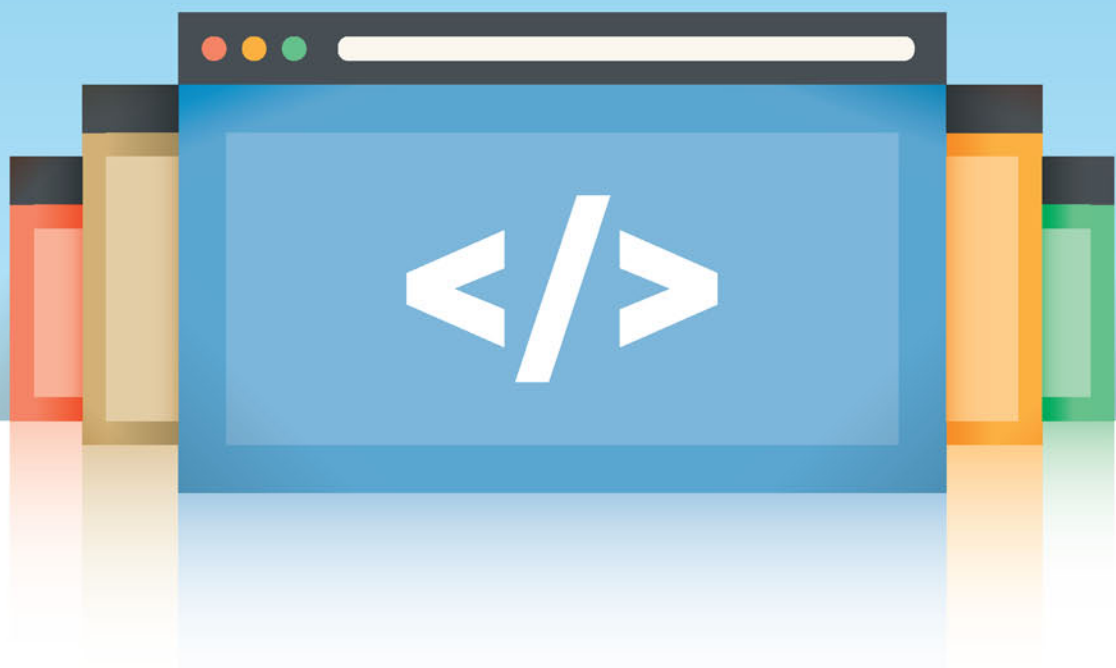


Michael Mikowski, Josh Powell

Single Page Web Applications

Programowanie aplikacji internetowych
z JavaScript



Niesamowite aplikacje internetowe!

Helion 

Tytuł oryginału: Single Page Web Applications: JavaScript end-to-end

Tłumaczenie: Lech Lachowski

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

ISBN: 978-83-283-0521-2

Original edition copyright © 2014 by Manning Publications Co.
All rights reserved.

Polish edition copyright © 2015 by HELION SA.
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
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

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

Printed in Poland.

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

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

Spis treści

<i>Przedmowa</i>	11
<i>Wstęp</i>	13
<i>Podziękowania</i>	15
<i>O książce</i>	17

CZĘŚĆ I. WPROWADZENIE DO APLIKACJI SPA 21

Rozdział 1. Pierwsza aplikacja jednostronicowa 23

1.1.	Definicja, trochę historii oraz nasze cele	24
1.1.1.	Trochę historii	24
1.1.2.	Dlaczego musieliśmy tak długo czekać na aplikacje SPA JavaScript?	25
1.1.3.	Cele	28
1.2.	Budowanie pierwszej aplikacji SPA	29
1.2.1.	Definiowanie celu	30
1.2.2.	Zainicjowanie struktury plików	30
1.2.3.	Konfigurowanie narzędzi dla programistów przeglądarki Google Chrome	31
1.2.4.	Projektowanie kodów HTML i CSS	31
1.2.5.	Dodawanie kodu JavaScript	33
1.2.6.	Sprawdzanie działania aplikacji za pomocą narzędzi dla programistów przeglądarki Google Chrome	38
1.3.	Korzyści dla użytkownika z dobrze napisanej aplikacji SPA	42
1.4.	Podsumowanie	43

