

Buduj zaawansowane i interaktywne strony WWW!

# JavaScript i jQuery

**nieoficjalny** podręcznik



David Sawyer McFarland

O'REILLY®



Tytuł oryginału: JavaScript & jQuery: The Missing Manual, Third Edition

Tłumaczenie: Piotr Rajca

ISBN: 978-83-283-0547-2

© 2015 Helion S.A.

Authorized Polish translation of the English edition of JavaScript & jQuery: The Missing Manual, 3rd Edition ISBN 9781491947074 © 2014 Sawyer McFarland Media, Inc.

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

Polish edition copyright © 2015 by Helion S.A.  
All rights reserved.

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 jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

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

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie 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](mailto:helion@helion.pl)  
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:  
<ftp://ftp.helion.pl/przyklady/jsjqn3.zip>

Drogi Czytelniku!  
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres  
<http://helion.pl/user/opinie/jsjqn3>  
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

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

# Spis treści

Nieoficjalna czołówka .....	13
Wprowadzenie .....	17
<b>Część I. Wprowadzenie do języka JavaScript .....</b>	<b>35</b>
<b>Rozdział 1. Pierwszy program w języku JavaScript .....</b>	<b>37</b>
Wprowadzenie do programowania .....	38
Czym jest program komputerowy? .....	40
Jak dodać kod JavaScript do strony? .....	40
Zewnętrzne pliki JavaScript .....	42
Pierwszy program w języku JavaScript .....	46
Dodawanie tekstu do stron .....	48
Dołączanie zewnętrznych plików JavaScript .....	49
Wykrywanie błędów .....	51
Konsola JavaScript w przeglądarce Chrome .....	52
Konsola przeglądarki Internet Explorer .....	55
Konsola JavaScript w przeglądarce Firefox .....	56
Konsola błędów w przeglądarce Safari .....	57
<b>Rozdział 2. Gramatyka języka JavaScript .....</b>	<b>59</b>
Instrukcje .....	59
Wbudowane funkcje .....	60
Typy danych .....	60
Liczby .....	61
Łańcuchy znaków .....	61
Wartości logiczne .....	62
Zmienne .....	63
Tworzenie zmiennych .....	63
Używanie zmiennych .....	66

Używanie typów danych i zmiennych .....	67
Podstawowe operacje matematyczne .....	68
Kolejność wykonywania operacji .....	69
Łączenie łańcuchów znaków .....	69
Łączenie liczb i łańcuchów znaków .....	70
Zmienianie wartości zmiennych .....	71
Przykład — używanie zmiennych do tworzenia komunikatów .....	72
Przykład — pobieranie informacji .....	74
Tablice .....	77
Tworzenie tablic .....	78
Używanie elementów tablicy .....	79
Dodawanie elementów do tablicy .....	80
Usuwanie elementów z tablicy .....	82
Przykład — zapisywanie danych na stronie za pomocą tablic .....	83
Krótką lekcja o obiektach .....	86
Komentarze .....	88
Kiedy używać komentarzy? .....	89
Komentarze w tej książce .....	90
<b>Rozdział 3. Dodawanie struktur logicznych i sterujących .....</b>	<b>93</b>
Programy reagujące inteligentnie .....	93
Podstawy instrukcji warunkowych .....	94
Uwzględnianie planu awaryjnego .....	98
Sprawdzanie kilku warunków .....	98
Bardziej skomplikowane warunki .....	102
Zagnieżdżanie instrukcji warunkowych .....	104
Wskazówki na temat pisania instrukcji warunkowych .....	104
Przykład — używanie instrukcji warunkowych .....	105
Obsługa powtarzających się zadań za pomocą pętli .....	109
Pętle while .....	109
Pętle i tablice .....	111
Pętle for .....	112
Pętle do-while .....	114
Funkcje — wielokrotne korzystanie z przydatnego kodu .....	115
Krótki przykład .....	117
Przekazywanie danych do funkcji .....	118
Pobieranie informacji z funkcji .....	120
Unikanie konfliktów między nazwami zmiennych .....	121
Przykład — prosty quiz .....	124
<b>Część II. Wprowadzenie do biblioteki jQuery .....</b>	<b>131</b>
<b>Rozdział 4. Wprowadzenie do jQuery .....</b>	<b>133</b>
Kilka słów o bibliotekach JavaScript .....	133
Jak zdobyć jQuery? .....	135
Dołączanie pliku jQuery z serwera CDN .....	137
Pobieranie pliku jQuery .....	138

Dodawanie jQuery do strony .....	139
Podstawowe informacje o modyfikowaniu stron WWW .....	142
Zrozumieć DOM .....	145
Pobieranie elementów stron na sposób jQuery .....	147
Proste selektory .....	148
Selektory zaawansowane .....	151
Filtry jQuery .....	153
Zrozumienie kolekcji jQuery .....	155
Dodawanie treści do stron .....	157
Zastępowanie i usuwanie wybranych elementów .....	160
Ustawianie i odczyt atrybutów znaczników .....	160
Klasy .....	161
Odczyt i modyfikacja właściwości CSS .....	163
Jednoczesna zmiana wielu właściwości CSS .....	164
Odczyt, ustawienia i usuwanie atrybutów HTML .....	166
Wykonanie akcji na każdym elemencie kolekcji .....	167
Funkcje anonimowe .....	168
this oraz \$(this) .....	169
Automatycznie tworzone, wyróżniane cytaty .....	171
Opis rozwiązania .....	172
Kod rozwiązania .....	172

## **Rozdział 5. Akcja i reakcja — ożywianie stron za pomocą zdarzeń .....177**

Czym są zdarzenia? .....	177
Zdarzenia związane z myszą .....	179
Zdarzenia związane z dokumentem i oknem .....	180
Zdarzenia związane z formularzami .....	181
Zdarzenia związane z klawiaturą .....	182
Obsługa zdarzeń przy użyciu jQuery .....	182
Przykład — prezentacja obsługi zdarzeń .....	185
Zdarzenia specyficzne dla biblioteki jQuery .....	190
Oczekiwanie na wczytanie kodu HTML .....	190
Umieszczanie i usuwanie wskaźnika myszy z elementu .....	192
Obiekt reprezentujący zdarzenie .....	194
Blokowanie standardowych reakcji na zdarzenia .....	195
Usuwanie zdarzeń .....	196
Zaawansowane zarządzanie zdarzeniami .....	197
Inne sposoby stosowania funkcji on() .....	199
Delegowanie zdarzeń przy użyciu funkcji on() .....	200
Przykład — jednostronicowa lista FAQ .....	204
Omówienie zadania .....	204
Tworzenie kodu .....	205

## **Rozdział 6. Animacje i efekty .....211**

Efekty biblioteki jQuery .....	211
Podstawowe wyświetlanie i ukrywanie .....	212
Wygaszanie oraz rozjaśnianie elementów .....	213
Przesuwanie elementów .....	216

Przykład — wysuwany formularz logowania .....	216
Tworzenie kodu .....	217
Animacje .....	220
Tempo animacji .....	221
Wykonywanie operacji po zakończeniu efektu .....	223
Przykład — animowany pasek ze zdjęciami .....	225
Tworzenie kodu .....	227
jQuery i przejścia oraz animacje CSS3 .....	231
jQuery i przejścia CSS .....	232
jQuery i animacje CSS .....	234
<b>Rozdział 7. Popularne zastosowania jQuery .....</b>	<b>239</b>
Zamiana rysunków .....	239
Zmienianie atrybutu src rysunków .....	240
Podmiana obrazków przy użyciu jQuery .....	241
Wstępne wczytywanie rysunków .....	242
Efekt rollover z użyciem obrazków .....	243
Przykład — dodawanie efektu rollover z użyciem rysunków .....	245
Omówienie zadania .....	245
Tworzenie kodu .....	246
Przykład — galeria fotografii z efektami wizualnymi .....	249
Omówienie zadania .....	249
Tworzenie kodu .....	251
Kontrola działania odnośników .....	255
Pobieranie odnośników w kodzie JavaScript .....	255
Określanie lokalizacji docelowej .....	255
Blokowanie domyślnego działania odnośników .....	256
Otwieranie zewnętrznych odnośników w nowym oknie .....	258
Tworzenie nowych okien .....	260
Właściwości okien .....	261
Przedstawienie wtyczek jQuery .....	265
Czego szukać we wtyczce jQuery? .....	266
Podstawy stosowania wtyczek jQuery .....	268
Responsywne menu nawigacyjne .....	270
Kod HTML .....	270
Kod CSS .....	273
Kod JavaScript .....	273
Przykład .....	273
Dostosowywanie wyglądu wtyczki SmartMenus .....	277
<b>Rozdział 8. Wzbogacanie formularzy .....</b>	<b>279</b>
Wprowadzenie do formularzy .....	279
Pobieranie elementów formularzy .....	281
Pobieranie i ustawianie wartości elementów formularzy .....	283
Sprawdzanie stanu przycisków opcji i pól wyboru .....	284
Zdarzenia związane z formularzami .....	285

Inteligentne formularze .....	290
Aktywowanie pierwszego pola formularza .....	290
Wyłączanie i włączanie pól .....	291
Ukrywanie i wyświetlanie opcji formularza .....	293
Przykład — proste wzbogacanie formularza .....	294
Aktywowanie pola .....	295
Wyłączanie pól formularza .....	295
Ukrywanie pól formularza .....	298
Walidacja formularzy .....	299
Wtyczka Validation .....	301
Podstawowa walidacja .....	302
Zaawansowana walidacja .....	305
Określanie stylu komunikatów o błędach .....	310
Przykład zastosowania walidacji .....	311
Prosta walidacja .....	312
Walidacja zaawansowana .....	313
Walidacja pól wyboru i przycisków opcji .....	316
Formatowanie komunikatów o błędach .....	319

## **Część III. Wprowadzenie do biblioteki jQuery UI ..... 321**

### **Rozdział 9. Rozbudowa interfejsu użytkownika ..... 323**

Czym jest jQuery UI? .....	323
Dlaczego warto używać jQuery UI? .....	325
Stosowanie jQuery UI .....	327
Dodawanie jQuery UI do strony .....	329
Wyświetlanie komunikatów przy użyciu okien dialogowych .....	330
Miniprzykład — tworzenie okna dialogowego .....	332
Określanie właściwości okna dialogowego .....	333
Miniprzykład — przekazywanie opcji do okna dialogowego .....	336
Otwieranie okna dialogowego w odpowiedzi na zdarzenia .....	338
Dodawanie przycisków do okien dialogowych .....	339
Miniprzykład — dodawanie przycisków do okien dialogowych .....	341
Prezentowanie informacji w etykietkach ekranowych .....	345
Miniprzykład — szybkie dodawanie etykietek ekranowych .....	347
Opcje etykietek ekranowych .....	348
Umieszczanie w etykietkach treści HTML .....	349
Miniprzykład — umieszczanie kodu HTML w etykietkach ekranowych ..	350
Dodawanie zestawów kart .....	351
Opcje zestawów kart .....	354
Miniprzykład — dodawanie zestawu kart .....	356
Karty prezentujące zawartość .....	360
Oszczędzanie miejsca z wykorzystaniem akordeonów .....	363
Miniprzykład — tworzenie akordeonu jQuery UI .....	366
Dodawanie menu .....	368
Tworzenie poziomego paska nawigacyjnego .....	371

<b>Rozdział 10. Formularze raz jeszcze .....</b>	<b>375</b>
Wybieranie dat ze stylem .....	375
Określanie właściwości kalendarzy .....	377
Przykład — pole do wyboru daty urodzenia .....	381
Stylowe rozwijane listy .....	383
Określanie właściwości list rozwijanych .....	385
Wykonywanie operacji po wybraniu opcji z listy .....	386
Stylowe przyciski .....	389
Dostosowywanie przycisków .....	390
Poprawianie wyglądu przycisków opcji i pól wyboru .....	391
Dostarczanie podpowiedzi przy użyciu automatycznego uzupełniania .....	393
Generowanie podpowiedzi przy użyciu tablicy danych .....	395
Stosowanie osobnych etykiet i wartości .....	397
Pobieranie danych automatycznego uzupełniania z serwera .....	398
Opcje widżetu Autocomplete .....	400
Przykład — widżety UI usprawniające formularze .....	401
<b>Rozdział 11. Dostosowywanie wyglądu jQuery UI .....</b>	<b>407</b>
Prezentacja narzędzia ThemeRoller .....	407
Pobieranie i stosowanie nowego tematu .....	413
Dodawanie własnego tematu do istniejących stron WWW .....	414
Więcej informacji o arkuszach stylów jQuery UI .....	415
Przesłanie stylów jQuery UI .....	415
Zasada szczegółowości .....	416
Jak są określane style widżetów jQuery UI? .....	418
<b>Rozdział 12. Interakcje i efekty jQuery UI .....</b>	<b>421</b>
Widżet Draggable .....	421
Dodawanie widżetu Draggable do strony .....	422
Miniprzykład — zastosowanie widżetu Draggable .....	423
Opcje widżetu Draggable .....	424
Zdarzenia widżetu Draggable .....	430
Widżet Droppable .....	434
Stosowanie widżetu Droppable .....	435
Opcje widżetu Droppable .....	436
Zdarzenia widżetu Droppable .....	438
Przykład — technika „przeciągnij i upuść” .....	443
Sortowanie elementów strony .....	449
Stosowanie widżetu Sortable .....	449
Opcje widżetu Sortable .....	451
Zdarzenia widżetu Sortable .....	455
Metody widżetów Sortable .....	458
Efekty jQuery UI .....	461
Efekty .....	462
Tempo animacji .....	465
Animowanie zmiany klas .....	466



## Część IV. Zaawansowane zastosowania jQuery i języka JavaScript .....469

### Rozdział 13. Wprowadzenie do technologii AJAX ..... 471

Czym jest AJAX? .....	471
AJAX — podstawy .....	473
Elementy układanki .....	474
Komunikacja z serwerem WWW .....	476
AJAX w bibliotece jQuery .....	479
Używanie metody load() .....	480
Przykład — korzystanie z metody load() .....	482
Metody get() i post() .....	486
Formatowanie danych przesyłanych na serwer .....	487
Przetwarzanie danych zwróconych z serwera .....	490
Obsługa błędów .....	494
Przykład — korzystanie z metody \$.get() .....	495
Format JSON .....	500
Dostęp do danych z obiektów JSON .....	502
Złożone dane JSON .....	503
Prezentacja JSONP .....	506
Dodawanie do witryny kanału Flickr .....	506
Tworzenie adresu URL .....	508
Łączenie opcji .....	510
Stosowanie metody \$.getJSON() .....	510
Prezentacja danych kanału Flickr w formacie JSON .....	511
Przykład — dodawanie zdjęć z Flickr na własnej stronie .....	512

