :partial => ' _____ ',
  :locals => { _____ => _____ }
```

Wypróbowanie całości

Zaostrz ołówek



Rozwiążanie

Musisz napisać dodatkowe wywołanie uaktualniające listę zarezerwowanych miejsc za pomocą zawartości odpowiedniego szablonu częściowego.

```
To jest element <div>,  
który uaktualniamy.  
  
page.replace_html 'seats', :partial => 'flights/seat_list'  
:locals => { :seats => @seat.flight.seats }
```

Ten kod powoduje zastosowanie
szablonu częściowego
app/view/flights/_seat_list.html.erb.

Tablica miejsc lotu.

Uzupełniony kod może teraz wykonywać kilka rzeczy

Oto jak powinien wyglądać uzupełniony kod:

```
def create  
  @seat = Seat.new(params[:seat])  
  render :update do |page|  
    if @seat.save  
      page.replace_html 'notice', 'Seat was successfully booked'  
    else  
      page.replace_html 'notice', 'Sorry - the seat could not be booked'  
    end  
    page.replace_html 'seats', :partial => 'flights/seat_list',  
    :locals => { :seats => @seat.flight.seats }  
  end  
end
```

Metody generatora JavaScriptu page można wykonywać tak często, jak tylko mamy ochotę. Jeśli rezerwacja miejsca została poprawnie zapisana, obiekt page wygeneruje kod uaktualniający powiadomienie, a także utworzy kod w JavaScriptie uaktualniający listę miejsc.



Jazda próbna

Gdy teraz miejsce zostaje zarezerwowane, nie tylko pojawia się komunikat o tym informujący, ale także uaktualniana jest lista rezerwacji:

Name	Baggage	Action
Brent Chase	19.0	Show Edit Destroy
Tom Christie	15.0	Show Edit Destroy
Ryan Cleary	19.0	Show Edit Destroy
Julien Collard	16.0	Show Edit Destroy
Charlie Collins	19.0	Show Edit Destroy

Listing seats				
Name	Baggage	Action	Action	Action
Brent Chase	19.0	Show	Edit	Destroy
Tom Christie	15.0	Show	Edit	Destroy
Ryan Cleary	19.0	Show	Edit	Destroy
Julien Collard	16.0	Show	Edit	Destroy
Charlie Collins	19.0	Show	Edit	Destroy
Jesse James Garrett	12.0	Show	Edit	Destroy

System zostaje uruchomiony, a użytkownicy mogą w szybki sposób zarezerwować większą liczbę miejsc.





Niezbędnik programisty Rails

Masz za sobą rozdział 7. i teraz do swojego niezbędnika programisty Rails możesz dodać umiejętność dodawania Ajaksa do swoich aplikacji.

Narzędzia Rails

Aplikacje oparte na Ajaksie wykonują żądania w tle za pomocą JavaScriptu.

Biblioteka Prototype udostępnia większość funkcji Ajaksa.

Platforma Rails udostępnia kilka metod pomocniczych Ajaksa:

`<%= link_to_remote %>` tworzy odnośnik oparty na Ajaksie.

`<%= periodically_call_remote %>` uruchamia licznik oparty na Ajaksie.

`<%= remote_form_for %>` tworzy formularz oparty na Ajaksie.

Jeśli do metody pomocniczej Ajaksa przekazemy parametr `":update"`, zastąpi ona część strony o odpowiednim identyfikatorze.

Jeśli parametr `":update"` zostanie pominięty, metoda wykona kod w JavaScriptie zwrocony przez kontroler.

8. XML i różne reprezentacje



Nie da się zawsze wszystkich zadowolić. A może jednak? Dotychczas widzieliśmy, jak można wykorzystać Rails do szybkiego i łatwego tworzenia aplikacji internetowych, które idealnie pasują do pewnego zbioru wymagań. Co jednak zrobić, kiedy pojawiają się inne wymagania? Co powinniśmy zrobić, jeśli niektóre osoby chcą otrzymać proste strony internetowe, inne interesuje mashup z aplikacji firmy Google, a jeszcze inne chcą, by aplikacja była dostępna w czytniku kanałów RSS? W tym rozdziale będziemy tworzyć różne reprezentacje tych samych danych, co da nam maksymalną elastyczność przy minimalnym wysiłku.

Zdobywanie szczytów świata

Head First Climbers to witryna internetowa dla wspinaczy z całego świata.

Amatorzy wspinaczki piszą swoje raporty z wypraw wraz z lokalizacjami i datami zdobycia określonych szczytów, a także zgłaszają odkryte niebezpieczeństwa, takie jak osuwiska skalne czy lawiny.

Informacje te są oczywiście bardzo istotne z punktu widzenia bezpieczeństwa innych wspinaczy, stąd wiele osób wykorzystuje telefony komórkowe w połączeniu z odbiornikami GPS do odczytywania oraz zapisywania informacji bezpośrednio na miejscu. Kiedy się z niego korzysta we właściwy sposób, system może komuś ocalić życie, a jednak z jakiegoś powodu nie ma zbyt wielu użytkowników.



Dlaczego zatem nie jest popularny?

Aplikacja jest bardzo prosta. To po prostu oparta na rusztowaniu wersja poniższej struktury danych:

Incident (zdarzenie)	
mountain (szczyt)	<i>string</i>
latitude (szerokość geograficzna)	<i>decimal</i>
longitude (długość geograficzna)	<i>decimal</i>
when (kiedy)	<i>datetime</i>
title (tytuł)	<i>string</i>
description (opis)	<i>text</i>

id	mountain	latitude	longitude	when	title	description
1	Mount Rushless	63.04348055...	-150.993963...	2009-11-21 11:...	Rock slide	Rubble on the ...
2	Mount Rushless	63.07805277...	-150.977869...	2009-11-21 17:...	Hidden crev...	Ice layer cove...
3	Mount Lotopaxo	-0.683975	-78.4365055...	2009-06-07 12:...	Ascent	Living only on...
4	High Kanuklima	11.123925	72.72135833...	2009-05-12 18:...	Altitude si...	Overcome by th...

Jak już z pewnością zauważyłeś, rusztowanie to świetny sposób na rozpoczęcie aplikacji, jednak prawie zawsze konieczne jest zmodyfikowanie kodu w celu zmiany uniwersalnego kodu rusztowania na coś bardziej zblżonego do problemów, z jakimi stykają się użytkownicy.

Co musi się zmienić w tej aplikacji?

Zrób tak!

Utwórz opartą na rusztowaniu aplikację odpowiadającą tej strukturze danych.

Użytkownicy nienawidzą interfejsu aplikacji!

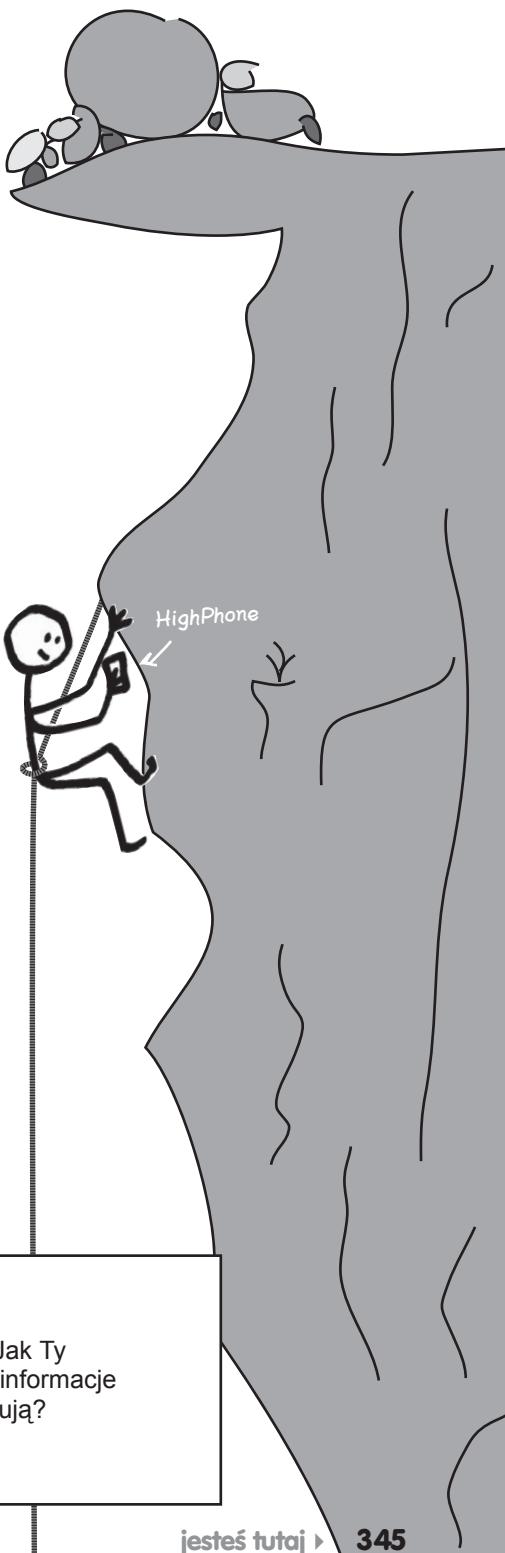
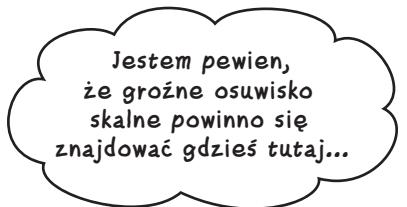
Wkrótce okazuje się, dlaczego witryna internetowa nie jest popularna — problemem jest **interfejs użytkownika**.

System służy do zarządzania danymi *przestrzennymi* — zapisuje zdarzenia mające miejsce w określonym miejscu i czasie na całym świecie. Informacje lokalizacyjne zapisywane są za pomocą dwóch liczb:

- **Szerokości geograficznej.** Określa ona, jak daleko na północ lub południe od równika znajduje się dane miejsce.
- **Długości geograficznej.** Określa ona, jak daleko na zachód lub wschód od południka zerowego znajduje się dane miejsce.

Użytkownicy mogą z powodzeniem zapisywać swoje dane — wystarczy, że odczytają szerokość oraz długość geograficzną z odbiorników GPS.

Z trudem jednak przychodzi im *odczytanie* oraz *interpretacja* informacji pochodzących od innych wspinaczy.



Użytkownicy mogą zatem dodawać do aplikacji dane, jednak nie potrafią zrozumieć danych, które otrzymują. To dziesiątkuje liczbę odwiedzających stronę, a im mniej odwiedzających, tym mniej informacji zostaje dodanych... co sprawia, że z aplikacji korzysta jeszcze mniejsza liczba osób. To błędne koło.

Coś trzeba z tym zrobić, bo inaczej witryna straci tylu klientów, że będzie musiała zostać zlikwidowana.



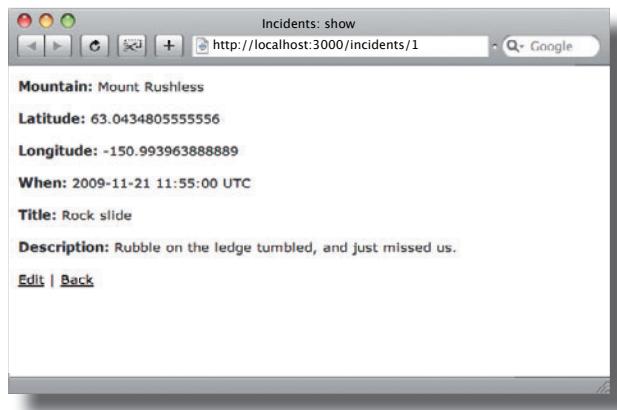
Zastanów się nad danymi, jakie musi wyświetlać aplikacja. Jak Ty wyświetliłbyś te informacje? Jaki format byłby najlepszy, by informacje były łatwe do zrozumienia dla wspinaczy, którzy ich potrzebują?

Dane muszą się znaleźć na mapie

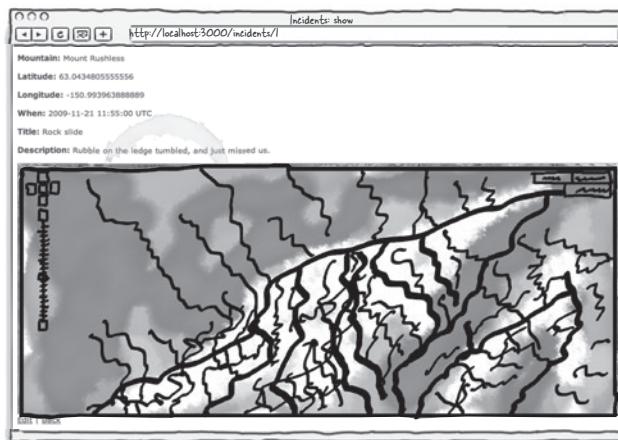
System zapisuje dane geograficzne i powinien je wyświetlać na mapie.

Dane zapisywane są poprawnie; dostępne są wszystkie podstawowe funkcje (tworzenia, odczytywania, aktualniania oraz usuwania). Problem leży w **prezentacji**. Lokalizacja *przechowywana* jest jako dwie liczby — szerokość oraz długość geograficzna — jednak nie oznacza to, że musi być w ten sposób *wyświetlana*.

Zamiast widzieć takie coś...



...wspinaczom przyda się coś takiego:



To oczywiście naprawdę spora modyfikacja interfejsu, dlatego właściciele strony zdecydowali, że zamiast zmieniać całą aplikację, na początek uruchomią niewielki program pilotażowy tworzący wersję strony wyświetlającą zdarzenie na mapie. Nie mają jednak pomysłu na to, jak to zrobić — potrzebują zatem Twojej pomocy.

Co będzie pierwszą rzeczą, jaką **TY byś zrobił?**

Musimy utworzyć nową akcję

Nie chcemy zmieniać istniejącego kodu — chcemy tylko coś do niego dodawać. Dopóki nie będziemy pewni, że nowy interfejs działa, nie chcemy denerwować dotychczasowych użytkowników. W końcu nie tak wielu ich znowu zostało...

Dodamy zatem nową akcję o nazwie `show_with_map`. W tej chwili użytkownik może zobaczyć stronę zdarzenia pod adresem URL takim jak:

```
http://localhost:3000/incidents/1
```

Nową wersję strony utworzymy natomiast pod adresem:

```
http://localhost:3000/incidents/map/1
```

W ten sposób uczestnicy pilotażu będą jedynie musieli dodać do adresu część `/map`, by trafić na nową wersję strony. Wykorzystamy taką trasę:

```
map.connect 'incidents/map/:id', :action=>'show_with_map', :controller=>'incidents'
```

Pamiętaj, by dodać ją jako pierwszą trasę w pliku config/routes.rb.



Zaostrz ołówek



Szablon strony utworzymy, kopując plik `app/views/incidents/show.html.erb`. Jaką nazwę będzie nosił nowy plik?

Kontroler `incidents` będzie potrzebował nowej metody, która wczyta odpowiedni obiekt modelu `Incident` i przechowa go w zmiennej instancji o nazwie `@incident`. Napisz tę metodę poniżej:

Przetestowanie nowej akcji

Zaostrz ołówek



Rozwiążanie

Szablon strony utworzymy, kopując plik `app/views/incidents/show.html.erb`.
Jaką nazwę będzie nosić nowy plik?

`app/views/incidents/show_with_map.html.erb`

Kontroler `incidents` będzie potrzebował nowej metody, która wczyta odpowiedni obiekt modelu `Incident` i przechowa go w zmiennej instancji o nazwie `@incident`. Napisz tę metodę poniżej:

```
..... „show_with_map” → def show_with_map  
      to nazwa akcji. @incident = Incident.find(params[:id]) ← To będzie identyfikator  
..... end z adresu URL.
```

Nowa akcja wydaje się działać...

Gdy teraz spojrzesz na obie wersje strony zdarzenia, zobaczyś, że wyświetlają one poprawne dane. Co jeszcze widzisz?

To oryginalna strona oparta na rusztowaniu.

Ta wersja ma inny adres URL.

To wersja wywolująca nową akcję „show_with_map”.

↑
Obie wersje pokazują te same dane. ↑

Obie wersje strony zdarzenia wyglądają identycznie — i na tym polega nasz problem.

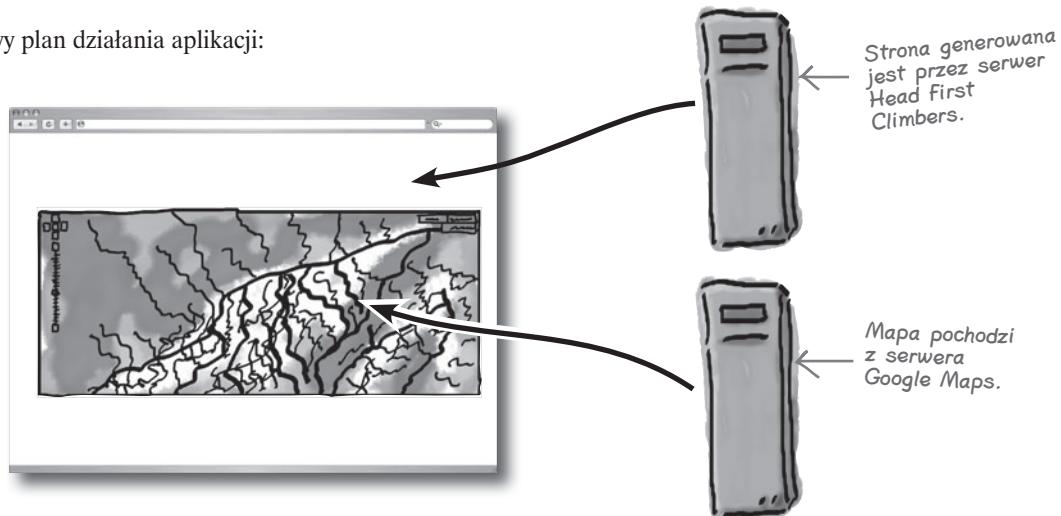
Nowa strona potrzebuje mapy... w tym właśnie rzecz!

Oczywiście *nie chcemy*, by nowa wersja strony wyglądała tak samo jak stara. Chcemy dodać do niej mapę.

Jak możemy to zrobić? Nie da rady, byśmy zbudowali swój własny system tworzący mapy. Zamiast tego utworzymy **mashup**. Mashup to aplikacja internetowa integrująca ze sobą dane oraz usługi pochodzące z różnych serwisów dostępnych w Internecie.

Większość usług udostępniających mapy pozwala na osadzanie ich wewnątrz własnych aplikacji internetowych użytkowników; my skorzystamy z serwisu firmy Google. Usługa Google Maps daje nam dużą elastyczność. Nie tylko możemy osadzać mapy w stronach internetowych, ale także bez większego nakładu pracy dodawać własne dane do map oraz programować sposób interakcji użytkownika z mapą i danymi.

Oto wysokopoziomowy plan działania aplikacji:



Mapa będzie wyświetlała przybliżoną lokalizację zanotowanego zdarzenia wraz z symbolem oznaczającym dokładnie miejsce.

Aplikacja Head First Climbers wygeneruje kod wywołujący mapę wraz z wyświetlonymi na niej danymi, jednak sama mapa wraz z większością kodu pozwalającego użytkownikowi na operacje takie, jak przeciąganie mapy czy jej powiększanie będzie pochodziła z serwera Google Maps. Choć większość kodu będzie pochodziła z firmy Google, nadal musimy udostępnić dwa elementy:

- Kod HTML oraz JavaScript wywołujący mapę. Ta część będzie nieco skomplikowana, dlatego niezbędny HTML oraz JavaScript umieścimy w osobnym szablonie częściowym, który możemy wywołać z naszego szablonu strony.
- Dane, które należy wyświetlić na mapie. Na początek skorzystamy z pliku z przykładowymi danymi, by upewnić się, że mapa działa.

Jak powinien wyglądać taki kod mapy?

Pobranie klucza Google Maps

Jakiego typu kod jest nam potrzebny?

W szablonie częściowym `_map.html.erb` powinniśmy mieć następujący kod:

```
<%
  google_key='ABQIAAAAAnfs7bKE82qgb3Zc2YyS-oBT2yXp_` +
    'ZAY8_ufC3CFXhHIE1NvwkxSySz_REpPq-4WZA270wgbtyR3VCA'
  full_page ||= false
  show_action ||= nil
  new_action ||= nil
  data ||= nil
%>
<div id="map"
  align="right"
  style="border: 1px solid #979797;
         min-width: 400px;
         if full_page -%>
         min-height: 800px;
         height: 800px;
         else -%>
         min-height: 400px;
         height: 400px;
         end -%>
         background-color: #FFFFFF;
         border: 1px solid #999999;
         padding: 10px;"></div>
...
...
```

Dzięki temu kluczowi mapa działa dla serwera lokalnego.

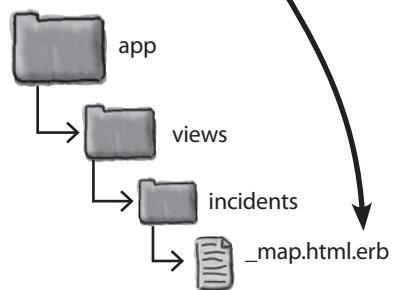
Pobierz to!

Nie mamy tutaj miejsca, by wyświetlić pełny kod szablonu częściowego, jednak możesz pobrać ten plik z <ftp://ftp.helion.pl/przyklady/hfor.zip>. Znajdziesz go w katalogu `app/views/incidents_map.html.erb` kodów z rozdziału 8.

Co robi powyższy kod? Po pierwsze wywołuje na serwerze Google Maps kod w JavaScriptie generujący mapę na naszej stronie internetowej. Mapa będzie miała wbudowane wszystkie podstawowe funkcje przeciągania oraz powiększania.

Podstawowy kod map Google nie robi jednak *wszystkiego*, co byśmy chcieli. Nie ładuje i nie wyświetla żadnych naszych danych lokalnych. Kod z szablonu `_map.html.erb` ładuje również z pliku dane lokalizacji, które wykorzystywane są do przeniesienia mapy do określonego miejsca i wyświetlenia w podanym punkcie ikony.

Z kodem ten wiąże się jednak pewna komplikacja...



Kod ten działa jedynie dla serwera lokalnego

Google ogranicza wykorzystanie kodu. Konieczne jest podanie serwera, na którym będzie się z niego korzystać. Oznacza to, że zanim użyjemy go na serwerze www.twojadomena.pl, będziemy musieli poinformować o tym Google. By użytkownicy przestrzegali tego warunku, kod będzie wykonywany jedynie wtedy, gdy dodamy do niego **klucz Google Maps**. Klucz ten generowany jest dla określonej nazwy serwera i jeśli spróbujemy osadzić mapę Google na stronie pochodzącej z innego miejsca, mapa ta nie zostanie wyświetlona.

Na razie jednak nie jest to dla nas problemem. Szablon częściowy `_map.html.erb`, którego mamy zamiar użyć, ma podany klucz Google Maps dla serwera lokalnego — dlatego dopóki będziemy go wykonywać na własnym komputerze, wszystko będzie dobrze. Pamiętaj jednak o zastosowaniu własnego klucza przed wykonaniem tego kodu z jakiegokolwiek innego miejsca.



Ciekawostki

Jeśli chcesz osadzać mapy Google Maps w swoich aplikacjach internetowych, musisz zarejestrować się w serwisie prowadzonym przez Google. By to zrobić, przejdź pod następujący adres URL:
<http://tinyurl.com/mapreg>.

Zaostrz ołówek



Teraz musimy dołączyć szablon częściowy mapy do szablonu strony `show_with_map.html.erb`. Musimy przekazać zmienną lokalną o nazwie `data` zawierającą ścieżkę do danych mapy. W tym celu wykorzystamy plik testowy `/test.xml`.

Napisz poniżej kod wywołujący szablon częściowy.

.....

Zaostrz ołówek



Rozwiążanie

Musisz wstawić ten wiersz kodu do pliku `show_with_map.html.erb`.

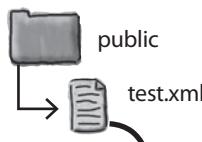
Teraz musimy dołączyć szablon częściowy mapy do szablonu strony `show_with_map.html.erb`. Musimy przekazać zmienną lokalną o nazwie `data` zawierającą ścieżkę do danych mapy. W tym celu wykorzystamy plik testowy `/test.xml`.

Napisz poniżej kod wywołujący szablon częściowy.

```
<%= render (:partial=>'map', :locals=>{:data=>'/test.xml'}) %>
```

Teraz potrzebne nam dane mapy

Zanim wypróbujemy osadzoną mapę, będziemy musieli przekazać jej odpowiednie dane. Na początek użyjemy po prostu pliku testowego `test.xml`. Wygląda on następująco:



```
<data>
  <description>This is an example description</description>
  <latitude>63.0434805555556 </latitude>
  <longitude>-150.993963888889</longitude>
  <title>Test Data</title>
</data>
```

Pobierz to!

By oszczędzić Ci wpisywania długich liczb, pod adresem <ftp://ftp.helion.pl/przykłady/hfror.zip> w kodach przeznaczonych dla tego rozdziału udostępniliśmy plik z danymi testowymi. Znajdziesz go w katalogu `public/test.xml`.

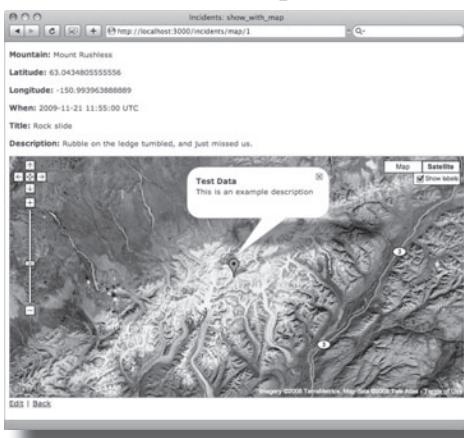
Dane mapy zawierają szerokość oraz długość geograficzną testowego zdarzenia. Kiedy mapa Google zostanie załadowana, nasz szablon częściowy przekaże jej zawartość tego pliku. Zdarzenie powinno zostać wyświetlone oraz wyśrodkowane na mapie.



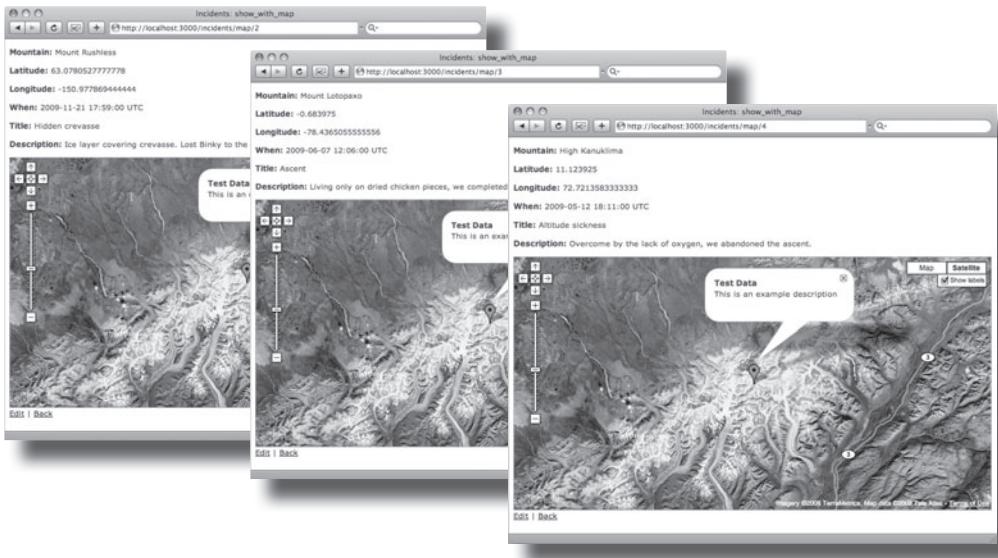
Jazda próbna

Co się zatem stanie, gdy przejdziemy pod adres URL taki jak:

`http://localhost:3000/incidents/map/1`



Mapa działa! Ale co jeśli przejdziemy pod inny adres URL?



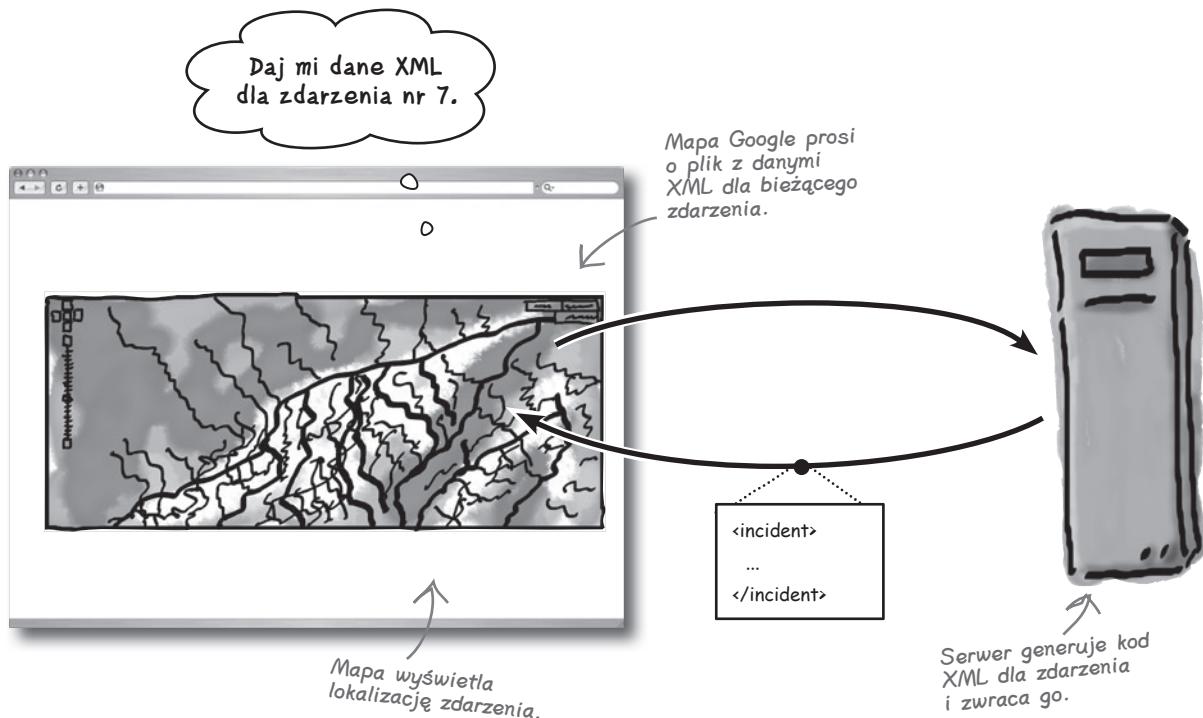
Każda mapa wygląda dokładnie tak samo, bez względu na dane. Jest tak, ponieważ każda mapa korzysta z tych samych danych — zawartości pliku testowego `test.xml`.

By mapa wyświetlała lokalizację podanego zdarzenia, musimy wygenerować plik z danymi dla każdej strony.

Co zatem powinniśmy wygenerować?

Do mapy przekazujemy dane XML, które opisują lokalizację każdego zdarzenia. Lokalizacja określona jest przez szerokość i długość geograficzną, tytuł oraz opis. Musimy wygenerować taki kod XML dla *każdego* zdarzenia.

System będzie działał mniej więcej tak:



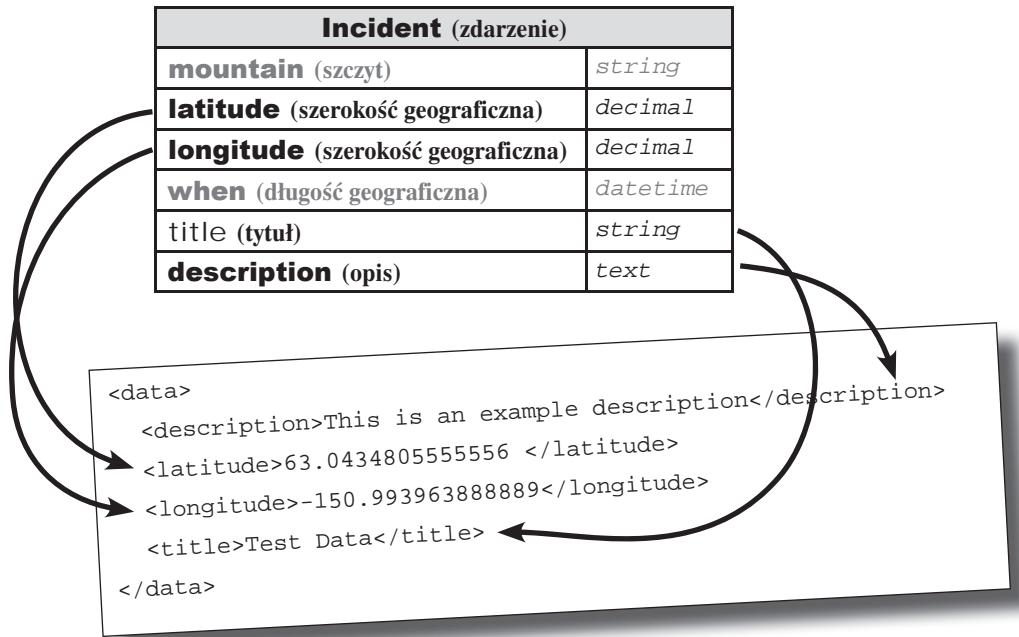
Jeśli to zaczyna wyglądać znajomo, świetnie! Tak naprawdę Google Maps wykorzystują do działania technologię Ajax. Pamiętasz, jak wykorzystaliśmy ją w poprzednim rozdziale do pobrania nowej wersji listy zarezerwowanych miejsc? W ten sam sposób Google Maps żąda danych XML dla lokalizacji zdarzenia.

Kolejnym krokiem jest zatem wygenerowanie danych.

Skąd możemy otrzymać dane?

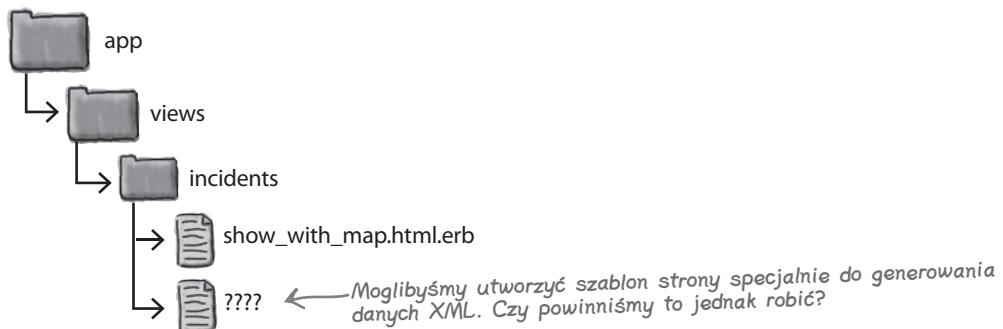
Wygenerujemy kod XML z modelu

Dane dla wygenerowanego kodu XML będą pochodziły z modelu `Incident`. Skorzystamy jedynie z czterech atrybutów — szerokości oraz długości geograficznej, tytułu i opisu.



Jak jednak generujemy kod XML? W pewien sposób przypomina to generowanie strony internetowej. W końcu XML i XHTML są do siebie bardzo podobne. I tak jak strony internetowe zawierają dane z modelu, również pliki XML będą je zawierały.

Jedną z opcji byłoby utworzenie szablonu strony zawierającego znaczniki XML zamiast znaczników HTML:

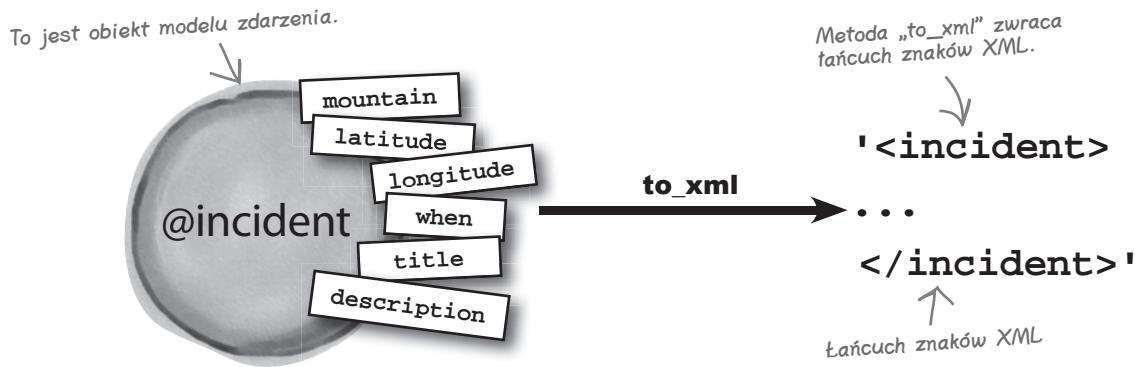


Ten sposób zadziałałby, ale istnieje lepsze rozwiązanie...

Obiekt modelu może generować kod XML

Obiekty modelu zawierają dane. Pliki XML zawierają dane.

W pewien sposób ma sens to, by obiekty modelu generowały wersje samych siebie w formacie XML. Każdy obiekt modelu ma metodę `to_xml` zwracającą łańcuch znaków XML:



`@incident.to_xml`

↑
Metoda „to_xml” zwracała串cuch znaków XML reprezentujący obiekt modelu.

Utworzenie kodu XML to tylko połowa zadania. Drugą połową jest zwrócenie tego kodu do przeglądarki. Nie używamy szablonu strony, dlatego całe zadanie spada na barki kontrolera wyświetlającego XML...

Jak powinien wyglądać taki kod kontrolera?

Możemy ulepszyć metodę show_with_map w taki sposób, by zwracała ona kod XML:

```
To obiekt zdarzenia,  
który już wczytywaliśmy.  
def show_with_map  
  → @incident = Incident.find(params[:id])  
  → render :text=>@incident.to_xml ← Ten kod utworzyła串znaków  
    Metoda „render”  
    zwraca kod XML.  
  end  
  ↑  
  Parametr „:text” mówi,  
  co będziemy zwracać  
  do przeglądarki.
```

Metoda render zwraca kod XML do przeglądarki. Spotkaliśmy się z nią już wcześniej, jednak ta wersja jest nieco inna. Zazwyczaj metodę render wykorzystuje się do generowania strony internetowej z szablonu strony lub szablonu częściowego. Można jej jednak również przekazać obiekt łańcucha znaków — i to właśnie robimy powyżej.



Ciekawostki

By ułatwić Ci życie, twórcy Rails pozwalają również przekazać do metody render parametr o nazwie :xml:

```
render :xml=>@incident
```

Jeśli obiekt zostanie przekazany do metody render za pomocą parametru :xml, metoda ta wywoła na obiekcie metodę to_xml i odeśle całość do przeglądarki. Wersja polecenia render z parametrem :xml wygeneruje tę samą zawartość co polecenie render w naszym kontrolerze, a dodatkowo ustawi typ MIME odpowiedzi na text/xml. Na razie skorzystamy z pierwszej wersji z parametrem :text.

Nieistniejące głupie pytania

P: Czy możecie mi przypomnieć jeszcze raz, czym zajmuje się metoda render?

O: Metoda render generuje odpowiedź dla przeglądarki. Kiedy przeglądarka prosi o stronę internetową, jest to żądanie. Metoda render generuje to, co zostanie przesłane z powrotem.



Jazda próbna

Co zatem otrzymamy teraz, kiedy udamy się pod adres:

`http://localhost:3000/incidents/map/1`

The screenshot shows a web browser window with the title "Source of: http://localhost:3000/incidents/map/1". The content of the page is the XML code for incident 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<incident>
  <created-at type="datetime">2009-11-21T11:59:31Z</created-at>
  <description>Rubble on the ledge tumbled, and just missed us.</description>
  <id type="integer">1</id>
  <latitude type="decimal">63.0434805555556</latitude>
  <longitude type="decimal">-150.993963888889</longitude>
  <mountain>Mount Rushless</mountain>
  <title>Rock slide</title>
  <updated-at type="datetime">2009-11-21T11:59:31Z</updated-at>
  <when type="datetime">2009-11-21T11:55:00Z</when>
</incident>
```

Kontroler zwraca teraz kod XML zawierający dane z obiektu zdarzenia o identyfikatorze 1 — co możesz zobaczyć w źródle strony.

Czy jest tu jednak jakiś problem? Wygenerowany kod XML przypomina *nieco* przykładowy XML, jednak widać pomiędzy nimi kilka różnic:

- Generujemy zbyt dużą liczbę atrybutów. Plik z przykładowymi danymi zawierał jedynie informacje o szerokości i długości geograficznej, tytule oraz opisie. Ten fragment kodu XML zawiera **wszystkie** informacje o zdarzeniu, włącznie z datą i czasem odnotowania go.
- Element główny pliku XML nosi niewłaściwą nazwę. Wygenerowany kod XML wziął nazwę elementu głównego z wykorzystywanej przez nas zmiennej, `<incident>`. My jednak potrzebujemy kodu XML z elementem głównym o nazwie `<data>`.

```
<data>
  <description>This is an example
  description</description>
  <latitude>63.0434805555556 </latitude>
  <longitude>-150.993963888889</longitude>
  <title>Test Data</title>
</data>
```

Ten kod XML ma *prawie* poprawny format, jednak *nie do końca*.

Musimy zmodyfikować kod XML tworzony przez metodę `to_xml`.



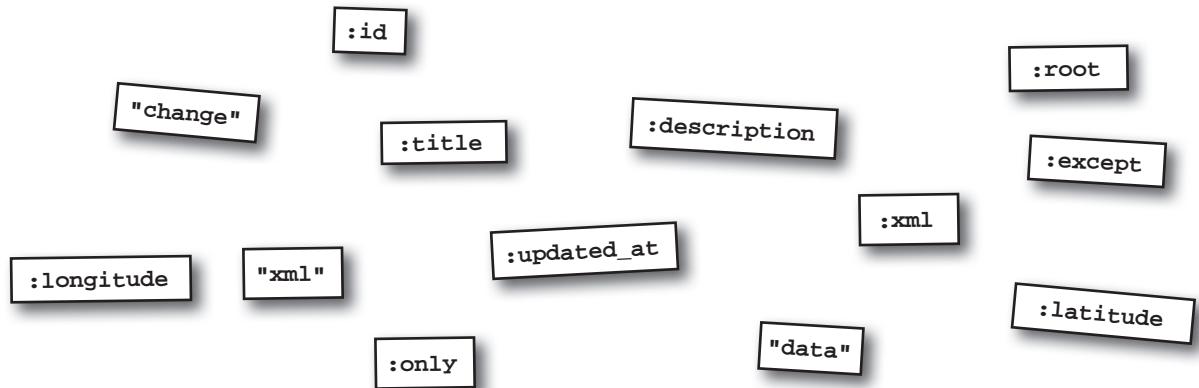
Magnesiki z kodem

Metoda `to_xml` ma kilka opcjonalnych parametrów pozwalających nam modyfikować zwracany przez nią kod XML. Sprawdź, czy będziesz w stanie odgadnąć, jakie będą wartości tych parametrów:

```
def show_with_map

  @incident = Incident.find(params[:id])

  render :text=>@incident.to_xml(
    _____ => [ ..... , ..... , ..... , ..... ],
    _____ => ..... )
  end
```





Magnesiki z kodem: Rozwiążanie

Metoda `to_xml` ma kilka opcjonalnych parametrów pozwalających nam modyfikować zwracany przez nią kod XML. Sprawdź, czy będziesz w stanie odgadnąć, jakie będą wartości tych parametrów:

```
def show_with_map

@incident = Incident.find(params[:id])

render :text=>@incident.to_xml(
  :only => [ :latitude , :longitude , :title , :description ],
  :root => "data"
)

end
```

Ponieważ korzystamy z wersji metody „render” z parametrem „:text=>...”, możemy do zmodyfikowania danych wyjściowych wykorzystać opcje metody „to_xml”.



Nie istnieja
głupie pytania

P: Czy nie powinniśmy generować danych XML w modelu?

O: Moglibyśmy, ale nie jest to dobry pomysł. W różnych sytuacjach być może konieczne będzie generowanie różnego kodu XML. Gdybyśmy dodawali kod do modelu dla każdego z formatów XML, model szybko stałby się przeładowany.



Jazda próbna

Gdy teraz udamy się pod adres:

`http://localhost:3000/incidents/map/1`

otrzymamy kod XML wyglądający nieco inaczej, co widać w źródle strony.

The screenshot shows a window titled "Source of: http://localhost:3000/incidents/map/1". The content of the window is the XML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <description>Rubble on the ledge tumbled, and just missed us.</description>
  <latitude type="decimal">63.043480555556</latitude>
  <longitude type="decimal">-150.993963888889</longitude>
  <title>Rock slide</title>
</data>
```

Udało Ci się zmodyfikować kod XML w taki sposób, że wyświetla on teraz jedynie potrzebne nam dane, a także zawiera odpowiednio nazwany element główny. Jest on teraz o wiele bardziej zbliżony do przykładowego pliku XML.

Metoda `to_xml` nie pozwala na wprowadzanie zbyt wielu zmian do zwracanego kodu XML, jednak dla większości zastosowań będzie wystarczająca — także w przypadku przesyłania kodu XML do serwisu Google Maps w celu utworzenia własnej mapy.

Przy niewielkim nakładzie pracy metoda `to_xml` daje nam kod XML, którego potrzebowaliśmy.

Wspinaczom potrzebne są również strony

Tymczasem na wysokości kilku tysięcy metrów...



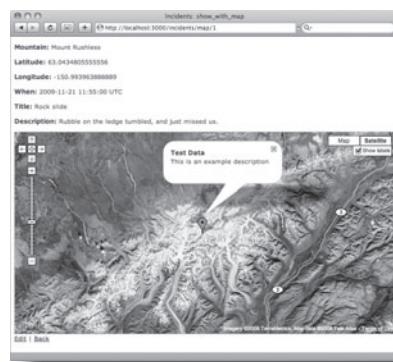
Niektórzy użytkownicy programu pilotażowego mają problem.

Strony internetowe zniknęły! Przed ostatnimi poprawkami adres URL taki jak poniższy:

`http://localhost:3000/incidents/map/1`

generował stronę internetową. Problem polega na tym, że teraz ten adres URL zwraca po prostu kod XML zamiast ładnej mapy Google.

Przed ostatnimi zmianami:



Przed zmianami mieliśmy stronę internetową pokazującą nasze dane na mapie Google.

Po ostatnich zmianach:

A screenshot of a terminal window titled "Source of: http://localhost:3000/incidents/map/1". The window displays the following XML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <description>Rubble on the ledge tumbled, and just missed us.</description>
  <latitude type="decimal">63.0434805555556</latitude>
  <longitude type="decimal">-150.993963888889</longitude>
  <title>Rock slide</title>
</data>
```

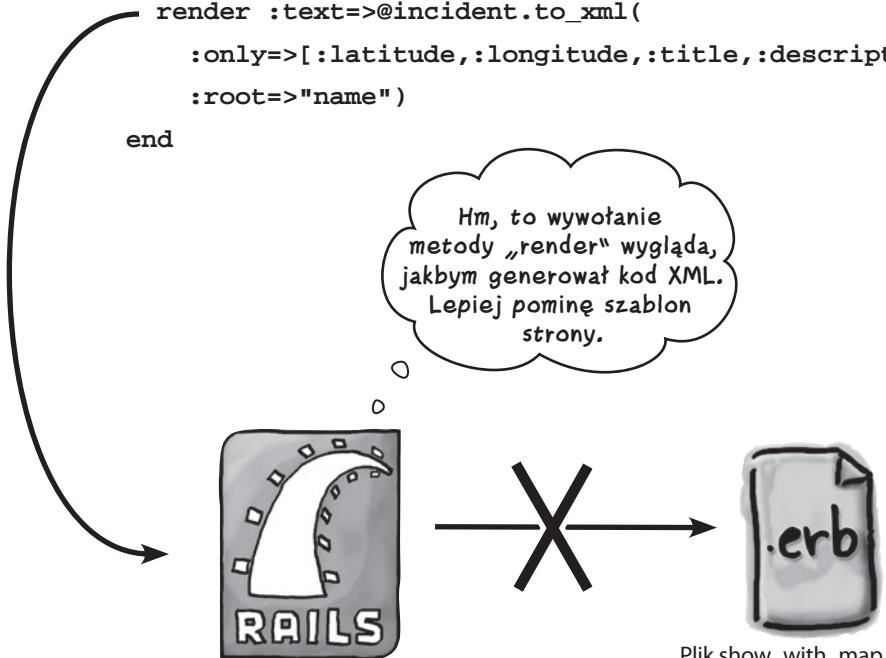
Po zmianach z powrotem otrzymaliśmy jedynie kod XML.

Musimy generować XML oraz HTML

Akcja `show_with_map` początkowo generowała stronę internetową za pomocą szablonu strony `show_with_map.html.erb`. Po dodaniu do metody kontrolera wywołania `render` Rails ignoruje szablon i po prostu generuje kod XML:

```
def show_with_map
  @incident = Incident.find(params[:id])
  render :text=>@incident.to_xml(
    :only=>[:latitude,:longitude,:title,:description],
    :root=>"name")
end
```

Hm, to wywołanie metody „`render`” wygląda, jakbym generował kod XML. Lepiej pominę szablon strony.



Plik `show_with_map.html.erb`

To oczywiście ma sens, ponieważ niemożliwe jest, by akcja generowała **jednocześnie** XML oraz HTML.

Mimo to potrzebna nam będzie strona internetowa wyświetlająca mapę, a mapa z kolei potrzebuje danych XML. Co zatem możemy zrobić?

Potrzebny jest nam jeden sposób wywołania kontrolera w celu wygenerowania kodu HTML, a inny do wygenerowania kodu XML.



Mark: Jeszcze jedna akcja?

Bob: Jasne. Jedna będzie generowała XML, a druga HTML.

Laura: Wydaje mi się, że nie jest to najlepszy pomysł...

Bob: Dlaczego nie?

Laura: Oznaczałoby to duplikowanie kodu. Obie metody musiałyby zawierać kod wczytujący obiekt zdarzenia.

Bob: I co z tego? Przecież to tylko jeden wiersz.

Laura: Na razie jest to jeden wiersz. Co jednak, jeśli zmienimy coś w przyszłości?

Mark: Masz na myśli zmianę modelu?

Laura: Albo jeśli dane będą dostarczane z innego miejsca, na przykład z usługi sieciowej.

Bob: To żaden problem. Martwmy się lepiej problemami, które mamy teraz, dobrze?

Mark: Sam nie wiem. Laura, co proponujesz?

Laura: To proste. Ja przekazałabym do akcji parametr. W ten sposób powiedziałabym, jakiego formatu oczekuję.

Mark: To może zadziałać.

Bob: Dajcie spokój, to mnóstwo pracy.

Laura: Mniej niż przy tworzeniu kolejnej akcji.

Mark: Jedno mnie interesuje...

Laura: Tak?

Mark: Czy adres URL nie identyfikuje informacji, jakich oczekujemy?

Laura: I co z tego?

Mark: Czy nie powinniśmy wykorzystywać tego samego adresu URL dla obu formatów?

XML i HTML to po prostu reprezentacje

Choć HTML i XML wyglądają zupełnie inaczej, tak naprawdę są wizualnymi reprezentacjami *tej samej rzeczy*. Zarówno strona HTML, jak i dane mapy w formacie XML opisują te same dane obiektu `Incident`. Zdarzenie to dane podstawowe, czasami nazywane **zasobami**.

Zasoby (ang. *resource*) to dane prezentowane przez stronę internetową. Strona internetowa jest z kolei nazywana **reprezentacją** zasobów. Weźmy jako przykład obiekt `Incident`. Obiekt `Incident` jest zasobem. Strona internetowa zdarzenia oraz plik XML z danymi mapy są reprezentacjami zasobów.



Rozumienie Internetu w kategoriach zasobów i reprezentacji to fragment architektury zwanej **REST**. REST to **architektura Rails**. Im bardziej aplikacja oparta jest na architekturze REST, tym lepiej będzie działać w Rails.

W czym nam to jednak pomoże? By być w pełni opartym na architekturze REST, zarówno dane XML, jak i strona internetowa powinny mieć ten sam adres URL (Uniform Resource Locator), ponieważ reprezentują te same zasoby. Potrzebne będzie coś takiego:

`http://localhost:3000/incidents/maps/1`

By jednak wszystko uprościć, możemy nieco zmodyfikować projekt oparty na architekturze REST (jedynie odrobinę) i użyć następujących adresów URL dla dwóch reprezentacji:

`http://localhost:3000/incidents/maps/1.xml` ← Jeden adres URL zwraca dane XML;
`http://localhost:3000/incidents/maps/1.html` ← drugi zwraca kod HTML.

Wybierz swój format

W jaki sposób powinniśmy decydować, z którego formatu skorzystać?

Jeśli dodamy dodatkową trasę zawierającą format w ścieżce:

```
map.connect 'incidents/map/:id.:format', :action=>'show_with_map',  
:controller=>'incidents'
```

będziemy w stanie wczytać żądanego formatu z pliku XML, a następnie podając

To zapisze format
z rozszerzeniem.

w kodzie decyzję, jak poniżej:

```
if params[:format] == 'html'  
    # Wygenerowanie reprezentacji HTML  
else  
    # Wygenerowanie reprezentacji XML  
end
```

http://localhost:3000/incidents/map/1.html

To rozszerzenie zostanie
przechowane w polu „:format”.

http://localhost:3000/incidents/map/1.xml

Nie jest to jednak sposób, w jaki większość aplikacji Rails wybiera format do wygenerowania. Zamiast tego wywołuję metodę o nazwie `respond_to` do oraz obiekt zwany **responderem**:

```
respond_to do |format|  
  format.html {  
  }  
  format.xml {  
  }
```

end

„format” jest obiektem respondera.
Tutaj trafia kod generujący stronę internetową.
Tutaj trafia kod generujący plik XML.

Kod ten robi mniej więcej to samo. Obiekt `format` jest responderem. Responder może decydować, czy wykonać kod w zależności od formatu wymaganego przez żądanie. Jeśli zatem użytkownik prosi o HTML, powyższy kod wykona kod przekazany do `format.html`. Jeśli użytkownik prosi o XML, responder wykona kod przekazany do `format.xml`.

Dlaczego programiści Rails nie korzystają po prostu z polecenia `if`? W końcu czy taki kod nie byłby prostszy? Cóż, responder ma pewne **ukryte możliwości**. Przykładowo ustawia on typ MIME odpowiedzi. Typ MIME informuje przeglądarkę, jakiego typu danymi jest odpowiedź. Zaleca się wykorzystanie `respond_to` do decydowania o tym, jaki format reprezentacji należy wygenerować.



Ćwiczenie

Metoda kontrolera `show_with_map` musi wybrać, czy powinna generować kod w formacie XML, czy też HTML. Napisz nową wersję metody wykorzystującą responder do generowania poprawnej reprezentacji.

Wskazówka: jeśli musisz wygenerować HTML, co jeszcze oprócz wczytania obiektu modelu musi zrobić kontroler?

Szablon strony `show_with_map.html.erb` wywołuje obecnie szablon częściowy mapy i przekazuje mu plik `/test.xml`. Jak będzie wyglądało wywołanie szablonu częściowego, jeśli ma ono wywoływać wygenerowany plik XML?

.....

Metoda respond_to do robi, co należy



Ćwiczenie
Rozwiążanie

Metoda kontrolera show_with_map musi wybrać, czy powinna generować kod w formacie XML, czy też HTML. Napisz nową wersję metody wykorzystującą responder do generowania poprawnej reprezentacji.

Wskazówka: jeśli musisz wygenerować HTML, co jeszcze oprócz wczytania obiektu modelu musi zrobić kontroler?

Nic! Przy generowaniu
HTML możemy pozostawić
Rails wywołanie szablonu
show_with_map.html.erb.

```
def show_with_map
  @incident = Incident.find(params[:id])
  respond_to do |format|
    format.html {
      render :text=>@incident.to_xml(
        :only=>[:latitude, :longitude, :title, :description],
        :root=>"data")
    }
  end
end
```

Możemy zostawić to puste
— Rails wywoła szablon za nas.

Szablon strony show_with_map.html.erb wywołuje obecnie szablon częściowy mapy i przekazuje mu plik /test.xml. Jak będzie wyglądało wywołanie szablonu częściowego, jeśli ma ono wywoływać wygenerowany plik XML?

```
<%= render(:partial=>'map', :locals=>{:data=>"#{@incident.id}.xml"}) %>
```

Nie istnieja
głupie pytania

P: Czy skoro część z format.html naprawdę nie wymaga żadnego kodu, nie możemy jej po prostu pominąć?

G: Nie. Uwzględnienie format.html jest konieczne, gdyż inaczej platforma Rails nie będzie wiedziała, że musi odpowiadać na dane wyjściowe w formacie HTML.

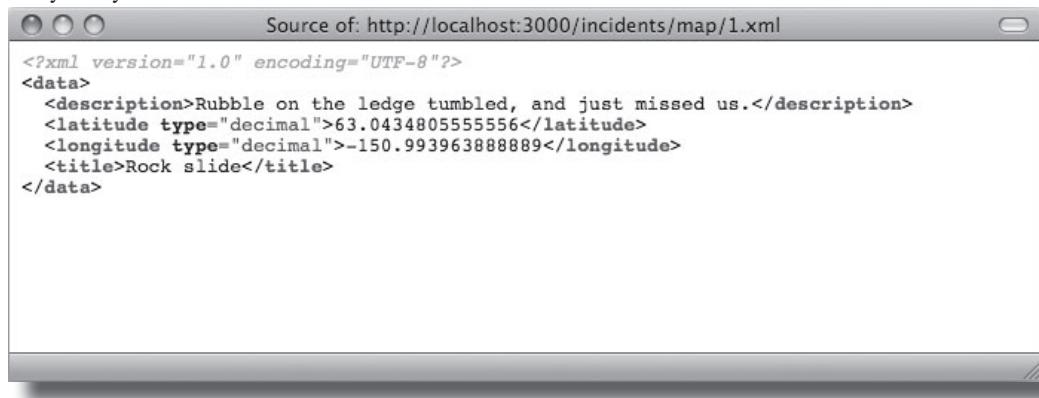


Jazda próbna

Jeśli odwiedzimy wersję strony znajdującą się pod adresem:

<http://localhost:3000/incidents/map/1.xml>

otrzymamy zdarzenie w formacie XML:



```

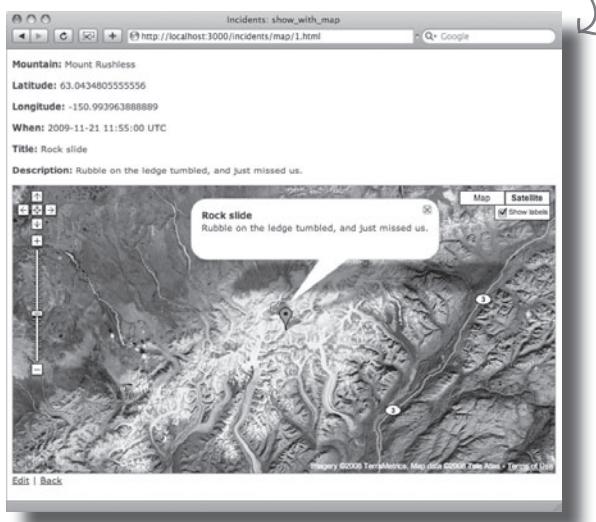
Source of: http://localhost:3000/incidents/map/1.xml

<?xml version="1.0" encoding="UTF-8"?>
<data>
  <description>Rubble on the ledge tumbled, and just missed us.</description>
  <latitude type="decimal">63.0434805555556</latitude>
  <longitude type="decimal">-150.993963888889</longitude>
  <title>Rock slide</title>
</data>

```

A jak wygląda wersja HTML?

<http://localhost:3000/incidents/map/1.html>



Wszystko działa! Teraz różne zdarzenia pokazują różne mapy.
Zanim jednak zastąpimy kod działającej aplikacji, lepiej upewnijmy się,
że dokładnie rozumiemy, jak ten kod działa.

Co się zatem tak naprawdę tutaj dzieje?

Jakiego formatu żądamy?

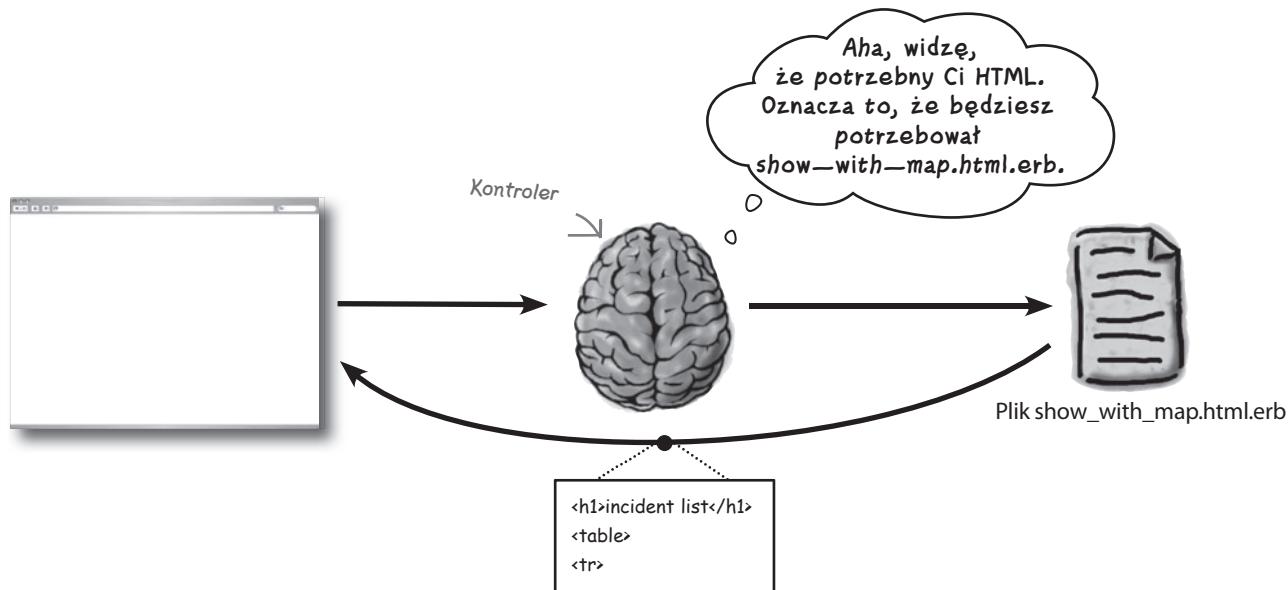
Jak działa strona z mapą?