Rozdział 2. JavaScript wraca do łask 45

2.1.	Zakres zmiennej	47
2.2.	Wynoszenie zmiennej	49
2.3.	Zaawansowane wynoszenie zmiennej i obiekt kontekstu wykonywania	51
2.3.1.	Wynoszenie	51
2.3.2.	Kontekst wykonywania i obiekt kontekstu wykonywania	52
2.4.	Łańcuch zakresów	56
2.5.	Obiekty i łańcuch prototypów języka JavaScript	59
2.5.1.	Łańcuch prototypów	62
2.6.	Głębsze spojrzenie na funkcje	67
2.6.1.	Funkcje oraz funkcje anonimowe	68
2.6.2.	Samowykonujące się funkcje anonimowe	68
2.6.3.	Wzorzec modułu — wprowadzanie do języka JavaScript zmiennych prywatnych	71
2.6.4.	Domknięcia	76
2.7.	Podsumowanie	79

CZĘŚĆ II. KLIENT APLIKACJI SPA 81**Rozdział 3. Projektowanie powłoki 83**

- 3.1. Gruntowne zrozumienie powłoki 84
- 3.2. Konfigurowanie plików i przestrzeni nazw 85
 - 3.2.1. Tworzenie struktury plików 85
 - 3.2.2. Pisanie kodu HTML aplikacji 86
 - 3.2.3. Tworzenie głównej przestrzeni nazw CSS 87
 - 3.2.4. Tworzenie głównej przestrzeni nazw JavaScript 89
- 3.3. Tworzenie kontenerów funkcji 90
 - 3.3.1. Wybór strategii 90
 - 3.3.2. Pisanie kodu HTML powłoki 91
 - 3.3.3. Pisanie kodu CSS powłoki 92
- 3.4. Renderowanie kontenerów funkcji 95
 - 3.4.1. Konwersja HTML na JavaScript 95
 - 3.4.2. Dodawanie szablonu HTML do JavaScript 96
 - 3.4.3. Pisanie arkusza stylów powłoki 97
 - 3.4.4. Nakazanie aplikacji, aby korzystała z powłoki 99
- 3.5. Zarządzanie kontenerami funkcji 100
 - 3.5.1. Pisanie metody rozwijania i zwijania suwaka czatu 101
 - 3.5.2. Dodanie procedury obsługi zdarzeń kliknięcia suwaka czatu 103
- 3.6. Zarządzanie stanem aplikacji 107
 - 3.6.1. Zachowania przeglądarki oczekiwane przez użytkowników 107
 - 3.6.2. Wybór strategii zarządzania kontrolkami historii 108
 - 3.6.3. Zmiana kotwicy przy wystąpieniu zdarzenia historii 109
 - 3.6.4. Użycie kotwicy do sterowania stanem aplikacji 111
- 3.7. Podsumowanie 116

Rozdział 4. Dodawanie modułów funkcji 119

- 4.1. Strategia modułu funkcji 120
 - 4.1.1. Porównanie z modułami zewnętrznymi 121
 - 4.1.2. Moduły funkcji i fraktalowy wzorzec MVC 123
- 4.2. Konfigurowanie plików modułu funkcji 125
 - 4.2.1. Planowanie struktury plików 125
 - 4.2.2. Zapewnianie plików 126
 - 4.2.3. Co mamy napisane 132
- 4.3. Projektowanie interfejsów API metod 132
 - 4.3.1. Wzorzec interfejsu kotwicy 133
 - 4.3.2. Interfejsy API konfiguracji Czatu 134
 - 4.3.3. Interfejs API inicjowania Czatu 135
 - 4.3.4. Interfejs API metody `setSliderPosition` Czatu 136
 - 4.3.5. Kaskada konfiguracji i inicjowania 137
- 4.4. Implementowanie interfejsu API funkcji 139
 - 4.4.1. Arkusze stylów 139
 - 4.4.2. Modyfikacja Czatu 144
 - 4.4.3. Czyszczenie powłoki 149
 - 4.4.4. Prześledzenie wykonywania aplikacji 154

- 4.5. Dodanie często potrzebnych metod 156
 - 4.5.1. *Metoda removeSlider* 157
 - 4.5.2. *Metoda handleResize* 158
- 4.6. Podsumowanie 162