### Rozdział 14. Tworzenie aplikacji do obsługi listy zadań ..... 519

Przegląd aplikacji .....	519
Dodanie przycisku .....	520
Dodanie okna dialogowego .....	522
Dodawanie zadań .....	525
Oznaczanie zadania jako wykonanego .....	531
Delegowanie zdarzeń .....	531
Usuwanie zadań .....	536
Dalsze kroki .....	538
Edycja zadań .....	538
Potwierdzanie usunięcia .....	539
Zapisywanie listy .....	539
Inne pomysły .....	540



## Część V. Wskazówki, sztuczki i rozwiązywanie problemów ... 541

<b>Rozdział 15. Wykorzystywanie wszystkich możliwości jQuery .....</b>	<b>543</b>
Przydatne informacje i sztuczki związane z jQuery .....	543
\$( ) to to samo, co jQuery( ) .....	543
Zapisywanie pobranych elementów w zmiennych .....	544
Jak najrzadsze dodawanie treści .....	546
Optymalizacja selektorów .....	547
Korzystanie z dokumentacji jQuery .....	548
Czytanie dokumentacji na stronie jQuery .....	552
Poruszanie się po DOM .....	554
Inne funkcje do manipulacji kodem HTML .....	560
<b>Rozdział 16. Zaawansowane techniki języka JavaScript .....</b>	<b>565</b>
Stosowanie łańcuchów znaków .....	565
Określanie długości łańcucha .....	566
Zmiana wielkości znaków w łańcuchu .....	566
Przeszukiwanie łańcuchów znaków: zastosowanie indexOf( ) .....	567
Pobieranie fragmentu łańcucha przy użyciu metody slice( ) .....	569
Odnajdywanie wzorów w łańcuchach .....	570
Tworzenie i stosowanie podstawowych wyrażeń regularnych .....	571
Tworzenie wyrażeń regularnych .....	572
Grupowanie fragmentów wzorców .....	576
Przydatne wyrażenia regularne .....	577
Dopasowywanie wzorców .....	582
Zastępowanie tekstów .....	585
Testowanie wyrażeń regularnych .....	585
Stosowanie liczb .....	587
Zamiana łańcucha znaków na liczbę .....	587
Sprawdzanie występowania liczb .....	589
Zaokrąglanie liczb .....	589
Formatowanie wartości monetarnych .....	590
Tworzenie liczb losowych .....	591
Daty i godziny .....	592
Pobieranie miesiąca .....	593
Określanie dnia tygodnia .....	594
Pobieranie czasu .....	594
Tworzenie daty innej niż bieżąca .....	597
Tworzenie bardziej wydajnego kodu JavaScript .....	599
Zapisywanie ustawień w zmiennych .....	600
Zapisywanie ustawień w obiektach .....	601
Operator trójargumentowy .....	602
Instrukcja Switch .....	603
Łączenie tablic i dzielenie łańcuchów znaków .....	605
Łączenie różnych elementów .....	606
Używanie zewnętrznych plików JavaScript .....	606
Tworzenie kodu JavaScript o krótkim czasie wczytywania .....	609

<b>Rozdział 17. Diagnozowanie i rozwiązywanie problemów .....</b>	<b>611</b>
Najczęstsze błędy w kodzie JavaScript .....	611
Brak symboli końcowych .....	612
Cudzysłowy i apostrofy .....	616
Używanie słów zarezerwowanych .....	617
Pojedynczy znak równości w instrukcjach warunkowych .....	617
Wielkość znaków .....	618
Nieprawidłowe ścieżki do zewnętrznych plików JavaScript .....	618
Nieprawidłowe ścieżki w zewnętrznych plikach JavaScript .....	619
Znikające zmienne i funkcje .....	620
Testowanie aplikacji przy użyciu konsoli .....	621
Otwieranie konsoli .....	621
Przeglądanie błędów przy użyciu konsoli .....	623
Śledzenie działania skryptu za pomocą funkcji console.log() .....	623
Przykład — korzystanie z konsoli .....	624
Diagnozowanie zaawansowane .....	628
Przykład diagnozowania .....	633
<b>Część VI. Dodatki .....</b>	<b>641</b>
<b>Dodatek A. Materiały związane z językiem JavaScript .....</b>	<b>643</b>
Źródła informacji .....	643
Witryny .....	643
Książki .....	644
Podstawy języka JavaScript .....	644
Witryny .....	644
Książki .....	644
jQuery .....	645
Witryny .....	645
Książki .....	645
Zaawansowany język JavaScript .....	645
Artykuły i prezentacje .....	646
Witryny .....	646
Książki .....	646
CSS .....	647
Witryny .....	647
Książki .....	647
<b>Skorowidz .....</b>	<b>649</b>



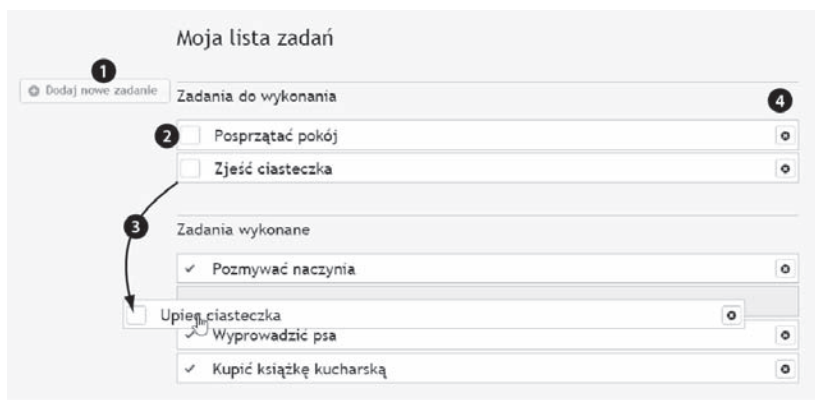
# Tworzenie aplikacji do obsługi listy zadań

Biblioteki jQuery i jQuery UI udostępniają narzędzia do tworzenia zaledwie w kilku prostych krokach profesjonalnie wyglądających aplikacji internetowych. Pierwsza z nich może zadbać o wybieranie elementów strony, dodawanie do niej nowych elementów oraz aktualizowanie DOM. Z kolei biblioteka jQuery UI udostępnia wspaniale wyglądające widżety, sposoby interakcji z użytkownikiem oraz efekty animacji, rozwiązując tym samym wiele najczęściej występujących problemów związanych z opracowaniem interfejsu użytkownika. Korzystając z obu tych bibliotek, można uniknąć żmudnych oraz czasochłonnych zadań i skoncentrować się wyłącznie na tworzeniu dynamicznej, interaktywnej aplikacji. W tym rozdziale przedstawiony został proces pisania prostej aplikacji do zarządzania listą zadań.

## Przegląd aplikacji

Lista zadań, którą napiszemy w tym rozdziale, będzie pozwalała użytkownikom na wykonywanie następujących operacji.

- **Dodawanie nowych zadań do listy.** W tym celu dodamy do niej widżet przycisku jQuery UI (nr 1 na rysunku 14.1), a następnie po kliknięciu wyświetlimy okno dialogowe jQuery UI.
- **Oznaczanie zadania jako wykonanego.** Każde zadanie na liście będzie mieć pole wyboru z lewej strony (numer 2 na rysunku 14.1). Kliknięcie tego pola będzie automatycznie usuwać zadanie z listy *Zadania do wykonania* i przenieść na listę *Zadania wykonane*.
- **Przeciąganie zadań z listy *Zadania do wykonania* na listę *Zadania wykonane* i na odwrót** (numer 3 na rysunku 14.1). Choć klikanie pola wyboru niewątpliwie spełni swoją rolę, to niby czemu mamy się ograniczać do tylko jednego sposobu oznaczania zadań jako wykonanych? Korzystając z widżetu



**Rysunek 14.1.** Biblioteki jQuery oraz jQuery UI sprawiają, że napisanie prostej listy zadań do zrobienia jest relatywnie łatwe. Pozwalają one na rozwiązywanie złożonych problemów związanych z tworzeniem interfejsów użytkownika, takich jak opracowanie okien dialogowych, list z możliwością sortowania oraz animowanie wybranych elementów stron. Dzięki nim będziemy mogli skoncentrować się wyłącznie na zaimplementowaniu możliwości funkcjonalnych aplikacji

Sortable jQuery UI, możemy także pozwolić użytkownikom na oznaczanie zdań jako wykonanych poprzez przeciąganie ich na listę *Zadania wykonane*. Co więcej, możemy także pozwolić na przeciąganie zadań z *powrotem* na listę *Zadania do wykonania*.

- **Usuwanie zadań z listy po kliknięciu przycisku *Usuń*** (numer 4 na rysunku 14.1). Aby zapewnić użytkownikom możliwość całkowitego usunięcia zadania, dodamy przycisk *Usuń*, którego kliknięcie całkowicie usunie zadanie ze strony. Zastosujemy przy tym jeden z efektów jQuery UI, by cała operacja wyglądała interesująco.

Ponieważ biblioteka jQuery UI udostępnia większość widżetów i mechanizmów interakcji, pozostaje jedynie zaimplementowanie podstawowej logiki działania aplikacji. Zadanie to wykonamy krok po kroku, zgodnie z przedstawionym powyżej planem. Zaczniemy od dodania przycisku, a następnie dokończymy pozostałą część aplikacji. Zadanie programistyczne podczas rozwiązywania warto podzielić na mniejsze elementy, które można łatwo przetestować. Podobnie będzie w tym przykładzie — najpierw zrobimy jedną rzecz, upewnimy się, że działa prawidłowo, i dopiero potem przejdziemy do kolejnej.

## Dodanie przycisku

Najpierw dodamy do strony przycisk i sformatujemy go przy użyciu jQuery UI. W tym przykładzie będziemy pracować nad dwoma plikami dostępnymi w katalogu *R14*; są to *index.html* oraz *todo.js*. Cały kod JavaScript będzie umieszczany w pliku *todo.js*, natomiast kod HTML niezbędny do utworzenia przycisku oraz okna dialogowego trafi do pliku *index.html*.

**Uwaga:** Informacje na temat pobierania przykładów do książki można znaleźć na stronie 46.

### 1. Otwórz w edytorze tekstu plik *index.html* dostępny w katalogu *R14*.

Plik zawiera już odwołania do potrzebnych plików CSS, jQuery oraz jQuery UI. Jednak nie ma w nim jeszcze odwołania do skryptu *todo.js* — czyli pliku, w którym zapiszesz kod JavaScript tworzonej aplikacji.

### 2. W wierszu bezpośrednio poniżej `<script src="js/jquery-ui.min.js"></script>` dodaj następujący wiersz kodu :

```
<script src="todo.js"></script>
```

Bardzo ważne jest to, by plik *todo.js* został dołączony do strony jako ostatni, gdyż wymaga on zarówno biblioteki jQuery, jak i jQuery UI. Jeśli wczytasz go przed którąkolwiek z tych bibliotek, podczas wyświetlania strony przeglądarka zgłosi błąd syntaktyczny.

Teraz zajmiesz się dodaniem przycisku. Użytkownik będzie mógł go kliknąć w celu dodania do listy nowego zadania.

### 3. W treści pliku odszukaj komentarz `<!-- Tu dodaj przycisk. -->` i zastąp go poniższym wierszem kodu HTML:

```
<button id="add-todo">Dodaj nowe zadanie</button>
```

W tym kodzie nie ma nic szczególnego — dodajesz do strony zwyczajny element `<button>`. Gdybyś teraz wyświetlił stronę w przeglądarce, przekonałbyś się, że wygląda on jak zwyczajny przycisk do wysyłania formularza — bezbarwny i nieinteresujący. Już zaraz przekształcisz go w przycisk jQuery UI.

### 4. W edytorze tekstów otwórz plik *todo.js*, a następnie wewnątrz funkcji `$(document).ready(function(e) {` wpisz następujący wiersz kodu:

```
$("#add-todo").button();
```

Powyższy kod spowoduje zastosowanie w przycisku formatowania określonego przez aktualnie używany temat graficzny jQuery UI (patrz strona 407). Możesz go dodatkowo uatrakcyjnić, dodając do niego ikonę jQuery UI.

### 5. Wewnątrz funkcji `button()` dodaj literał obiektowy definiujący ikonę, która będzie wyświetlona na przycisku (kod literału został wyróżniony pogrubioną czcionką):

```
$("#add-todo").button({
  icons: {
    primary: "ui-icon-circle-plus"
  }
});
```

Powyższy kod umieści z lewej strony napisu widocznego na przycisku niewielką ikonę ze znakiem „+”. Zgodnie z informacjami podanymi na stronie 390, jQuery UI pozwala na wyświetlanie na przyciskach dwóch ikon: jednej po lewej stronie (ikona główna — `primary`), a drugiej po prawej (ikona pomocnicza — `secondary`). W przypadku tego przycisku jedna ikona w zupełności wystarczy.

### 6. Zapisz oba pliki, *todo.js* oraz *index.html*, a następnie wyświetl stronę *index.html* w przeglądarce.

Teraz przycisk powinien już wyglądać tak samo jak widżet przycisku jQuery UI widoczny na rysunku 14.1. Jeśli tak nie wygląda, oznacza to, że przygotowany kod JavaScript nie działa. Dokładnie go sprawdź, używając konsoli JavaScript, by upewnić się, czy nie ma komunikatów o błędach.

## Dodanie okna dialogowego

A zatem mamy już przycisk, który jednak nic nie robi. Docelowo kliknięcie tego przycisku powinno powodować wyświetlenie okna dialogowego — jego zaimplementowanie będzie Twoim kolejnym zadaniem. Najpierw dodasz kod HTML okna dialogowego.

1. W pliku *index.html* odzyskaj komentarz `<!-- Tutaj dodaj okienko dialogowe. -->` i zastąp go poniższym kodem HTML:

```
<div id="new-todo" title="Dodaj zadanie">
  <form>
    <p>
      <label for="task">Nazwa zadania:</label>
      <input type="text" name="task" id="task">
    </p>
  </form>
</div>
```

Zgodnie z tym, czego dowiedziałeś się na stronie 331, w okno dialogowe możesz zamienić dowolny fragment kodu HTML. Tu użyjemy elementu `<div>`, wewnątrz którego znajduje się formularz z jednym polem tekstowym. Podczas dodawania zadania użytkownicy będą wpisywali w tym polu nazwę zadania.

Aby przekształcić ten kod HTML w okno dialogowe, będziesz musiał użyć kodu JavaScript.

2. Wróć do pliku *todo.js*. Poniżej kodu, który dodałeś w kroku 5. na stronie 521, wpisz:

```
$("#new-todo").dialog();
```

Zapisz plik *index.html* i odśwież stronę w przeglądarce. Powinno się na niej pojawić okno dialogowe (patrz górny zrzut na rysunku 14.2). Jednak jest ono wyświetlane od razu, a nie po kliknięciu przycisku. To standardowe zachowanie okien dialogowych tworzonych przy użyciu biblioteki jQuery UI. Aby ukryć to okno, będziesz musiał przekazać w wywołaniu funkcji `dialog()` odpowiednie opcje.