Przyjrzyjmy się bliżej temu, co się właśnie wydarzyło, oraz sposobowi wygenerowania strony HTML.

1

Kontroler zauważa, że potrzebna jest strona HTML.

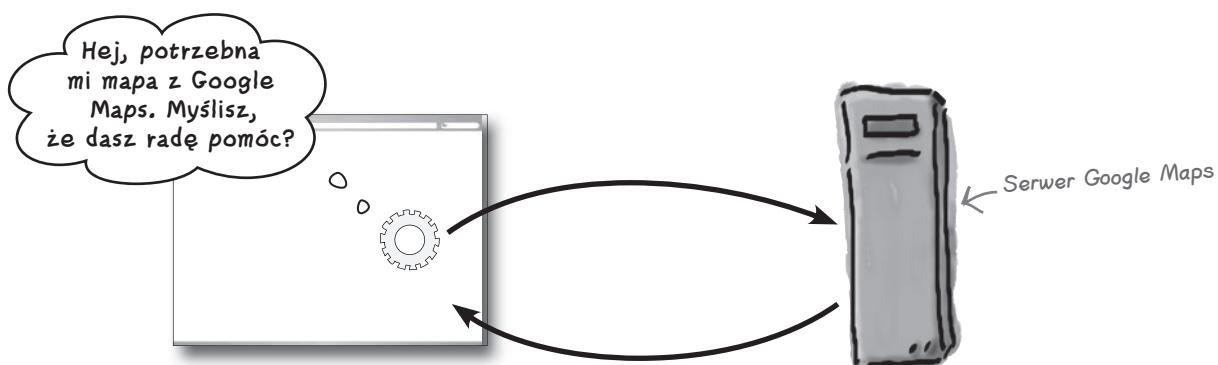
Przeglądarka kieruje się do wersji HTML strony. Kontroler widzi, że wymagany jest kod HTML, a nie XML, dlatego wywołuje szablon `show_with_map.html.erb`. Do przeglądarki klienta odsyłana jest strona HTML.



2

Kod w języku JavaScript żąda mapy z serwisu Google Maps.

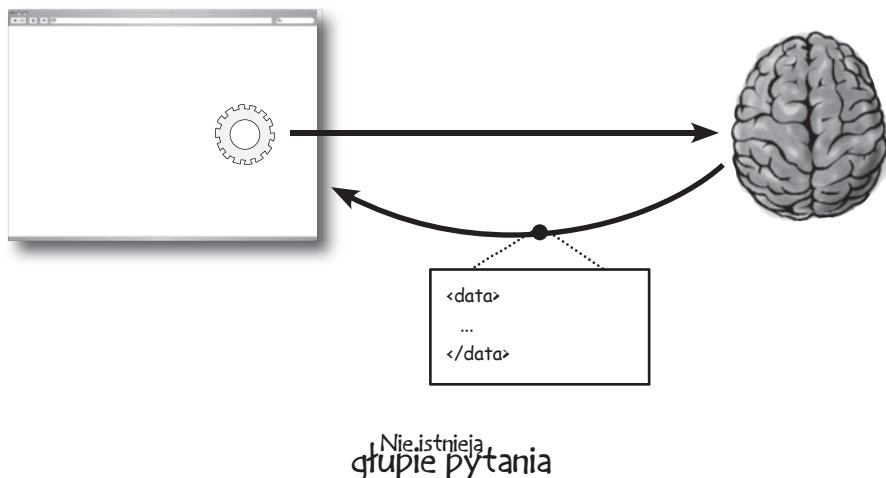
Kod w JavaScriptie znajdujący się na stronie żąda danych mapy z serwera Google Maps. Serwer Google Maps zwraca te dane.



3

Kod w języku JavaScript żąda danych XML zdarzenia.

Kod w JavaScriptie znajdujący się na stronie internetowej żąda od kontrolera danych XML dla zdarzenia. Następnie wyświetla dane na mapie.



P: Twierdzicie, że zasoby powinny zawsze mieć te same adresy URL. Dlaczego tak jest?

O: Nie muszą, ale architektura REST — podstawowa reguła projektowa Rails — mówi, że tak powinno być.

P: Ale jeśli format podany jest w adresie URL, czy nie oznacza to, że różne adresy URL wykorzystywane są dla tych samych zasobów?

O: Oczywiście, że tak. Dodanie formatu do adresu URL nieco nagina architekturę REST... tylko troszeczkę. Jest to jednak często stosowana sztuczka. Jest prosta i działa dobrze.

P: Nie ma zatem możliwości użycia tego samego adresu URL dla różnych formatów?

O: Istnieje rozwiązanie na to pozwalające. Jeśli żądanie zawiera nagłówek „Accepts:” mówiący na przykład, że żądanie dotyczy „text/xml”, responder wykona kod dla formatu XML.

P: Czy istnieje jakiś sposób na wymienienie atrybutów, których nie chcemy uwzględnić w danych wyjściowych metody `to_xml?`

O: Tak. Zamiast korzystać z parametru `:only`, można użyć parametru `:except`. Platforma Rails jest niesamowicie spójna i znajdziesz w niej wiele miejsc, w których żądania mają opcjonalne parametry `:only`. W każdym przypadku możesz zamienić je na parametry `:except` określające, czego *nie chcesz*.

P: Czy kontroler może w jakiś sposób odróżnić żądanie oparte na Ajaksie skierowane za pomocą JavaScriptu od żądania przeglądarki?

O: W pewnym sensie tak. Wyrażenie `request.xhr?` zazwyczaj zwraca `true` dla żądań opartych na Ajaksie i `false` dla prostych żądań przeglądarki. Problem polega na tym, że choć działa to dla żądań wygenerowanych za pomocą biblioteki Prototype, nie działa dla wszystkimi bibliotekami Ajksa.

P: Dlaczego czasami muszę wywoływać metodę `render`, a czasami nie?

O: Jeśli wystarczy Ci wykonanie szablonu domyślnego (tego, którego nazwa pasuje do akcji), możesz pominąć wywołanie metody `render`.

P: Piszcie, że wygenerowane kody XML oraz HTML to różne reprezentacje, ale nie zawierają one tych samych informacji, prawda?

O: To prawda — nie zawierają. Kod XML generowany dla pojedynczego zdarzenia zawiera mniejszą ilość danych od reprezentacji HTML. Obie formy prezentują jednak informacje dotyczące tych samych zasobów, dlatego są reprezentacjami tej samej rzeczy.

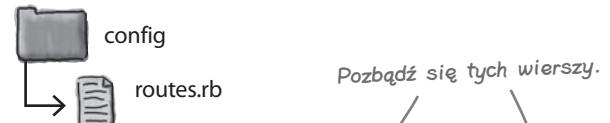
Kod jest gotowy do opublikowania

Nowa wersja strony z lokalizacją zdarzenia działa dobrze, dlatego zastąpimy teraz opartą na rusztowaniu akcję show kodem show_with_map.

1

Usunięcie tras.

Na potrzeby kodu testowego utworzyliśmy własne trasy, dlatego musimy usunąć je teraz z pliku routes.rb:

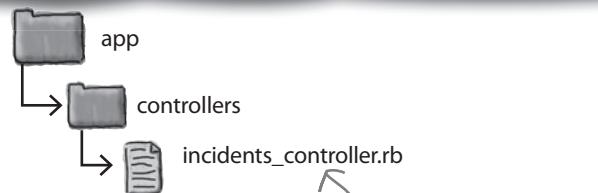


```
ActionController::Routing::Routes.draw do |map|
  map.connect 'incidents/map/:id', :action=>'show_with_map', :controller=>'incidents'
  map.connect 'incidents/map/:id.:format', :action=>'show_with_map', :controller=>'incidents'
end
map.resources :incidents
```

2

Zmiana nazwy metody show_with_map w kontrolerze.

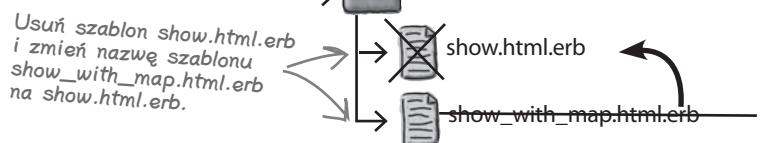
Metoda show_with_map stanie się teraz naszą nową metodą show. Usuń zatem istniejącą metodę show i zmień nazwę metody show_with_map na show.



3

Następnie zmień nazwę szablonu show_with_map.html.erb.

Oznacza to, że musimy usunąć istniejący szablon show.html.erb i zastąpić go szablonem show_with_map.html.erb.



Nie istniejąca
główne pytania

P: Skoro trasa zniknęła, jak wybrany zostanie teraz właściwy format?

O: Trasa map.resources tworzy cały zbiór tras. Wszystkie te trasy zawierają format.

P: Jak to się dzieje, że strona indeksująca przeszła do /incidents/1 zamiast do /incidents/1.html? Skąd platforma Rails wieǳiała, że będzie to strona HTML?

O: Jeśli format nie jest podany, Rails zakłada, że jest nim HTML... co mogliśmy tutaj wykorzystać.

P: Co oznacza map.resources?

O: Kod ten generuje standardowy zbiór tras wykorzystywany przez rusztowanie.



Jazda próbna

Teraz strony z mapami zastąpiły domyślną akcję show. Główna strona indeksująca zawiera już odnośniki do stron z mapami, a nie ich wersjami tekstowymi.

The screenshot illustrates a web application for managing incidents on mountains. It consists of three main windows:

- Main Listing Page:** Shows a table of incidents with columns: Mountain, Latitude, Longitude, When, Title, and Description. Each row has "Show", "Edit", and "Destroy" links. A "New Incident" button is at the bottom left.
- Incident Detail View:** Shows a single incident with fields: Mountain, Latitude, Longitude, When, Title, and Description. Below this is a large text area containing the description. At the bottom are "Edit" and "Back" buttons.
- Map View:** Shows a satellite map of southern India and Sri Lanka. A specific location is highlighted with a marker and a callout box containing the incident details from the second window. The map includes zoom controls and a legend.

Arrows indicate the flow of data from the main listing to the detail view, and from the detail view to the map view.

Pozostaje jedna rzecz — czy strona indeksująca nie jest nieco... nudna? Szczególnie w porównaniu ze wszystkimi ciekawymi wizualnie stronami map!



Użytkownicy pytali, czy strona indeksująca mogłaby wyświetlać pełny zbiór odnotowanych zdarzeń. Na szczęście szablon częściowy `_map.html.erb` potrafi generować większą liczbę punktów na mapie, jeśli przekażemy do niego poprawne dane XML.

Poniżej znajduje się kod metody `index` kontrolera `incidents`. Przepisz tę metodę tak, by generowała ona kod XML z tablicy wszystkich zdarzeń. Będziesz musiał zmienić element główny na `data`.

```
def index
  @incidents = Incident.find(:all)

  respond_to do |format|
    format.html # index.html.erb
    format.xml { render :xml => @incidents }
  end
end
```



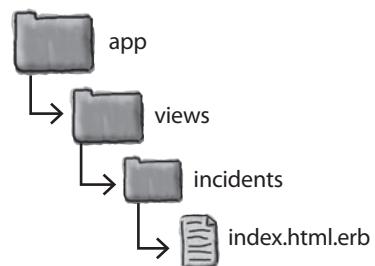
Strona indeksująca będzie musiała zawierać mapę. Napisz kod wstawiający mapę we wskazanym miejscu. Będziesz musiał przekazać ścieżkę strony indeksującej w formacie XML jako dane mapy.

```
<h1>Listing incidents</h1>


| Mountain                   | Latitude                   | Longitude                   | When                   | Title                   | Description                   |                                 |                                                     |                                                                                     |
|----------------------------|----------------------------|-----------------------------|------------------------|-------------------------|-------------------------------|---------------------------------|-----------------------------------------------------|-------------------------------------------------------------------------------------|
| <%= h incident.mountain %> | <%= h incident.latitude %> | <%= h incident.longitude %> | <%= h incident.when %> | <%= h incident.title %> | <%= h incident.description %> | <%= link_to 'Show', incident %> | <%= link_to 'Edit', edit_incident_path(incident) %> | <%= link_to 'Destroy', incident, :confirm => 'Are you sure?', :method => :delete %> |


```

```
.....
<br />
<%= link_to 'New incident', new_incident_path %>
```





Ćwiczenie (nieco dłuższe)

Rozwiązanie

Użytkownicy pytali, czy strona indeksująca mogłaby wyświetlać pełny zbiór odnotowanych zdarzeń. Na szczęście szablon częściowy `_map.html.erb` potrafi generować większą liczbę punktów na mapie, jeśli przekażemy do niego poprawne dane XML.

Poniżej znajduje się kod metody `index` kontrolera `incidents`. Przepisz tę metodę tak, by generowała ona kod XML z tablicy wszystkich zdarzeń. Będziesz musiał zmienić element główny na `data`.

```
def index
  @incidents = Incident.find(:all)

  respond_to do |format|
    format.html # index.html.erb
    format.xml { render :xml => @incidents }
  end
end

def index
  @incidents = Incident.find(:all)

  respond_to do |format|
    format.html # index.html.erb
    format.xml {
      render :text=>@incidents.to_xml(:root=>"data")
    }
  end
end
```



Strona indeksująca będzie musiała zawierać mapę. Napisz kod wstawiający mapę we wskazanym miejscu. Będziesz musiał przekazać ścieżkę strony indeksującej w formacie XML jako dane mapy.

```

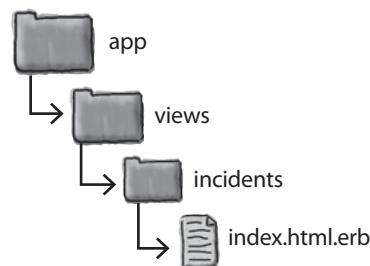
<h1>Listing incidents</h1>


| Mountain                         | Latitude                               | Longitude                                | When                                                         | Title                                                                                        | Description                      |
|----------------------------------|----------------------------------------|------------------------------------------|--------------------------------------------------------------|----------------------------------------------------------------------------------------------|----------------------------------|
| <% for incident in @incidents %> | <td><%= h incident.mountain %></td>    | <td><%= h incident.latitude %></td>      | <td><%= h incident.longitude %></td>                         | <td><%= h incident.when %></td>                                                              | <td><%= h incident.title %></td> |
| <% end %>                        | <td><%= h incident.description %></td> | <td><%= link_to 'Show', incident %></td> | <td><%= link_to 'Edit', edit_incident_path(incident) %></td> | <td><%= link_to 'Destroy', incident, :confirm => 'Are you sure?', :method => :delete %></td> |                                  |
|                                  |                                        |                                          |                                                              |                                                                                              |                                  |


<% render (:partial=>'map', :locals=>{:data=>"/incidents.xml"}) %>
  

<br />
<%= link_to 'New incident', new_incident_path %>

```





Jazda próbna

Gdy teraz użytkownicy odwiedzają stronę główną, widzą zdarzenia na liście, a także na mapie. Po kliknięciu zdarzenia wyświetlane są jego szczegóły wraz z odnośnikiem do osobnej podstrony zdarzenia.

Do utworzenia punktów mapa wykorzystuje kod XML wygenerowany przez metodę „index” kontrolera.

Wszystkie zdarzenia zaznaczone są teraz na mapie.

Okno informacyjne zawiera odnośnik do własnej podstrony „show” zdarzenia.

The screenshot shows two browser windows. The top window is titled 'Incidents: index' and lists five incidents with columns for Mountain, Latitude, Longitude, When, Title, and Description. The bottom window is titled 'Incidents: show' and displays a detailed view of the third incident, which occurred on Mount Lotopaxo on June 7, 2009, at approximately -0.683975 latitude and -76.436505555556 longitude. The description states: 'Ascent Living only on dried chicken pieces, we completed our 4 day...'. A callout bubble points from this description to the 'More...' link in the original list. Arrows also point from the 'More...' link to the 'Description' field in the detailed view and from the 'Description' field to the map below. The map shows the location of the incident in South America, specifically in Ecuador. The bottom window also includes a 'Map' and 'Satellite' button and a 'Show labels' checkbox.



Większość witryn internetowych udostępnia teraz **kanały RSS**, które zawierają zgromadzone w prosty sposób odnośniki do głównych zasobów strony.

Jak jednak wygląda kanał RSS?

Kanały RSS to po prostu kod XML

Tak wyglądałby plik kanału RSS ze strony dla wspinaczy:

```
<rss version="2.0">
  <channel>
    <title>Head First Climbers News</title>
    <link>http://localhost:3000/incidents/</link>
    <item>
      <title>Rock slide</title>
      <description>Rubble on the ledge tumbled, and just missed us.</description>
      <link>http://localhost:3000/incidents/1</link>
    </item>
    <item>
```

To po prostu plik XML. Jeśli korzystasz z czytnika kanałów RSS albo jeśli Twoja przeglądarka potrafi subskrybować kanały RSS, pobierasz takie właśnie pliki, zawierające listę odnośników wraz z opisami aktualności.

Jak możemy wygenerować kanał RSS taki jak ten?



Czy ktorekolwiek ze znaczników w pliku kanału RSS wyglądają szczególnie zaskakująco lub niejasno? Jak sądzisz, co robi znacznik `channel`? A co znacznik `link`?

Utworzmy akcję o nazwie news

Utworzmy następującą nową trasę na samej górze pliku definicji tras:

```
map.connect '/incidents/news', :action=>'news', :controller=>'incidents', :format=>'xml'
```



Zaostrz ołówek

Napisz metodę kontrolera dla nowej akcji. Będzie ona musiała odnaleźć wszystkie zdarzenia uaktualnione w ciągu ostatnich 24 godzin. Powinna następnie wygenerować domyślny kod XML, wywołując metodę `to_xml` na tablicy pasujących zdarzeń.

Wskazówka: wyrażenie języka Ruby `Time.now.yesterday` zwraca wartość daty i czasu sprzed dokładnie 24 godzin.



Zaostrz ołówek

Rozwiążanie

Napisz metodę kontrolera dla nowej akcji. Będzie ona musiała odnaleźć wszystkie zdarzenia uaktualnione w ciągu ostatnich 24 godzin. Powinna następnie wygenerować domyślny kod XML, wywołując metodę `to_xml` na tablicy pasujących zdarzeń.

Wskazówka: wyrażenie języka Ruby `Time.now.yesterday` zwraca wartość daty i czasu sprzed dokładnie 24 godzin.

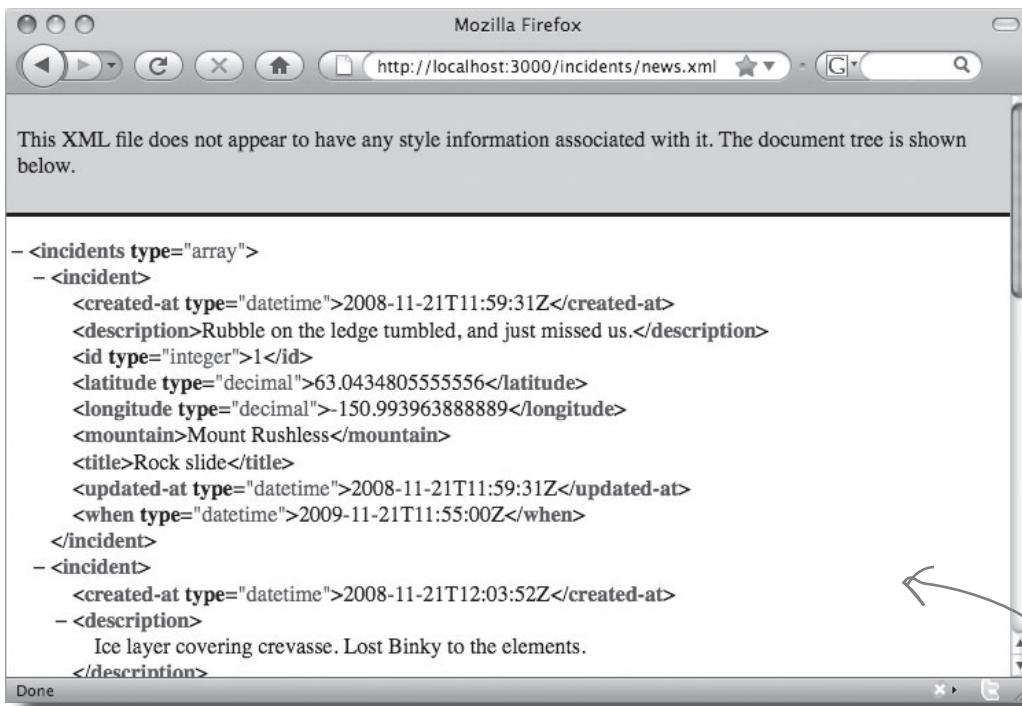
```
def news
  @incidents = Incident.find(:all, :conditions=>'updated_at > ?', Time.now.yesterday)
  render :xml=>@incidents
end
```

Można było również użyć „:text=>@incidents.to_xml”.



Jazda próbna

Oto plik XML wygenerowany przez akcję news:



```

<incidents type="array">
  <incident>
    <created-at type="datetime">2008-11-21T11:59:31Z</created-at>
    <description>Rubble on the ledge tumbled, and just missed us.</description>
    <id type="integer">1</id>
    <latitude type="decimal">63.0434805555556</latitude>
    <longitude type="decimal">-150.993963888889</longitude>
    <mountain>Mount Rushless</mountain>
    <title>Rock slide</title>
    <updated-at type="datetime">2008-11-21T11:59:31Z</updated-at>
    <when type="datetime">2009-11-21T11:55:00Z</when>
  </incident>
  <incident>
    <created-at type="datetime">2008-11-21T12:03:52Z</created-at>
    <description>
      Ice layer covering crevasse. Lost Binky to the elements.
    </description>
  </incident>

```

Wygenerowaliśmy kod XML dla poprawnych danych, jednak nie jest to rodzaj XML, jaki potrzebujemy dla kanału RSS. Nie powinno nas to jednak szczególnie martwić, spotkaliśmy się już wcześniej z takim problemem. Kiedy generowaliśmy dane XML dla lokalizacji, miały one niepoprawny format — i wtedy udało nam się to poprawić.

**Musimy po prostu zmodyfikować ten kod XML
w ten sam sposób... prawda?**

Pamiętaj: kod ten
uzależniony jest
od czasu, dlatego
pojawia się tu
tylko zdarzenia
zmodyfikowane
w ciągu ostatnich
24 godzin.



WYSIL
SZARE KOMÓRKI

Czy konwersja kodu XML w taki sposób, by odpowiadał on strukturze kanałów RSS, może być dla nas problemem?

Musimy zmienić strukturę kodu XML

Metoda `to_xml` pozwala na wprowadzenie kilku prostych zmian do zwracanego kodu XML. Możemy zmieniać nazwy i wybierać, które dane uwzględnić. Czy da nam jednak wystarczające możliwości pozwalające na przekształcenie kodu XML, który *mamy*, na ten, który *chcemy* mieć?

To mamy...
↓

```
<?xml version="1.0" encoding="UTF-8"?>
<incidents type="array">
  <incident>
    <created-at type="datetime">2008-11-21T11:59:31Z</created-at>
    <description>Rubble on the ledge tumbled, and just missed us.</description>
    <id type="integer">1</id>
    <latitude type="decimal">63.043480555556</latitude>
    <longitude type="decimal">-150.993963888889</longitude>
    <mountain>Mount Rushless</mountain>
    <title>Rock slide</title>
  </incident>
<rss version="2.0">
  <channel>
    <title>Head First Climbers News</title>
    <link>http://localhost:3000/incidents/</link>
    <item>
      <title>Rock slide</title>
      <description>Rubble on the ledge tumbled, and just missed us.</description>
      <link>http://localhost:3000/incidents/1</link>
    </item>
    <item>
```

... a to chcemy mieć.
↑

Potrzebne nam większe MOŻLIWOŚCI

Kod XML kanału RSS nie może być generowany za pomocą metody `to_xml`. Choć metoda ta lekko modyfikuje zwracany kod XML, nie potrafi radykalnie zmienić jego struktury. Przykładowo metoda `to_xml` nie jest w stanie przenosić elementów pomiędzy poziomami. Nie potrafi grupować elementów wewnętrz innych elementów. Metoda ta zaprojektowana została tak, by można z niej było łatwo i szybko korzystać, jednak sprawia to, że jest nieco mało elastyczna.

Pełne możliwości w zakresie XML da nam coś innego...

Użyjemy nowego typu szablonu — XML Builder

Gdybyśmy utworzyli kolejny szablon strony HTML, moglibyśmy generować dowolne dane wyjściowe w formacie XML. W końcu HTML jest bardzo podobny do XML:

```
<rss version="2.0">
  <channel>
    <title>Head First Climbers News</title>
    <link>http://localhost:3000/incidents/</link>
    <% for incident in @incidents %>
      <item>
        <title><%= h incident.title %></title>
        <description><%= h incident.description %></description>
```

Ten kod wygląda w zasadzie podobnie do HTML...

Platforma Rails udostępnia specjalny typ szablonu zaprojektowany do generowania XML. Nosi on nazwę **szablonu XML Builder**.

Szablony XML Builder znajdują się w tym samym katalogu co szablony stron i są wykorzystywane w podobny sposób. Jeśli ktoś żąda odpowiedzi w formacie XML (dodając rozszerzenie `.xml` na końcu adresu URL), kontroler musi jedynie wczytać dane z modelu, a Rails automatycznie wywoła szablon XML Builder.

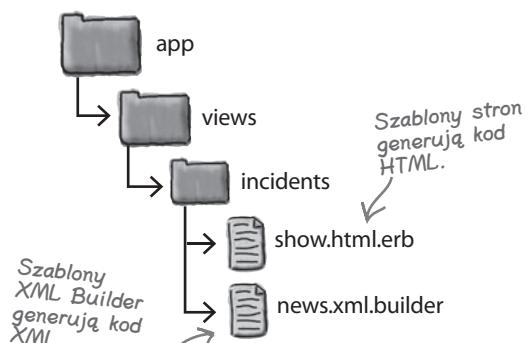
Oznacza to, że z akcji `news` możemy usunąć jeden wiersz:

```
def news
  @incidents = Incident.find(:all, :conditions=>['updated_at > ? ', Time.now.yesterday])
  render :xml=>@incidents
end
```

To kod metody „news” kontrolera „incidents”.

Powyższy kod wczyta teraz jedynie dane z modelu, a szablon XML Builder zrobi resztę.

Jak wygląda zatem szablon XML Builder?





Szablony XML Builder z bliska

Szablony stron zaprojektowane zostały tak, by wyglądały jak pliki HTML z odrobiną języka Ruby. Szablony XML Builder są inne. To czysty język Ruby, a szablony zaprojektowane są w taki sposób, by zachować strukturę podobną do kodu XML. Na przykład takie coś:

```
xml.sentence(:language=>'English') {  
  for word in @words do  
    xml.word(word)  
  end  
}
```

może wygenerować coś, co wygląda tak:

```
<sentence language="English"> ← Atrybut  
  <word>XML</word>  
  <word>Builders</word>  
  <word>Kick</word> ← Elementy  
  <word>Ass!</word>  
</sentence>
```

Dlaczego autorzy Rails utworzyli inny typ szablonu? Czemu szablony XML Builder nie działają tak samo jak szablony stron? Dlaczego nie używają Embedded Ruby?

Choć XML i HTML są bardzo podobne — a w przypadku XHTML z technicznego punktu widzenia są równe — sposoby używania HTML oraz XML nieco się od siebie różnią.

- Strony internetowe zawierają zazwyczaj mnóstwo znaczników **HTML** sprawiających, że strona wygląda ładnie, a tylko *niewiele* danych pochodzących z bazy danych.
- Większość zawartości plików **XML** pochodzi z kolei z danych oraz logiki warunkowej; o wiele mniej ze znaczników XML.

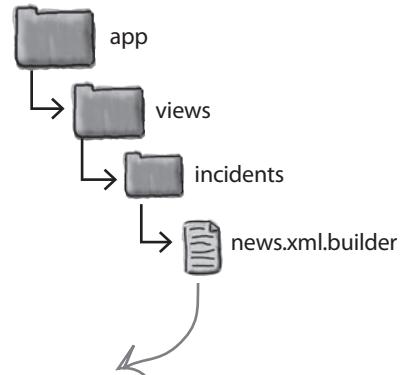
Dzięki użyciu Ruby (zamiast XML) jako podstawowego języka szablony XML Builder są bardziej zwiędłe i łatwiejsze w utrzymaniu.

Łamigłówka

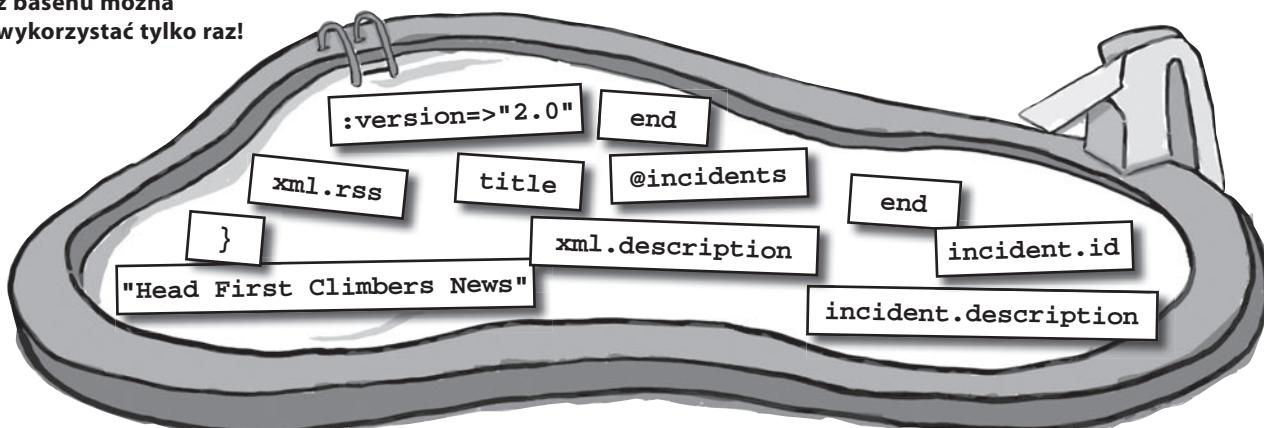


Twoje **zadanie** polega na zebraniu fragmentów kodu z basenu i umieszczeniu ich w pustych miejscach w kodzie szablonu. **Nie możesz** użyć tego samego fragmentu więcej niż jeden raz i nie będziesz potrzebował wszystkich fragmentów. Twoim **celem** jest uzupełnienie szablonu XML Builder generującego kanał RSS.

```
.....( _____ ) {  
    xml.channel {  
        xml.title( _____ )  
        xml.link("http://localhost:3000/incidents/")  
        for incident in .....  
            xml.item {  
                xml._____ (incident.title)  
                _____ ( _____ )  
                xml.link("http://localhost:3000/incidents/#{ _____ }")  
            }  
        .....  
    }  
}
```



Uwaga: każdy element z basenu można wykorzystać tylko raz!