Rozdział 5. Budowanie Modelu 163

- 5.1. Czym jest Model 164
 - 5.1.1. *Co zamierzamy zbudować* 165
 - 5.1.2. *Zadania Modelu* 166
 - 5.1.3. *Czego Model nie robi* 167
- 5.2. Konfigurowanie Modelu oraz innych plików 168
 - 5.2.1. *Planowanie struktury plików* 168
 - 5.2.2. *Wypełnianie plików* 169
 - 5.2.3. *Zastosowanie ujednoliconej biblioteki do obsługi wprowadzania danych za pomocą dotyku i myszy* 174
- 5.3. Projektowanie obiektu people 175
 - 5.3.1. *Projektowanie obiektów person* 176
 - 5.3.2. *Projektowanie interfejsu API obiektu people* 178
 - 5.3.3. *Dokumentowanie interfejsu API obiektu people* 181
- 5.4. Budowanie obiektu people 182
 - 5.4.1. *Tworzenie listy symulowanych osób* 182
 - 5.4.2. *Rozpoczęcie budowy obiektu people* 185
 - 5.4.3. *Dokończenie obiektu people* 189
 - 5.4.4. *Testowanie interfejsu API obiektu people* 195
- 5.5. Implementacja logowania i wylogowania w powłoce 197
 - 5.5.1. *Projektowanie wrażeń użytkownika podczas logowania* 198
 - 5.5.2. *Aktualizacja kodu JavaScript powłoki* 199
 - 5.5.3. *Aktualizacja arkusza stylów powłoki* 201
 - 5.5.4. *Testowanie procesów logowania i wylogowania za pomocą interfejsu użytkownika* 202
- 5.6. Podsumowanie 203

Rozdział 6. Dokończenie budowy modułów modelu i danych 205

- 6.1. Projektowanie obiektu chat 206
 - 6.1.1. *Projektowanie metod i zdarzeń* 206
 - 6.1.2. *Dokumentowanie interfejsu API obiektu chat* 209
- 6.2. Budowanie obiektu chat 210
 - 6.2.1. *Rozpoczęcie budowy obiektu chat przez utworzenie metody join* 210
 - 6.2.2. *Aktualizacja Atrapy, aby reagowała na metodę chat.join* 213
 - 6.2.3. *Testowanie metody chat.join* 214
 - 6.2.4. *Dodanie do obiektu chat obsługi przesyłania wiadomości* 216
 - 6.2.5. *Aktualizacja Atrapy w celu emulowania komunikatów* 220
 - 6.2.6. *Testowanie obsługi komunikatów dla obiektu chat* 222
- 6.3. Dodanie obsługi Awatara do Modelu 224
 - 6.3.1. *Dodanie obsługi Awatara do obiektu chat* 224
 - 6.3.2. *Modyfikacja Atrapy w celu emulacji awatarów* 225
 - 6.3.3. *Testowanie obsługi awatarów* 226
 - 6.3.4. *Technika projektowania sterowanego testami* 227

- 6.4. Dokończenie modułu funkcji czatu 229
 - 6.4.1. Aktualizacja kodu JavaScript Czatu 230
 - 6.4.2. Aktualizacja arkusza stylów 237
 - 6.4.3. Testowanie interfejsu użytkownika Czatu 241
- 6.5. Tworzenie modułu funkcji awatara 242
 - 6.5.1. Utworzenie pliku JavaScript Awatara 243
 - 6.5.2. Tworzenie arkusza stylów Awatara 248
 - 6.5.3. Aktualizacja powłoki i dokumentu przeglądarkowego 249
 - 6.5.4. Testowanie modułu funkcji awatara 250
- 6.6. Wiązanie danych oraz jQuery 250
- 6.7. Tworzenie modułu danych 252
- 6.8. Podsumowanie 255