3. W wywołaniu funkcji `dialog()` umieść obiekt zawierający dwie właściwości:

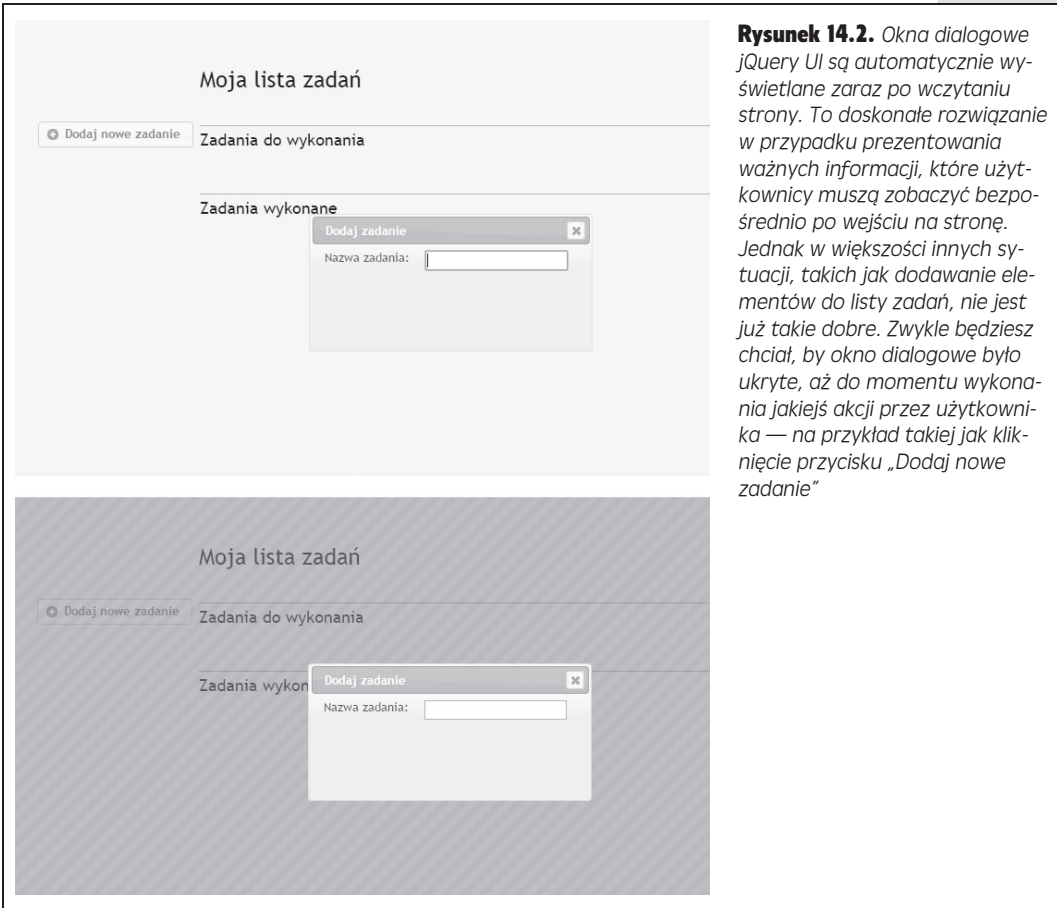
```
$("#new-todo").dialog({
  modal : true,
  autoOpen : false
});
```

Zgodnie z tym, czego się dowiedziałeś na stronie 335, opcja `modal` zmusza użytkownika do zamknięcia okna dialogowego, zanim będzie mógł wykonać na stronie jakąkolwiek inną operację. I właśnie o to chodzi — kiedy użytkownik kliknie przycisk *Dodaj nowe zadanie*, wypełnienie okna dialogowego powinno być jedyną rzeczą, na której użytkownik ma się skoncentrować.

Z kolei przypisanie właściwości `autoOpen` wartości `false` sprawi, że okno dialogowe nie będzie już wyświetlane natychmiast po wczytaniu strony. Teraz będzie początkowo ukryte i trzeba wyświetlić je programowo, w odpowiedzi na kliknięcie przycisku!



**Rysunek 14.2.** Okna dialogowe jQuery UI są automatycznie wyświetlane zaraz po wczytaniu strony. To doskonale rozwiązanie w przypadku prezentowania ważnych informacji, które użytkownicy muszą zobaczyć bezpośrednio po wejściu na stronę. Jednak w większości innych sytuacji, takich jak dodawanie elementów do listy zadań, nie jest już takie dobre. Zwykle będziesz chciał, by okno dialogowe było ukryte, aż do momentu wykonania jakiejś akcji przez użytkownika — na przykład takiej jak kliknięcie przycisku „Dodaj nowe zadanie”



#### 4. Dodaj funkcję obsługującą zdarzenia click, dopisując do wywołania funkcji `.button()` wywołanie `.click()`:

```
$("#add-todo").button({
  icons: {
    primary: "ui-icon-circle-plus"
  }
}).click(function() {
  $('#new-todo').dialog('open');
});
```

Wywołania funkcji jQuery można łączyć w sekwencje — w tym celu na końcu jednej z nich wystarczy dopisać kropkę, a za nią umieścić wywołanie kolejnej funkcji. W tym przypadku kod najpierw wybiera element o identyfikatorze `add-todo` (czyli przycisk), potem wywołuje funkcję jQuery UI `button()` i w końcu dodaje do przycisku funkcję, która będzie obsługiwać zdarzenia `click`. Dodana funkcja obsługująca zdarzenia wywołuje z kolei funkcję `open()` okna dialogowego (patrz strona 338). Innymi słowy, kliknięcie przycisku powinno teraz powodować wyświetlenie okna dialogowego.

Nadszedł czas, żeby przetestować działanie kodu.

5. Zapisz plik *todo.js*, po czym wyświetl w przeglądarce stronę *index.html*. Kliknij przycisk *Dodaj nowe zadanie*.

Teraz okno dialogowe powinno być wyświetlane wyłącznie po kliknięciu przycisku (u dołu rysunku 14.2). Co więcej, cały obszar poniżej okna dialogowego powinien zostać nieco zaciemniony — przykryty przez prążkowaną, półprzezroczystą warstwę. To wizualne oznaczenie informuje użytkownika, że zostało wyświetlone *modalne* okno dialogowe, co z kolei znaczy, że dopóki go nie zamknie, nie będzie mógł zrobić nic innego. Jednak aktualnie nie ma jak zamknąć tego okna dialogowego!

Na szczęście za pomocą jQuery UI dodawanie przycisków do okien dialogowych jest bardzo proste.

6. Wróć do pliku *todo.js*. Do listy opcji okna dialogowego dodaj kolejną — *buttons*, jej wartością ma być pusty literał obiektowy:

```
$("#new-todo").dialog({
  modal : true,
  autoOpen : false,
  buttons : {

  }
});
```

Opcja *buttons* pozwala określać przyciski, które jQuery UI dynamicznie doda do okna dialogowego. Oprócz tego, do każdego z nich można dodać funkcję, dzięki czemu coś się stanie, kiedy użytkownik kliknie dany przycisk. Tworzymy kod aplikacji wolno, krok po kroku, ponieważ dzieje się w nim całkiem dużo. Teraz na przykład dysponujesz nowym obiektem (właściwością *buttons*) umieszczonym wewnątrz innego obiektu (literału obiektowego z opcjami, przekazywanego w wywołaniu funkcji *dialog()*).

Najpierw dodaj przycisk z pustą funkcją.

7. Do literału obiektowego wpisanego w poprzednim kroku dodaj nową właściwość (wyróżnioną pogrubioną czcionką):

```
$("#new-todo").dialog({
  modal : true,
  autoOpen : false,
  buttons : {
    "Dodaj zadanie" : function () {

    }
  }
});
```

Ten kod powoduje dodanie do okna dialogowego przycisku z napisem *Dodaj zadanie*. Kiedy użytkownik kliknie ten przycisk, zostanie wykonana funkcja. Obecnie funkcja jest pusta, więc nic się nie stanie. Funkcję tę zaimplementujesz do końca w następnej części przykładu. A teraz dodasz do okna dialogowego drugi przycisk.

8. Za zamykającym nawiasem klamrowym funkcji umieszczonej we właściwości "Dodaj zadanie" wpisz przecinek i dodaj kolejną właściwość z funkcją anonimową:

```
$("#new-todo").dialog({
  modal : true,
  autoOpen : false,
```

```

buttons : {
  "Dodaj zadanie" : function () {
  },
  "Anuluj" : function () {
    $(this).dialog('close');
  }
}
});

```

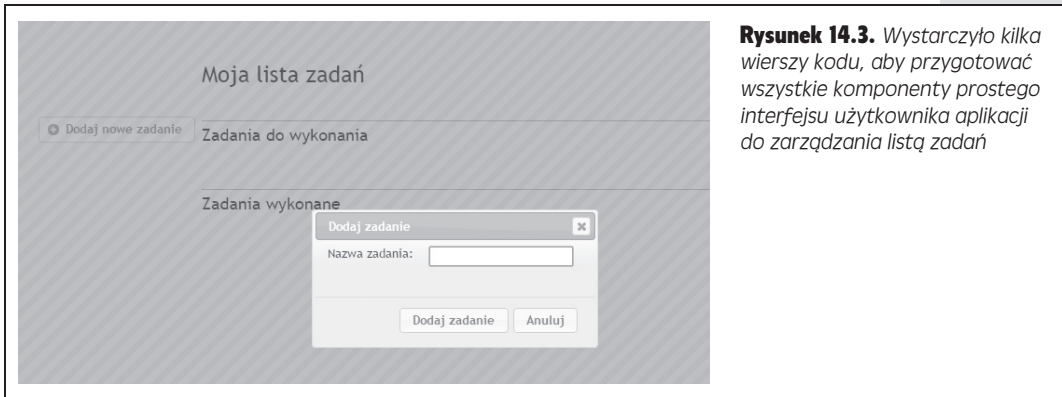
Ten nowy fragment kodu dodaje do okna dialogowego przycisk *Anuluj*. Co więcej, przycisk już coś robi. Wyrażenie `$(this)` odwołuje się do elementu, który wywołał funkcję, czyli do samego okna dialogowego. Kiedy użytkownik kliknie przycisk *Anuluj*, zostaje wywołana funkcja `close()` widżetu okna dialogowego (patrz strona 340). Powoduje ona natychmiastowe zamknięcie tego okna.

Nadszedł czas, by zobaczyć efekt wykonanej pracy.

### 9. Zapisz plik *todo.js* i odśwież stronę *index.html* w przeglądarce. Kliknij przycisk *Dodaj nowe zadanie*.

Okno dialogowe powinno teraz zawierać dwa przyciski (patrz rysunek 14.3). Kliknij przycisk *Dodaj zadanie* — oczywiście nic się nie powinno stać, gdyż obsługa przycisku nie została jeszcze zaimplementowana. Kiedy jednak klikniesz drugi przycisk, czyli *Anuluj*, okno powinno zostać zamknięte.

W następnej części przykładu zajmiesz się pisaniem kodu, który pozwoli na dodawanie zadań do listy.



**Rysunek 14.3.** Wystarczyło kilka wierszy kodu, aby przygotować wszystkie komponenty prostego interfejsu użytkownika aplikacji do zarządzania listą zadań

## Dodawanie zadań

Nasza aplikacja do zarządzania listą zadań wygląda już całkiem dobrze, jednak nie dzieje się w niej zbyt dużo. Nadszedł zatem czas, byś zajął się kodem umożliwiającym dodawanie zadań. Zostanie on umieszczony w funkcji skojarzonej z przyciskiem *Dodaj zadanie*, czyli w pustej funkcji anonimowej, którą dopisałeś w kroku 5. na stronie 524. Implementację tego kodu wykonasz w opisanych poniżej czterech krokach.

### 1. Pobierz nazwę zadania, którą użytkownik wpisał w polu tekstowym w oknie dialogowym.

Wartość ta jest przechowywana w polu tekstowym, a to oznacza, że możesz ją łatwo pobrać, używając funkcji `val()` jQuery (patrz strona 283).

### 2. Utwórz element `<li>`, który następnie dodasz do treści strony.

Każde zadanie będzie reprezentowane przez jeden element listy wypunktowanej. Poniżej przedstawiona została struktura kodu HTML takiego elementu:

```
<li>
  <span class="done">%</span>
  <span class="delete">x</span>
  <span class="task">Upiec ciasteczka</span>
</li>
```

Pierwszy element `<span>` reprezentuje obszar, który użytkownik może kliknąć, by oznaczyć zadanie jako wykonane. Drugi element `<span>` reprezentuje przycisk do usunięcia zadania. I w końcu ostatni element `<span>` zawiera nazwę zadania podaną przez użytkownika w oknie dialogowym. Taki kod HTML możesz skonstruować, łącząc ze sobą literały łańcuchowe zawierające odpowiednie znaczniki i dodając do nich wpisaną przez użytkownika nazwę zadania.

---

**Uwaga:** Postać obu przycisków określisz, stosując odpowiednie style CSS. Dzięki temu znak procenta (%) oraz x umieszczone w kodzie po wyświetleniu strony zostaną zamienione odpowiednio na znacznik charakterystyczny dla pól wyboru oraz przycisk z ikoną krzyżyka. Tajemnica tych przycisków tkwi w zastosowaniu specjalnej czcionki, w której zamiast liter są ikony.

---

### 3. Dodaj do listy zadań do wykonania nowy element listy, znacznik `<li>`.

Biblioteka jQuery sprawia, że bardzo łatwo można zamienić łańcuch znaków na element DOM, a następnie dodać go do strony (szczegółowe informacje na ten temat można znaleźć w rozdziale 4.). Ponieważ dysponujemy potężnym zestawem efektów jQuery UI, możesz skorzystać z nich, by dodawać nowe zadania w sposób atrakcyjny wizualnie.

### 4. Zamknij okno dialogowe.

Ta operacja jest bardzo prosta. Wykonałeś ją już wcześniej, przy okazji implementowania przycisku do zamykania okna dialogowego, w kroku 8. opisanym na stronie 524.

Skoro już znasz podstawowe kroki, jakie należy zaimplementować, czas zabrać się za pisanie kodu. Najpierw pobierzesz dane wpisane przez użytkownika, czyli nazwę zadania wpisaną w oknie dialogowym.

#### 1. Odszukaj funkcję skojarzoną z przyciskiem *Dodaj zadanie* okna dialogowego (została ona podana we właściwości `buttons` literału obiektowego przekazywanego w wywołaniu funkcji `dialog()`), a następnie wpisz w niej poniższy kod:

```
"Dodaj zadanie" : function () {
  var taskName = $('#task').val();
},
```

Powyższa instrukcja tworzy nową zmienną, `taskName`, i zapisuje w niej wartość pola tekstowego umieszczonego w oknie dialogowym. Pamiętaj, że struktura kodu tworzącego okno dialogowe, została ona przedstawiona w kroku 1.

na stronie 522. Kod ten zawiera między innymi pole tekstowe o określonym identyfikatorze. A zatem wywołanie `$('#task').val()` pobiera wartość tego pola — czyli to, co użytkownik wpisał jako nazwę zadania.