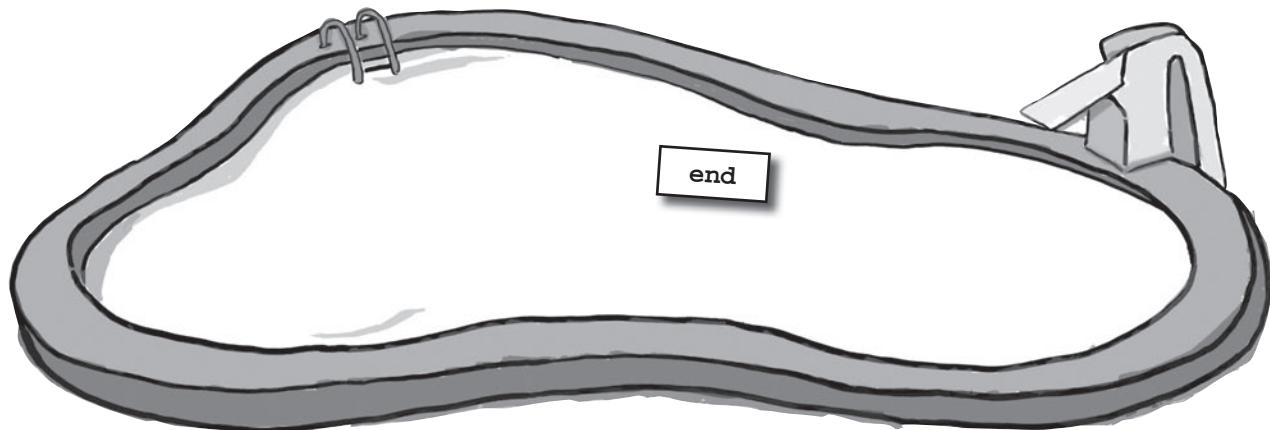
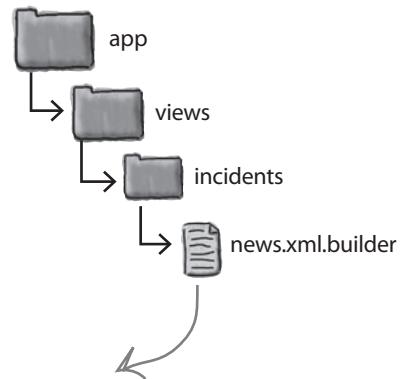


Łamigłówka: Rozwiążanie



Twoje **zadanie** polega na zebraniu fragmentów kodu z basenu i umieszczeniu ich w pustych miejscach w kodzie szablonu. **Nie możesz** użyć tego samego fragmentu więcej niż jeden raz i nie będziesz potrzebował wszystkich fragmentów. Twoim **celem** jest uzupełnienie szablonu XML Builder generującego kanał RSS.

```
xml.rss ( :version=>"2.0" ) {  
    xml.channel {  
        xml.title( "Head First Climbers News" )  
        xml.link( "http://localhost:3000/incidents/" )  
        for incident in @incidents ...  
            xml.item {  
                xml.title(incident.title)  
                xml.description ( incident.description )  
                xml.link("http://localhost:3000/incidents/#{{ incident.id }}")  
            }  
        }  
    }  
}
```



Teraz dodajmy kanały RSS do stron

Jak użytkownicy mogą odnaleźć kanały RSS? Przeglądarki odkrywają obecność kanału RSS dzięki odnalezieniu odwołania `<link... />` na stronie internetowej.

Właściciele witryny Head First Climbers chcą, by kanały RSS widoczne były na każdej stronie, dlatego dodamy odwołanie do kanału RSS do pliku układu strony zdarzenia za pomocą metody pomocniczej `auto_discovery_link`:

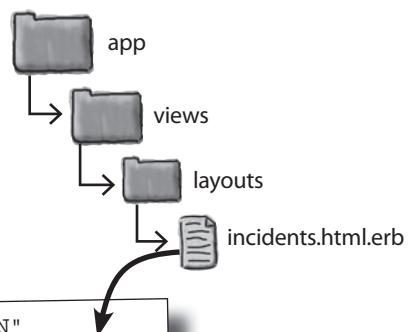
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <title>Incidents: <%= controller.action_name %></title>
    <%= stylesheet_link_tag 'scaffold' %>
    <%= auto_discovery_link_tag(:rss, { :action=>/news'}) %>
  </head>
  <body>

    <p style="color: green"><%= flash[:notice] %></p>

    <%= yield %>

  </body>
</html>
```



Powinna ona utworzyć następujący odnośnik:

```
<link href="http://localhost:3000/incidents/news.xml"
  rel="alternate" title="RSS" type="application/rss+xml" />
```

**By jednak zobaczyć, czy to działa,
musimy znowu uruchomić przeglądarkę.**

Dla kogo kanał RSS?



Jazda próbna

Gdy teraz użytkownik udaje się na stronę internetową, w przeglądarce pojawia się ikona kanału RSS:

Różne przeglądarki pokazują znalezienie kanału RSS w odmienny sposób.

The image shows two side-by-side browser windows. Both have the URL <http://localhost:3000/incidents/> in the address bar. The left browser is a Mac OS X style with a standard toolbar. The right browser is a Google Chrome style with a magnifying glass icon. In both windows, there is a small orange circular icon with a white 'RSS' symbol located in the top right corner of the main content area. The content itself is a table titled 'Listing incidents' with columns for Mountain, Latitude, Longitude, When, Title, and Description. It lists two entries: 'Rock slide' and 'Hidden crevasses'.

Mountain	Latitude	Longitude	When	Title	Description
Mount Rushmore	63.0434805555556	150.993963888889	2009-11-21 11:55:00 UTC	Rock slide	Rubble on the ledge tumbled, and just missed us.
Mount	63.0780527222222		2009-11-21	Hidden	Ice layer covering crevasses. Lost Binky to the elements.

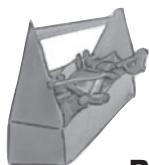
Po zasubskrybowaniu kanału RSS (albo po prostu odczytaniu go) użytkownik zobaczy odnośniki do zdarzeń, które zostały odnotowane w ciągu ostatnich 24 godzin.

The image shows a desktop application window titled 'Head First Climbers News'. The address bar says 'feed:<http://localhost:3000/incidents/news.xml>'. The main content area displays four news items: 'Rock slide' (Today, 6:27 PM), 'Hidden crevasses' (Today, 6:27 PM), 'Ascent' (Today, 6:27 PM), and 'Altitude sickness' (Today, 6:27 PM). Each item has a brief description and a 'Read more...' link. To the right of the main content is a sidebar with filters for 'Search Articles', 'Article Length', 'Sort By' (Date, Title, Source, New), and a 'Recent Articles' section with links for All, Today, Yesterday, and Last Seven Days.

Zdobyłeś szczyt!

Jedna z pierwszych wiadomości opublikowana zostaje na stronie przez naszego nieustraszonego wspinacza; tysiące użytkowników mogą dowiedzieć się o jego niesamowitych osiągnięciach.





Niezbędnik programisty Rails

Masz za sobą rozdział 8. i teraz
do swojego niezbędnika programisty
Rails możesz dodać umiejętność
wykorzystania XML do tworzenia różnych
reprezentacji Twojej strony.

Narzędzia Rails

Metoda „to_xml” generuje kod XML dla dowolnego obiektu modelu.

Parametry „:only” oraz „:root” pozwalają na modyfikację formatu „to_xml”.

Metoda „respond_to” tworzy obiekt „respondera”, który pomoże Ci wygenerować kilka reprezentacji dla jednego zasobu.

Szablony XML Builder przypominają szablony stron stujące do generowania XML.

Szablony XML Builder dają większą elastyczność niż użycie metody „to_xml”.

Respondery ustawiają typ MIME odpowiedzi, a także decydują, czy należy wywołać szablon strony, czy też szablon XML Builder.

9. Architektura REST i Ajax



Kolejne kroki



To taka wspaniała
trasa...

Czas skonsolidować umiejętności w zakresie korzystania z aplikacji typu mashup.

Dotychczas widzieliśmy, jak w celu pokazania danych geograficznych można dodać do naszych aplikacji mapy z serwisu **Google Maps**. Co jednak, jeśli chcemy **rozszerzyć istniejącą już funkcjonalność**? Czytaj dalej, a przekonasz się, jak można wzbogacić aplikacje typu **mashup o bardziej zaawansowane cudańka oparte na Ajaksie**. Co więcej, przy okazji nauczysz się też nieco o architekturze **REST**.

Dla kogo osuwisko?

Zdarzeń jest zbyt dużo!

Po poprawieniu interfejsu użytkownika aplikacji liczba osób odwiedzających witrynę Head First Climbers znacznie wzrosła. Problem polega na tym, iż odnotowywanych jest tyle zdarzeń, że użytkownicy mają problem, by je wszystkie przejrzeć.



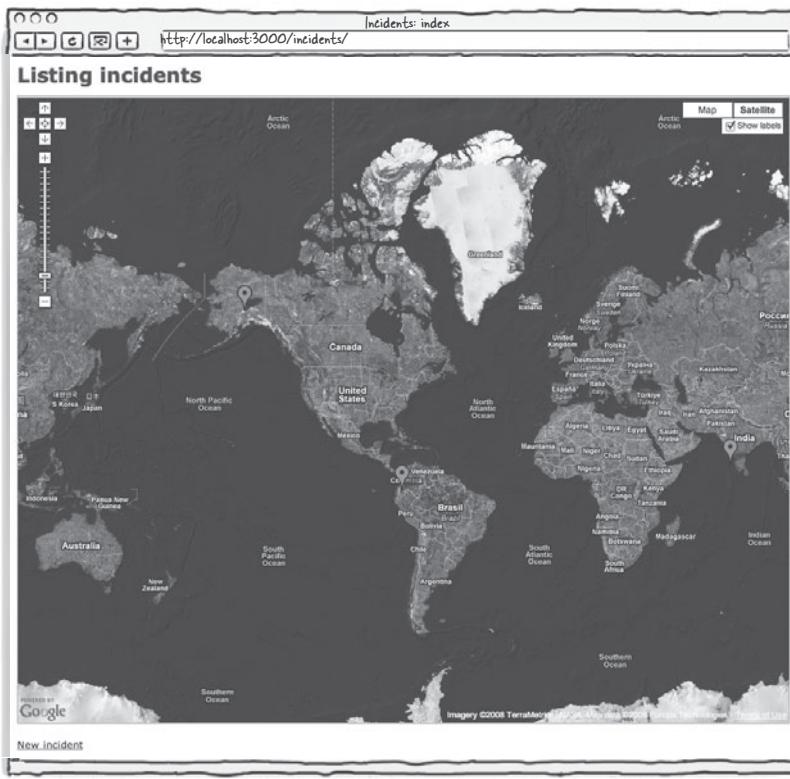
Strona indeksująca wyświetla informacje na dwa sposoby:

- 1 Na górze strony znajduje się szczegółowa lista zdarzeń wraz z szerokościami i długościami geograficznymi. Problem polega na tym, że większość osób przewija tę część w celu dostania się do mapy znajdującej się na dole strony.
- 2 Na mapie pokazującej się po kliknięciu zdarzenia widoczna jest ograniczona liczba szczegółów. Problem polega na tym, że na mapie nie są wyświetlane wszystkie dane.

Żadne z tych rozwiązań nie jest w pełni satysfakcyjne. Trudno jest zlokalizować zdarzenia na liście, dlatego też dodaliśmy mapę. Mapa nie jest jednak w stanie wyświetlać wszystkich dostępnych danych. Co powinniśmy zrobić w tej sytuacji?

Mapa mogłaby pokazywać więcej szczegółów

Idealnym rozwiązaniem byłoby, gdyby mapa *pokazywała więcej*. Gdyby można ją było zmodyfikować, tak by wyświetlała bardziej przydatne informacje dotyczące zdarzeń, moglibyśmy najprawdopodobniej usunąć listę zdarzeń i sprawić, by strona główna składała się z jednej wielkiej mapy z wieloma funkcjami. Oznaczałoby to na przykład, że nie musielibyśmy przechodzić do osobnych podstron w celu wprowadzenia większej ilości danych.



Jest tylko jeden problem: szablon częściowy mapy został *pobrany z Internetu*. Jest prosty w *użyciu*, ale czy naprawdę powinniśmy zmieniać kod? Na szczęście istnieje technika programistyczna wykorzystywana przez szablon częściowy mapy, która wiąże się z tym, że nie będziemy musieli w ogóle dotykać pobranego kodu. Co to będzie?



WYSIL
SZARE KOMÓRKI

Jak sądzisz, jak moglibyśmy poradzić sobie z wprowadzeniem zmian do mapy **bez** modyfikowania pobranego kodu?

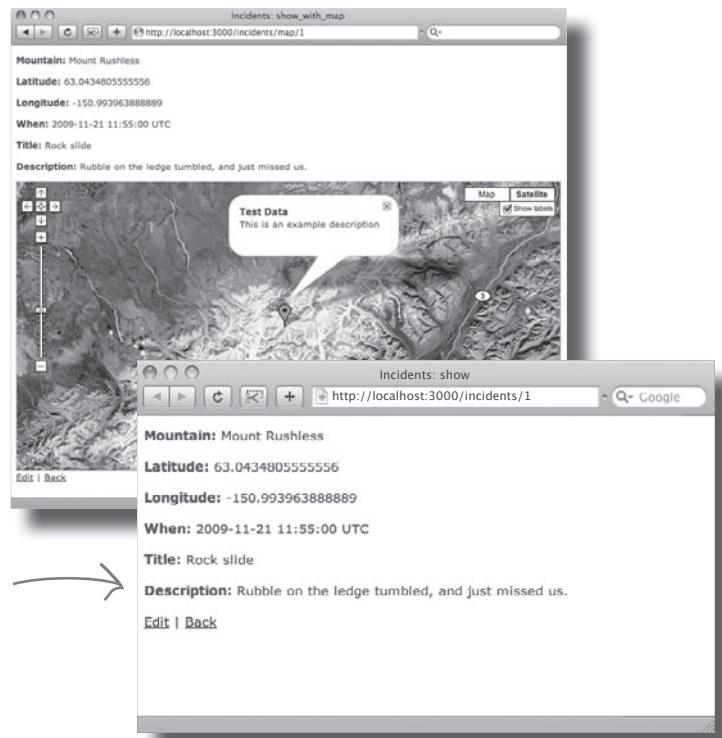
Możemy rozszerzyć funkcjonalność mapy za pomocą Ajaksa

Osoby, które utworzyły szablon częściowy mapy, zauważą, że użytkownicy wkrótce będą chcieli rozszerzyć sposób działania mapy. Ponieważ szablon częściowy mapy wywołuje Google Maps, a serwis ten zbudowany został w oparciu o technologię Ajax, rozszerzanie funkcjonalności mapy można osiągnąć za pomocą Ajaksa.

W tej chwili mapa działa dzięki wykonywaniu żądań do serwera proszących o plik XML zawierający szczegóły wszystkich zdarzeń wspinaczkowych odnotowanych w systemie. Domyślnie mapa wyświetla tytuł oraz opis zawarty w kodzie XML.

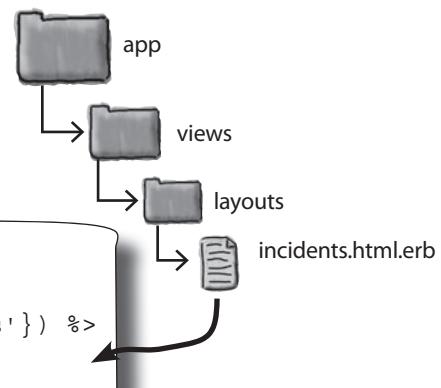
Szablon częściowy mapy pozwala również przekazywać nazwę **akcji**, która wyświetli informacje o zdarzeniu. Może to być dowolna akcja, dlatego gdybyśmy chcieli, moglibyśmy wygenerować coś, co przypominałoby oryginalną wersję strony pokazującej zdarzenie.

Musimy wyświetlić te dane wyjściowe w oknie informacyjnym mapy.



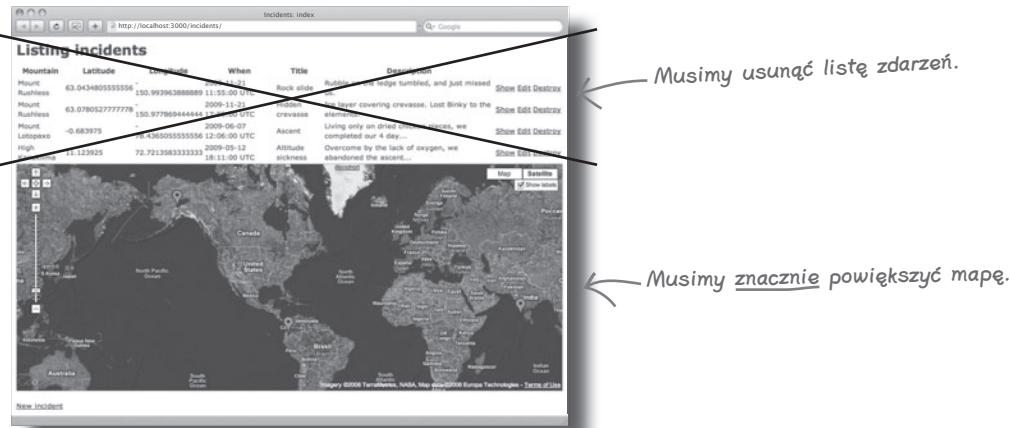
Szablon częściowy `_map.html.erb` powinien wykonywać żądania oparte na Ajaksie, co oznacza, że będzie on potrzebował dostępu do biblioteki Prototype. Możemy to umożliwić w taki sam sposób jak wcześniej, dodając odwołanie do biblioteki w pliku układu strony — jak poniżej:

```
<%= stylesheet_link_tag 'scaffold' %>
<%= auto_discovery_link_tag(:rss, { :action=>'news' } ) %>
<%= javascript_include_tag '/prototype' %>
</head>
```



Jak jednak możemy przekształcić stronę indeksującą?

Pierwszymi rzecząmi, jakie musimy zrobić, by zmienić stronę główną, jest usunięcie listy zdarzeń oraz powiększenie mapy:

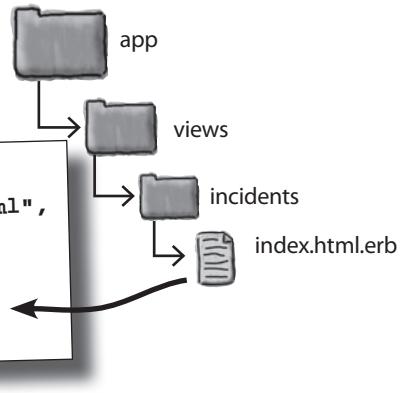


Musimy również przekazać szablonowi częściowemu mapy nazwę akcji, która wyświetli informacje o zdarzeniu. To sporo zmian, które jednak sprawią, że szablon `index.html.erb` zostanie znacznie uproszczony.

Tak naprawdę zostanie nam tylko tyle:

To oznacza, że będziemy generować większą mapę wypełniającą całą stronę.

```
<h1>Listing incidents</h1>
<%= render (:partial=>'map', :locals=>{:data=>"/incidents.xml",
  >:full_page=>true, :show_action=>'show'}) %>
<br />
<%= link_to 'New incident', new_incident_path %>
```



Zmodyfikujmy akcję „show”, tak by generowała zawartość okna informacyjnego.

Kiedy szablon częściowy mapy wywołujemy w taki sposób, **zmienia on swoje zachowanie**. Wcześniej gdy zdarzenie zostało kliknięte, szablon częściowy wykonywał fragment domyślnego kodu w języku JavaScript, który w wyskakującym oknie informacyjnym wyświetlał tytuł oraz opis zdarzenia. Wprowadzenie tej zmiany oznacza, że po kliknięciu zdarzenia na mapie szablon częściowy wywołuje akcję show i wyświetla jej odpowiedź w oknie.

**A przynajmniej tak będzie w przyszłości,
po zmodyfikowaniu akcji show w taki sposób,
by generowała ona poprawne dane wyjściowe.**

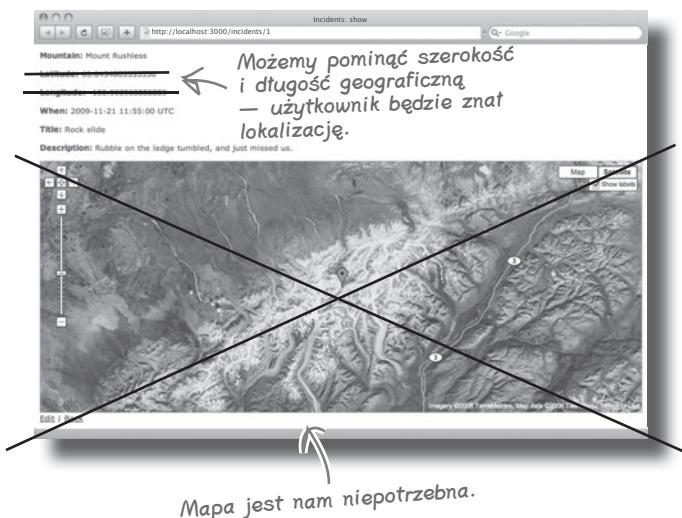
Pokaż mi więcej

Co będzie musiała wygenerować akcja show?

Mamy już akcję show, która generuje stronę internetową zawierającą szczegóły zdarzenia wraz z mapą pokazującą jego lokalizację.

To jednak o wiele więcej, niż potrzebujemy teraz. Akcja show musi wygenerować jedynie szczegóły zdarzenia w formie tekstuowej, a ponieważ informacje te będą pokazywane obok punktu na mapie, wyświetlanie szerokości oraz długości geograficznej nie będzie potrzebne.

Pojawia się też kolejna różnica: potrzebny nam jest jedynie **fragment strony**. Nie chcemy standardowego kodu HTML, który będzie utworzony przez szablon strony. Oznacza to, że nasza akcja będzie musiała być generowana z szablonu częściowego. Nazwiemy go `_show.html.erb`.



Zaostrz ołówek



Uzupełnij wiersz metody show kontrolera, tak by wywoływała ona szablon częściowy `_show.html.erb`:

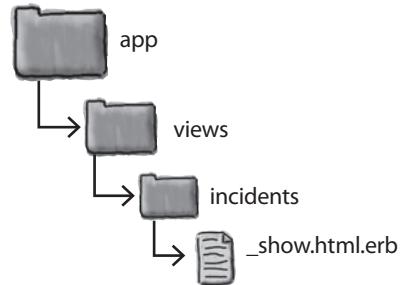
```
def show
  @incident = Incident.find(params[:id])
  respond_to do |format|
    format.html {
      .....
    }
    format.xml {
      render :text=>@incident.to_xml(
        :only=>[:latitude,:longitude,:title,:description],
        :root=>"data")
    }
  end
end
```





Magnesiki z kodem

Uzupełnij kod nowego szablonu częściowego _show.html.erb.
Pamiętaj – nie będziesz musiał wyświetlać wszystkich informacji.



```

<p>
  <b> _____ </b>
  <%=h _____ %>
</p>

<p>
  <b> _____ </b>
  <%=h _____ %>
</p>

<p>
  <b> _____ </b>
  <%=h _____ %>
</p>

<p>
  <b> _____ </b>
  <%=h _____ %>
</p>
  
```

incident.longitude

Mountain:

incident.title

Longitude:

incident.description

When:

incident.mountain

Latitude:

incident.latitude

incident.when

Description:

Title:

Zaostrz ołówek



Rozwiążanie

Uzupełnij wiersz metody show kontrolera, tak by wywoływała ona szablon częściowy `_show.html.erb`:

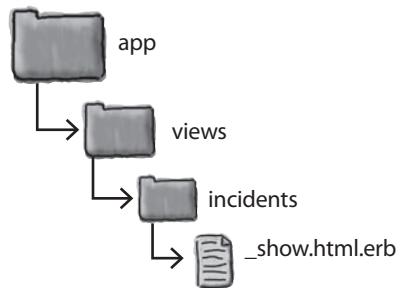
```
def show
  @incident = Incident.find(params[:id])
  respond_to do |format|
    format.html {
      render :partial=>'show', :locals=>{:incident=>@incident}
    }
    format.xml {
      render :text=>@incident.to_xml(
        :only=>[:latitude,:longitude,:title,:description],
        :root=>"data")
    }
  end
end
```





Magnesiki z kodem: Rozwiążanie

Uzupełnij kod nowego szablonu częściowego _show.html.erb.
Pamiętaj – nie będziesz musiał wyświetlać wszystkich informacji.



```

<p>
  <b> Mountain:</b>
  <%=h incident.mountain %>
</p>

<p>
  <b> When:</b>
  <%=h incident.when %>
</p>

<p>
  <b> Title:</b>
  <%=h incident.title %>
</p>

<p>
  <b> Description:</b>
  <%=h incident.description %>
</p>
  
```

incident.longitude

Latitude:

Longitude:

incident.latitude



Jazda próbna

Gdy teraz przejdziemy do strony głównej, lista zdarzeń zniknęła, a mapa jest o wiele większa. Najważniejsze jednak dzieje się, kiedy użytkownik kliknie jedno ze zdarzeń na mapie:

The screenshot shows a web browser window with the title "Incidents: index". The address bar displays "http://localhost:3000/incidents/". A callout box is overlaid on the map, containing the following information:

Mountain: Mount Lotopaxo
When: 2009-06-07 12:06:00 UTC
Title: Ascent
Description: Living only on dried chicken pieces, we completed our 4 day...

The map itself shows the world with various countries labeled in multiple languages. A white callout box is positioned over a location in South America, specifically around Colombia and Venezuela. The map interface includes standard zoom controls (plus/minus) and orientation indicators (compass rose). The bottom of the map features a footer with "Powered by Google" and copyright information: "Imagery ©2008 TerraMetrics, NASA. Map data ©2008 Eurodata Technologies - Terms of Use".

Kiedy mapa wykrywa kliknięcie myszą na zdarzeniu, generuje żądanie do akcji show oparte na Ajaksie; akcja ta generuje szczegóły zdarzenia w formacie HTML. Mapa otrzymuje kod HTML i wykorzystuje go do zastąpienia zawartości okna informacyjnego zdarzenia. Wszystko to zostaje następnie wyświetlone użytkownikowi.

Nowa funkcjonalność mapy jest pełnym sukcesem!

Nowa funkcjonalność mapy jest witana entuzjastycznie przez wspinaczy. Nie muszą oni już przewijać długiej listy zdarzeń, by dostać się do mapy. Teraz mogą otrzymać wszystkie niezbędne informacje bezpośrednio z mapy. Jest tylko jeden problem...

Wspinacze chcą zgłaszać nowe zdarzenia za pomocą mapy.

Mogą oczywiście wykorzystać urządzenia GPS do odnalezienia szerokości oraz długości geograficznej, a później wpisać te dane, jednak strona byłaby dla nich o wiele łatwiejsza w użyciu, gdyby mogli po prostu odnaleźć odpowiedni punkt na mapie i wypełnić wszystkie informacje bezpośrednio tam.

W ten sposób wprowadzanie danych stałoby się dla nich o wiele szybsze.

Jak wprowadzanie danych za pomocą mapy pasuje do tego, co aktualnie robimy?

Musimy utworzyć żądania wykorzystujące Ajaksa

Kiedy ktoś chce utworzyć nowy raport dotyczący zdarzenia, obecnie musi wykonać następujące kroki:

1

Kliknięcie odnośnika **New Incident** na stronie głównej.

Nie da się wprowadzić danych bezpośrednio na stronie głównej, konieczne jest kliknięcie odnośnika kierującego do strony *new*.



2

Ręczne wprowadzenie szerokości oraz długości geograficznej na stronie nowego zdarzenia.

Na stronie tej nie ma mapy, zatem konieczne jest ręczne wpisanie szerokości oraz długości geograficznej i zapisanie rekordu.

Incidents: new

New incident

Mountain: Mountain

Latitude:

Longitude:

When: 2008-11-17 00:00

Title:

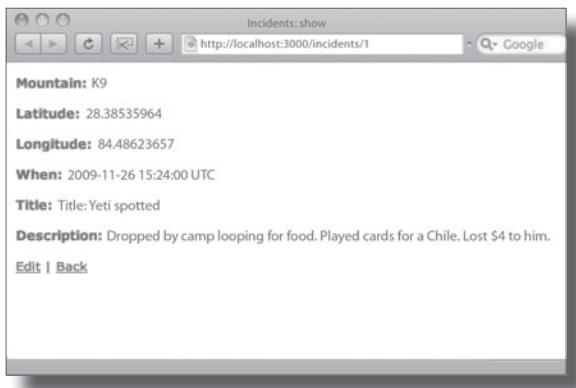
Description:

Create Back

3

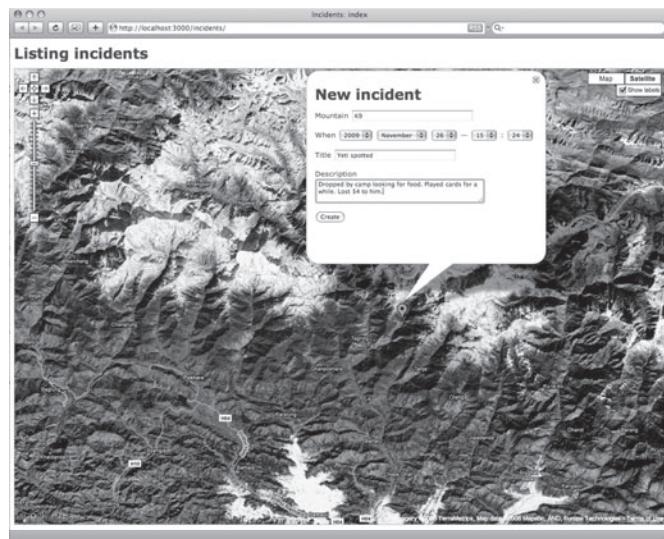
Utworzzone zdarzenie zostaje wyświetlone.

Po kliknięciu przycisku zapisującego zdarzenie zostajemy przekierowani do strony pokazującej właśnie utworzone zdarzenie. Jeśli chcemy utworzyć kolejne zdarzenie lub powrócić do głównej mapy, musimy kilka razy nacisnąć przycisk *Wstecz* przeglądarki, by powrócić do strony głównej — skąd możemy rozpoczęć wszystko od nowa.



Co powinno się zmienić?

Zamiast wykonywać wszystkie te kroki, użytkownicy woleliby, by interfejs był znacznie uproszczony. Chcą po prostu kliknąć mapę i uzupełnić szczegóły zdarzenia za pomocą formularza z wyskakującego okna.



By wprowadzić te zmiany, musimy wygenerować formularz tworzenia nowego zdarzenia za pomocą Ajaksa. Mapa musi także wywoływać ten formularz za każdym razem, gdy ktoś kliknie myszą w nowym miejscu. Ale jak możemy to zrobić?

Jedno polecenie render może mieć kilka akcji

Szablon częściowy mapy pozwala nam wybrać akcję new

Dotychczas zajmowaliśmy się pokazywaniem szczegółów zdarzenia na mapie. Jak jednak możemy tworzyć nowe zdarzenia?

Szablon częściowy `_map.html.erb` pozwala nam wybrać akcję obsługującą nowe zdarzenia w ten sam sposób co akcję pokazującą szczegóły zdarzenia. Oznacza to, że możemy dodać akcję new do pliku `index.html.erb` w następujący sposób:

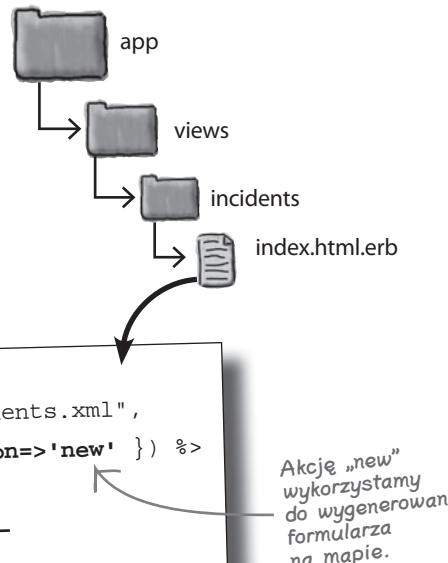
```
<h1>Listing incidents</h1>
<%= render (:partial=>'map', :locals=>{ :data=>"/incidents.xml",
  :full_page=>true, :show_action=>'show', :new_action=>'new' }) %>
<br />
<%= link_to 'New incident', new_incident_path %>
```

Nie będziemy potrzebowali już na stronie głównej odnośnika tworzącego nowe zdarzenia, dlatego możemy wykreślić te wiersze.

Kiedy ktoś kliknie jakieś miejsce na mapie, szablon częściowy `_map.html.erb` utworzy nowy znaczek i wyświetli wyskakujące okno informacyjne zawierające informacje zwracane przez akcję new.

Mamy już akcję new zdefiniowaną dla tej aplikacji, jednak generuje ona pełną stronę internetową. Zamiast wyświetlać pełną stronę internetową, chcemy, by akcja new tworzyła fragment strony, który zostanie wyświetlony wewnątrz okna wyskakującego. Musimy się również upewnić, że kiedy użytkownik przesyła formularz new, pozostaje on na mapie.

Utworzmy zatem szablon częściowy `_new.html.erb` generujący formularz oparty na Ajaksie.



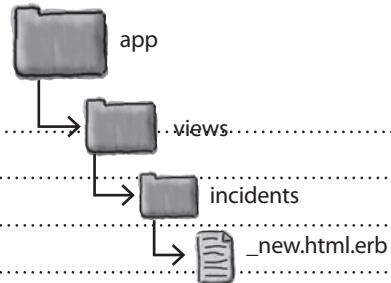
Akcję „new” wykorzystamy do wygenerowania formularza na mapie.



Ćwiczenie

Na potrzeby formularza musisz utworzyć szablon częściowy `_new.html.erb`. Uzupełnij kod metody formularza. Pamiętaj — użytkownicy nie będą musieli wprowadzać danych dla szerokości i długości geograficznej, ale formularz będzie musiał te dane zapisać.