CZĘŚĆ III. SERWER APLIKACJI SPA 257

Rozdział 7. Serwer WWW 259

- 7.1. Rola serwera 259
 - 7.1.1. Uwierzytelnianie i autoryzacja 260
 - 7.1.2. Walidacja 260
 - 7.1.3. Przechowywanie i synchronizacja danych 261
- 7.2. Node.js 261
 - 7.2.1. Dlaczego Node.js? 262
 - 7.2.2. Tworzenie aplikacji „Witaj, świecie!” za pomocą serwera Node.js 262
 - 7.2.3. Instalowanie frameworku Connect i korzystanie z niego 266
 - 7.2.4. Dodawanie funkcji pośredniczących frameworku Connect 267
 - 7.2.5. Instalowanie frameworku Express i korzystanie z niego 268
 - 7.2.6. Dodawanie funkcji pośredniczących frameworku Express 271
 - 7.2.7. Korzystanie z różnych środowisk za pomocą frameworku Express 272
 - 7.2.8. Serwowanie plików statycznych za pomocą frameworku Express 273
- 7.3. Zaawansowany routing 274
 - 7.3.1. Trasy operacji CRUD obiektu user 275
 - 7.3.2. Ogólny routing operacji CRUD 281
 - 7.3.3. Umieszczenie routingu w osobnym module Node.js 284
- 7.4. Dodanie uwierzytelniania i autoryzacji 288
 - 7.4.1. Uwierzytelnianie podstawowe 288
- 7.5. Technologia WebSocket i biblioteka Socket.IO 289
 - 7.5.1. Prosta aplikacja Socket.IO 290
 - 7.5.2. Socket.IO oraz serwery wymiany komunikatów 293
 - 7.5.3. Aktualizacja kodu JavaScript z wykorzystaniem Socket.IO 294
- 7.6. Podsumowanie 297

Rozdział 8. Baza danych serwera 299

- 8.1. Rola bazy danych 300
 - 8.1.1. Wybór magazynu danych 300
 - 8.1.2. Eliminacja transformacji danych 300
 - 8.1.3. Przeniesienie logiki w wymagane miejsce 301

- 8.2. Wprowadzenie do MongoDB 302
 - 8.2.1. Przechowywanie dokumentowe 303
 - 8.2.2. Dynamiczna struktura dokumentów 303
 - 8.2.3. Pierwsze kroki z MongoDB 304
- 8.3. Użycie sterownika MongoDB 305
 - 8.3.1. Przygotowanie plików projektu 306
 - 8.3.2. Instalacja i podłączenie do MongoDB 307
 - 8.3.3. Zastosowanie metod CRUD bazy danych MongoDB 308
 - 8.3.4. Dodanie operacji CRUD do aplikacji serwera 311
- 8.4. Walidacja danych klienta 315
 - 8.4.1. Walidacja typu obiektu 316
 - 8.4.2. Walidacja obiektu 317
- 8.5. Tworzenie osobnego modułu CRUD 325
 - 8.5.1. Przygotowanie struktury plików 326
 - 8.5.2. Przeniesienie logiki operacji CRUD do osobnego modułu 329
- 8.6. Budowanie modułu czatu 334
 - 8.6.1. Rozpoczęcie budowy modułu czatu 334
 - 8.6.2. Tworzenie procedury obsługi komunikatu adduser 337
 - 8.6.3. Tworzenie procedury obsługi komunikatu updatechat 341
 - 8.6.4. Tworzenie procedur obsługi komunikatów rozłączania 343
 - 8.6.5. Tworzenie procedury obsługi komunikatu updateavatar 345
- 8.7. Podsumowanie 348

Rozdział 9. Przygotowanie aplikacji SPA do pracy w środowisku produkcyjnym 349

- 9.1. Optymalizacja naszej aplikacji SPA dla wyszukiwarek 350
 - 9.1.1. Jak Google indeksuje aplikację SPA 350
- 9.2. Chmura i usługi zewnętrzne 354
 - 9.2.1. Analityka stron WWW 354
 - 9.2.2. Rejestrowanie błędów po stronie klienta 356
 - 9.2.3. Systemy CDN 358
- 9.3. Buforowanie i pomijanie pamięci podręcznej 359
 - 9.3.1. Możliwości buforowania 359
 - 9.3.2. Web Storage 360
 - 9.3.3. Buforowanie HTTP 362
 - 9.3.4. Buforowanie serwera 365
 - 9.3.5. Buforowanie zapytań bazy danych 371
- 9.4. Podsumowanie 372

DODATKI 375

Dodatek A Standard kodowania JavaScript 377

- A.1. Dlaczego potrzebujemy standardu kodowania 378
- A.2. Układ kodu i komentarze 379
- A.3. Nazwy zmiennych 389
- A.4. Deklarowanie i przypisywanie zmiennych 398
- A.5. Funkcje 400