Istnieje także możliwość, że użytkownik kliknie przycisk *Dodaj zadanie*, mimo że pole tekstowe będzie puste. Ponieważ nie chcemy dodawać do naszej listy pustych zadań, zatem powinieneś się upewnić, że wartość zmiennej `taskName` będzie inna od pustego łańcucha znaków.

**2. Poniżej wiersza kodu dodanego w poprzednim kroku wpisz trzy kolejne (wyróżnione pogrubioną czcionką):**

```
"Dodaj zadanie" : function () {
    var taskName = $('#task').val();
    if (taskName === '') {
        return false;
    }
},
```

Ten kod jedynie upewnia się, że zmienna `taskName` zawiera coś innego niż pusty łańcuch znaków (został on zapisany przy użyciu dwóch znaków apostrofu, z których pierwszy oznacza początek łańcucha, a drugi jego koniec), a użytkownik nie kliknął przycisku *Dodaj zadanie* bez podania tytułu zadania. Instrukcja `return false` powoduje zakończenie funkcji, co z kolei sprawia, że okno dialogowe pozostanie widoczne. A zatem użytkownik musi wpisać nazwę zadania lub zamknąć okno dialogowe (co może także zrobić, klikając niewielki przycisk „X” widoczny w jego prawym, górnym rogu).

Teraz zajmiesz się tworzeniem kodu HTML reprezentującego zadanie.

**3. Dodaj kolejną zmienną, a następnie skonstruuj i zapisz w niej łańcuch znaków; do tego celu posłużą trzy kolejne wiersze kodu (wyróżnione pogrubioną czcionką):**

```
"Dodaj zadanie" : function () {
    var taskName = $('#task').val();
    if (taskName === '') {
        return false;
    }
    var taskHTML = '<li><span class="done">%%</span>';
    taskHTML += '<span class="delete">x</span>';
    taskHTML += '<span class="task"></span></li>';
},
```

Do utworzenia zmiennej `taskHTML` i zapisania w niej całego, długiego łańcucha znaków wystarczyłby jeden wiersz kodu. Jednak w ten sposób powstałaby instrukcja, którą trudno przeanalizować. Podzielenie długiego łańcucha znaków na części i zapisanie go w kilku kolejnych wierszach pozwala poprawić czytelność kodu. Operator `+=` służy do dołączenia podanego łańcucha znaków do łańcucha, który już jest zapisany w zmiennej (patrz strona 72).

Zauważ, że jeszcze nie dodałeś do tego łańcucha nazwy zadania podanej przez użytkownika i przechowywanej w zmiennej `taskName`. Zrobisz to już za chwilę.

**4. Dodaj kolejną zmienną i zapisz w niej obiekt jQuery zawierający obiekt utworzony na podstawie kodu HTML (nowy kod został wyróżniony pogrubioną czcionką):**

```
"Dodaj zadanie" : function () {
    var taskName = $('#task').val();
    if (taskName === '') {
```

```

    return false;
}
var taskHTML = '<li><span class="done">%</span>';
taskHTML += '<span class="delete">x</span>';
taskHTML += '<span class="task"></span></li>';
var $newTask = $(taskHTML);
},

```

Biblioteka jQuery umożliwia przekształcenie łańcucha znaków, takiego jak "`<h1>To jest nagłówek</h1>`", na element DOM. Innymi słowy, `taskHTML` jest zmienną zawierającą łańcuch znaków. Łańcuch ten nie jest jednak „prawdziwym” kodem HTML; natomiast przekazanie łańcucha zawierającego znaczniki HTML w wywołaniu funkcji `$()` spowoduje przekształcenie go na element DOM. A nawet jeszcze lepiej — funkcja `$()` przekształca łańcuch znaków na obiekt jQuery, dzięki czemu można na nim wywoływać standardowe funkcje tej biblioteki. Choć znak `$` na początku zmiennej `$newTask` nie jest konieczny, jednak umieszczanie go na początku nazw zmiennych stanowi praktykę powszechnie stosowaną przez programistów używających jQuery. Ten znak dolara stanowi wizualną podpowiedź informującą, że dana zmienna zawiera obiekt jQuery i można jej używać do wywoływania wszystkich metod biblioteki, takich jak `.show()` czy też `.addClass()`.

W następnym kroku dodasz nazwę zadania podaną przez użytkownika.

#### 5. Do kodu funkcji obsługującej przycisk *Dodaj zadanie* dopisz kolejny wiersz:

```

"Dodaj zadanie" : function () {
    var taskName = $('#task').val();
    if (taskName === '') {
        return false;
    }
    var taskHTML = '<li><span class="done">%</span>';
    taskHTML += '<span class="delete">x</span>';
    taskHTML += '<span class="task"></span></li>';
    var $newTask = $(taskHTML);
    $newTask.find('.task').text(taskName);
},

```

Metoda `find()` poszukuje elementu odpowiadającego podanemu selektorowi *wewnątrz* aktualnie wybranego elementu (patrz strona 555). W tym przypadku metoda operuje na elemencie `<li>` przechowywanym w zmiennej `$newTask` i poszukuje w nim innego elementu, należącego do klasy `task` — czyli elementu `<span>`, w którym powinna zostać zapisana nazwa zadania (patrz krok 3.). Następnie wywołanie funkcji `text()` zapisuje zawartość zmiennej `taskName` w odnalezionym elemencie.

Ale dlaczego masz zadawać sobie tyle trudu, zamiast po prostu dodać wartość zmiennej `taskName` do łańcucha znaków tworzonego w kroku 4. w następujący sposób:

```
taskHTML += '<span class="task">' + taskName + '</span></li>';
```

Gdybyś postąpił w ten sposób, złośliwy użytkownik mógłby utworzyć zadanie o następującej nazwie: `<script>alert('ha, ha, ha... włamałem się do tej listy');</script>`. Ten złośliwy kod zostałby dodany bezpośrednio do strony i wykonany. Natomiast funkcja `text()` jQuery zamienia wszystkie znaczniki HTML na ich bezpieczne odpowiedniki, czyli zamienia `<script>` na `&lt;script&gt;`.

Jeśli nawet użytkownik nie ma żadnych złych intencji, takie rozwiązanie pozwoli mu wpisać w nazwie zadania całkowicie prawidłowy tekst, taki jak „Dodac do strony głównej znacznik <h1>”, bez spowodowania awarii aplikacji.

W końcu nadszedł czas, by umieścić nowe zadanie na stronie.

#### 6. Do kodu funkcji dodaj kolejne dwa wiersze (wyróżnione pogrubioną czcionką):

```
"Dodaj zadanie" : function () {
  var taskName = $('#task').val();
  if (taskName === '') {
    return false;
  }
  var taskHTML = '<li><span class="done">%</span>';
  taskHTML += '<span class="delete">x</span>';
  taskHTML += '<span class="task"></span></li>';
  var $newTask = $(taskHTML);
  $newTask.find('.task').text(taskName);
  $newTask.hide();
  $('#todo-list').prepend($newTask);
},
```

Ponieważ \$newTask jest obiektem jQuery, można wywoływać na jego rzecz funkcje biblioteki. W dodanym fragmencie kodu najpierw ukrywasz nowe zadanie, dzięki czemu później będziesz mógł je wyświetlić, używając efektu animacji. Po ukryciu elementu kolejna instrukcja najpierw wybiera element listy — jest to lista wypunktowana o identyfikatorze todo-list — a następnie dodaje na jej początku nowy (wciąż ukryty) element (więcej informacji na temat działania metody prepend() można znaleźć na stronie 159).

W kolejnym kroku wyświetlisz nowy element listy i ukryjesz okno dialogowe.

#### 7. Do kodu funkcji dodaj kolejne dwa wiersze (zmiany zostały wyróżnione pogrubioną czcionką):

```
"Dodaj zadanie" : function () {
  var taskName = $('#task').val();
  if (taskName === '') {
    return false;
  }
  var taskHTML = '<li><span class="done">%</span>';
  taskHTML += '<span class="delete">x</span>';
  taskHTML += '<span class="task"></span></li>';
  var $newTask = $(taskHTML);
  $newTask.find('.task').text(taskName);
  $newTask.hide();
  $('#todo-list').prepend($newTask);
  $newTask.show('clip', 250).effect('highlight', 1000);
  $(this).dialog('close');
},
```

Pierwsza z dodanych instrukcji operuje na dodanym do listy ukrytym elemencie i wyświetla go przy użyciu metody show(). Dla dodatkowego poprawienia atrakcyjności strony zastosowane zostały dwa efekty wizualne jQuery UI. Pierwszy z nich, clip (patrz strona 463), sprawia, że element wydaje się powiększać. Kiedy element będzie już widoczny, wywołujemy metodę effect(), każąc jej odtworzyć efekt highlight, który na chwilę wyświetla element na żółto, przyciągając uwagę użytkownika (i nieodmiennie powodując zachwyt osób korzystających z aplikacji).

Ostatni wiersz kodu zamyka okno dialogowe (ten sam kod zastosowałeś już wcześniej, w kroku 8. na stronie 524, dodając go do przycisku *Anuluj*). Teraz trzeba sprawdzić, jak to wszystko działa.

8. **Zapisz plik `todo.js`, a następnie odśwież w przeglądarce stronę `index.html`. Kliknij przycisk *Dodaj nowe zadanie*, wpisz nazwę zadania i kliknij przycisk *Dodaj zadanie*.**

Nowe zadanie powinno się pojawić poniżej nagłówka *Zadania do wykonania*. Jeśli jednak nie widać go, sprawdź kod i zajrzyj do konsoli JavaScript przeglądarki, by zobaczyć, czy nie ma w niej komunikatów o błędach.

Spróbuj dodać kolejne zadanie. Przy tej okazji zauważysz zapewne kolejny, niewielki problem — w polu tekstowym w oknie dialogowym pozostała nazwa poprzedniego zadania. Aby dodać nowe, musisz najpierw zaznaczyć ten tekst i go usunąć. Na szczęście ten problem można rozwiązać bardzo łatwo.

9. **W literale obiektowym przekazywanym w wywołaniu funkcji `dialog()`, za właściwością `buttons` wpisz przecinek i dodaj poniższy fragment kodu:**

```
close: function() {
    $('#new-todo input').val('');
}
```

Pełny kod funkcji `dialog()` powinien mieć następującą postać:

```
$("#new-todo").dialog({
    modal: true,
    autoOpen: false,
    buttons: {
        "Dodaj zadanie": function() {
            var taskName = $('#task').val();
            if (taskName === '') {
                return false;
            }
            var taskHTML = '<li><span class="done">%</span>';
            taskHTML += '<span class="delete">x</span>';
            taskHTML += '<span class="task"></span></li>';
            var $newTask = $(taskHTML);
            $newTask.find('.task').text(taskName);
            $newTask.hide();
            $('#todo-list').prepend($newTask);
            $newTask.show('clip', 250).effect('highlight', 1000);
            $(this).dialog('close');
        },
        "Anuluj": function() {
            $(this).dialog('close');
        }
    },
    close: function() {
        $('#new-todo input').val('');
    }
});
```

To naprawdę całkiem sporo kodu. Upewnij się, że ten ostatni fragment umieściłeś poza obiektem `buttons`. Opcja `close` widżetu okna dialogowego jQuery UI pozwala na wykonanie podanej funkcji w momencie, gdy użytkownik zamknie okno dialogowe. W tym przypadku wykonywane operacje ograniczą się do wyczyszczenia pola tekstowego, kiedy zatem użytkownik następnym razem otworzy okno dialogowe, pole będzie puste i gotowe do wpisania nazwy kolejnego zadania. (Zawartość pola można by także usunąć w ramach obsługi kliknięcia przycisku *Dodaj zadanie*, jednak chodziło tu o zaprezentowanie opcji `close`).



## Oznaczanie zadania jako wykonanego

Jedną z czynności, które podczas korzystania z listy zadań dają najwięcej satysfakcji, jest oznaczanie zadań jako wykonanych. Niestety nasza aplikacja jeszcze nie udostępnia tej jakże przyjemnej operacji. W tym podrozdziale naprawisz to niedopatrzenie. Ogólnie rzecz biorąc, zadanie jest całkiem proste: użytkownik zaznacza puste pole wyboru z lewej strony nazwy i zadanie — czyli element listy — jest przenoszone z jednej listy na drugą.

### Delegowanie zdarzeń

Aby oznaczyć zadanie jako wykonane, użytkownik musi kliknąć pole wyboru umieszczone w danym elemencie listy. Zazwyczaj funkcje obsługujące zdarzenia są określane poprzez wybranie odpowiedniego elementu i skojarzenie z nim funkcji; operację tę można wykonać w następujący sposób:

```
$('.done').click(function () {
  // Operacja wykonywana po kliknięciu elementu.
});
```

Jednak sposób obsługi zdarzeń w tworzonej liście zadań będzie nieco inny. Bezpośrednio po wyświetleniu strony w przeglądarce lista zadań jest pusta — nie ma żadnych zadań ani żadnych pól wyboru, które użytkownik mógłby kliknąć. Gdybyśmy spróbowali określić funkcję obsługującą zdarzenia `click` bezpośrednio po wczytaniu strony, nic by się nie stało. Ponieważ w tym momencie na stronie nie ma żadnych zadań ani żadnych pól wyboru, nie byłoby na niej żadnych elementów, w których można by zastosować tę funkcję. Oczywiście można by wywoływać metodę `click()` podczas tworzenia każdego nowego zadania, jednak biblioteka jQuery udostępnia znacznie lepsze rozwiązanie, nazywane delegowaniem zdarzeń (patrz strona 200).

Ogólnie rzecz biorąc, mechanizm delegowania zdarzeń pozwala wybrać jakiś inny element strony — element już istniejący, w którym będą umieszczane elementy dynamicznie dodawane do strony po jej wczytaniu. To właśnie ten element kontenera będzie odbierał zdarzenia skierowane do elementów umieszczonych wewnątrz niego, w naszym przypadku będą to zdarzenia `click`. Kiedy takie zdarzenie zostanie odebrane, kontener sprawdzi, czy faktycznie było ono skierowane do odpowiedniego elementu (w naszym przypadku do pola wyboru w którymś z zadań) i jeśli było, wykona odpowiednią funkcję.

W tym przypadku pusta lista wypunktowana znajduje się na stronie od momentu jej pobrania:

```
<ul id="todo-list">
</ul>
```

Kiedy użytkownik tworzy nowe zadanie, aplikacja dynamicznie dodaje je do tej listy:

```
<ul id="todo-list">
  <li>
    <span class="done">%</span>
    <span class="delete">x</span>
    <span class="task">Upiec ciasto</span>
  </li>
</ul>
```

Aby zareagować na kliknięcie elementu `<span>` należącego do klasy `done`, należy delegować funkcję obsługującą zdarzenia do istniejącej już listy wypunktowanej.