```
<h1>New incident</h1>
<% remote_form_for(incident) do |f| %>
```



```

<p>
  <%= f.submit "Create" %>
</p>
<% end %>
```

Kiedy mapa wywołuje akcję `new`, przesyła lokalizację nowego zdarzenia jako parametr żądania. Uzupełnij kod metody `new` kontrolera `incidents_controller.rb`, tak by poprawnie wywoływała ona szablon częściowy:

```
format.html {
  @incident.latitude=params[:latitude]
  @incident.longitude=params[:longitude]
}
}
```



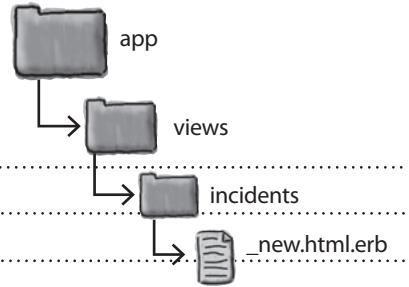
Dostosowanie nowego kodu wywołującego



Ćwiczenie
Rozwiązanie

Na potrzeby formularza musisz utworzyć szablon częściowy `_new.html.erb`. Uzupełnij kod metody formularza. Pamiętaj — użytkownicy nie będą mogli wprowadzać danych dla szerokości i długości geograficznej, ale formularz będzie musiał te dane zapisać.

```
<h1>New incident</h1>
<% remote_form_for(incident) do |f| %>
<p>
  <%= f.label :mountain %> <%= f.text_field :mountain %>
</p>
<%= f.hidden_field :latitude %> Szerokość i długość
<%= f.hidden_field :longitude %> geograficzną nadal musimy
                                    zapisać w formularzu, jednak
                                    tylko jako pola ukryte.
<p>
  <%= f.label :when %> <%= f.datetime_select :when %>
</p>
<p>
  <%= f.label :title %> <%= f.text_field :title %>
</p>
<p>
  <%= f.label :description %><br/>
  <%= f.text_area :description, :rows=>3 %>
</p>
<p>
  <%= f.submit "Create" %>
</p>
<% end %>
```



Twój kod może wyglądać nieco inaczej.

Kiedy mapa wywołuje akcję `new`, przesyła lokalizację nowego zdarzenia jako parametr żądania. Uzupełnij kod metody `new` kontrolera `incidents_controller.rb`, tak by poprawnie wywoływała ona szablon częściowy:

```
format.html {
  @incident.latitude=params[:latitude]
  @incident.longitude=params[:longitude]
  render :partial=>'new', :locals=>{:incident=>@incident}
}
```





Jazda próbna

Gdy teraz użytkownik udaje się na stronę główną i kliką nowy punkt na mapie, może dodać szczegóły zdarzenia za pomocą wyskakującego formularza. Po kliknięciu przycisku *Create* formularz pozostaje na ekranie, a nowy rekord pojawia się w bazie danych.

id	mountain	latitude	longitude	when	title	description
1	Mount Rushless	63.04348055...	-150.993963...	2009-11-21 11:...	Rock slide	Rubble on the ...
2	Mount Rushless	63.07805277...	-150.977869...	2009-11-21 17:...	Hidden crev...	Ice layer cove...
3	Mount Lotopaxo	-0.683975	-78.4365055...	2009-06-07 12:...	Ascent	Living only on...
4	High Kanuklima	11.123925	72.72135833...	2009-05-12 18:...	Altitude si...	Overcome by th...
5	K9	28.38535964...	84.48623657...	2009-11-26 15:...	Yeti spotted	Dropped by cam...

Wszystko jest zatem w porządku, tak?



Wspinacze są zakłopotani.

Choć formularz działa dobrze, a zdarzenia zapisywane są do bazy danych, tak naprawdę *wygląda*, jakby nic się nie działa. Kiedy użytkownik kliką przycisk *Create*, nie otrzymuje żadnych informacji zwrotnych potwierdzających fakt zapisania rekordu. Oznacza to, że użytkownicy raz za razem naciskają przycisk *Create*, a w bazie danych pojawia się coraz więcej zduplikowanych rekordów.

Coś trzeba z tym zrobić. Dawno temu, gdy mieliśmy do dyspozycji tylko rusztowanie, kiedy użytkownik zgłosił zdarzenie za pomocą strony *new*, przeglądarka przenosiła go natychmiast do strony *show*. Było to potwierdzenie faktu zapisania danych w bazie.

The screenshot shows two browser windows side-by-side. The left window is titled "Incidents: new" and contains a form for creating a new incident. It has fields for "Mountain" (with a placeholder "Mountain"), "Latitude", "Longitude", "When" (set to 2009-11-21 11:55:00 UTC), "Title", and "Description" (a large text area). A "Create" button is at the bottom. The right window is titled "Incidents: show" and displays the details of a specific incident: Mountain: Mount Rushless, Latitude: 63.04348055555556, Longitude: -150.9939638888889, When: 2009-11-21 11:55:00 UTC, Title: Rock slide, and Description: Rubble on the ledge tumbled, and just missed us. Below the description are links for "Edit" and "Back".

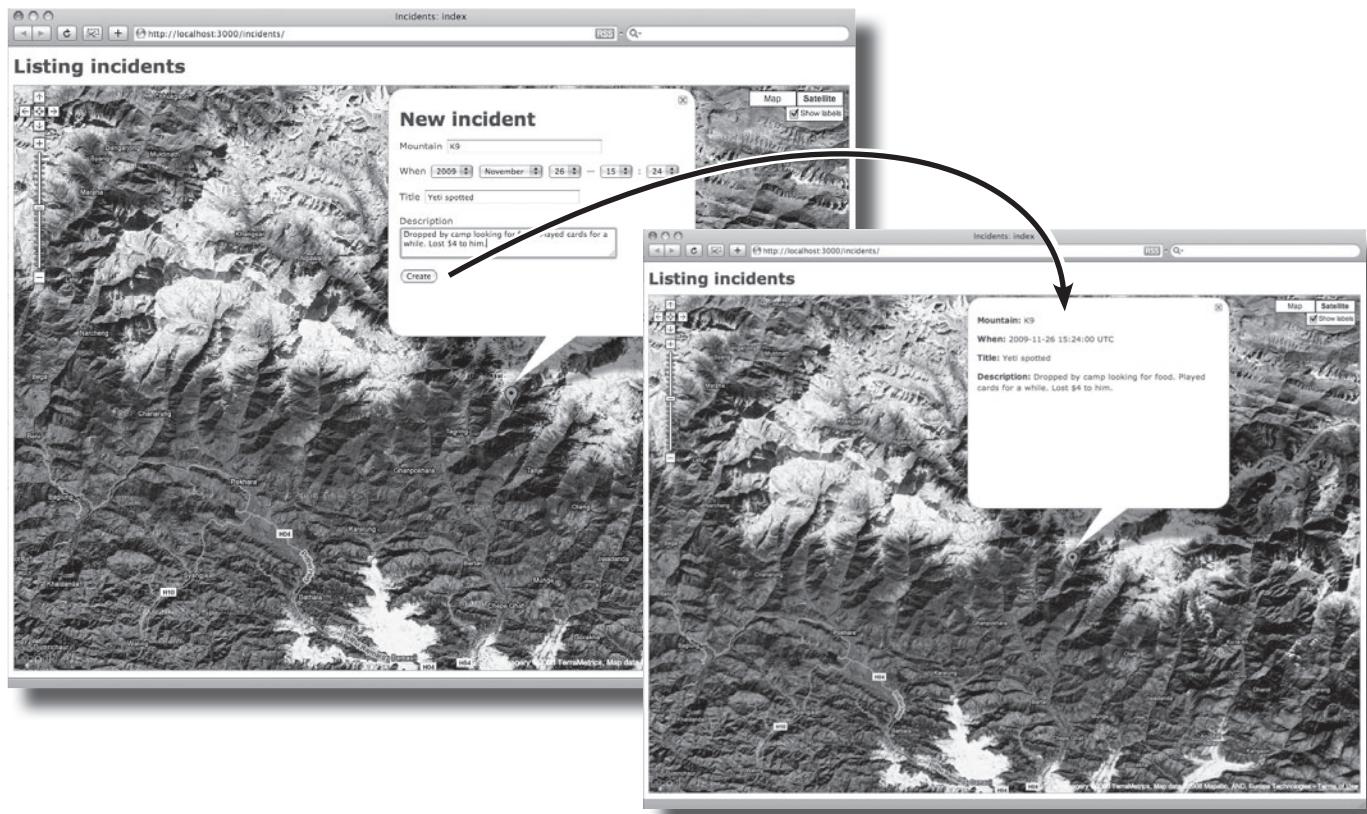
System wyświetla stronę pokazującą zdarzenie („show”) w celu potwierdzenia zapisania rekordu.

Oryginalny formularz nowego zdarzenia („new”).

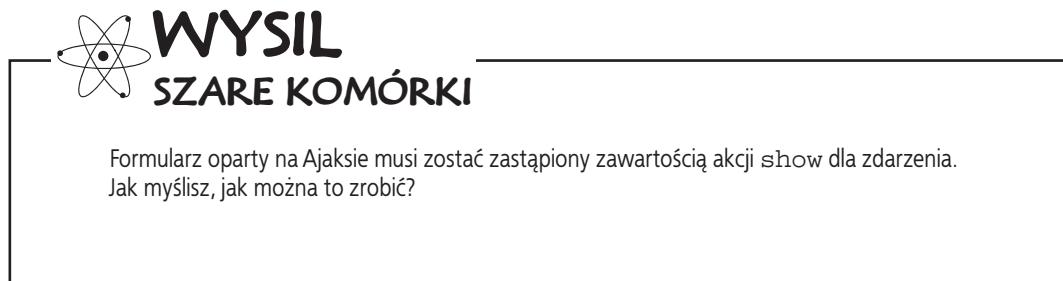
Czy moglibyśmy uzyskać coś takiego w aplikacji opartej na Ajaksie?

Jak możemy udowodnić, że zdarzenie zostało zapisane?

System musiałby tak naprawdę wyświetlać utworzony rekord za pomocą akcji *show* w wyskakującym oknie. Kiedy zatem ktoś wprowadzi szczegóły zdarzenia, wyskakujące okno powinno się zmienić, pokazując zdarzenie w wersji *tylko do odczytu*.



W ten sposób okno wyskakujące będzie działało podobnie jak przeglądarka, przekierowując użytkownika do nowych informacji. Tyle tylko, że nasz kod nie może przekierować przeglądarki do nowych informacji. Musimy utrzymać użytkowników na tej samej stronie... pokazując im tylko nowe informacje.



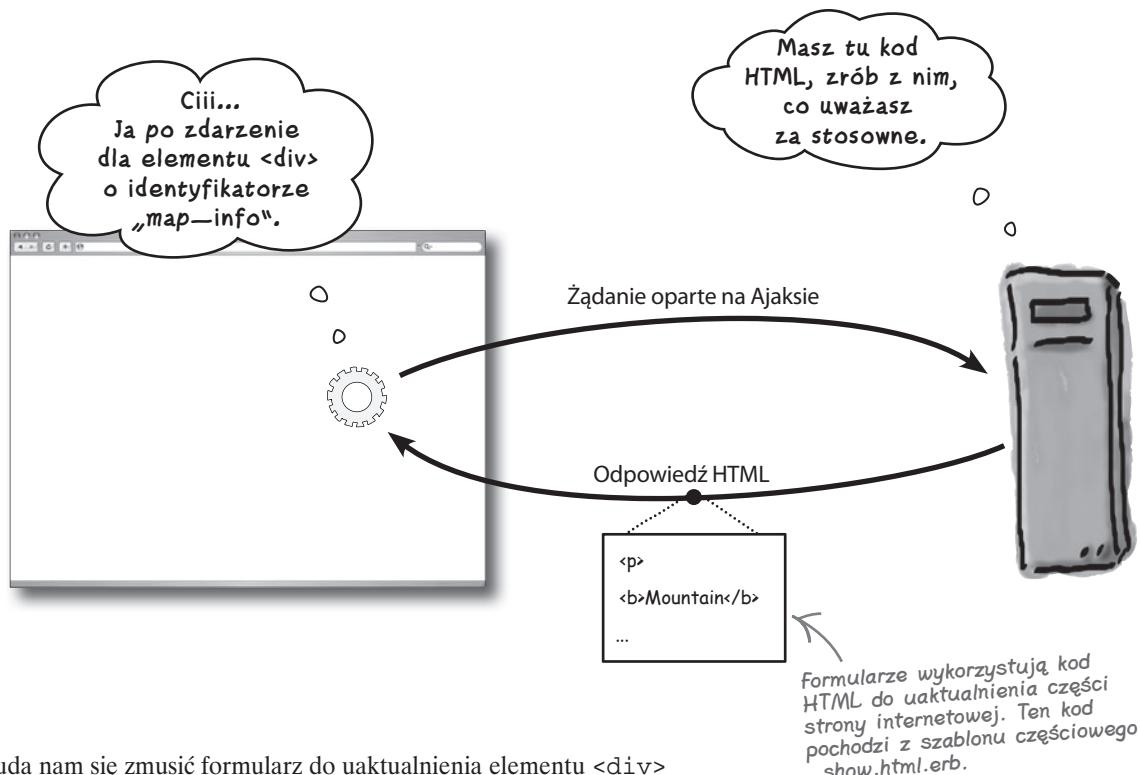
Formularz musi uaktualnić zawartość elementu <div> wyskakującego okna

Choć aplikacja Google Maps wygląda prawie jak aplikacja desktopowa, tak naprawdę składa się ona z kodu HTML oraz JavaScriptu. To po prostu strona internetowa. Oznacza to, że wyskakujące okno informacyjne — tak jak wszystko inne — to po prostu fragment kodu HTML.

Zawartość tego okna definiowana jest za pomocą elementu `<div>` o identyfikatorze:

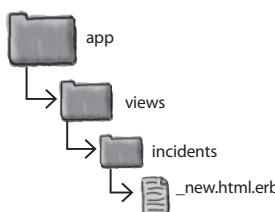
```
id='map_info'
```

Jest to istotne, ponieważ do utworzenia raportu nowego zdarzenia wykorzystujemy formularz oparty na Ajaksie, a formularze tego typu mogą być wykorzystywane do dynamicznego uaktualnienia części strony internetowej **za pomocą ich identyfikatorów**.



Jeśli uda nam się zmusić formularz do uaktualnienia elementu `<div>` o identyfikatorze `map_info` za pomocą zawartości akcji `show` zdarzenia, powinno to dać użytkownikom oczekiwane przez nich informacje zwrotne.

Zaostrz ołówek

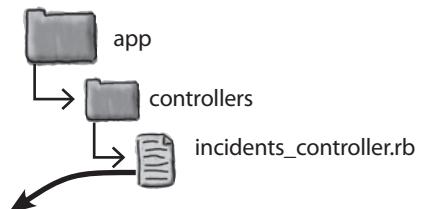


To część kodu formularza new. Pamiętając, że formularz będzie musiał uaktualnić stronę za pomocą odpowiedzi, uzupełnij kod:

```

<% remote_form_for @incident, _____ ) do |f| %>
<p>
  <%= f.label :mountain %> <%= f.text_field :mountain %>
</p>
  
```

Formularz new wysyła się do akcji create. To jest metoda create kontrolera. Wprowadź do niej wszelkie niezbędne zmiany.



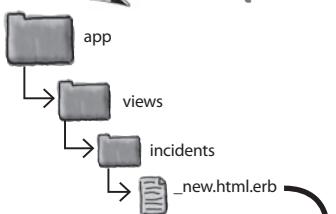
```

def create
  @incident = Incident.new(params[:incident])

  respond_to do |format|
    if @incident.save
      flash[:notice] = 'Incident was successfully created.'
      format.html { redirect_to(@incident) }
      format.xml { render :xml => @incident, :status => :created,
                   :location => @incident }
    else
      format.html { render :action => "new" }
      format.xml { render :xml => @incident.errors,
                   :status => :unprocessable_entity }
    end
  end
end
  
```

Zaostrz ołówek

Rozwiążanie



To część kodu formularza new. Pamiętając, że formularz będzie musiał uaktualnić stronę za pomocą odpowiedzi, uzupełnij kod:

```
<% remote_form_for(incident, _____ :update=>'map_info') do |f| %>
<p>
  <%= f.label :mountain %> <%= f.text_field :mountain %>
</p>
```

Formularz new wysyła się do akcji create. To jest metoda create kontrolera. Wprowadź do niej wszelkie niezbędne zmiany.

```
def create
@incident = Incident.new(params[:incident])

respond_to do |format|
  if @incident.save
    flash[:notice] = 'Incident was successfully created.'
    format.html { redirect_to(@incident) }
    format.xml { render :xml => @incident, :status => :created,
      :location => @incident }
  else
    format.html { render :action => "new" }
    format.xml { render :xml => @incident.errors,
      :status => :unprocessable_entity }
  end
end
end
```



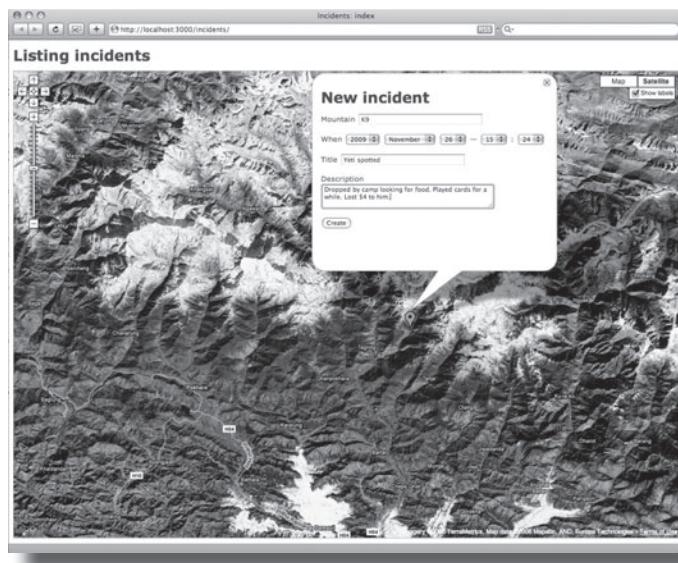
NIC nie musi się zmienić w akcji „create”. Po przestaniu formularza akcja „create” wstawia rekord do bazy danych, a następnie przekierowuje żądanie do akcji „show”. Akcja „show” generuje teraz fragment kodu HTML wyświetlający szczegóły nowego zdarzenia. A właśnie o to nam chodzi.



Jazda próbna

Co się teraz dzieje, kiedy użytkownik tworzy zdarzenie?

Kliknięcie nowego punktu na mapie wyświetla formularz oparty na Ajaksie — tak jak wcześniej:



Po kliknięciu przycisku *Create* system nie tylko zapisuje rekord w bazie danych, ale i zwraca fragment strony zawierający szczegóły zdarzenia, które formularz oparty na Ajaksie wykorzystuje teraz do uaktualnienia elementu `<div>` o identyfikatorze `map_info` wewnętrz w wyskakującego okna.



Element `<div>` o identyfikatorze „`map_info`” wewnątrz wyskakującego okna zostaje uaktualniony.

Nie istnieją głupie pytania

P: Co się dzieje, jeśli przeglądarka użytkownika ma wyłączoną obsługę JavaScriptu?

O: Mapa aplikacji nie zostanie uruchomiona. Aplikacje oparte na Ajaksie, takie jak Google Maps, wymagają obsługi JavaScriptu.

P: Skoro Google Maps jest aplikacją opartą na Ajaksie, dlaczego nie musielibyśmy w poprzednim rozdziale dodawać biblioteki Prototype?

O: Google Maps wywołuje własne biblioteki Ajaksa na serwerach firmy Google, dlatego nie musi korzystać z biblioteki Prototype.

P: Dlaczego zatem tym razem musimy odwołać się do Prototype?

O: Jeśli przekażemy nazwy akcji do szablonu częściowego mapy, szablon ten musi wykonać oparte na Ajaksie żądanie do serwera. Z tego powodu potrzebna jest biblioteka Prototype... niezależnie od tego, co robi Google Maps.

P: Czy istnieje jakiś sposób, by aplikacja działała bez mapy, jeśli ktoś wyłączył obsługę JavaScriptu?

O: Mogliby tak być, gdybyś zmodyfikował kontroler. Kontroler decyduje o tym, które widoki należy wyświetlić, dlatego mógłby wykonać inne szablony stron i szablony częściowe, gdyby obsługa JavaScriptu nie była możliwa.

P: Nadal nie rozumiem, jak działa respond_to. Jeśli format.html jest jakimś rodzajem wywołania metody, dlaczego pomiędzy następującymi po nim znakami {...} znajduje się jakiś kod?

O: W języku Ruby metody przyjmują jako parametry fragmenty kodu znajdujące się pomiędzy znakami { . . . } (lub do . . . end). Kod pomiędzy znakami { oraz } zostaje przekazany dla format .html i tu decyduje się, czy zostanie on wykonany.

P: Jak działa szablon częściowy mapy?

O: Nie jest to zbyt skomplikowane — to przede wszystkim kod w języku JavaScript, dlatego nie będziemy się w niego szczególnie zagłębiać. Więcej tego rodzaju szczegółów znajdziesz jednak w książce *Head First JavaScript. Edycja polska*.

P: Ale ja naprawdę chcę wiedzieć, jak to działa!

O: Warto przyjrzeć się kodowi pliku _map.html.erb. Jeśli chcesz dowiedzieć się czegoś więcej o JavaScriptie, czy wspominaliśmy już, że *Head First JavaScript. Edycja polska*. to świetna książka? :-)

Lawina!

Dotychczas pozwalaliśmy użytkownikom oglądać szczegółowe informacje na mapie, a także tworzyć bezpośrednio na niej nowe zdarzenia.
Ale co z edytowaniem?



Ale co z edytowaniem?

Jak działa to teraz...

Rusztowanie udostępnia nam opcję edycji w dwóch miejscach.

W oryginalnej wersji strony *index* pochodzącej z rusztowania można było kliknąć odnośnik *Edit* znajdujący się obok każdego z rekordów, by przejść do formularza edycji. Teraz jednak nie możemy zrobić niczego takiego, ponieważ lista zdarzeń na stronie głównej została zastąpiona mapą. A wiemy, że szablon częściowy mapy nie ma wbudowanych żadnych funkcji edycji.

The screenshot shows a web browser window titled "Incidents: index". The URL in the address bar is "http://localhost:3000/incidents/". The page displays a table of incidents with columns: Mountain, Latitude, Longitude, When, Title, and Description. Each row has three links at the end: "Show", "Edit", and "Destroy". A large callout bubble points to the "Edit" link in the first row, containing the text "Tutaj znajdują się odnośniki do edycji".

Mountain	Latitude	Longitude	When	Title	Description
Mount Rushless	63.0434805555556	-150.993963888889	2009-11-21 11:55:00 UTC	Rock slide	Rubble on the ledge tumbled, and just missed us.
Mount Rushless	63.0780527777778	-150.977869444444	2009-11-21 17:59:00 UTC	Hidden crevasse	Ice layer covering crevasse. Lost Binky to the elements.
Mount Lotopaxo	-0.683975	-78.4365055555556	2009-06-07 12:06:00 UTC	Ascent	Living only on dried chicken pieces, we completed our 4 day...
High Kanuklima	11.123925	72.7213583333333	2009-05-12 18:11:00 UTC	Altitude sickness	Overcome by the lack of oxygen, we abandoned the ascent.

[New incident](#)

Gdzie jeszcze w oryginalnym rusztowaniu mogliśmy coś edytować?
Kolejnym miejscem była strona *show* zdarzenia. W wersji aplikacji
opartej na rusztowaniu na stronie *show* znajdował się odnośnik *Edit*.

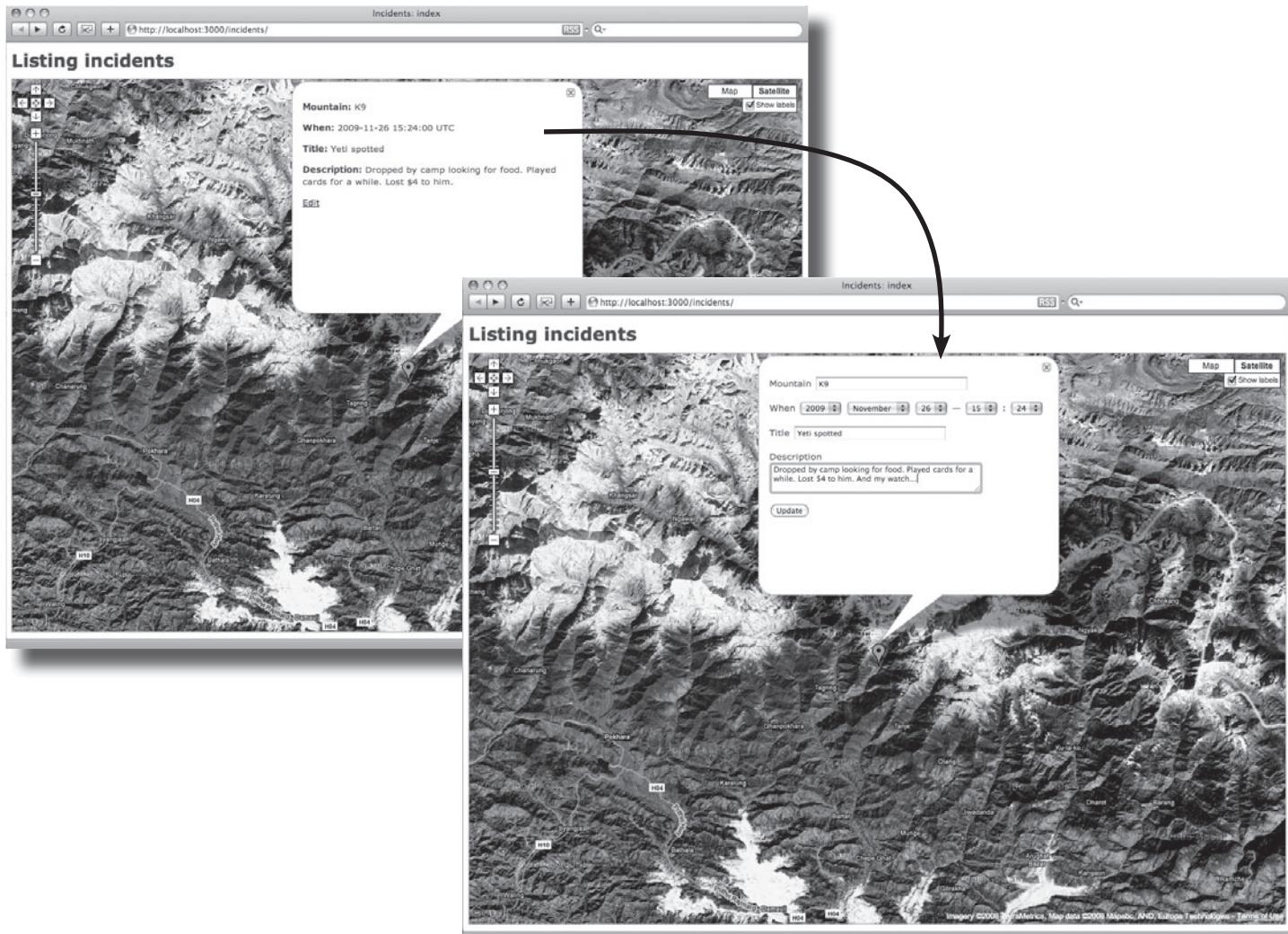
The screenshot shows a web browser window titled "Incidents: show". The URL in the address bar is "http://localhost:3000/incidents/1". The page displays the details of an incident for "Mount Rushless":
- Mountain: Mount Rushless
- Latitude: 63.0434805555556
- Longitude: -150.993963888889
- When: 2009-11-21 11:55:00 UTC
- Title: Rock slide
- Description: Rubble on the ledge tumbled, and just missed us.
At the bottom, there are two buttons: "Edit" and "Back". A callout bubble points to the "Edit" button with the text "Tutaj także znajduje się odnośnik do edycji".

Czy moglibyśmy zrobić coś podobnego? Może dodalibyśmy odnośnik *Edit* do zbioru szczegółów wyświetlanych w wyskakującym oknie informacyjnym?

Czy takie rozwiązanie by zadziałało?

Możemy umieścić odnośnik „Edit” w oknie wyskakującym

Musimy dodać odnośnik *Edit* do szczegółowych informacji, które pokazują się, kiedy użytkownik wybierze zdarzenie z mapy. Gdy ktoś kliknie odnośnik *Edit*, podmienimy zawartość elementu `<div>` o identyfikatorze `map_info` w celu wyświetlenia formularza edycji, a następnie wykorzystamy formularz do zmodyfikowania rekordu.



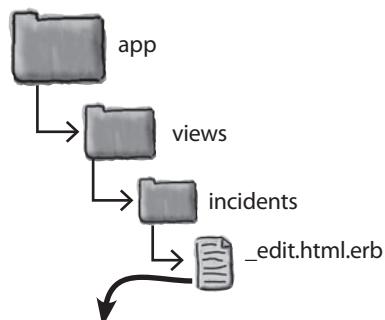
Mamy już wbudowaną funkcję `show`. Formularz `edit` powinien być podobny do formularza `new`, a w kontrolerze mamy już kod modyfikujący rekord.

To zadanie nie może chyba być zbyt trudne?

Zaczniemy od zmodyfikowania akcji edit

Potrzebny nam jakiś sposób generowania formularza edycji, który pojawi się w wyskakującym oknie informacyjnym.

Utworzymy szablon częściowy o nazwie `_edit.html.erb`. Szablon ten wygląda dość podobnie do `_new.html.erb`:



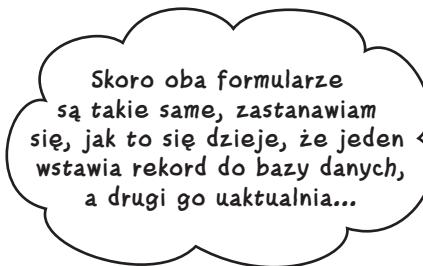
```
<% remote_form_for(incident, :update=>'map_info') do |f| %>
<p><%= f.label :mountain %> <%= f.text_field :mountain %></p>
<%= f.hidden_field :latitude %>
<%= f.hidden_field :longitude %>
<p><%= f.label :when %> <%= f.datetime_select :when %></p>
<p><%= f.label :title %> <%= f.text_field :title %></p>
<p><%= f.label :description %><br/><%= f.text_area :description, :rows=>3 %></p>
<p>
<%= f.submit "Update" %>
</p>
<% end %>
```

Skąd dwa szablony częściowe?

Dwa szablony częściowe zawierają właściwie ten sam kod, jednak rozdzielenie ich jest dobrym pomysłem. W tej chwili oba wyglądają tak samo, jednak nie zawsze musi tak być.

Przykładowo możemy kiedyś chcieć zmienić funkcjonalność dostępną za pomocą stron `new` oraz `edit`. Możemy zadecydować, że użytkownicy mogą wstawić datę zdarzenia, ale nie pozwolić im na późniejsze zmodyfikowanie tej informacji. Możliwe też, że będziemy chcieli, by obie strony różniły się od siebie wyglądem.

Kod w plikach `_edit.html.erb` oraz `_new.html.erb` jest ten sam, jednak w tym przypadku zachowamy go w osobnych plikach.



Rails wie, czy obiekt modelu formularza był wcześniej zapisany w bazie danych.

Kod generowany przez metodę pomocniczą `form_for` zmienia się w zależności od tego, czy ma do czynienia z niezapisanym obiektem (w tym przypadku formularz wywołuje akcję `create`), czy też z obiektem, który kiedyś był już zapisany (wtedy formularz wywołuje akcję `update`).

A skoro mówimy już o akcjach i tworzeniu szablonu częściowego, musimy zmodyfikować metodę akcji `edit` w kontrolerze, by po wywołaniu zwracała ona szablon częściowy `_edit.html.erb`:

```
def edit
  @incident = Incident.find(params[:id])
  render :partial=>'edit', :locals=>{:incident=>@incident}
end
```

Nie istniejąca grupa pytań

P: Skąd Rails wie, czy obiekt został już zapisany?

O: Wyołuje metodę o nazwie `new_record?`. Zwraca ona `true`, jeśli obiekt nigdy nie został zapisany.

P: Dlaczego metoda `new_record?` ma na końcu znak zapytania?

O: To konwencja języka Ruby. Większość metod zwracających wartości `true` lub `false` ma w nazwie znak zapytania.

P: Dlaczego pola dla szerokości i długości geograficznej są ukryte?

O: Bo nie chcemy, by użytkownicy je edytowali.

P: No tak, to wiem. Ale po co o nich w ogóle wspominać?

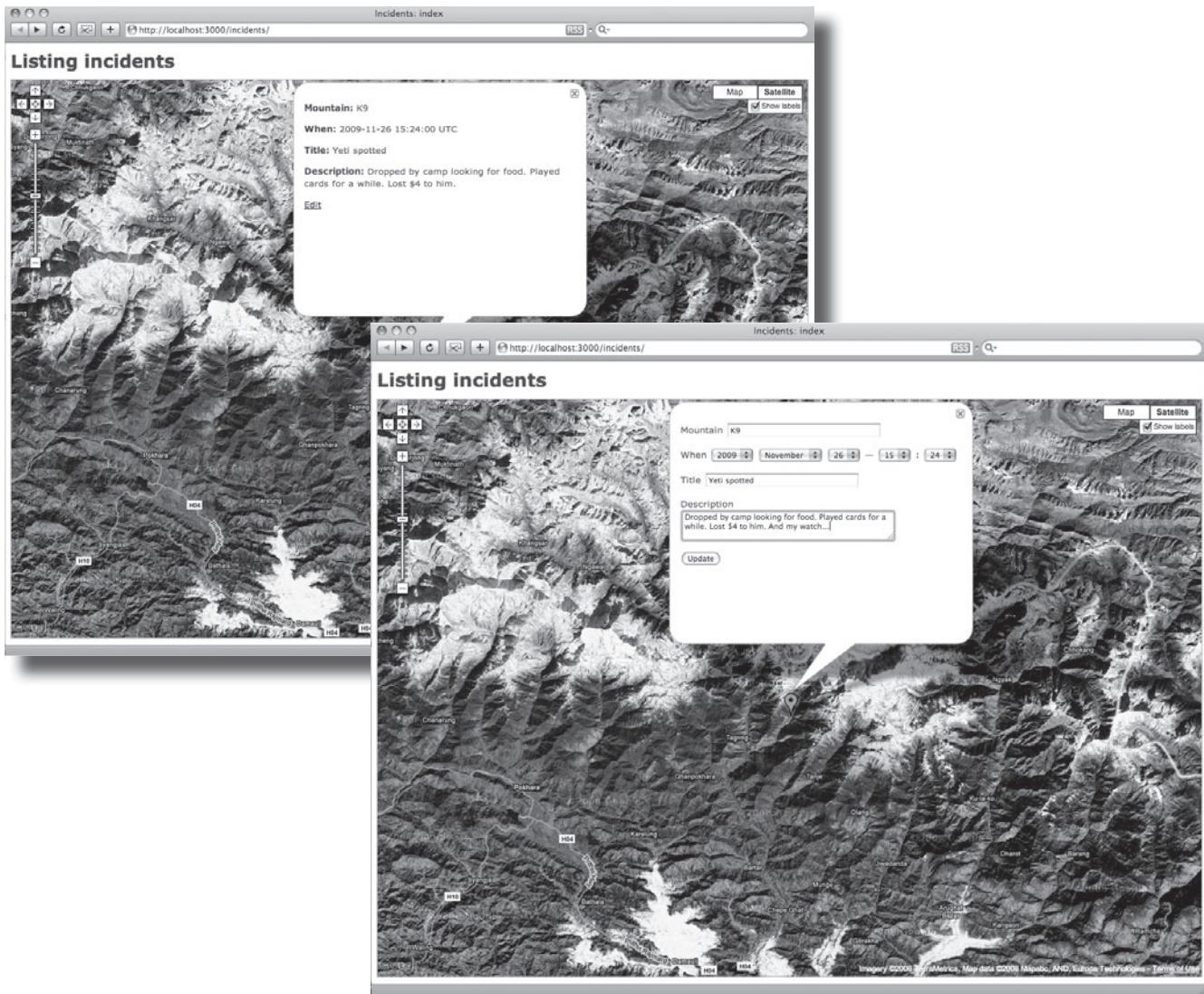
O: Obiekt formularza zostaje przekształcony w pola formularza. Gdybyśmy nie wspomnieli o nich w polach formularza, zniknęłyby.

P: Nie mamy jednak pól dla `id`, `created_at` oraz `updated_at`?

O: Nie, ale Rails wie, że są one niezbędne, dlatego metoda pomocnicza `form_for` utworzy je za nas.

Na stronie show potrzebny nam jest także nowy odnośnik

Powinniśmy teraz być w stanie wygenerować formularz edycji, jak jednak ma się do niego dostać użytkownik? Musimy dodać odnośnik *Edit*, który pojawi się, kiedy użytkownik będzie oglądał szczegóły zdarzenia. Odnośnik ten trzeba dodać do szablonu częściowego *_show.html.erb*.

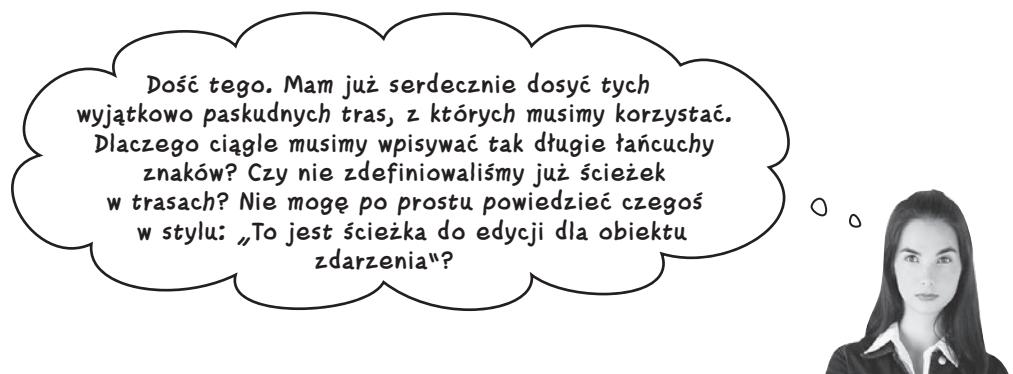


By wygenerować odnośnik, użyjemy metody pomocniczej *link_to*.

Jak stosuje się metodę pomocniczą link_to?

Metoda pomocnicza link_to ma dwa parametry — tekst odnośnika oraz miejsce, do którego on prowadzi.

```
<p><%= link_to "Edit", "/incidents/#{incident.id}/edit" %></p>
```



To bardzo cenna uwaga.

Jak na razie utworzyliśmy mnóstwo ścieżek oraz adresów URL, korzystając z łańcuchów znaków. Co jednak, jeśli w przyszłości zmienimy format odnośników? Trasy uda nam się szybko poprawić, jednak zostaniemy z całym mnóstwem zbędnych łańcuchów znaków w kodzie zawierającym ścieżki.

Posiadanie jakiegoś typu informacji w dwóch miejscach jest niewłaściwe, ponieważ łamie ważną regułę Rails:

Nie powtarzaj się

Jeśli jednak trasy zapisują już strukturę ścieżek oraz adresów URL, może warto przyjrzeć się im bardziej szczegółowo?

Czy już o tym
nie mówiliśmy?



Trasy oparte na architekturze REST z bliska

Dotychczas tworzyliśmy poszczególne trasy w pliku *config/routes.rb* za pomocą polecenia `map.connect`:

```
map.connect 'incidents/map/:id', :action=>'show_with_map', :controller=>'incidents'
```

Być może zauważyłeś już przy edycji pliku *routes.rb*, że rusztowanie Rails generuje swoje trasy za pomocą innego polecenia:

```
map.resources :incidents
```

To pojedyncze polecenie generuje zbiór standardowych tras zwanych *trasami opartymi na architekturze REST*. Trasy umożliwiają dostęp do standardowych operacji CRUD w aplikacji. Można je przeglądać w konsoli za pomocą polecenia `rake routes`:



		/incidents/news.xml		{ :controller=>"incidents", :action=>"news" }
incidents	GET	/incidents		{ :controller=>"incidents", :action=>"index" }
formatted_incidents	GET	/incidents.:format		{ :controller=>"incidents", :action=>"index" }
	POST	/incidents		{ :controller=>"incidents", :action=>"create" }
	POST	/incidents.:format		{ :controller=>"incidents", :action=>"create" }
new_incident	GET	/incidents/new		{ :controller=>"incidents", :action=>"new" }
formatted_new_incident	GET	/incidents/new.:format		{ :controller=>"incidents", :action=>"new" }
edit_incident	GET	/incidents/:id/edit		{ :controller=>"incidents", :action=>"edit" }
formatted_edit_incident	GET	/incidents/:id/edit.:format		{ :controller=>"incidents", :action=>"edit" }
incident	GET	/incidents/:id		{ :controller=>"incidents", :action=>"show" }
formatted_incident	GET	/incidents/:id.:format		{ :controller=>"incidents", :action=>"show" }
	PUT	/incidents/:id		{ :controller=>"incidents", :action=>"update" }
	PUT	/incidents/:id.:format		{ :controller=>"incidents", :action=>"update" }
	DELETE	/incidents/:id		{ :controller=>"incidents", :action=>"destroy" }
				/:controller/:action/:id

Każdy z tych wierszy jest pojedynczą trasą, a w niektórych przypadkach trasy mają swoje nazwy. Trasa `/incidents/:id/edit` nosi na przykład nazwę `edit_incident`.

Jak może nam to jednak pomóc w oczyszczeniu ścieżki w kodzie?

Rails udostępnia metodę pomocniczą dla każdej z nazwanych tras

Te nazwy tras opartych na architekturze REST są istotne, ponieważ pomagają nam odnieść się do trasy **z wewnątrz kodu aplikacji**. Pozwalają zamienić takie coś:

```
"/incidents/#{incident.id}/edit"
```

na takie:

```
edit_incident_path(@incident)
```

Dla każdej nazwanej trasy Rails udostępnia metody pomocnicze generujące ścieżki na serwerze lokalnym oraz pełne adresy URL.

Ścieżki na serwerze lokalnym

```
edit_incident_path(@incident) zwraca /incidents/3/edit jeśli @incident ma id = 3
```

Pełne adresy URL

```
edit_incident_url(@incident) zwraca http://localhost:3000/incidents/3/edit
```

Są one nazywane metodami pomocniczymi tras **opartych na architekturze REST**, ponieważ jako parametry przyjmują **zasoby**, czy inaczej obiekty modelu. Pamiętaj — jedną z reguł projektu opartego na architekturze REST jest wyobrażenie sobie aplikacji internetowych jako pojemników na zasoby.

Metody pomocnicze incidents oraz new_incident wywoływane są bez zasobów — na przykład jako incidents_url czy incidents_path.

Metody pomocnicze tras nie tylko usuwają zbędne formatowanie ścieżek z kodu, ale są także łatwiejsze do odczytania i redukują szansę popełnienia błędu w opisie ścieżki.

Do czego przyda nam się to w naszym kodzie? Zmieni ten kod:

```
<p><%= link_to "Edit", "/incidents/#{incident.id}/edit" %></p>
```

na ten:

```
<p><%= link_to "Edit", edit_incident_path(incident) %></p>
```

Jazda próbna



Jazda próbna

Co stanie się po dodaniu odnośnika *Edit* do szablonu częściowego *show*, kiedy klikniemy istniejące zdarzenie na mapie?

Incidents: index

<http://localhost:3000/incidents/>

Listing incidents

Mountain: K9
When: 2009-11-26 15:24:00 UTC
Title: Yeti spotted
Description: Dropped by camp looking for food. Played cards for a while. Lost \$4 to him.
[Edit](#)

Patrz! Odnośnik *Edit* znajduje się teraz w szablonie częściowym!

Auć! Przeglądarka przeszła do innej strony internetowej...

Map Satellite Show labels

Mountain [K9]
When [2009] [November] [26] [15] : [24]
Title Yeti spotted
Description Dropped by camp looking for food. Played cards for a while. Lost \$4 to him.

<http://localhost:3000/incidents/5/edit>

Odnośnik przeniósł przeglądarkę do strony:

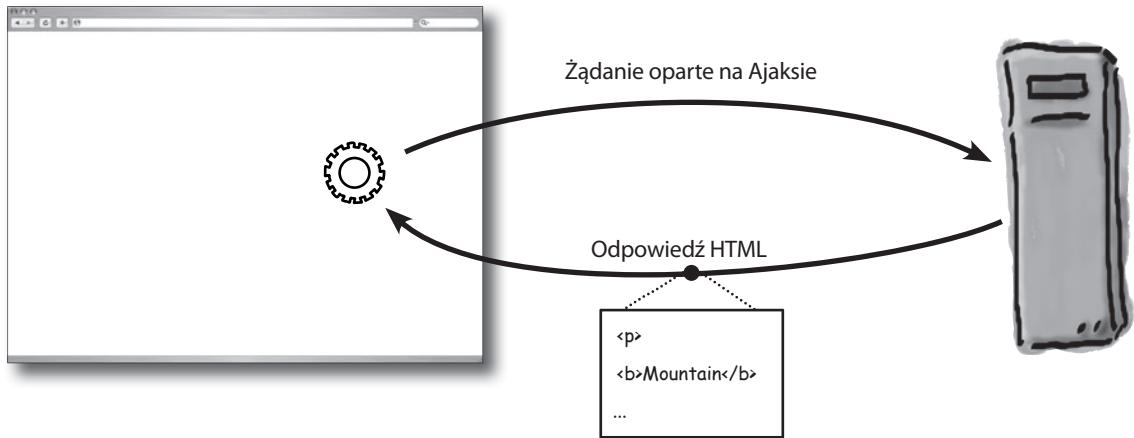
<http://localhost:3000/incidents/5/edit>

Problem polega na tym, że teraz przesyła on zawartość szablonu częściowego *_edit.html.erb* z powrotem do przeglądarki, tak jakby była to inna strona.

A my naprawdę musimy pozostać w przeglądarce na tej samej stronie — jak to zatem naprawić?

Na pomoc spieszny odnośnik oparty na Ajaksie

Utworzony wcześniej formularz new był w stanie zastąpić zawartość wyskakującego okna informacyjnego, ponieważ wykonywał żądanie oparte na Ajaksie. Wykorzystywał odpowiedź serwera do zastąpienia zawartości elementu <div> o identyfikatorze map_info.



Dodany przez nas przed chwilą odnośnik tego nie zrobił. Nakazał po prostu przeglądarkę utworzenie odnośnika do innej strony internetowej. Gdybyśmy zamiast tego umieścili odnośnik oparty na Ajaksie, moglibyśmy obejść ten problem.

Odnośnik oparty na Ajaksie działa dość podobnie do formularza opartego na tej technologii. Po kliknięciu odnośnika opartego na Ajaksie przeglądarkę nie jest nakazywane przejście do innej strony; zamiast tego wygenerowane zostaje oparte na Ajaksie żądanie do serwera, a odpowiedź wykorzystana zostaje do uaktualnienia części strony. Jeśli brzmi to znajomo, to dlatego, że odnośniki oparte na Ajaksie wykorzystaliśmy wcześniej do odświeżenia listy zarezerwowanych miejsc w witrynie linii Coconut Airways.

By przekształcić zwykły odnośnik w oparty na Ajaksie, musimy zmienić poniższy kod:

```
<p><%= link_to "Edit", edit_incident_url(incident) %></p>
```

na takie rozwiązanie:

```
<p><%= link_to_remote "Edit", :update => "map_info",  
                      :url=>edit_incident_path(incident) %></p>
```

To część strony, która odnośnik ma uaktualnić.

To adres URL, który wygeneruje kod HTML uaktualnienia.

Odnośnik powinien teraz generować formularz edycji i wyświetlać go w wyskakującym oknie informacyjnym. Zobaczmy, jak to działa.

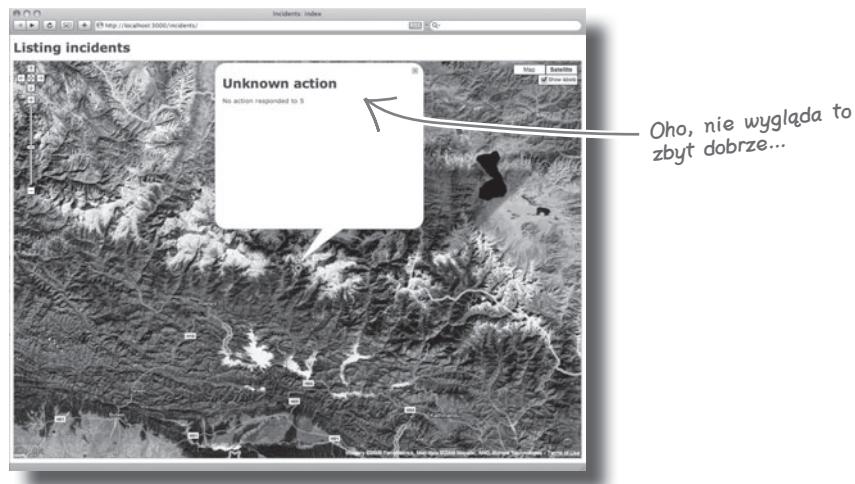


Jazda próbna

Po kliknięciu zdarzenia okno informacyjne wygląda dokładnie tak samo jak wcześniej.



Odnośnik wygląda tak samo, ale pamiętajmy, że tak naprawdę nie jest to już zwykły odnośnik. Zamiast tego w tle działa JavaScript, który czeka na wygenerowanie po kliknięciu odnośnika żądania opartego na Ajaksie. Co się dzieje, kiedy klikniemy odnośnik?



Zamiast wyświetlać formularz edycji, otrzymujemy dziwny błąd „Unknown action”. Co się stało?

Musimy zagłębić się nieco bardziej w trasy...

Używamy niewłaściwej trasy!

Kiedy Rails otrzymuje z odnośnika żądanie oparte na Ajaksie, odnośnik ten poprawnie przesyła żądanie do:

```
http://localhost:3000/incidents/5/edit
```

Zamiast dopasować żądanie do trasy `edit_incident`, zostaje ono dopasowane do jednej z tras domyślnych:

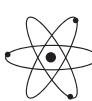
		/incidents/news.xml	{ :controller=>"incidents", :action=>"news" }
incidents	GET	/incidents	{ :controller=>"incidents", :action=>"index" }
formatted_incidents	GET	/incidents.:format	{ :controller=>"incidents", :action=>"index" }
	POST	/incidents	{ :controller=>"incidents", :action=>"create" }
	POST	/incidents.:format	{ :controller=>"incidents", :action=>"create" }
new_incident	GET	/incidents/new	{ :controller=>"incidents", :action=>"new" }
formatted_new_incident	GET	/incidents/new.:format	{ :controller=>"incidents", :action=>"new" }
edit_incident	GET	/incidents/:id/edit	{ :controller=>"incidents", :action=>"edit" }
formatted_edit_incident	GET	/incidents/:id/edit.:format	{ :controller=>"incidents", :action=>"edit" }
incident	GET	/incidents/:id	{ :controller=>"incidents", :action=>"show" }
formatted_incident	GET	/incidents/:id.:format	{ :controller=>"incidents", :action=>"show" }
	PUT	/incidents/:id	{ :controller=>"incidents", :action=>"update" }
	PUT	/incidents/:id.:format	{ :controller=>"incidents", :action=>"update" }
	DELETE	/incidents/:id	{ :controller=>"incidents", :action=>"destroy" }
		/:controller/:action/:id	

System routingu dopasowuje żądanie do tej trasy zamiast do znajdującej się wyżej trasy „edit_incident”.

Rails próbuje dopasować żądanie do trasy domyślnej znajdującej się na dole i ustawia parametr `:action` na 5, a parametr `:id` na `edit`. Nie istnieje jednak akcja o nazwie 5, dlatego żądanie się nie powiedzie.

Dlaczego tak jednak jest? Nasz adres URL (`http://localhost:3000/incidents/5/edit`) ma ten sam format ścieżki co trasa `edit_incident` (`/incidents/:id/edit`).

Dlaczego nie został on dopasowany? W końcu odnośnik działał dobrze przed konwersją na wersję Ajax.



WYSIL
SZARE KOMÓRKI

Przyjrzyj się raz jeszcze liście tras. Oryginalny odnośnik oraz odnośnik oparty na Ajaksie prowadzą do tego samego adresu URL. Dlaczego, Twoim zdaniem, odnośnik oparty na Ajaksie został dopasowany do niewłaściwej trasy?

Więcej dopasowywania tras

Na wybór trasy ma wpływ metoda HTTP

W liście tras znajduje się jedna kolumna, którą dotychczas się nie interesowaliśmy:

Spójrzmy, co tutaj mamy...

incidents	GET	/incidents/news.xml	{ :controller=>"incidents", :action=>"news" }
formatted_incidents	GET	/incidents	{ :controller=>"incidents", :action=>"index" }
	POST	/incidents	{ :controller=>"incidents", :action=>"index" }
	POST	/incidents.:format	{ :controller=>"incidents", :action=>"create" }
	GET	/incidents	{ :controller=>"incidents", :action=>"create" }
new_incident	GET	/incidents/new	{ :controller=>"incidents", :action=>"new" }
formatted_new_incident	GET	/incidents/new.:format	{ :controller=>"incidents", :action=>"new" }
edit_incident	GET	/incidents/:id/edit	{ :controller=>"incidents", :action=>"edit" }
formatted_edit_incident	GET	/incidents/:id/edit.:format	{ :controller=>"incidents", :action=>"edit" }
incident	GET	/incidents/:id	{ :controller=>"incidents", :action=>"show" }
formatted_incident	GET	/incidents/:id.:format	{ :controller=>"incidents", :action=>"show" }
	PUT	/incidents/:id	{ :controller=>"incidents", :action=>"update" }
	PUT	/incidents/:id.:format	{ :controller=>"incidents", :action=>"update" }
	DELETE	/incidents/:id	{ :controller=>"incidents", :action=>"destroy" }
		/:controller/:action/:id	

Czym są te wszystkie GET, POST, PUT oraz DELETE?

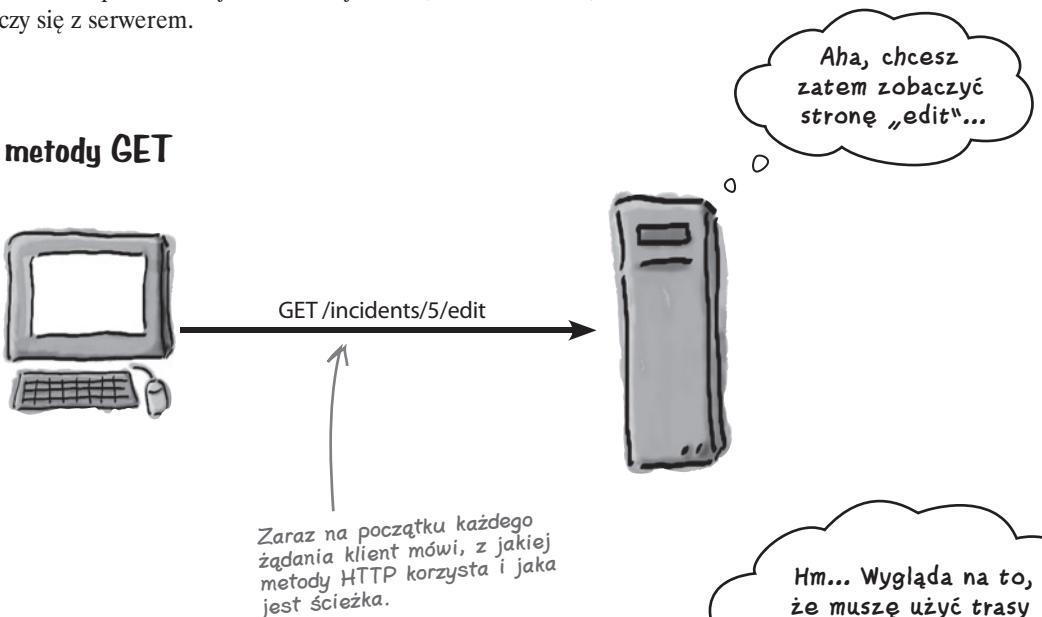
Są to **metody HTTP**. Każde żądanie wykorzystuje określona metodę HTTP, a Rails przy decydowaniu, której trasy użyć, wykorzystuje zarówno metodę, jak i ścieżkę.

Czym są tak naprawdę metody HTTP?

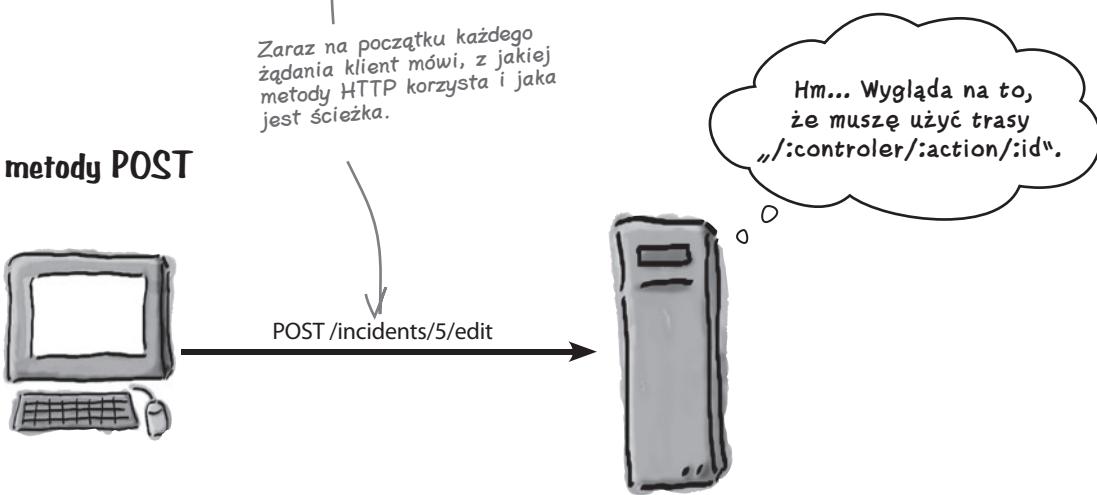
Czym jest zatem metoda HTTP?

Pomimo nazwy metody HTTP nie przypominają metod języka Ruby, które znajdziesz na przykład w kontrolerze. Zamiast tego metoda HTTP wymieniana jest w niskopoziomowej komunikacji HTTP, która zachodzi, kiedy klient łączy się z serwerem.

Za pomocą metody GET



Za pomocą metody POST



Dlaczego dwie wersje odnośnika robią różne rzeczy? Cóż — zwykłe odnośniki HTML przesyłają do serwera żądanie **GET**. Ajax domyślnie wysyła żądania **POST**.

By odnośnik działał, musimy *także* przekazać mu, z której metody HTTP ma korzystać — jak poniżej:

```
<p><%= link_to_remote "Edit", :update => "map_info",
:url=>edit_incident_path(incident), :method=>'get' %></p>
```

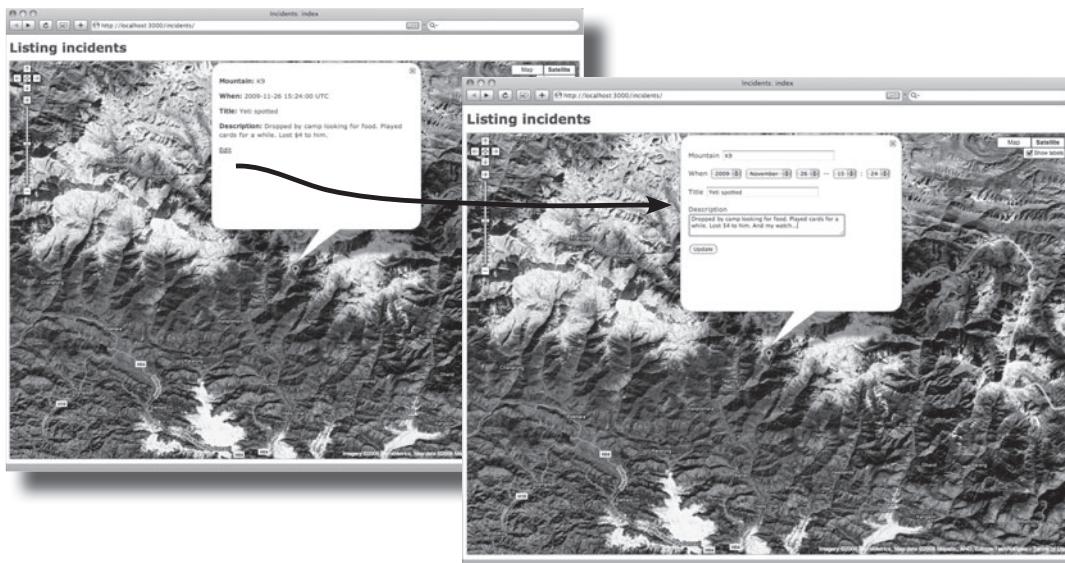


Jazda próbna

Wszystko działa!

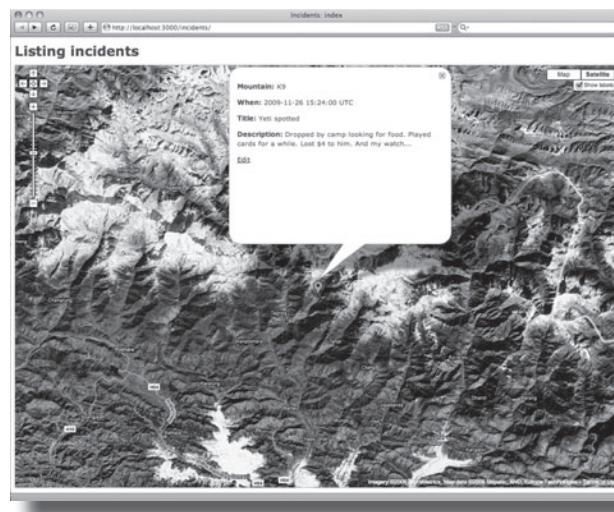
Po kliknięciu odnośnika zdarzenia widzimy informacje wyświetlane wraz z odnośnikiem *Edit*.

Po kliknięciu *Edit* utworzone zostaje oparte na Ajaksie żądanie do formularza edycji, które wykorzystane zostaje do zastąpienia zawartości wyskakującego okna informacyjnego.



A co z formularzem? Działa tak samo dobrze jak formularz new.

Uaktualnia zdarzenie, a następnie ponownie je wyświetla:



Nie istnieja grupie pytania

P: Czy wykorzystywanie łańcuchów znaków jako ścieżek naprawdę jest takie złe?

O: Łącuchy znaków będą działać, ale będą trudniejsze do odczytania i bardziej podatne na błędy od metod pomocniczych.

P: Dlaczego będą bardziej podatne na błędy?

O: Jeśli niepoprawnie wpiszesz nazwę metody pomocniczej trasy, Rails utworzy błąd i poinformuje Cię o tym. Jeśli źle wpiszesz ścieżkę w łańcuchu znaków, system albo nie zgłosi błędu, albo zgłosi inny błąd spowodowany przez złą ścieżkę.

P: Dlaczego link_to_remote tworzy żądanie POST, a link_to — żądanie GET?

O: Metoda pomocnicza link_to tworzy zwykły odnośnik HTML. Przeglądarki w zwykłych odnośnikach zawsze wykorzystują metodę GET. Metoda pomocnicza link_to_remote tworzy natomiast żądanie oparte na Ajaksie, a żądania tego typu domyślnie zawsze przesyłane są za pomocą metody POST.

P: Dlaczego HTTP w ogóle ma metody GET i POST? Na co się one przydają?

O: Żądanie GET zaprojektowane jest tak, by można je było powtarzać. Nie powinno mieć znaczenia, ile razy wykonuje się to samo żądanie GET. Metoda GET jest często wykorzystywana dla żądań, które po prostu odczytują informacje. Metoda POST wykorzystywana jest w żądaniach, które za każdym razem mogą zmienić dane na serwerze, dlatego normalnie używane są one do uaktualniania bazy danych.

P: A co z PUT i DELETE?

O: Metoda PUT wykorzystywana jest w żądaniach uaktualniających rekordy w bazie danych. Natomiast DELETE używana jest dla operacji usuwania z bazy danych.

P: Czy tak jest we wszystkich aplikacjach internetowych?

O: Jest tak we wszystkich aplikacjach opartych na Rails. Używanie poprawnej metody HTTP jest bardzo istotną częścią projektu opartego na architekturze REST.

P: Czy dlatego właśnie form_for jest w stanie używać tego samego kodu do generowania formularzy, które mogą uaktualniać i wstawiać dane?

O: Tak. Jeśli obiekt został już zapisany, form_for generuje formularz, który będzie korzystał z metody PUT. Jeśli obiekt jest nowy, wygeneruje formularz wykorzystujący metodę POST.

P: Ktoś powiedział mi, że przeglądarki nie mogą wykorzystywać metod PUT i DELETE. Czy to prawda?

O: Jedynie kilka przeglądarek obsługuje metody PUT i DELETE. By upewnić się, że wszystko będzie działać, Rails dodaje kolejne ukryte pole o nazwie _method, w którym przechowywana jest nazwa metody HTTP. Jeśli otrzymane zostanie żądanie z _method= "PUT", Rails potraktuje je jako żądanie PUT, nawet jeśli tak naprawdę zostało przesłane za pomocą metody POST.

P: Czym jednak jest projektowanie oparte na architekturze REST?

O: To sposób projektowania aplikacji internetowych, który próbuje pozostać zbliżony do oryginalnego projektu samego Internetu. Więcej informacji na ten temat znajdziesz pod adresem <http://tinyurl.com/28nguu>.

Czas na konkurs!

Witryna Head First Climbers Cieę potrzebuje!

Wspinacze kochają tę aplikację. Myślimy jednak, że możesz zrobić coś lepszego.

Czas zabrać się do pracy i znacznie ulepszyć tę aplikację. Dodaj więcej szczegółów, więcej widgetów, coś interesującego dla oka. Oto kilka pomysłów:

Można tu tworzyć, odczytywać i aktualniać zdarzenia. Ale jak to — nie da się ich usuwać??? Może byćście to naprawili?

Może datoby się tworzyć animacje zdarzeń z ekspedycji?

A może... jeszcze coś innego?

A może połączyć tę aplikację z innymi aplikacjami typu mashup w duchu Web 2.0? Dlaczego nie możemy przesytać informacji za pomocą serwisu Twitter bezpośrednio ze szczytu K2?

Dlaczego użytkownicy nie mogą przesytać dodatkowych plików? Zdjęć, odnośników, filmów wideo, komentarzy?

A może połączyć ze sobą zdarzenia w jakiś rodzaj obiektu ekspedycji?

Mögliby użytkownicy przekazać punkty na mapie? (Żeby to zrobić, będziesz się pewnie musiał nauczyć czegoś więcej o Google Maps API. Zobacz więcej na stronie <http://tinyurl.com/2bfbm2>).

Zbuduj najlepszą wersję aplikacji Head First Climbers, a następnie zgłoś jej adres URL na forum Head First Labs „Head First Rails”. Masz szansę wygrać coś fajnego z O'Reilly, a także zyskać międzynarodową sławę dzięki umieszczeniu na stronie Head First Labs!

Odwiedź nas na forum, gdzie zobaczyś, jak wziąć udział w konkursie.

Head First Labs from O'Reilly Media, Inc.

http://www.headfirstlabs.com/

O'REILLY Brain-Friendly Guides from O'Reilly Media, Inc.

Head First Labs

Home Books Forums Blog About write for us

Algebra and Rails now available! See our Special Announcement for details. First Algebra and Head First Rails information.

WireSide Chat with Rebecca Rides! RIA Revolution interviewed Rebecca Rides. Read it here!

Call for technical reviews!

Forum Main Page

- Book News, Info, and Discussion
- Polls and Surveys
- Head First Ajax
- Head First C#
- Head First Design Patterns
- Head First HTML with CSS & XHTML

Search Head First Labs O'Reilly.com Go

Search Tips

Subscribe to OUR RSS FEED

FOLLOW US ON



Niezbędnik programisty Rails

Masz za sobą rozdział 9. i teraz do swojego niezbędnika programisty Rails możesz dodać umiejętność dodawania do swoich aplikacji bardziej zaawansowanych opcji.

Narzędzia Rails

Polecenie „rake routes” wyświetla trasy aplikacji.

`<nazwa_trasy>_path(obiekt)` zwraca ścieżkę dla podanej nazwanej ścieżki, wykorzystując do tego identyfikator podanego obiektu.

`edit_<nazwa_modelu>_path(obiekt)` zwraca ścieżkę do edytora obiektu.

`new_<nazwa_modelu>_path` zwraca ścieżkę do edytora nowego obiektu.

Zmiana „`_path`” na „`_url`” zwraca pełny adres URL zamiast ścieżki lokalnej.

10. Prawdziwe aplikacje

Rails w świecie rzeczywistym

On tak bardzo urósł...
Mam nadzieję,
że nie porzuci Rails.



Nauczyłeś się już wiele o Ruby on Rails. By jednak zastosować tę wiedzę **w prawdziwym świecie**, będziesz musiał zastanowić się nad kilkoma sprawami. W jaki sposób połączyć aplikację **z inną bazą danych?** Jak **testuje** się aplikacje Rails? Jak można wydobyć maksimum możliwości z Rails oraz **języka Ruby?** I skąd można dowiedzieć się o **najświeższych nowościach** w świecie Rails? Czytaj dalej, a pokażemy Ci kierunek, dzięki któremu jeszcze bardziej rozwiniiesz swoje umiejętności programistyczne.



Widzę, że omówiliśmy już wiele aspektów tworzenia aplikacji internetowych opartych na Rails, jednak to tylko książka. Jak będzie to wyglądało w prawdziwym świecie? Nie powiesz mi przecież, że wszystko będzie wyglądało tak samo, kiedy zaczniemy programować prawdziwe aplikacje...

Wszystkie techniki, które opanowałeś, NAPRAWDĘ są przydatne... jednak wiele możesz się jeszcze nauczyć.

Tak, to prawda. W świecie rzeczywistym nie wszystko zawsze idzie tak, jak powinno. Co więcej, Rails jest naprawdę rozbudowaną platformą. Istnieje o wiele więcej niż tylko to, co ta książka — czy dowolna inna mająca mniej niż kilka milionów stron — mogłaby kiedykolwiek omówić.

Nie należy jednak zakładać, że nie zostałeś dobrze przygotowany! Przejrzyj ostatnie strony książki, by zobaczyć, co wiesz, a nawet znaleźć kilka wskazówek dotyczących rzeczy, o których jeszcze nie wiesz.



*JAKI JEST MÓJ CEL?

Omówiliśmy jedynie kilka metod pomocniczych dostępnych w Rails, jednak do wyboru masz o wiele większą ich liczbę. Sprawdź, czy będziesz potrafił dopasować każdą z poniższych metod pomocniczych do jej rzeczywistej funkcji.

`number_to_phone`

Pozwala przeglądarkom automatycznie wykrywać kanał RSS.

`number_to_percentage`

Pozwala zmierzyć, ile czasu trwa wykonanie fragmentów kodu szablonu.

`error_message_on`

Formatuje liczbę jako numer telefonu z USA.

`auto_discovery_link_tag`

Formatuje liczbę jako wartość procentową.

`image_tag`

Zwraca zbiór list wyboru dla roku, miesiąca oraz dnia.

`benchmark`

Pomaga sformatować komunikaty o błędach.

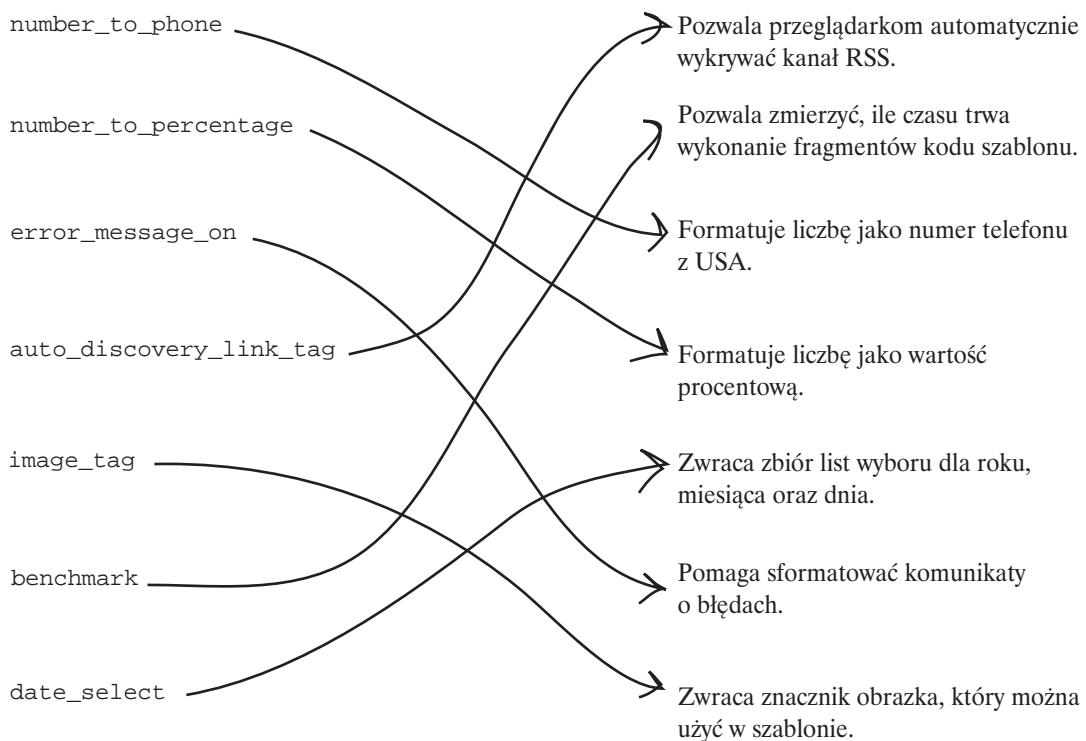
`date_select`

Zwraca znacznik obrazka, który można użyć w szablonie.

JAKI JEST MÓJ CEL?

ROZWIAZANIE

Omówiliśmy jedynie kilka metod pomocniczych dostępnych w Rails, jednak do wyboru masz o wiele większą ich liczbę. Sprawdź, czy będziesz potrafił dopasować każdą z poniższych metod pomocniczych do jej rzeczywistej funkcji.



Ciekawostki

By dowiedzieć się więcej o metodach pomocniczych dostępnych w najnowszej wersji Rails, odwiedź stronę
<http://tinyurl.com/railshelpers>.

Patrz! Eksperymenty z językiem Ruby!

Czy zauważłeś, jak niewiele języka Ruby musisz znać, by tworzyć świetne aplikacje internetowe oparte na Rails? Mimo to znajomość języka Ruby może się czasami przydać. Poniżej znajduje się kilka przykładowych fragmentów kodu w tym języku — może chcesz nieco poeksperymentować? Wpisz poniższe fragmenty i zobacz, co się stanie...



Wczytuje wiersze pliku do tablicy o nazwie a:

```
a=File.readlines("filename.txt")
```

Sortuje tablicę:

```
a.sort
```

Odwraca łańcuch znaków:

```
"Bass ackwards".reverse!
```

Zwraca kopię odwróconego łańcucha znaków:

```
"Bass ackwards".reverse
```

Czy łańcuch znaków s zawiera podłańcuch Waldo?

```
/Waldo/ =~ s
```

Czy łańcuch znaków jest amerykańskim kodem pocztowym?

```
/^\d{5}$/ =~ "90210"
```

Konwersja łańcucha znaków na liczbę całkowitą typu Fixnum:

```
"12345".to_i
```

Konwersja łańcucha znaków na liczbę zmiennoprzecinkową typu Float:

```
"3.1415".to_f
```

Konwersja obiektu a na łańcuch znaków:

```
a.to_s
```

Formatuje zawartość tablicy:

```
[1, 2, 3, 4, 5].inspect
```

Formatuje zawartość tablicy asocjacyjnej:

```
{:a=>1, :b=>"c"}.inspect
```

Tworzy łańcuch znaków składający się z 50 znaków =:

```
"=" * 50
```

Tworzy tablicę ze słów składających się na łańcuch znaków:

```
"to be or not to be".split
```

Zwraca klasę (typ danych) obiektu o:

```
o.class
```

Zaokrąglą liczbę zmiennoprzecinkową do najbliższej liczby całkowitej:

```
(3.14).round
```

Oblicza pierwiastek kwadratowy:

```
Math.sqrt(16)
```

Usuwa plik:

```
File.delete("filename.txt")
```

Zwraca aktualną datę i czas:

```
Time.now
```

Zwraca aktualny rok:

```
Time.now.year
```

Nadaje metodzie inną nazwę:

```
alias my_method
```

Zwraca tablicę plików z katalogu:

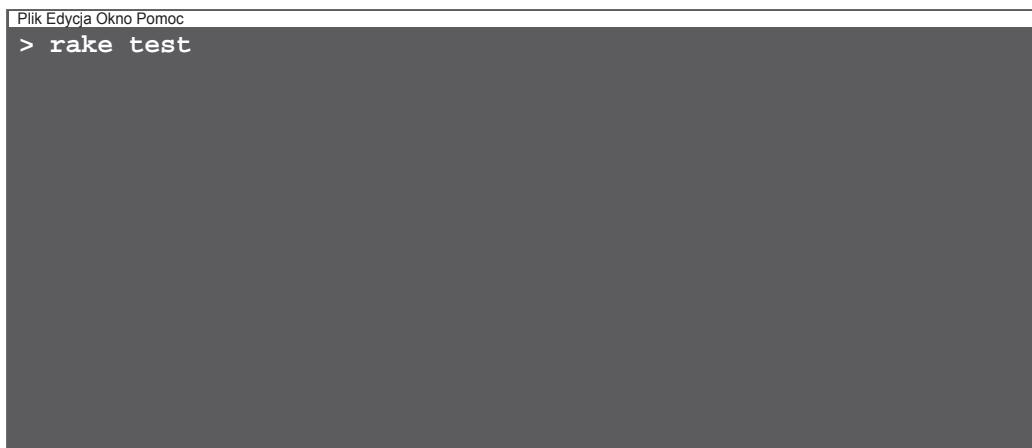
```
Dir.entries("directoryName")
```

Aplikacje internetowe muszą być testowane

Zautomatyzowane testy to jeden z najważniejszych etapów programowania aplikacji, a dotychczas nic o tym nie wspominaliśmy. Dlaczego? Testowanie oprogramowania oparte jest na głębokim zrozumieniu wykorzystywanych narzędzi, a projektowanie testów może być o wiele trudniejsze (ale i zabawniejsze) od pisania samego kodu. Z tego powodu w niniejszej książce skoncentrowaliśmy się na umiejętnościach niezbędnych do zrozumienia sposobu działania Rails. Dopiero kiedy to zrozumiesz, będziesz mógł zacząć myśleć o testowaniu aplikacji.

Nie oznacza to jednak, że testowanie wykonuje się długo po skończeniu tworzenia systemu. Zupełnie tak nie jest. Najlepsze testy pisane są *przed* napisaniem najważniejszych części kodu.

Rails zawiera mnóstwo narzędzi wspomagających testowanie — o wiele więcej niż jakakolwiek inna platforma. Każda aplikacja zawiera zbiór skryptów testowych (znajdujących się w katalogu *test*); za każdym razem gdy generujesz rusztowanie, Rails generuje dla Ciebie również zbiór standardowych testów. Jeśli zatem przejdziesz do folderu, w którym w pierwszym rozdziale książki napisałeś opartą na rusztowaniu aplikację `tickets` i wpiszesz:



Plik Edycja Okno Pomoc
> **rake test**

Więcej informacji na ten temat
znajdziesz w książce „Extreme
Programming Explained”, ISBN-13:
978-0321278654.

Rails wykona dla Ciebie cały zestaw testów. Czy oznacza to, że nigdy nie będziesz musiał pisać własnych? Tak naprawdę wcale nie. Jako programista Rails sporo czasu spędzasz na pisaniu i utrzymywaniu testów.

Jakie rodzaje testów są dostępne?

Istnieją trzy główne typy testów:

Testy jednostkowe

Rails czasami używa pewnych pojęć w sposób nieco odmienny od tego, co znajdziesz gdzieś indziej. W większości systemów test jednostkowy (ang. *unit test*) to test jakiegokolwiek samodzielnego fragmentu kodu. W Rails pojęcie to ma bardziej zawężone znaczenie. Tutaj „test jednostkowy” oznacza test klasy modelu. Rails tworzy dla Ciebie standardowe testy jednostkowe znajdujące się w katalogu *test/unit* za każdym razem, gdy generujesz model — albo w sposób bezpośredni, albo za pomocą rusztowania.

Testy funkcjonalne

Pod nazwą testów funkcjonalnych (ang. *functional test*) Rails rozumie test pojedynczego kontrolera. Testy funkcjonalne sprawdzają, czy jeśli wykonasz określony rodzaj żądania, otrzymasz określony rodzaj odpowiedzi. Testy funkcjonalne znajdziesz w katalogu *test/functional*. Rails tworzy testy funkcjonalne za każdym razem, gdy generujesz kontroler — albo w sposób bezpośredni, albo za pomocą rusztowania.

Testy integracyjne

To testy wysokopoziomowe, które wyglądają nieco podobnie do skryptów testowych wykonywanych przez ręcznych testerów. Testy integracyjne sprawdzają system jako całość. Automatyzują pewne zbiory działań, które w systemie mógłby wykonać typowy użytkownik. W strukturze katalogów aplikacji znajduje się specjalny folder przeznaczony dla testów integracyjnych (*test/integration*), jednak nie są one generowane automatycznie. Są one blisko powiązane z tym, co ma robić nasz system, dlatego trzeba je utworzyć ręcznie.

Dane testowe dla wszystkich typów testów przechowywane są w plikach danych znajdujących się w katalogu *test/fixtures*. Fikstura (ang. *fixture*) to po prostu elegancka nazwa zbioru danych testowych. Rails przechowuje dane fikstur w specjalnej, osobnej bazie danych testów, by mieć pewność, że dane programistyczne (czy też prawdziwe dane działającej aplikacji) nie zostaną pomieszczone z danymi potrzebnymi do testów.

Więcej informacji na temat testowania w Rails można znaleźć na stronie <http://tinyurl.com/railstest>.

Udostępnienie aplikacji użytkownikom

Aplikacja nie pozostanie wiecznie na etapie programowania i w końcu kiedyś będziesz musiał udostępnić ją użytkownikom. Co należy wtedy zrobić? Nie jest dobrym pomysłem, by na tym etapie aplikacja zawierała kod podający lokalizację bazy danych czy podobne informacje. Nie chcemy jednak także, by działająca aplikacja oraz testowa wersja kodu robiły różne rzeczy. Chcemy, by po prostu korzystała z różnych baz danych.

Z tego powodu Rails pozwala na określenie *środowiska* (ang. *environment*). Środowisko konfiguruje lokalizację oraz typ bazy danych, a także kilka innych ustawień, na przykład czas przechowywania komunikatów.

Domyślnie aplikacja skonfigurowana została tak, by używać trzech środowisk:

1 Środowisko programistyczne

To środowisko wykorzystywane domyślnie. Z niego właśnie korzystaliśmy dotychczas w całej książce. Środowisko programistyczne korzysta z bazy danych *db/development.sqlite3*.

2 Środowisko testowe

To środowisko służy wyłącznie potrzebom zautomatyzowanych skryptów testowych.

3 Środowisko produkcyjne

To środowisko działającej aplikacji.

Jak przełączamy się pomiędzy środowiskami?

Kiedy uruchamiasz serwer, Rails szuka zmiennej środowiskowej o nazwie `RAILS_ENV`. Mówi ona, z którego środowiska należy skorzystać. Jeśli chcesz przełączyć się między środowiskiem programistycznym a produkcyjnym, musisz odpowiednio ustawić zmienną `RAILS_ENV`:

Plik Edycja Okno Pomoc
> set RAILS_ENV=production
> ruby script/server

Takie coś musisz wpisać w systemie Windows...

Plik Edycja Okno Pomoc
> RAILS_ENV=production
> ruby script/server

... a takie w systemach Linux, Unix lub Mac.

Jak zmienia się bazę danych?

Jeśli spojrzyesz do pliku *config/database.yml*, znajdziesz tam szczegóły bazy danych dla każdego ze środowisk.

Przykładowo oryginalne środowisko programistyczne dla bazy danych **SQLite** może mieć takie ustawienia:

```
development:  
  adapter: sqlite3  
  database: db/development.sqlite3  
  timeout: 5000
```



Jeśli jednak chcesz zmienić środowisko produkcyjne tak, by korzystało z bazy danych **Oracle**, ustawienia będą najprawdopodobniej wyglądały mniej więcej tak:

```
production:  
  adapter: oracle  
  host: mydatabaseserver  
  username: scott  
  password: tiger
```



A jeśli chcesz, by środowisko produkcyjne wykorzystywało bazę danych **MySQL** znajdującą się na tym samym serwerze co Rails, musiałbyś zmienić konfigurację w następujący sposób:

```
production:  
  adapter: mysql  
  database: my_db_name  
  username: root  
  password:  
  host: localhost
```



Czym jest architektura REST?

W niniejszej książce sporo mówiliśmy o REST. Jak Rails wykorzystuje REST. Jak architektura REST stanowi jedną z reguł projektowych Rails. Jak to, że gdy będziesz używał REST, Twojezęby będą miały jeszcze bielszy odcień bieli, Twoje życie stanie się pasmem sukcesów, a na świecie zapanuje powszechnie dobro i szczęście.

Zacznijmy od podstaw. **REST** to skrót od *Representational State Transfer* i jest sposobem ustrukturyzowania tego, jak ludzie pracują z systemami komputerowymi. Oczywiście najważniejszym systemem komputerowym na świecie jest Internet i znaczące jest to, że osoba, która stworzyła REST — Roy Fielding — była jednym z autorów specyfikacji HTTP.

Dlaczego to, że osoba będąca twórcą REST jest jednocześnie twórcą HTTP, ma takie znaczenie? Dlatego, że projekt oparty na architekturze REST oznacza projekt aplikacji, która działa tak, jak od początku miał wyglądać Internet.

Jakie są zatem główne zasady architektury REST?

1

Wszystkie istotne elementy są zasobami.

Oznacza to, że wszystkie istotne dane w Twoim systemie są osobno zdefiniowanymi elementami, z którymi możesz coś zrobić. Jeśli masz stronę internetową sprzedającą pączki, pączki są Twoimi zasobami.

2

Każdy zasób ma swoją nazwę własną.

W przypadku Internetu oznacza to, że wszystko ma swój adres URL.

3

Na zasobach można wykonywać standardowy zbiór operacji.

Operacje CRUD (tworzenia, odczytywania, aktualniania oraz usuwania) to typowy zbiór operacji; są one obsługiwane przez Rails oraz przez Internet.

4

Klient i serwer komunikują się ze sobą bez stanu.

Oznacza to, że kiedy klient (taki jak przeglądarka) komunikuje się z aplikacją opartą na architekturze REST, wygląda to jak odrębny zbiór żądań i odpowiedzi. Klient przemawia do serwera. Serwer odpowiada. Konwersacja się kończy.

Wszystko to wydaje się zupełnie oczywiste, prawda? To dość dobry opis sposobu działania Internetu.

Właściwie to dość dobry opis sposobu działania Internetu kiedyś. Zanim wszystko poszło nie tak...

Aplikacje internetowe pobłędziły

Wyobraź sobie, że była sobie aplikacja, która pozwalała użytkownikom sprzedawać części zapasowe do rakiet:

Jej twórcy mogli utworzyć system wyświetlający element rakiety w następujący sposób:

`http://www.boosters-r-us.com/airframes/472`

Strona internetowa służy do opisu komponentu rakiety, a to jest adres URL, którego można użyć jako nazwy komponentu.

Spójrz jednak, co się dzieje, kiedy ktoś uaktualnia szczegóły komponentu — na przykład jego cenę. Formularz internetowy systemu przesyła szczegóły do poniższego adresu URL:

`http://www.boosters-r-us.com/airframes/472/update`

Problem polega na tym, że takie rozwiązanie nie jest oparte na architekturze REST. Dlaczego? Ponieważ adresy URL w systemie opartym na architekturze REST powinny być nazwami zasobów.

A drugi adres URL nie reprezentuje **zasobu**, tylko **działanie**.

Dlaczego niezgodność z architekturą REST jest problemem?

Czy kiedykolwiek wróciłeś do jakiegoś adresu URL w przeglądarce i zostałeś zapytany, czy chcesz **ponownie przesłać dane**? Historia przeglądanych stron to po prostu lista adresów URL, co powinno oznaczać, że jest to lista nazw. Kiedy jednak aplikacja internetowa wykorzystuje adresy URL reprezentujące działania, gdy przeglądasz historię stron, przeglądarka nie wie, czy chcesz ponowić wykonane działania.

Wystarczy Ci metoda HTTP

W jaki sposób możemy zatem obejść ten problem? Trzecia reguła architektury REST mówi, że dostępna powinna być dobrze zdefiniowana lista działań. Aplikacja oparta na architekturze REST wykorzystuje metody HTTP do zdefiniowania działań i pozostawia adresom URL nazywanie zasobów:

To trasy oparte na architekturze REST w aplikacji pochodzącej z rusztowania.

Operacja CRUD	Metoda HTTP	Adres URL
Utworzenie komponentu	POST	<code>http://www.boosters-r-us.com/airframes/</code>
Odczytanie komponentu	GET	<code>http://www.boosters-r-us.com/airframes/472</code>
Uaktualnienie komponentu	PUT	<code>http://www.boosters-r-us.com/airframes/472</code>
Usunięcie komponentu	DELETE	<code>http://www.boosters-r-us.com/airframes/472</code>

Życie na krawędzi

Rails ciągle się zmienia — jak zatem masz pozostać na bieżąco ze wszystkimi fantastycznymi nowościami, które są dodawane do tej platformy? Jednym ze sposobów jest życie na krawędzi, czyli korzystanie z **Edge** (ang. *krawędź*).

Rails bardzo upraszcza korzystanie z najnowszej wersji, zwanej Edge Rails, dzięki pobieraniu i instalowaniu najświeższej wersji platformy bezpośrednio do aplikacji.

W niektórych platformach aplikacji przejście na najnowszą wersję jest bardzo skomplikowane. Wymaga uruchomienia przeglądarki internetowej. Pobrania plików ze strony. Odczytania najnowszych instrukcji instalacyjnych. Zabawy ze ścieżkami. Konfiguracji ustawień, tak by odpowiadały one naszemu systemowi. I tak dalej, i tak dalej. Byłyby to na tyle skomplikowane, że jedynie niewielka liczba osób by się na to zdecydowała.

Tymczasem mnóstwo osób korzysta z Edge Rails. Dlaczego? Tak naprawdę nie tylko dlatego, że chcą korzystać z najnowszych funkcji. Rails ciągle znajduje się w fazie intensywnego rozwoju i może się okazać, że nawet niewielkie uaktualnienie może sprawić, że jakiś fragment kodu w Twoim systemie przestanie działać. By zatem upewnić się, że aplikacje będą działać wraz z rozwojem Rails, ich twórcy nie muszą czekać całych tygodni czy miesięcy na uaktualnienie, zamiast tego uaktualniając Rails Edge codziennie.

Jak jednak można zainstalować Rails Edge w swojej aplikacji? To bardzo proste. Robisz tak:

```
Plik Edycja Okno Pomoc
> rake rails:freeze:edge
```

Wystarczy Ci to jedno polecenie. Narzędzie `rake` łączy się z serwerami Rails i pobiera najświeższą wersję skryptów Rails oraz instaluje je w katalogu `vendor/plugins` Twojej aplikacji. Z każdym uruchomieniem serwera Rails przed przejściem do głównej instalacji Rails na komputerze sprawdzany jest kod w katalogu `vendor`. Oznacza to, że dla tej aplikacji wykorzystana zostanie wersja Edge Rails.

**Życie na krawędzi może być emocjonujące.
Czasami jednak lepiej jest odkrywać problemy
z niezgodnością po jednym na raz...**

Uzyskanie dodatkowych informacji

Choć Rails pozwala na tworzenie w pełni funkcjonalnych aplikacji internetowych w sposób szybki i bezpieczny, bez wątpienia pełne opanowanie tej platformy wymaga sporo czasu. Wszystkiego jest po prostu za dużo.

Dlatego właśnie potrzebna jest Ci porządną baza wiedzy. Najlepszą bazę wiedzy znajdziesz w Internecie. Rails ciągle się zmienia. Do kodu źródłowego Rails codziennie dodawane są nowe elementy i jedyną szansę na pozostanie na bieżąco daje Ci Internet. Oto lista świetnych stron internetowych na początek:

1 <http://www.rubyonrails.pl/>

To główna siedziba platformy Rails. Znajdziesz tam nie tylko samo oprogramowanie, ale także prezentacje, filmy i odnośniki do dalszej lektury.

2 <http://wiki.rubyonrails.org/rails>

Tutaj otrzymasz szczegółowe instrukcje dotyczące instalacji oraz rozwiązywania problemów, a także odnośniki do dalszych zasobów dostępnych w Internecie.

3 <http://ryandaigle.com/>

Blog Ryana zawiera bogactwo informacji na temat najnowszych sztuczek, jakie możesz wykonać za pomocą Rails.

4 <http://www.ruby-lang.org/pl/>

Tu znajdziesz najświeższe informacje o języku Ruby.

Wbudowana dokumentacja

Oprócz bogactwa materiałów dostępnych w Internecie Ruby on Rails zawiera większość potrzebnych Ci rzeczy zaraz po zainstalowaniu.

Dwa najważniejsze narzędzia wiersza poleceń to:

```
ri <klasa>
```

gdzie **<klasa>** to klasa języka Ruby, o której chcesz dowiedzieć się więcej.

Przykładowo `ri Array` pokaze Ci więcej informacji na temat klasy tablicy `Array`.

Innym przydatnym źródłem informacji jest serwer gem. Gem to najpopularniejsze narzędzie do zarządzania pakietami w języku Ruby i polecenie wykorzystywane chyba najczęściej do instalowania Rails. Gem ma wbudowany serwer udostępniający tę samą dokumentację API, którą znajdziesz na stronie <http://api.rubyonrails.org>. By uruchomić to narzędzie, wpisz:

```
gem server
```

a następnie otwórz w przeglądarce stronę:

```
http://localhost:8808/
```

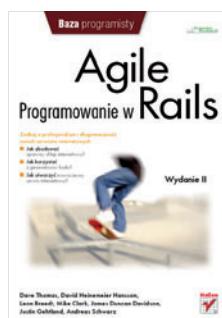
Nieco dodatkowej lektury...

Oczywiście w naszej firmie wszyscy jesteśmy ludźmi książek. I bez względu na to, jak świetne są materiały dostępne w Internecie, nic nie pobije możliwości czytania prawdziwej, drukowanej książki. Skoro dotarłeś już do końca niniejszej książki, a Twój mózg pełny jest świeżych wiadomości o Ruby on Rails, być może będziesz chciał zerknąć do tych obłędnych pozycji:



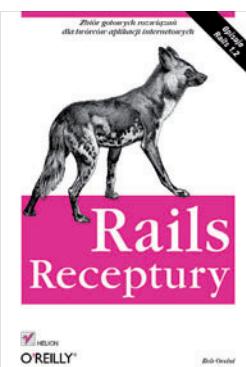
Ruby. Tao programowania w 400 przykładach

W Head First Labs kochamy tę książkę. To przepastna, solidna pozycja, wspaniale napisana przez Hala Fultona. Ta książka zabierze Cię w daleką podróż z językiem Ruby. Najlepsze w niej jest to, że nie tylko opowie Ci o szczegółach tego języka, ale wyjaśni również filozofię leżącą u podstaw jego projektu. Wiele elementów czyniących platformę Rails tak wspaniałą pochodzi bezpośrednio z języka Ruby. Wiele z nich omówiono w tej właśnie książce.



Agile. Programowanie w Rails

To świetna książka, która wprowadzi Cię w bardziej zaawansowane programowanie w Rails. Ciekawe jest to, że sama pisana jest jak projekt programistyczny. Na kilka miesięcy przed wydaniem nowej wersji anglojęzycznej w Internecie dostępna jest wersja beta, którą czytelnicy mogą wypróbować i skomentować.

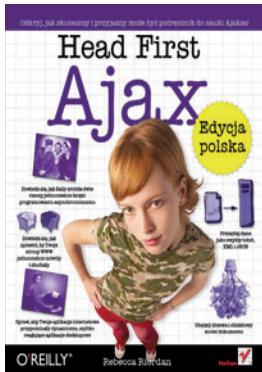


Rails. Receptury

Kiedy już lepiej poznasz Rails, z pewnością spotkasz się z koniecznością rozwiązywania tych samych typów problemów, z jakimi wcześniej musiało sobie radzić wiele innych osób. Bez obaw! Książka *Rails. Receptury* udostępnia świetny zbiór gotowych fragmentów kodu, dzięki którym poradzisz sobie z wszelkimi trudnościami.

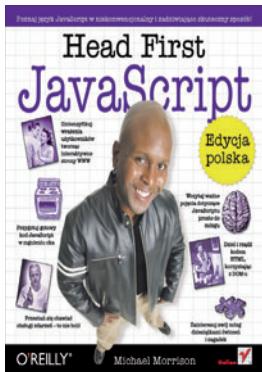
Książki Head First o podobnej tematyce

Oprócz książek poświęconych Ruby on Rails być może przydadzą Ci się także pozycje dotyczące powiązanych zagadnień. A jak najlepiej zaangażować mózg w nowe zagadnienie? Oczywiście za pomocą książki z serii *Head First*!



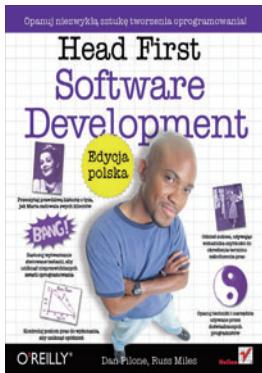
Head First Ajax. Edycja polska.

Rails zawiera wbudowaną obsługę technologii Ajax, jednak by naprawdę zyskać jak najwięcej, powinieneś zapoznać się ze sposobem działania Ajaksu. A czy można zrobić to lepiej niż z *Head First Ajax*?



Head First JavaScript. Edycja polska.

Ajax oparty jest na JavaScriptie i jeśli dowiesz się, jak programować w tym języku, Twoja aplikacja będzie błyszczeć. *Head First JavaScript. Edycja polska*. to świetne wprowadzenie do tego języka.



Head First Software Development. Edycja polska.

W niniejszej książce dowiedziałeś się, jak programować w platformie Ruby on Rails. Jeśli jednak chcesz przejść od *programowania do rozwijania oprogramowania*, sięgnij po tę książkę. Pokaż Ci ona, jak robią to prawdziwi zawodowcy — od planowania projektu, przez automatyczne testowanie, po ciągłą integrację.



Niezbędnik programisty Rails

Masz za sobą rozdział 10. i teraz do swojego niezbędnika programisty Rails możesz dodać kilka elementów przydatnych w tworzeniu prawdziwych aplikacji.

Narzędzia Rails

Rails zawiera mnóstwo dodatkowych metod pomocniczych, które można wykorzystywać w aplikacjach.

Język Ruby ma ogromne możliwości. Choć można tworzyć świetne aplikacje internetowe, nie znając języka Ruby, poznanie go jest bardzo przydatne.

Polecenie „rake test” wykonuje zautomatyzowane testy aplikacji.

Polecenie „rake rails:freeze:edge” instaluje najświeższą wersję Rails w aplikacji.

Ustawienie „RAILS_ENV=production” oznacza, że system działa na prawdziwej, produkcyjnej bazie danych.

Polecenie „ri <klasa>” udostępnia informacje o metodach obiektu języka Ruby.

Polecenie „gem server” uruchamia serwer dokumentacji języka Ruby.

Twój mózg — to narzędzie programistyczne o największych możliwościach, jakie masz do dyspozycji.

Koniec wycieczki...



**Twoje odwiedziny w Railsville były
dla nas prawdziwą przyjemnością!**

Smutno nam, że nas opuszczasz, jednak nie ma nic ciekawszego od zastosowania tego, czego się nauczyłeś, w prawdziwym świecie. Dopiero zaczynasz swoją przygodę z platformą Rails, a my usadziliśmy Cię na miejscu kierowcy. **BAW SIĘ DOBRZE.**



Skorowidz

:action, 94
:conditions, 217
:controller, 94
:id, 96
:text, 357
:xml, 357
`<% ... %>`, 125, 206
`<%= ... %>`, 105, 124, 125, 206
`<%= yield %>`, 131
`<a>`, 123
`<div>`, 412
``, 123
`<link... />`, 389
`<title>`, 124

A

action, 110
ActionPack, 38
ActiveRecord, 38
administrator systemu, 178
adres URL, 93, 166, 365, 371
 mapowanie, 94
Ajax, 299, 307, 320, 328, 330, 393
 biblioteki, 308
 edycja, 422
 formularze, 406
 Google Maps, 354, 396
 metody HTTP, 328
 odnośnik Odśwież, 309
 odnośniki, 427
 Prototype, 308
 synchroniczna asynchroniczność, 332
 token uwierzytelniający, 311

wyłączona obsługa JavaScriptu, 416
żądania, 311, 322
 żądania asynchroniczne, 404
akcje, 110, 153, 165, 347, 396
 akcje uaktualniające, 167
 akcje wyświetlające, 167
 aktualizacja bazy danych, 158
 aktualny rok, 441
 aktywność mózgu, 25
alias, 441
aplikacje bazodanowe, 38
aplikacje internetowe, 33, 447
 dane, 38
 generowanie, 39
 testowanie, 442
 uruchamianie, 38
aplikacje mashup, 349, 393
aplikacje Rails, 41, 52
 foldery, 57
 kontroler, 53
 logika biznesowa, 54
 model, 53
 widok, 53
architektura MVC, 57, 79
architektura Rails, 52, 365
architektura REST, 365, 371, 393, 446
 projektowanie, 433
trasy, 424
 zasady, 446
arkusze stylów, 133
array, 117
associative array, 167
atrybuty, 104, 282
authenticity_token, 311

auto_discovery_link_tag, 389, 396
automatyczne uaktualnianie strony, 314

B

baza danych, 36, 38, 46, 82
migracja, 64
modyfikacja struktury, 63
MySQL, 445
obsługa, 110
Oracle, 445
SQLite, 38, 445
tabele, 46
wyszukiwanie, 189
belongs_to, 292
bezpieczeństwo, 178
ograniczanie dostępu do funkcji, 178
biblioteka mapowania relacyjno-objektowego, 36, 38
biblioteki Ajaksu, 308
błędy, 223
 komunikaty, 243

C

CamelCase, 47, 88, 96
ciąg pętli, 122
class, 441
config/routes.rb, 93
controller, 53
create, 42, 157, 194
created_at, 87, 148
CRUD, 42, 50, 194, 446
czas, 381, 441

Skorowidz

częstotliwość żądania, 316
czytnik kanałów RSS, 380

D

dane, 100, 105, 346
dane przestrzenne, 345
dane XML, 354, 371
database.yml, 445
daty, 381, 441
definiowanie powiązania, 283
delete, 42, 194
DELETE, 430, 433
Dir.entries, 441
dodawanie
 dane, 143
 kanały RSS do strony, 389
 kolumny do tabeli, 63
 scriptlety, 126
 wyszukiwanie do interfejsu, 199
dokumentacja, 449
dołączanie szablonu częściowego do szablonu strony, 267
dopasowywanie tras, 430
dostęp do funkcji, 178
dostęp do obiektów w tablicy, 117
DRY, 46
duplikacja kodu, 130
dynamiczne uaktualnianie części strony internetowej, 412
dzielenie informacji na części, 28
dzielenie zawartości strony na odrębne pliki, 261

E

Edge Rails, 448
edit, 173
edit_incident_url, 425
edytacja
 kod HTML, 59
 niepoprawne dane, 226

w stylu Ajaksa, 422
zbiór samodzielnych wartości pól, 202
elementy tablicy, 118
Embedded Ruby, 90, 91, 105, 124
end, 122
environment, 444
ERB, 90, 91, 105, 124, 262, 264
 tworzenie stron internetowych, 90
 wczytywanie obiektów z pamięci, 105
error_messages, 248

F

fikstura, 443
File.delete, 441
File.readlines, 441
filtry, 179
find, 203, 206
find(:all), 116
find_all_by_nazwa_atrybutu, 210
Fixnum, 441
Float, 441
foldery, 52, 57, 79, 86
 public, 133
 test, 442
for, 122
form_for, 202, 206, 433
form_remote_for, 328
form_tag, 202, 206
formatowanie zawartości tablicy, 441
 tablice asocjacyjne, 441
formaty danych, 366
formularz dodawania danych, 143
formularz wyszukiwania, 196
formularze, 145, 151, 196, 421
 form_tag, 202
 identyfikatory, 273
 JavaScript, 323
 oparte na Ajaksie, 322, 406, 412
 oparte na modelu, 198, 202

parametry, 202
pola, 155
tworzenie, 196
wartości pól, 157
wymagane pola, 232
żądania oparte na Ajaksie, 322
fragmenty, 292, 305
functional test, 443

G

generate controller, 88
generator JavaScriptu, 334, 337
generowanie
 aplikacje internetowe, 39
 jednocześnie kodu XML oraz HTML, 363
 kanały RSS, 380
 kod, 42
 kod JavaScript, 334
 kod Ruby, 124
 kod XML, 354, 355
generowanie
 migracja, 65
GET, 430, 431, 433
Google Maps, 349
 Ajax, 354, 396
 dane mapy, 352
 klucz, 350, 351
 obsługa JavaScriptu, 416
 przesyłanie danych, 352
 szablon częściowy, 396
Google Maps, 349
 wykorzystanie kodu, 351
 żądanie map, 370
GPS, 345
graficzna strona indeksująca, 378

H

has_many, 283, 292
hash, 167
hasła, 178

href, 123
HTML, 91, 365, 368
HTTP, 160
 odpowiedzi, 160
 uwierzytelnianie, 178
 żądania, 160

I

id, 87, 106, 148
identyfikator części strony, 317
identyfikatory, 106, 145, 412
Identyfikatory, 144
if, 246, 366
indeksy tablicy, 118, 120
index, 374
index.html, 41
index.html.erb, 212
informacje o błędach, 250
INSERT, 158
inspect, 441
instalacja Ruby on Rails, 28
interakcja z systemem, 53
interfejs użytkownika, 345
interpretator języka Ruby, 50

J

JavaScript, 133, 307, 311
javascript_include_tag, 311, 396
javascripts, 133
język
 interpretowany, 41
 JavaScript, 133, 307, 311
 Ruby, 29, 65

K

kanały RSS, 379
 dodawanie kanału do stron, 389
generowanie, 380, 382
szablony XML Builder, 385
zawartość, 380
katalogi, 441

klasa kontrolera, 88
klasy, 441
klient, 446
kliknięcie myszą, 315
klucz Google Maps, 350, 351
kod, 41
kod Ajax, 308
kod generowany, 42
kod HTML, 91, 127
kod języka Ruby, 124
kod SQL, 217
kod szablonu nadziednego, 131
kod widoku, 91
kod XML, 354, 355
kody pocztowe, 441
kompilacja, 41
komunikacja z bazą danych, 156
komunikaty o błędach, 45, 233, 243
 wyświetlanie, 248
konfiguracja, 93, 94
kontrola przechodzenia między stronami, 244
kontroler, 53, 54, 85, 88, 96, 102, 137
 akcje, 165
 dane, 103
 klasa, 88
 metoda, 165
 nazwy, 89
 obsługa danych, 155
 plik klasy, 88
 przesyłanie danych do widoku, 114
 zapisywanie rekordu, 158
Konwencja ważniejsza od konfiguracji, 41, 50, 52
konwencje, 41
konwersja
 łańcuch znaków na liczbę, 441
 obiekt na łańcuch znaków, 441
 szablony stron na kod języka Ruby, 124
kopia odwróconego łańcucha znaków, 441

krawędź, 448
kryteria wyszukiwania, 214

L

layout, 130
liczby, 230
licznik, 315
 częstotliwość żądania, 316
link_to, 423, 433
link_to_remote, 309, 427, 431, 433
logika biznesowa, 54
logowanie, 179

Ł

łańcuch przekierowania, 167
łańcuch wyszukiwania, 208
łańcuchy znaków, 59, 69, 230
 tworzenie, 441

M

magiczne kolumny, 87, 89
map.resources, 372
mapowanie adresów URL, 94
mapowanie relacyjno-obiektowe, 36
mapy, 346, 349
mashup, 349
Math.sqrt, 441
metapoznanie, 25
metoda kontrolera, 144, 153, 165
metoda logowania, 179
metoda wyszukująca, 116, 209, 215, 220, 288, 292
metody HTTP, 328, 430, 431, 447
 DELETE, 433
 GET, 431, 433
 POST, 431, 433
 PUT, 433
 trasy, 430
metody pomocnicze, 69, 440
migracja, 47, 50, 64
 generowanie, 65

Skorowidz

migracja, 47, 50, 64

kod, 65

nazwy, 66

wykonywanie, 67

migration, 47

MIME, 96, 357

model, 53, 85, 137

formularze, 198

generowanie kodu XML, 356

nazwy, 89

odczytywanie danych z bazy, 103

powiązanie, 281

model, 53, 85, 137

rusztowanie, 96

sprawdzanie poprawności danych, 226

tworzenie, 86

tworzenie obiektu z surowych danych, 156

validatory, 229

Model-Widok-Kontroler, 57, 79

moduł obsługi języka JavaScript, 307

modyfikacja

kod, 192

kod XML, 358, 384

rekordy, 170

struktura bazy danych, 63

szablony, 132

wygenerowane strony, 51

mózg, 23

MVC, 57, 79

MySQL, 110, 445

N

nadawanie innej nazwy metodzie, 441

nazwa użytkownika, 178

nazwy, 47, 66

kontroler, 89

migracja, 66

model, 89

pola formularza, 146

tabele, 89

trasy oparte na architekturze REST, 425

new, 152, 156

new_record?, 421

Nie powtarzaj się, 46, 50, 130

niepoprawne dane, 224

niezgodność z architekturą REST, 447

nil, 150

notacja CamelCase, 47, 88, 96

notacja ze znakiem, 47

O

O/R mapping, 36

obiekt formularza, 151

obiekt respondera, 366

obiekty, 38, 53, 104, 106, 145, 152

atrybuty, 104

tworzenie, 152

obiekty biznesowe, 96

obiekty domenowe, 96

obowiązkowe pola, 233

obrazki, 133

obsługa błędów, 243, 252

komunikaty o błędach, 243

obsługa danych, 155

obsługa JavaScriptu, 416

odczytywanie, 42

dane, 142

dane z bazy, 103

pojedyncze rekordy, 207

rekordy pasujące do określonych kryteriów, 207

surowe dane formularza, 157

wszystkie rekordy, 207

odnośniki, 123, 126

Edit, 419

odnośniki oparte na Ajaksie, 427

odnośnik Odśwież, 309

odpowiedzi szablonu częściowego, 326

odpowiedź, 96, 160

przekierowanie, 167

Odśwież, 300, 302

odświeżanie strony, 302, 311

odświeżanie za pomocą kliknięcia, 318

odwracanie łańcucha znaków, 441

odwrotne powiązanie, 289

odwzorowanie tablic, 376

ograniczanie dostępu do funkcji, 178

okna wyskakujące, 419

operacje CRUD, 42, 194, 198, 446

Oracle, 110, 445

ORM, 36

P

parametry formularza, 202

parametry zapytania SQL, 216

params [...], 155, 157, 167

partial, 261

periodically_call_remote, 316

pętle, 122

ciało, 122

for, 122

scriptlets, 125

pierwiastek kwadratowy, 441

platforma aplikacji, 36, 38

platforma Rails, 37, 85

pliki

ERb, 262

statyczne, 133

widok, 75

pliki XML, 356

element główny, 358

pobieranie klucza Google Maps, 350

podłańcuchy, 441

pola formularza, 146, 155

połączenia, 255

połączenia stron, 260

połączenia tabel, 258

POST, 328, 430, 431, 433
 powiązanie, 280, 281, 288
 definiowanie, 283
 odwracanie, 290
 powiązanie pomiędzy polami formularza a obiektem modelu, 145
 prezentacja, 53, 346
 problemy biznesowe, 53
 programowanie, 27
 projektowanie oparte na architekturze REST, 433
 projektowanie opcji wyszukiwania, 195
 protokół HTTP, 160
 Prototype, 308, 320, 328, 416
 prototype.js, 308
 przechowywanie danych, 63
 przeciążenie serwera, 300
 przekierowanie, 166, 167
 przekierowanie żądań, 304
 przekształcanie rekordu w obiekt, 104
 przesyłanie danych
 do Google Maps, 352
 do widoku, 114
 public, 41, 133
 public/images, 133
 public/javascripts, 133
 public/stylesheets, 133
 publikacja danych z bazy, 141
 PUT, 430, 433
 puts, 204, 206

R

rails, 39, 50
 Rails, 27, 29, 33, 36, 37, 438
 RAILS_ENV, 444
 rake, 47, 50, 65, 87, 448
 rake db:migrate, 47, 50, 67
 rake routes, 424
 read, 42, 194
 redukcja obciążenia serwera, 301

rekordy, 104
 remote_form_for, 338
 render, 265, 267, 268, 271, 280, 292,
 352, 357, 371
 :text, 357
 :xml, 357
 Representational State Transfer, 446
 reprezentacje, 365
 request.xhr, 371
 respond_to, 366, 368, 416
 responder, 366
 REST, 365, 371, 393, 446
 reverse, 441
 reverse!, 441
 round, 441
 route, 93
 routes.rb, 93, 95
 routing, 155
 rozmiar tablicy, 294
 rozwinięcie kodu, 192
 RSS, 379
 Ruby, 29, 65
 Ruby on Rails, 28
 ruby script/generate, 65
 ruby script/server, 40, 50
 rusztowanie, 41, 42, 48, 84, 194, 344
 model, 96
 tworzenie, 72

S

save, 158
 scaffold, 44
 scaffolding, 42
 scriptlets, 125
 SELECT, 216
 serwer, 446
 serwer WWW, 36, 38, 40, 50
 uruchamianie, 40
 show_with_map, 357, 363
 skrypty, 38

skrypty JavaScript, 133
 sort, 441
 sortowanie tablic, 441
 split, 441
 sprawdzanie poprawności danych, 223,
 225
 komunikaty o błędach, 233, 243
 liczby, 230
 model, 226
 validatory, 227, 228, 233, 236
 wymagane pola, 232
 SQL, 158, 216
 SQLite, 38, 110, 445
 SQLite Manager, 89
 statyczna zawartość, 133
 strona indeksująca, 109, 115, 373
 strony internetowe, 81
 stylesheets, 133
 stylizacja aplikacji, 130
 submit, 407
 symbole, 59, 69, 230
 synchroniczna asynchroniczność, 332
 system bazy danych, 36
 system routingu, 155
 system sprzedaży biletów, 38
 szablony, 90, 262
 szablony części stron, 261
 szablony częściowe, 261, 262, 266, 292,
 396, 416, 420
 dane, 270
 dołączanie do szablonu strony, 267
 odpowiedzi, 326
 render, 268
 tablica miejsc, 278
 tworzenie, 266
 zmienne lokalne, 271
 szablony nadrzędne, 130
 szablony stron, 90, 105, 106, 123, 153,
 278
 dołączanie szablonu częściowego,
 267

Skorowidz

szablony stron, 90, 105, 106, 123, 153, 278
kod HTML, 91
konwersja na kod języka Ruby, 124
modyfikacja, 132
pliki, 96
szablony XML Builder, 385, 386
stosowanie, 388

Ś

ścieżki na serwerze lokalnym, 425
środowisko, 444
produkcyjne, 444
programistyczne, 444
testowe, 444

T

tabele, 46
created_at, 87
dodawanie kolumn, 64
id, 87
kolumny, 63
magiczne kolumny, 87
nazwy, 89
połączenia, 258
tworzenie, 47, 87
updated_at, 87
tablice, 117, 118, 123, 376
dostęp do elementów, 117
elementy, 118
indeksy, 118
przetwarzanie, 123
tablice asocjacyjne, 155, 167
teksty, 75
test, 442
test/fixtures, 443
test/functional, 443
test/integration, 443
test/unit, 443

testowanie, 204, 442
testy, 443

funkcjonalne, 443
integracyjne, 443
jednostkowe, 443
wysokopoziomowe, 443
text/xml, 357, 371

Time.now, 441
Time.now.year, 441
Time.now.yesterday, 381

to_s, 441
to_xml, 356, 358, 361, 371, 384
token uwierzytelniający, 311
trasy, 93, 94, 106, 113, 143, 372, 423, 429

konfiguracja, 94
metody HTTP, 430

tworzenie, 206
trasy oparte na architekturze REST, 424, 425

nazwy, 425
treść statyczna, 133

tworzenie, 42
akcje, 347
aplikacje, 39, 85
formularze, 151, 196
formularze oparte na modelu, 202
formularze powiązane z obiektami modelu, 146

kod, 44

tworzenie, 42
kod JavaScript, 337

kod XML, 356
łańcuchy znaków, 441

model, 86
obiekty, 152
obiekty z surowych danych, 156
rusztowanie, 72
scriptlety, 126
strona indeksująca, 110

strona internetowa, 90
szablony częściowe, 266

tabele, 46, 87
trasy, 206
validatory, 286
żądania Ajaksa, 404
typy MIME, 96, 357, 366
typy proste, 110

U

uaktualnianie, 42
części strony internetowej, 412
rekordy, 170
strony, 302, 314
widok, 68
udostępnianie aplikacji użytkownikom, 444
układ strony, 130, 200, 262, 292
ukrywanie danych, 276
Uniform Resource Locator, 365
unit test, 443
update, 42, 180, 194
updated_at, 87, 148
URL, 85, 94, 166, 365, 371
uruchamianie
 aplikacje, 36
 serwer WWW, 40, 50
usuwanie, 42
 pliki, 441
 rekordy, 181
uwierzytelnianie HTTP, 178

V

validate, 286, 292
validates_format_of, 235
validates_inclusion_of, 235
validates_length_of, 235
validates_numericality_of, 230
validates_presence_of, 233

validates_too_much_baggage, 286
 validates_uniqueness_of, 235
 validator, 227
 view, 53
 views/ads, 96

W

validatory, 227, 228, 233, 236, 250
 tworzenie, 286
 validates_numericality_of, 230
 validates_presence_of, 233
 validates_too_much_baggage, 286
 warstwa prezentacyjna, 53
 wartości domyślne, 274
 wartości obiektów, 125
 wartości pól formularza, 157
 wartość nil, 150
 wczytywanie
 tabele, 116
 wiersze pliku do tablicy, 441
 wdrażanie aplikacji, 444
 wiązanie pola formularza z obiektem, 146
 widok, 53, 54, 75, 85, 96
 dostęp do danych, 100
 edycja kodu HTML, 59
 szablony strony, 90
 uaktualnienie, 68
 wprowadzanie danych, 403
 wstawianie niepoprawnych danych, 226
 wybór tras, 430

wykonywanie migracji, 67
 wyłączenie obsługi JavaScriptu, 416
 wymagane pola, 232
 wymagania użytkownika, 190
 wyrażenia, 105, 123
 wyskakujące okna, 412
 wyszukiwanie, 189, 194
 baza danych, 189
 find_all_by_nazwa_atrybutu, 210
 form_tag, 202, 206
 formularze, 196
 kod SQL, 217
 kryteria wyszukiwania, 214
 łańcuch wyszukiwania, 208
 metoda wyszukująca, 209
 projektowanie, 195
 zapytania SQL, 216
 wyświetlanie
 dane, 101, 394
 komunikaty o błędach, 248
 wywołanie szablonu częściowego z układu strony, 292
 wywoływanie zdarzeń, 316

X

XHR, 320
 XML, 320, 343, 365
 kanaly RSS, 380
 XML Builder, 385
 XML HTTP Requests, 320

Z

zaokrąglanie liczb, 441
 zapisywanie
 dane, 142
 rekordy, 158
 zapytania SQL, 216
 parametry, 216
 zarządzanie danymi przestrzennymi, 345
 zasada DRY, 46
 zasoby, 365, 371, 425, 446
 zawartość statyczna, 133
 zdarzenia, 315, 353, 394, 405
 licznik, 318
 systemowe, 328
 zmiana
 baza danych, 445
 podpisy na stronach internetowych, 58
 struktura kodu XML, 384
 struktura tabeli, 79

zmienne lokalne, 270, 271, 292

Ż

żądania, 96, 160
 Ajax, 311
 GET, 433
 POST, 433
 w tle, 307, 328
 XML HTTP, 320, 328
 żądania asynchroniczne, 307, 311
 tworzenie, 404

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**