1. W pliku *todo.js* za kodem funkcji `dialog()`, lecz wciąż wewnątrz wywołania funkcji `$(document).ready()`, wpisz następujące wywołanie:

```
$("#todo-list").on('click', '.done', function() {

});
```

Tak wygląda ogólna struktura delegowanej funkcji obsługi zdarzeń. Najpierw wybierasz element listy wypunktowanej. Następnie wywoływana jest metoda `on()` jQuery, do której przekazujesz trzy argumenty. Pierwszym z nich jest łańcuch znaków zawierający nazwę zdarzenia, `'click'`. Drugim argumentem jest selektor określający element wewnątrz listy wypunktowanej, który musi zostać kliknięty. W tym przypadku chodzi o element `<span>` klasy `done`. I w końcu ostatnim argumentem wywołania jest funkcja — kod, który ma zostać wykonany, kiedy użytkownik kliknie pole wyboru oznaczające zadanie jako wykonane.

2. Wewnątrz funkcji obsługującej zdarzenia dodaj poniższy wiersz kodu (wyróżniony pogrubioną czcionką):

```
$("#todo-list").on('click', '.done', function() {
  var $taskItem = $(this).parent('li');
});
```

Pamiętaj, że zadaniem tej funkcji jest przeniesienie zadania z listy *Zadania do wykonania* na listę *Zadania wykonane*. W chwili, gdy użytkownik kliknie pole wyboru, dla jQuery istnieje jedynie znacznik `<span>` umieszczony gdzieś na liście, jednak nas interesuje cały znacznik `<li>`. Możesz go pobrać, korzystając z metody `parent()` jQuery, która umożliwia pobranie elementu rodzica aktualnie wybranego elementu.

W kodzie dodanym w tym kroku wyrażenie `$(this)` odwołuje się do elementu klikniętego przez użytkownika, czyli: `<span class="done">%</span>`. A zatem całe wywołanie `$(this).parent('li')` pobiera najbliższego przodka znacznika `<span>`, którym jest znacznik `<li>`. Innymi słowy, wybiera ono dokładnie ten element, o który chodziło.

Element ten jest następnie zapisywany w zmiennej `$taskItem`, a w kolejnym kroku go ukryjesz.

---

**Uwaga:** Wyczerpujące informacje na temat różnic pomiędzy `this` i `$(this)` można znaleźć na stronie 169.

---

3. Wewnątrz tej samej funkcji dodaj poniższy fragment kodu (wyróżniony pogrubioną czcionką):

```
$("#todo-list").on('click', '.done', function() {
  var $taskItem = $(this).parent('li');
  $taskItem.slideUp(250, function() {

  });
});
```

Metoda `slideUp()` jest zabawnym sposobem ukrycia elementu (patrz strona 216). Jednak jej wywołanie nie usuwa elementu ze strony. Po zakończeniu jej wywołania element wciąż jest dostępny na stronie, choć został ukryty przy użyciu

odpowiednich stylów CSS. Zgodnie z informacjami zamieszczonymi na stronie 211, wszystkie funkcje jQuery tworzące efekty animacji (takie jak `hide()`, `show()` czy też `slideUp()`) pozwalają na podawanie argumentów. W tym przypadku pierwszym z argumentów jest liczba — 250 — określająca czas trwania animacji; w naszym przypadku zajmie ona 250 milisekund.

Drugim argumentem jest funkcja zwrotna. Jest to funkcja, która zostanie wywołana *po* zakończeniu odtwarzania animacji. Pamiętajsz zapewne, że po zakończeniu wywołania funkcji `slideUp()` znacznik `<li>` reprezentujący zakończone zadanie wciąż znajduje się na liście *Zadania do wykonania*, tyle że jest niewidoczny. Musisz zatem przenieść go na drugą listę. Kod, który to robi, umieścisz właśnie w funkcji zwrotnej.

#### 4. Wewnątrz funkcji zwrotnej dodaj poniższy fragment kodu (wyróżniony pogrubioną czcionką):

```
$("#todo-list").on('click', '.done', function() {
    var $taskItem = $(this).parent('li');
    $taskItem.slideUp(250, function() {
        var $this = $(this);
        $this.detach();
    });
});
```

Pierwszy z dwóch nowych wierszy kodu jedynie pobiera element listy — `$(this)` — i zapisuje go w kolejnej zmiennej. Robisz to dlatego, że każde wywołanie funkcji jQuery — `$( )` — zmusza przeglądarkę do wykonania pewnej pracy. Ponieważ na tym elemencie będziesz musiał wykonać kilka operacji, zatem zamiast niepotrzebnie za każdym razem wywoływać funkcję jQuery, lepiej będzie zapisać wynik jej wywołania w zmiennej. (Rozwiązanie to, opisane bardziej szczegółowo na stronie 544, stanowi jedną z ogólnie przyjętych najlepszych praktyk związanych ze stosowaniem biblioteki jQuery).

Drugi z dodanych wierszy kodu wywołuje metodę `detach()`, która usuwa wybrany element lub elementy z drzewa DOM, choć pozostawia je na stronie. Innymi słowy, wybrany element jest usuwany z listy, lecz wciąż znajduje się w pamięci przeglądarki. Co więcej, wciąż jest zapisany w zmiennej `$this`. Dzięki temu w następnym kroku będziesz mógł przenieść go w inne miejsce strony — a konkretnie, na drugą listę!

---

**Uwaga:** Wyczerpujące informacje na temat metody `detach()` można znaleźć na stronie <http://api.jquery.com/detach/>.

---

#### 5. Dokończ kod funkcji zwrotnej, dopisując wewnątrz niej kolejne dwa wiersze kodu (wyróżnione pogrubioną czcionką):

```
$("#todo-list").on('click', '.done', function() {
    var $taskItem = $(this).parent('li');
    $taskItem.slideUp(250, function() {
        var $this = $(this);
        $this.detach();
        $('#completed-list').prepend($this);
        $this.slideDown();
    });
});
```

Metodę `prepend()` poznałeś już wcześniej (w kroku 6. na stronie 529). Za jej pomocą wstawisz kod HTML wewnątrz innego elementu. W tym przypadku odłączony element listy — `$this` — zostaje zapisany na początku listy *Zadania wykonane* (listy wypunktowanej o identyfikatorze `completed-list`). I w końcu, ponieważ przenoszony element został ukryty przez wywołanie metody `slideUp()`, zatem teraz możesz go wyświetlić w nowym położeniu — w tym celu wywoływana jest metoda `slideDown()` (patrz strona 216).

6. Zapisz plik *todo.js*, a następnie odśwież stronę *index.html* w przeglądarce. Aby ją przetestować, najpierw dodaj kilka zadań, a następnie kliknij pole wyboru umieszczone z lewej strony nazwy zadania, aby oznaczyć je jako wykonane.

Teraz powinieneś już bez problemu dodawać zadania i oznaczać je jako wykonane (patrz rysunek 14.4). Jeśli przykładowa aplikacja nie działa, dokładnie sprawdź kod i zajrzyj do konsoli JavaScript, by zobaczyć, czy nie zostały w niej wyświetlone jakieś komunikaty o błędach.



**Rysunek 14.4.** Pole wyboru oraz przycisk, który pozwala oznaczyć zadanie jako wykonane oraz je usunąć, zmieniają wygląd po wskazaniu ich myszą. Możliwość tej nie zapewnia jednak kod JavaScript, lecz style CSS. A konkretnie, pseudoklasa `:hover` pozwalająca na zmianę wyglądu elementu, na którym znajduje się wskaźnik myszy. Aby sprawdzić, jak tworzony jest ten efekt, zajrzyj do pliku *todo.css* umieszczonego w podkatalogu *css* katalogu *R14*

A teraz kolejna sprawa — może warto byłoby mieć możliwość zmiany kolejności zadań na liście? Dzięki temu można by dodać kilka zadań, a następnie uporządkować je w takiej kolejności, w jakiej powinny zostać wykonane, na przykład: „Kupić książkę kucharską”, „Upiec ciasteczka”, „Zjeść ciasteczka”. Przy użyciu widżetu Sortable jQuery UI takie możliwości funkcjonalne można uzyskać bardzo szybko i łatwo. A skoro już przy tym jesteśmy, równie łatwo możesz zapewnić użytkownikom możliwość dowolnego przenoszenia zadań między oboma listami, zatem będzie można oznaczyć zadanie jako wykonane poprzez przeciągnięcie go na listę *Zadania wykonane*.

7. Otwórz plik *index.html* w edytorze tekstów i odszukaj wewnątrz niego znaczniki `<ul>` obu list: *Zadania do wykonania* oraz *Zadania wykonane*. Do obu dodaj atrybut `class="sortlist"`, tak by miały następującą postać:

```
<ul id="todo-list" class="sortlist">
```

oraz

```
<ul id="completed-list" class="sortable">
```

Przypisując tę samą nazwę klasy obu listom, będziesz mógł w prosty sposób zapewnić możliwość porządkowania elementów na każdej z nich.

8. Zapisz plik *index.html*, po czym otwórz w edytorze plik *todo.js*. Na samym końcu funkcji `$(document).ready()`, lecz wciąż wewnątrz niej, wpisz następujące wywołanie:

```
$('.sortable').sortable();
```

W ten sposób obie listy zapewniają już możliwość porządkowania elementów. Jeśli chcesz sprawdzić, jak to działa, zapisz oba pliki, a następnie odśwież stronę *index.html* w przeglądarce. Zauważysz jednak, że choć faktycznie można zmieniać kolejność elementów na liście, jednak nie da się przeciągać elementów jednej listy na drugą. Bardzo łatwo można jednak połączyć ze sobą dwa widżety Sortable.

9. Wewnątrz wywołania funkcji `sortable()` wstaw następujący literał obiektowy:

```
$('.sortable').sortable({
  connectWith : '.sortable'
});
```

Opcja `connectWith` (opisana szczegółowo na stronie 452) umożliwia połączenie dwóch list. Ponieważ w naszej aplikacji obie listy należą do tej samej klasy, aby zatem je połączyć, wystarczy podać selektor tej klasy. W ten sposób użytkownik będzie już mógł dowolnie przeciągać elementy pomiędzy oboma listami. Jednak zanim definitywnie zakończysz pracę nad listami, dodasz do nich jeszcze kilka ostatnich, drobnych uprawnień wizualnych.

10. Do tego samego literału obiektowego dodaj trzy kolejne właściwości (wyróżnione pogrubioną czcionką):

```
$('.sortable').sortable({
  connectWith : '.sortable',
  cursor : 'pointer',
  placeholder : 'ui-state-highlight',
  cancel : '.delete, .done'
});
```

Nie zapomnij o przecinku za opcją `connectWith`. Opcja `cursor` (opisana na stronie 425) zmienia postać wskaźnika myszy podczas przeciągania elementów, a opcja `placeholder` (patrz strona 454) wyróżnia miejsce listy, gdzie użytkownik może upuścić przeciągany element. I w końcu opcja `cancel` (patrz strona 451) określa te elementy umieszczone wewnątrz elementu sortowanego, które nie mogą posłużyć jako „uchwyty” do przeciągania. W naszym przypadku użytkownik nie będzie mógł przeciągać zadania, używając pola wyboru oznaczającego zadanie jako wykonane ani przycisku do usunięcia zadania.

11. Zapisz pliki *index.html* oraz *todo.js*, po czym odśwież plik *index.html* w przeglądarce.

Dodaj do listy kilka różnych zadań. Spróbuj je przeciągać pomiędzy obiema listami. Teraz powinieneś już oznaczyć zadanie jako wykonane poprzez samo przeciągnięcie go na listę *Zadania wykonane* (patrz rysunek 14.1).

## Usuwanie zadań

Pozostała do zaimplementowania już tylko jedna możliwość, czyli usuwanie zadań. Jest ona całkiem ważna, gdyż może się zdarzyć, że użytkownik przez pomyłkę doda zadanie, które nie powinno się znaleźć na liście. Poza tym, jeśli lista wykonanych zadań stanie się zbyt długa, użytkownik może zdecydować się na usunięcie kilku z nich.

1. W pliku *todo.js*, za kodem funkcji `sortable()`, lecz wciąż wewnątrz funkcji `$(document).ready()` wpisz następujący fragment kodu:

```
$('.sortlist').on('click', '.delete', function() {
});
```

Także w tym przypadku zastosowałeś delegowanie zdarzeń. W momencie wczytywania strony listy są puste i nie ma w nich żadnych przycisków, dlatego też do usuwania zadań używasz techniki delegowania. W tym przypadku wywołanie `$('.sortlist')` powoduje pobranie obu list wypunktowanych dostępnych na stronie (gdyż użytkownicy powinni mieć możliwość usuwania zadań z obu list), a wywołanie metody `on()` informuje jQuery, że należy obsługiwać zdarzenia kliknięcia skierowane do elementów klasy `delete`. W odpowiedzi na kliknięcie takiego elementu ma zostać wywołana przekazana funkcja.

W kolejnym punkcie zajmiesz się zaimplementowaniem kodu tej funkcji.

2. Znajdź kod między znacznikami `<script>` w sekcji nagłówkowej strony i usuń kod wyróżniony pogrubieniem:

```
$('.sortlist').on('click', '.delete', function() {
    $(this).parent('li').effect('puff', function() {
        $(this).remove();
    });
});
```

W tym kodzie dzieje się całkiem sporo, lecz powinieneś już być przyzwyczajony do takich rozwiązań. Poniżej zostały opisane wszystkie wykonywane operacje.

- Wyrażenie `$(this).parent('li')` pobiera element kliknięty przez użytkownika (reprezentowany przez `$this`), a następnie wśród jego przodków odnajduje znacznik `<li>`. Innymi słowy, wyrażenie to pobiera element zadania, które należy usunąć.
- Metoda `effect()` biblioteki jQuery UI odtwarza w elemencie określony efekt wizualny. W tym przypadku odtwarzamy efekt o nazwie `puff`, który powoduje że element się powiększa, stopniowo wygasa i w końcu znika.
- W wywołaniu metody `effect()` została umieszczona funkcja zwrotna, która będzie wywołana po zakończeniu efektu. W naszym przypadku funkcja pobiera element listy, na którym operowała metoda `effect()` — czyli element listy pobrany przy użyciu wyrażenia `$(this)` — a następnie usuwa go całkowicie ze strony, wywołując metodę `remove()` jQuery (patrz strona 160). Metoda `remove()`, w odróżnieniu od `detach()` (opisanej na stronie 533), całkowicie usuwa wskazany element strony.

3. Zapisz plik i odśwież w przeglądarce stronę *index.html*.

Teraz powinieneś już bez problemu dodawać, przenosić i usuwać zadania, a cała strona powinna wyglądać tak, jak na rysunku 14.1. Gdybyś miał jakieś problemy z wykonaniem tego przykładu, poniżej znajdziesz cały kod pliku *todo.js*.

```

$(document).ready(function(e) {
  $("#add-todo").button({
    icons: {
      primary: "ui-icon-circle-plus"
    }
  }).click(function() {
    $("#new-todo").dialog('open');
  });

  $("#new-todo").dialog({
    modal: true,
    autoOpen: false,
    buttons : {
      "Dodaj zadanie" : function() {
        var taskName = $('#task').val();
        if (taskName === '') {
          return false;
        }
        var taskHTML = '<li><span class="done">%</span>';
        taskHTML += '<span class="delete">x</span>';
        taskHTML += '<span class="task"></span></li>';
        var $newTask = $(taskHTML);
        $newTask.find('.task').text(taskName);
        $newTask.hide();
        $('#todo-list').prepend($newTask);
        $newTask.show('clip',250).effect('highlight',1000);
        $(this).dialog('close');
      },
      "Anuluj" : function() {
        $(this).dialog('close');
      }
    },
    close: function() {
      $('#new-todo input').val('');
    }
  });

  $("#todo-list").on('click', '.done', function() {
    var $taskItem = $(this).parent('li');
    $taskItem.slideUp(250, function() {
      var $this = $(this);
      $this.detach();
      $('#completed-list').prepend($this);
      $this.slideDown();
    });
  });

  $('.sortlist').sortable({
    connectWith : '.sortlist',
    cursor : 'pointer',
    placeholder : 'ui-state-highlight',
    cancel : '.delete,.done'
  });

  $('.sortlist').on('click', '.delete', function() {
    $(this).parent('li').effect('puff', function() {
      $(this).remove();
    });
  });
}); // Koniec funkcji ready.

```

**Uwaga:** Pełną, działającą kopię aplikacji napisanej w tym rozdziale znajdziesz w plikach *complete-index.html* oraz *complete-todo.js*, umieszczonych w katalogu *R14*.

## Dalsze kroki

Gratuluję — właśnie napisałeś swoją pierwszą aplikację internetową! Jednak tę aplikację można poprawić na kilka różnych sposobów. Być może już nawet masz przygotowaną listę ewentualnych usprawnień. W tym podrozdziale zamieszczona została lista potencjalnych poprawek wraz z odnośnikami do źródeł informacji, które mogą Ci się przydać podczas ich implementowania.

### Edycja zadań

Aktualnie aplikacja nie zapewnia użytkownikowi możliwości poprawy ewentualnych błędów typograficznych w nazwach zadań. Jeśli ktoś wpisze „Uwiec ciasteczka”, będzie musiał usunąć zadanie i utworzyć je od nowa. Problem edycji zadań można rozwiązać na dwa sposoby.

Pierwszym z nich jest dodanie do każdego z zadań przycisku *Edytuj*. Jego kliknięcie będzie powodować wyświetlenie okna dialogowego z polem tekstowym zawierającym nazwę zadania — czyli tekst umieszczony wewnątrz znacznika `<span>` klasy `task`.

Można także dodać do strony kolejne okno dialogowe, takie jak przedstawione na stronie 522. Kliknięcie przycisku *Edytuj* powodowałoby wyświetlenie tego okna dialogowego i umieszczenie nazwy zadania w polu tekstowym, a zamknięcie okna dialogowego — aktualizację nazwy zadania.

Innym rozwiązaniem może być skorzystanie z właściwości HTML o nazwie `contentEditable`. Jej użycie zapewnia możliwość edycji zawartości dowolnego elementu HTML. Przykładowo poniżej został przedstawiony kod HTML zapewniający możliwość edycji nazw zadań w naszej przykładowej aplikacji:

```
<span class="task" contentEditable>
```

Właściwość tę można nawet zastosować dynamicznie, używając metody `prop()` jQuery:

```
$('.task').prop('contentEditable', true);
```

Jednak z zastosowaniem właściwości `contentEditable` wiąże się jeden problem. Widżet `Sortable` jQuery UI nie pozwala na zaznaczanie tekstu w sortowalnych elementach, a zatem nawet po zastosowaniu właściwości `contentEditable` użytkownik nie będzie mógł zaznaczyć tekstu do edycji. Problem ten można ominąć, nakazując jQuery UI ignorowanie elementów klasy `task` poprzez przekazanie odpowiedniej opcji w wywołaniu funkcji `sortable()`. Dokładnie w taki sam sposób postąpiłeś z polem wyboru do oznaczania zadania jako wykonanego oraz przyciskiem do usuwania zadania w kroku 10. na stronie 535; a tu wystarczy dodać selektor `.task` tak, jak pokazano na poniższym przykładzie:

```
cancel: '.delete,.done,.task'
```

Jeśli jednak zastosujesz to rozwiązanie, w elementach zadań nie pozostanie zbyt wiele miejsc, za które użytkownik będzie mógł je przeciągać. W takim przypadku powinieneś dodać do elementu `<li>` zadania jakiś wyraźnie widoczny element, którego użytkownik mógłby używać do przeciągania. Aby skorzystać z tego rozwiązania, wystarczy użyć opcji `handle` widżetu `Sortable` (patrz strona 453).



## Potwierdzanie usunięcia

Obecnie kiedy użytkownik kliknie przycisk usuwający zadanie, zostaje ono usunięte raz na zawsze. Mógłbyś jednak dodać modalne okno dialogowe, zawierające prośbę o potwierdzenie usunięcia. Jeśli użytkownik kliknie przycisk *Tak*, zadanie zostanie usunięte, jeśli przycisk *Nie*, zadanie pozostanie na liście.

## Zapisywanie listy

Jednak największym problemem naszej aplikacji jest to, że nie zapamiętuje ona zadań w momencie zamykania okna przeglądarki. Innymi słowy, lista ma charakter całkowicie tymczasowy i nie zostanie odtworzona po zamknięciu i ponownym uruchomieniu przeglądarki bądź też po wyświetleniu strony na innym komputerze. Istnieje kilka sposobów pozwalających na zapamiętanie stanu listy zadań.

### Magazyn lokalny

Wszystkie nowoczesne przeglądarki udostępniają mechanizm określany jako *magazyn lokalny* (ang. *local storage*). Pozwala on na zapisywanie danych na komputerze użytkownika i odczytanie ich po ponownym wyświetleniu strony. Mógłbyś skorzystać z magazynu lokalnego, by zapisywać aktualny stan listy zadań po każdej zmianie ich stanu. W takim przypadku, kiedy użytkownik wróci na stronę, powinienś sprawdzić, czy są zapisane jakieś dane w magazynie lokalnym, a jeśli są, musisz odpowiednio zaktualizować stronę.

Więcej informacji na temat magazynu lokalnego można znaleźć na witrynie Mozilla Developer Network: <https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Storage>. Dostępna jest nawet wtyczka jQuery ułatwiająca korzystanie z tego mechanizmu: <https://github.com/julien-maurel/jquery-Storage-API>.

### Zapis na serwerze

Kolejną możliwością jest zapisanie listy zadań na serwerze. To rozwiązanie ma tę zaletę, że takiej listy zadań można używać na dowolnym komputerze. Z drugiej strony jego wadą jest konieczność utworzenia jakiegoś systemu kontroli dostępu do listy, w przeciwnym razie każdy będzie mógł wejść na stronę i wyświetlić Twoje zadania (a nawet je usunąć).

Książka ta nie jest poświęcona zagadnieniom programowania aplikacji działających po stronie serwera. Jednak do przekazania zadań na serwer będziesz musiał użyć kodu JavaScript. Zapewne będzie to kod korzystający z technologii AJAX, opisanej w poprzednim rozdziale.

Najprostszym rozwiązaniem byłoby pobranie wszystkich elementów list i zastosowanie metody `.each()` do pobrania nazwy każdego z zadań. Następnie mógłbyś utworzyć obiekt jQuery zawierający listę wszystkich zadań, zarówno tych do wykonania, jak i już wykonanych. W końcu, abyś mógł je przesłać na serwer, musiałbyś *serializować* ten obiekt, czyli zapisać go w postaci łańcucha znaków. Poniżej został zamieszczony kod funkcji, która realizuje wszystkie te operacje:

```
function getData() {
  var todoData = {
    todo : [],
    completed : []
  };
  $('#todo-list').each( function() {
    var task = $(this).find('.task').text();
    todoData.todo.push(task);
  });
  $('#completed-list').each( function() {
    var task = $(this).find('.task').text();
    todoData.completed.push(task);
  });
  return $.param(todoData);
}
```

Tę funkcję mógłbyś wywoływać za każdym razem, kiedy będziesz chciał pobrać wszystkie zadania w postaci nadającej się do przesłania na serwer. Odczytanie tych zadań i zrobienie z nimi czegoś użytecznego należałoby już do programu wykonywanego na serwerze.

## Inne pomysły

Jeśli masz więcej pomysłów na poprawienie zamieszczonej tu przykładowej listy zadań, warto, byś je dokładniej wypróbował. Ten projekt doskonale nadaje się do sprawdzenia nowych umiejętności stosowania języka JavaScript oraz bibliotek jQuery i jQuery UI. Bezustannie rozwijaną wersję tego projektu można znaleźć na serwisie GitHub, na stronie <https://github.com/sawmac/jquery-todo>.

# Skorowidz

## A

adres URL, 45, 259, 508  
AJAX, Asynchronous JavaScript and XML, 471  
  formatowanie danych, 487  
  obsługa błędów, 494  
  przetwarzanie danych, 490  
akcje, 167  
akordeony jQuery UI, 363  
aktywowanie pola, 290, 295  
animacje, 209, 211, 220, 465  
  CSS3, 231, 234  
  tempo, 221  
animowanie  
  kolorów, 220  
  zmiany klas, 466  
animowany pasek, 225  
API, Application Programming Interface, 548  
API key, 507  
atrybut, 146  
  checked, 284  
  href, 251  
  src, 44, 240  
  title, 346  
atrybuty  
  HREF, 167  
  HTML, 166  
  znaczników, 160  
automatyczne uzupełnianie, 393, 394, 398

## B

biblioteka, 49  
  Dojo Toolkit, 135  
  jQuery, 20, 131, 135  
  jQuery UI, 321  
  Mootools, 135  
  Yahoo User Interface Library, 135  
biblioteki JavaScript, 133, 135

blokowanie  
  działania odnośników, 256  
  przesyłania danych, 292  
  reakcji na zdarzenia, 195  
błąd składniowy, 54, 612  
błędy, 161  
  czasu wykonania, 615  
  logiczne, 615  
  składniowe, 615  
  w kodzie, 54, 611  
brak symboli końcowych, 612

## C

CDN, content distribution network, 136  
CSS, Cascading Style Sheets, 23, 150  
  blok deklaracji, 25  
  deklaracja, 25  
  modyfikacja właściwości, 163  
  odczyt właściwości, 163  
  selektor, 24  
  wartości, 25  
  właściwości, 25  
  zmiana właściwości, 164  
CSS3, 231  
cytat wyróżniany, 171  
czas wczytywania kodu, 609

## D

dane złożone JSON, 503  
daty i godziny, 592  
debugger, 629, 631, 636  
delegowanie zdarzeń, event delegation,  
  198-205, 531  
diagnozowanie  
  pliku, 633  
  skryptu, 636  
zaawansowane, 628

- dodawanie
  - efektu rollover, 245
  - elementów do tablicy, 80
  - etykietek ekranowych, 347
  - formatu JSON, 509
  - funkcji anonimowej, 184
  - identyfikatorów, 508
  - jQuery, 139
  - jQuery UI, 329
  - kanału Flickr, 506
  - końca JavaScript, 40
  - komunikatów, 301
  - komunikatów o błędach, 304
  - menu, 368
  - odwołania zwrotnego, 509
  - okna dialogowego, 522
  - operacji, 185
  - przycisków, 339, 341, 520
  - reguł walidacji, 303
  - tekstu, 48
  - tematu do strony, 414
  - treści, 157, 546
  - widżetu Draggable, 422
  - wyróżnionych cytatów, 171
  - zadań, 519, 525
  - zdjęć, 512
  - zestawu kart, 351, 356
- dokumentacja jQuery, 548, 552
- dołączanie
  - pliku JavaScript, 49
  - pliku jQuery, 137
  - wtyczki Validation, 301
  - zdarzenia, 184
- DOM, Document Object Model, 146, 554
- dopasowanie liczby, 573
- dopasowywanie wzorców, 582
- dopełnianie pojedynczych cyfr, 596
- dostarczanie podpowiedzi, 393
- dostęp do
  - błędów, 55
  - danych, 502
- dostosowywanie
  - przycisków, 390
  - wyglądu, 407
- dyrektywa @keyframes, 235
- działanie
  - funkcji, 118
  - odnośników, 256
  - znaczników, 22
- dzielenie łańcuchów znaków, 605

## E

- edycja zadań, 538
- efekt, 211, 325, 421
  - blind, 462
  - bounce, 463
  - clip, 463
  - distance, 463

- drop, 463
- effect(), 462
- explode, 461, 463
- fade, 463
- fold, 463
- hide(), 461
- highlight, 464
- puff, 464
- pulsate, 464
- rollover, 243, 245
- scale, 464
- shake, 464
- show(), 461
- size, 465
- slide, 465
- times, 463
- toggle(), 462
- efekty
  - jQuery UI, 461, 462
  - wizualne, 249
- elementy
  - formularza, 281
  - pętli for, 113
  - tablicy, 79
  - wyzwalające, 343
- etykiety ekranowe, 345
  - kod HTML, 350
  - opcje, 348
  - treści HTML, 349

## F

- FAQ, Frequently Asked Questions, 204
- FIFO, First In, First Out, 82
- filtr
  - :even, 154
  - :odd, 154
  - :first, 154
  - :last, 154
  - :not(), 154
  - :has(), 154
  - :contains(), 155
  - :hidden(), 155
  - :visible, 155
  - checked, 282, 284
  - selected, 283
- filtry jQuery, 153
- format
  - godziny, 595
  - JSON, 491, 500
- formatowanie
  - danych, 487
  - komunikatów, 319
  - wartości monetarnych, 590
- formularze, 279, 375
  - aktywowanie pola, 290, 295
  - daty ze stylem, 375
  - inteligentne, 290
  - logowania, 216

- proste wzbogacanie, 294
- ukrywanie opcji, 293
- ukrywanie pól, 298
- walidacja, 299
- widzety usprawniające, 401
- włączanie pól, 291
- wyłączanie pól, 295
- wyświetlanie opcji, 293
- funkcja, *Patrz także* metoda
- `$()`, 544
- `$(document).ready()`, 169, 191, 205, 358
- `$.each()`, 504
- `$.getJSON()`, 501
- `$.get()`, 501
- `$.post()`, 501
- `.after()`, 159, 561
- `.append()`, 158, 560
- `.before()`, 159, 561
- `.button()`, 523
- `.children()`, 556
- `.closest()`, 557
- `.empty()`, 563
- `.end()`, 559
- `.find()`, 548, 556
- `.html()`, 157, 560
- `.next()`, 560
- `.parent()`, 557
- `.prepend()`, 159, 561
- `.ready()`, 191
- `.remove()`, 561
- `.replaceWith()`, 561
- `.siblings()`, 558
- `.text()`, 158, 560
- `.unwrap()`, 563
- `.wrap()`, 561
- `.wrapInner()`, 562
- `accordion()`, 367
- `addClass()`, 162, 466
- `alert()`, 187
- `animate()`, 220–228
- `append()`, 546
- `appendTo()`, 318
- `attr()`, 166, 241, 256
- `autocomplete()`, 398
- `buildAnswer()`, 638
- `button()`, 390, 392, 406
- `buttonset()`, 392
- `click()`, 218, 254, 257
- `console.log()`, 623, 624
- `css()`, 163, 165, 297, 445, 553
- `datepicker()`, 403
- `dialog()`, 336–339, 522, 530
- `document.write()`, 86
- `draggable()`, 431, 447
- `drop`, 447
- `droppable()`, 437, 447
- `each()`, 168, 255, 505
- `effect()`, 462
- `fadeIn()`, 214, 253

- `fadeOut()`, 167, 209, 214, 225
- `fadeTo()`, 214
- `fadeToggle()`, 214
- `focus()`, 291
- `get()`, 488
- `getMonth()`, 593
- `hide()`, 206, 212, 461
- `hover()`, 192, 227, 244
- `jQuery()`, 544
- `not()`, 259
- `off()`, 196
- `on()`, 197–202
- `openExt()`, 260
- `parseFloat()`, 588
- `parseInt()`, 588
- `post()`, 488
- `prepend()`, 499, 546
- `preventDefault()`, 195, 257
- `processData()`, 498
- `processResponse()`, 492
- `prop()`, 297
- `ready()`, 141, 186, 190
- `remove()`, 254
- `removeAttr()`, 166
- `removeClass()`, 162, 466
- `selectmenu()`, 387, 405
- `serialize()`, 490
- `show()`, 212, 257, 461
- `slideDown()`, 208, 216
- `slideToggle()`, 216
- `slideUp()`, 209, 216, 533
- `sortable()`, 452, 455, 458
- `stopPropagation()`, 197
- `tabs()`, 354, 358, 360
- `text()`, 528
- `toggle()`, 213, 462
- `toggleClass()`, 234, 466
- `tooltip()`, 346–349
- `val()`, 281, 283, 284, 300
- `validate()`, 302, 306, 309
- funkcje, 115, 620
  - anonimowe, 168, 184, 186, 524
  - do manipulacji kodem, 560
  - nazwy zmiennych, 121
  - pobieranie informacji, 120
  - przekazywanie danych, 118
  - quiz, 124
  - systemu operacyjnego, 31
  - wbudowane, 60
  - zwrotne, 223, 477, 493, 533

## G

- galeria fotografii, 249, 250
- generowanie
  - liczby losowej, 592
  - podpowiedzi, 395
- GitHub, 267
- grupowanie fragmentów wzorców, 576

## H

HTML, Hypertext Markup Language, 21  
HTML5, 42, 301

## I

identyfikator tooltip, 545  
ikona  
  główna, 390  
  pomocnicza, 390  
instrukcja, 59  
  else if, 99, 101  
  if, 95  
  Switch, 603  
instrukcje  
  warunkowe, 94  
  warunkowe zagnieżdżone, 104  
interakcje, 325, 421, 493  
interfejs użytkownika, 323, 326  
interpreter języka JavaScript, 40

## J

JavaScript, 17  
  funkcje, 115  
  gramatyka, 59  
  instrukcje, 59  
  instrukcje warunkowe, 94, 105  
  komentarze, 88  
  literały obiektowe, 165  
  obiekty, 86  
  operacje matematyczne, 68  
  pętle, 109  
  pierwszy program, 37  
  słowa kluczowe, 65  
  struktury logicznych, 93  
  struktury sterujące, 93  
  tablice, 77, 111  
  typy danych, 60  
  wbudowane funkcje, 60  
  zmiennie, 63  
jednostki miary, 220  
język  
  CSS, 23  
  HTML, 21  
  HTML5, 42  
  JavaScript, 17  
języki  
  kompilowane, 39  
  skryptowe, 39  
jQuery, 20, 133, 135  
  AJAX, 479  
  animacje, 211  
  efekt rollover, 243  
  efekty, 211  
  filtry, 153

klasy, 161  
kolekcje, 155  
menu responsywne, 270  
metoda load(), 480  
obsługa zdarzeń, 182  
pobieranie elementów, 147  
podmiana obrazków, 241  
rozjaśnianie elementów, 213  
selektory, 282  
ukrywania elementów, 212  
wczytywanie rysunków, 242  
wtyczka Validation, 301, 318  
wtyczki, 265  
wygaszanie elementów, 213  
wyświetlanie elementów, 212  
wzbogacanie formularzy, 279  
zamiana rysunków, 239  
zastosowania, 239  
zdarzenia, 177  
zdarzenia specyficzne, 190  
jQuery UI, 323, 325  
  akordeony, 363  
  animacje, 465  
  arkusze stylów, 415  
  dokładne umiejscawianie, 343  
  dostosowywanie wyglądu, 407  
  efekty, 421, 461  
  formularze, 375  
  interakcje, 421  
  interfejs użytkownika, 323  
  okna dialogowe, 330, 523  
  pasek nawigacyjny, 371  
  prezentowanie informacji, 345  
  przesłanie stylów, 415  
  stosowanie nowego tematu, 413  
  tematy graficzne, 383  
  widżet Autocomplete, 394–396, 400  
  widżet DatePicker, 375  
  widżet Draggable, 421  
  widżet Droppable, 434  
  widżet przycisku, 389, 390  
  widżet Selectmenu, 368  
  widżet Sortable, 449  
  widżet Tabs, 354  
  zastosowania, 327  
  zastosowania zaawansowane, 469  
  zdarzenia niestandardowe, 357  
JSON, JavaScript Object Notation, 500  
JSON z wypełnieniem, 506  
JSONP, JSON with padding, 506

## K

kalendarze, 377, 383  
kanał Flickr, 506  
karta Sources, 629, 637  
karty, 351  
karty prezentujące zawartość, 360

## klasa

- Ajax, 551
  - Attributes, 550
  - CSS, 550
  - Data, 551
  - Deferred objects, 551
  - Dimensions, 552
  - Effects, 551
  - Events, 550
  - Forms, 551
  - Internals, 552
  - Manipulation, 550
  - Offset, 552
  - Selectors, 549
  - Traversing, 550
  - Utilities, 551
- klasa, 161
- animateDiv, 236
  - button, 232
  - close, 209
  - faded, 233
  - required, 303, 304
  - ui-dialog-title, 420
  - ui-menu, 369
  - ui-widget, 419
- klatki kluczowe, keyframes, 234
- klauzula else, 98, 101
- klucz, 507
- kod HTML
- formularza, 280
  - menu nawigacyjnego, 270
  - sekcji strony, 295
- kolejka FIFO, 82
- kolejność wykonywania operacji, 69
- kolekcje jQuery, 155
- komentarze, 88
- komunikacja z serwerem WWW, 476
- komunikat o błędzie, 99, 301, 304, 309, 319, 613
- konfigurowanie serwera WWW, 476
- konsola, 621, 624
- błędów, 57
  - JavaScript, 52, 53, 56, 628
- kontrola działania odnośnik, 255
- kontrolki formularza, 279
- kontrolowanie działania skryptu, 631

## L

- liczba, 61
- błędów, 621
  - losowa, 591
- lista
- FAQ, 204
  - rozwijana stylowa, 383
  - wypunktowana, 270
  - zadań, 519, 525
  - zagnieżdżona, 272
- literały obiektowe, 165, 489

## Ł

- łańcuch znaków, 61
- dzielenie, 605
  - odnajdywanie wzorów, 570
  - określanie długości, 566
  - pobieranie fragmentu, 569
  - przeszukiwanie łańcuchów, 567
  - z zapytaniem, 487
  - zamiana na liczbę, 587
  - zmiana wielkości znaków, 566
- łańcuchy
- formatujące, 379
  - wywołań funkcji, 156
- łączenie
- liczb, 70
  - łańcuchów znaków, 69
  - opcji, 510
  - różnych elementów, 606
  - tablic, 605

## M

- magazyn lokalny, 539
- manipulacja kodem HTML, 560
- menu, 31
- nawigacyjne, 270
  - wielopoziomowe, 372
- metoda
- \$.each(), 515
  - \$.get(), 495
  - \$.getJSON(), 510
  - addClass(), 467
  - blur(), 263
  - cancel, 458
  - close(), 263
  - destroy, 458
  - detach(), 533
  - disable, 458
  - draggable(), 441
  - each(), 504
  - effect(), 529, 536
  - enable, 458
  - find(), 528
  - focus(), 263
  - GET, 477, 488
  - get(), 486
  - getDay(), 594
  - getHours(), 594
  - indexOf(), 568
  - load(), 480, 482, 485
  - match(), 575, 584
  - Math.random(), 591
  - menu(), 373
  - moveBy(), 263
  - moveTo(), 264
  - Number(), 587
  - on(), 197
  - open(), 262

metoda  
  parent(), 532  
  parseInt(), 164  
  POST, 477, 488  
  post(), 486  
  prepend(), 529, 534  
  prop(), 285  
  remove(), 536  
  removeClass(), 467  
  resizeBy(), 264  
  resizeTo(), 264  
  scrollBy(), 264  
  scrollTo(), 264  
  search(), 582  
  send(), 478  
  serialize, 458, 459  
  slice(), 569  
  slideUp(), 532, 534  
  switchClass(), 467  
  toArray, 459  
  toggleClass(), 467  
metody  
  obiektu Date, 593  
  widżetów Sortable, 458  
modalne okna dialogowe, 335  
modyfikowanie  
  stron, 142  
  tematów graficznych, 407  
  treści, 546  
  właściwości CSS, 163

## N

nagłówki kart, 353  
narzędzia  
  do programowania, 26  
  inspekcji kodu, 419  
narzędzie inspektora, 420  
nawias  
  klamrowy, 95, 105, 165, 229  
  kwadratowy, 78  
nazwy zmiennych, 64, 121  
negowanie warunków, 103

## O

obiekt, 86  
  buttons, 344  
  Date, 593, 595, 597  
  JSON, 501, 502  
  rules, 306  
  XMLHttpRequest, 474, 476  
obiekty  
  reprezentujące zdarzenia, 194  
  jQuery, 147  
obserwowanie skryptu, 632  
obsługa  
  błędów, 494  
  danych, 477  
  kanałów, 509

  kilku elementów, 199  
  listy zadań, 519  
  quizów, 124, 129  
  stref czasowych, 599  
  zdarzeń, 182, 185, 226, 531  
odczyt  
  atrybutów HTML, 166  
  atrybutów znaczników, 160  
  właściwości CSS, 163  
odnośniki, 251  
  blokowanie działania, 256  
  działanie domyślne, 256  
  lokalizacja docelowa, 255  
  pobieranie, 255  
  zewnętrzne, 258  
odświeżanie strony, 472  
odwołanie zwrotne JSONP, 509  
okna dialogowe, 330, 332  
  dodawanie przycisków, 339, 341  
  modalne, 335  
  otwieranie, 338  
  przekazywanie opcji, 336  
okno przeglądarki, 261  
opcja  
  accept, 436  
  active, 365  
  activeClass, 436  
  animate, 365  
  axis, 424, 451  
  buttons, 524  
  cancel, 424, 451  
  cancelWith, 452  
  change, 387  
  changeMonth, 378  
  changeYear, 378  
  collapsible, 365  
  connectToSortable, 424  
  connectWith, 535  
  containment, 425, 452  
  cursor, 453, 535  
  cursorAt, 453  
  dateFormat, 378, 379  
  delay, 401, 453  
  disabled, 437  
  distance, 453  
  equalTo, 308  
  event, 365  
  grid, 426, 453  
  handle, 427, 453  
  heightStyle, 365  
  helper, 427  
  hoverClass, 437  
  icons, 365, 370, 386, 390  
  items, 454  
  max, 308  
  maxDate, 379  
  maxLength  
  min, 308  
  minDate, 380



- minLength, 307, 401
- monthNames, 378
- numberOfMonths, 378
- opacity, 428, 454
- placeholder, 454
- position, 370, 386
- range, 308
- rangelength, 307
- revert, 428
- revertDuration, 429
- scope, 429, 437
- snap, 429
- snapMode, 430
- snapTolerance, 430
- source, 401
- text, 391
- tolerance, 438
- width, 385
- yearRange, 380
- zIndex, 430, 446
- opcje
  - etykietek ekranowych, 348
  - formularza, 293
  - widżetu Autocomplete, 400
  - widżetu Draggable, 424
  - widżetu Droppable, 436
  - widżetu Sortable, 451
  - zestawów kart, 354
- operacje matematyczne, 68
- operator
  - !=, 96
  - !=", 96
  - \*=, 72
  - /=, 72
  - ++, 72
  - +=, 72
  - <, 96
  - <=, 96
  - =, 72
  - ==, 96
  - ===, 96
  - >, 96
  - >=, 96
  - LUB, 103
  - NIE, 103
  - trójargumentowy, 602
- optymalizacja selektorów, 547
- organizacja W3C, 23
- oszczędzanie miejsca, 363
- otwieranie konsoli, 621
- oznaczanie zadań, 519, 531

## P

- panele treści, 353
- pasek nawigacyjny, 371
- pętla
  - do-while, 114
  - for, 112
  - while, 109

- pętle automatyczne, 155
- plik
  - accordion.html, 366
  - advanced\_tooltips.html, 350
  - airports.js, 395, 396, 404
  - birthdate.html, 381
  - callback.html, 224
  - console.html, 625
  - debugger.html, 633
  - events\_intro.html, 185
  - faq.html, 205
  - flickr.html, 512
  - form.html, 295, 402, 404
  - gallery.html, 251
  - index.html, 534
  - interactions.css, 447
  - jquery.js, 301
  - jquery-ui.min.js, 329
  - load.html, 483
  - login.html, 217, 496
  - menu.html, 274
  - open\_external.js, 260
  - panel1.html, 361
  - products.php, 399
  - rollover.html, 245, 246
  - sm-core.css, 273
  - tabs.html, 356
  - todo.js, 524, 536
  - tooltips.html, 347
- pliki
  - .js, 44
  - JavaScript, 42, 49, 606
  - jQuery, 138
- pobieranie
  - czasu, 594
  - danych, 398
  - elementów strony, 147, 183
  - informacji, 74
  - kodu XML, 495
  - miesiąca, 593
  - odnośników, 184, 255
  - odpowiedzi, 478
  - pliku jQuery, 138
- podgląd źródła strony, 161
- podmienianie
  - obrazków, 241
  - rysunków, 239
- podpowiedzi, 395
- podwzorce, 586
- pole
  - tekstowe, 290
  - wyboru, 284, 534
- polecenie
  - alert(), 199
  - prompt(), 124
- porównywanie wartości, 105
- pozycjonowanie bezwzględne, 215

prezentacja  
  danych, 511  
  informacji, 345  
  JSONP, 506  
  obsługi zdarzeń, 185  
problemy, 611  
program  
  Aptana Studio, 27  
  Atom, 27  
  BBEdit, 27  
  Brackets, 26  
  CoffeeCup Free HTML Editor, 26  
  Dreamweaver, 28  
  Eclipse, 27  
  EditPlus, 27  
  Emacs, 27  
  HTML-Kit, 26  
  Notepad++, 26  
  SublimeText, 27  
  TextWrangler, 26  
  ThemeRoller, 327, 407  
  Vim, 27  
programowanie komputerowe, 38  
programy, 40  
  bezpłatne, 26  
  komercyjne, 27  
  reagujące inteligentnie, 93  
projekt jQuery UI, 144  
przeciąganie i upuszczanie, 94  
przeglądanie  
  błędów, 623  
  informacji, 473  
przeglądarka, 474  
  Chrome, 52, 53  
  Firefox, 56  
  Internet Explorer, 55  
  Safari, 57  
przejścia CSS3, 231, 232  
przekazywanie  
  funkcji do zdarzenia, 183  
  obsługi zdarzenia, 202  
  zdarzeń, 197  
przesłanie stylów, 415  
przesuwanie  
  elementów, 216  
  znacznika <div>, 225  
przesyłanie  
  danych, 292  
  formularzy, 473  
przetwarzanie danych, 490  
przezroczystość, 232  
przyciski, 389  
przypisywanie zdarzenia, 183  
pseudoklasa  
  active, 232  
  hover, 232, 534  
pusta instrukcja if, 207

## R

reakcje na zdarzenia, 195  
referencje do okien, 263  
reguły walidacji, 303  
  zaawansowane, 306  
responsywne menu nawigacyjne, 270  
  kod CSS, 273  
  kod HTML, 270  
  kod JavaScript, 273  
rodzaje  
  adresów URL, 45  
  błędów, 615  
rozwiązywanie problemów, 611

## S

sekcja  
  nagłówkowa, 22  
  Theme, 327  
selektor, 148  
  #gallery, 251  
  #mainMenu, 372  
  hidden, 208  
selektory  
  atrybutów, 152  
  CSS, 150  
  dzieci, 151  
  elementów, 149  
  elementów potomnych, 151  
  elementów sąsiadujących, 152  
  identyfikatorów, 148  
  klas, 150  
  zaawansowane, 151  
serwer  
  CDN, 137  
  WWW, 475, 476  
serwis GitHub, 267  
siatka pól, 427  
silnik zarządzania układem, 40  
składnia języka, 39  
skrótów klawiaturowe, 31  
skrypt JavaScript, 475  
skrypty po stronie  
  klienta, 41  
  serwera, 41  
słowa  
  kluczowe, 65  
  zarezerwowane, 65, 617  
słowo kluczowe  
  if, 95  
  this, 169  
sortowanie elementów strony, 449  
sprawdzanie  
  kilku warunków, 98  
  stanu przycisków, 284  
  warunków, 102  
  wprowadzonych danych, 94  
  występowania liczb, 589

- stan przycisków, 284
- stosowanie
  - elementów tablicy, 79
  - funkcji `css()`, 554
  - instrukcji warunkowych, 105
  - jQuery UI, 327
  - komentarzy, 89
  - liczb, 587
  - łańcuchów znaków, 565
  - metody `$.getJSON()`, 510
  - okien dialogowych, 331
  - osobnych etykiet, 397
  - osobnych wartości, 397
  - słów zarezerwowanych, 617
  - tablicy danych, 395
  - typów danych, 67
  - widżetu `Draggable`, 435
  - widżetu `Sortable`, 449
  - wtyczek jQuery, 268
  - wyrażeń regularnych, 571
  - zmiennych, 66, 67, 72
- strona dokumentacji, 553
- struktura
  - funkcji, 116
  - galerii fotografii, 250
- styl, 24
  - komunikatów, 310
  - widżetów, 418
- stylowe przyciski, 389

## Ś

- ścieżka
  - bezwzględna, 45
  - do zewnętrznego pliku, 618
  - względna, 45
- śledzenie działania skryptu, 623

## T

- tablica
  - danych, 395
  - obiektów, 397
  - współrzędnych, 425
- tablice, 77, 111
  - dodawanie elementów, 80
  - tworzenie, 78
  - usuwanie elementów, 82
  - zapisywanie danych, 83
- technika
  - JSONP, 506
  - przeciągnij i upuść, 443
- techniki języka JavaScript, 565
- technologia AJAX, 471
- tematy graficzne jQuery UI, 383
- tempo animacji, 221, 465
- testowanie
  - aplikacji, 621
  - wyrażeń regularnych, 585

- ThemeRoller
  - Clickable items, 411
  - Content, 411
  - Corner Radius, 410
  - Drop shadows, 412
  - Error, 412
  - Font settings, 409
  - Header/Toolbar, 410
  - Highlight, 412
  - Kolor ikon, 411
  - Kolor obramowania, 411
  - Kolor tekstu, 411
  - Kolor tła, 410
  - Modal Screen for overlays, 412
  - Nieprzezroczystość tekstury tła, 411
  - Tekstura tła, 410
- tworzenie
  - adresu URL, 508
  - akordeonu, 366
  - aplikacji, 519
  - daty, 597, 598
  - interfejsu użytkownika, 326
  - kodu JavaScript, 609
  - kolejek, 82
  - komunikatów, 72
  - liczb losowych, 591
  - nowych okien, 260
  - okna dialogowego, 332
  - paska nawigacyjnego, 371
  - skryptów, 479
  - sliderów, 266
  - tablic, 78
  - wyrażeń regularnych, 572
  - zmiennych, 63
- typy danych, 60

## U

- uciekający element pływający, 373
- ukośnik, 31
- ukrywanie
  - opcji formularza, 293
  - pól, 298
- umieszczanie wskaźnika myszy, 192
- URL, Uniform Resource Locator, 45
- ustawianie atrybutów znaczników, 160
- usuwanie
  - atrybutów HTML, 166
  - elementów, 160
  - elementów z tablicy, 82
  - wskaźnika myszy, 192
  - zadań, 520, 536, 539
  - zdarzeń, 196
- używanie, *Patrz* stosowanie

## W

- W3C, World Wide Web Consortium, 146
- walidacja, 302
  - formularzy, 94, 299
  - pól wyboru, 316
  - prosta, 312
  - przycisków opcji, 316
  - strony, 23
  - z wykorzystaniem serwera, 309
  - zaawansowana, 305, 313
- walidator kodu HTML, 258
- wartości
  - elementów formularzy, 283
  - logiczne, 62, 99
- wartość \$(this), 203
- wczytywanie rysunków, 242
- wersje biblioteki jQuery, 138, 140
- węzeł, node, 146
- widżet, 324
  - Autocomplete, 394–396, 400
  - automatycznego uzupełniania, 394
  - Datepicker jQuery UI, 375
  - Dialog, 333
  - Draggable, 421
    - opcje, 424
    - zastosowanie, 423
    - zdarzenia, 430
  - Droppable, 434
    - opcje, 436
    - stosowanie, 435
    - zdarzenia, 438
  - kalendarza, 377, 383
  - przycisku, 389, 390
  - Selectmenu, 368, 388
  - Sortable, 449
    - metody, 458
    - opcje, 451
    - stosowanie, 449
    - zdarzenia, 455
  - Tabs, 354
- widżety
  - okna dialogowego, 343
  - usprawniające formularze, 401
  - Wijmo UI, 326
- wielkość znaków, 618
- właściwości
  - CSS, 554
  - kalendarzy, 377
  - list rozwijanych, 385
  - okien, 261
  - okna dialogowego, 333
- właściwość
  - active, 354
  - animation-play-state, 236
  - collapsible, 355
  - content, 349
  - contentEditable, 538
  - disabled, 426
  - draggable, 333
  - easing, 466
  - event, 355
  - height, 262, 334
  - heightStyle, 355
  - hide, 335, 348, 354
  - left, 262
  - location, 262
  - location.hostname, 259
  - menubar, 263
  - modal, 335
  - my, 343
  - placeholder, 456
  - of, 343
  - opacity, 232
  - position, 215, 336, 348
  - resizable, 334
  - responseXML, 478
  - scrollbars, 262
  - show, 335, 348, 354
  - status, 262
  - toolbar, 262
  - tooltipClass, 348
  - top, 262
  - track, 348
  - transition, 233
  - ui.helper, 432, 439, 455
  - ui.item, 455
  - ui.item.index, 387
  - ui.item.label, 387
  - ui.item.value, 387
  - ui.offset, 433, 440, 456
  - ui.originalPosition, 433, 440, 456
  - ui.position, 433, 440, 456
  - ui.sender, 456
  - width, 262, 334
  - zdarzenia, 195
- włączanie pól, 291
- wskaźnik myszy, 192, 425
- wstępne wczytywanie rysunków, 242
- wstrzymywanie przekazywania zdarzeń, 197
- wtyczka
  - jPanel, 278
  - Multi-level Push Menu, 278
  - SmartMenus, 270, 277
  - Validation, 301, 305, 318
- wtyczki
  - jQuery, 265
  - jQueryWidgets, 326
  - Kendo UI, 326
- wydajność kodu JavaScript, 599
- wygląd przycisków, 391
- wykrywanie błędów, 51
- wyłączanie pól, 291, 295
- wrażenia regularne, 571, 573, 577
  - adresy e-mail, 579
  - adresy stron WWW, 581
  - daty, 580
  - kod pocztowy, 577

- numer telefonu stacjonarnego, 578
- symbole, 572
- testowanie, 585
- zastępowanie tekstów, 585
- wyrażenie
  - \$(), 544
  - \$(this), 170, 252, 448, 533, 536, 545
  - answers.length, 637
- wysuwany formularz logowania, 216
- wysyłanie żądania, 478
- wyświetlanie
  - danych HTML, 472
  - komunikatów, 330, 626
  - opcji formularza, 293
- wzbogacanie formularza, 294

## Z, ǀ

- zagnieżdżanie instrukcji warunkowych, 104
- zakończanie liczb, 589
- zapisywanie
  - danych, 83
  - listy, 539
  - pobranych elementów, 544
  - na serwerze, 539
  - ustawień, 600, 601
- zarządzanie zdarzeniami, 197
- zasada szczególności, 416
- zasięg zmiennych, 123, 620
- zastępowanie
  - elementów, 160
  - tekstów, 585, 586
- zastosowania jQuery, 239
- zastosowanie
  - indexOf(), 567
  - metody \$.get(), 495
  - metody load(), 482
  - walidacji, 311
  - widżetu Draggable, 423
  - wyrażeń regularnych, 574
- zdarzenia, 177
  - niestandardowe jQuery UI, 357
  - specyficzne, 190
  - usuwanie, 196
  - widżetu Draggable, 430
  - widżetu Droppable, 438
  - widżetu Sortable, 455
  - związane z dokumentem i oknem, 180
  - związane z formularzami, 181, 285
  - związane z klawiaturą, 182
  - związane z myszą, 179
- zdarzenie
  - activate, 441, 456
  - beforestop, 457
  - blur, 181, 288
  - create, 455
  - change, 181, 289, 457
  - click, 179, 289, 531
  - create, 431
  - dblclick, 179

- deactivate, 441, 457
- drag, 433
- drop, 439, 447
- focus, 181, 286
- keydown, 182
- keypress, 182
- keyup, 182, 627
- loa, 190
- load, 180, 190
- mousedown, 179
- mousemove, 180
- mouseout, 180, 193
- mouseover, 179, 193, 196
- mouseup, 179
- out, 442, 457
- over, 442, 457
- receive, 457
- remove, 457
- reset, 181
- resize, 180
- scroll, 180
- sort, 456
- start, 432, 444, 455
- stop, 434, 445, 457
- submit, 181, 195, 285
- unload, 181
- update, 457
- zestawy kart, 351
- zewnętrzne pliki JavaScript, 42
- zmiana
  - atributu src, 240
  - wartości zmiennych, 71
  - wielkości znaków, 566
  - właściwości CSS, 164
  - wyglądu elementu, 534
- zmiennie, 63, 620
- znacznik
  - <a>, 22, 555
  - <body>, 22
  - <div>, 146, 177, 183, 226–230, 245
  - <form>, 279
  - <html>, 22, 146
  - <img>, 243, 555
  - <input>, 281
  - <li>, 201, 272, 526, 532
  - <p>, 22, 218
  - <script>, 40, 42, 107, 635
  - <span>, 172, 516, 526, 532
  - <strong>, 22
  - <td>, 600
  - <ul>, 170, 202, 370
  - końcowy, 22
  - początkowy, 22
- znak
  - apostrofu, 616
  - cudzysłowu, 62, 616
  - równości, 617
- znaki
  - karetki, 67
  - tabulacji, 67
- źródła informacji, 643



# PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄZKI**
- 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**

# JavaScript i jQuery

## nieoficjalny podręcznik

JavaScript ma za sobą długą historię, w której bywały okresy lepsze i gorsze. Czasem język ten był wręcz masowo blokowany w przeglądarkach. Jednak te czasy minęły. W tej chwili nie obejdziesz się bez niego żadna poważna aplikacja internetowa lub choć trochę bardziej zaawansowana strona WWW. Użytkownicy serwisów internetowych wymuszają na projektantach coraz nowsze i lepsze rozwiązania. Dlatego na rynku wciąż pojawiają się dodatkowe narzędzia dla języka JavaScript, które ułatwiają wykorzystanie jego potencjału.

Najpopularniejszym dodatkiem tego typu jest biblioteka jQuery. Genialna w swojej prostocie, z ogromnymi możliwościami, zdobyła uznanie wszystkich programistów JavaScriptu. Nie potrafią sobie oni wyobrazić programowania bez jej wykorzystania. W tej książce znajdziesz najlepsze techniki, jakie oferuje JavaScript. Nauczysz się nawigować po drzewie DOM, modyfikować zachowanie elementów oraz obsługiwać zdarzenia. Poznasz również narzędzia, które ułatwią Ci pracę oraz debugowanie kodu. Jest to obowiązkowa pozycja dla każdego projektanta stron internetowych. Musisz ją mieć!

Dzięki tej książce:

- poznasz podstawy języka JavaScript
- zobaczysz, jak jQuery potrafi ułatwić pracę z JavaScriptem
- opanujesz mechanizm zdarzeń w jQuery
- zbudujesz lepszą i ciekawszą witrynę!

Twórz atrakcyjne strony WWW.

Wzbogać kod HTML o animacje, interaktywność i dynamiczne efekty wizualne!

**helion.pl**  
księgarnia  
internetowa

Nr katalogowy: 8754



Księgarnia internetowa:

<http://helion.pl>



Zamówienia telefoniczne:

**0 801 339900**



**0 601 339900**



**Helion**

Sprawdź najnowsze promocje:

• <http://helion.pl/promocje>

Książki najchętniej czytane:

• <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

• <http://helion.pl/nowosci>

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel.: 32 230 98 63

e-mail: [helion@helion.pl](mailto:helion@helion.pl)

<http://helion.pl>



KOD KORZYŚCI

ISBN 978-83-246-4381-3



Cena 79,00 zł

Informatyka w najlepszym wydaniu