- A.6. Przestrzenie nazw 402
- A.7. Nazwy i układ plików 403
- A.8. Składnia 404
- A.9. Walidacja kodu 408
- A.10. Szablon dla modułów 411
- A.11. Podsumowanie 413

Dodatek B Testowanie aplikacji SPA 415

- B.1. Konfiguracja trybów testowych 416
- B.2. Wybór frameworku testowego 420
- B.3. Konfiguracja frameworku nodeunit 421
- B.4. Tworzenie zestawu testów 422
- B.5. Dostosowanie modułów SPA do testów 441
- B.6. Podsumowanie 444

Skorowidz 445

Projektowanie powłoki

W tym rozdziale omawiamy następujące zagadnienia:

- moduł powłoki i jego miejsce w naszej architekturze;
- strukturyzacja plików i przestrzeni nazw;
- tworzenie i stylizacja kontenerów funkcji;
- używanie procedury obsługi zdarzeń do przełączania kontenerów funkcji;
- stosowanie wzorca interfejsu kotwicy do zarządzania stanem aplikacji.

W tym rozdziale omówiona zostanie **powłoka** (ang. *shell*), która jest wymaganym komponentem naszej architektury. Zaprojektujemy układ strony, który zawiera kontenery funkcji, a następnie dostosujemy powłokę do ich renderowania. Potem pokażemy, w jaki sposób powłoka zarządza kontenerami funkcji, wykorzystując ją do rozwijania i zwijania suwaka czatu. Następnie powłoka będzie przechwytywać zdarzenie kliknięcia przez użytkownika, aby otworzyć i zamknąć suwak. Na koniec użyjemy kotwicy adresu URI jako API stanu z zastosowaniem wzorca interfejsu kotwicy (ang. *anchor interface*). Zapewnia to użytkownikom oczekiwane kontrolki przeglądarki, takie jak przyciski *Wstecz* i *Dalej*, historia przeglądania oraz zakładki.

W rozdziale zbudujemy fundamenty dla skalowalnej i zarządzalnej aplikacji SPA. Nie wybiegajmy jednak zbyt daleko w przyszłość. Najpierw musimy zrozumieć, czym jest powłoka.

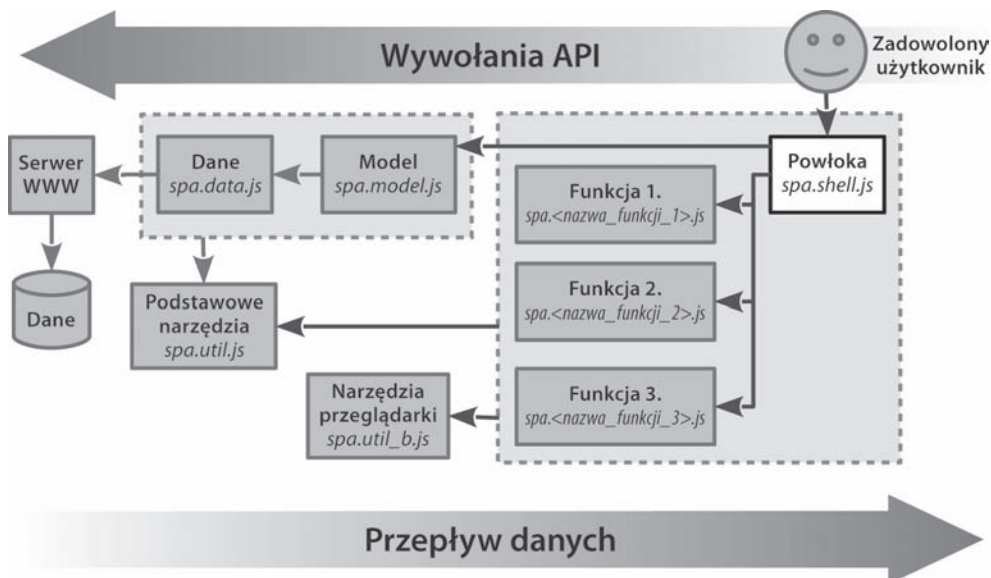
3.1. Gruntowne zrozumienie powłoki

Powłoka jest kontrolerem głównym dla naszej aplikacji SPA i jest niezbędna w architekturze. Rolę modułu powłoki możemy porównać do powłoki samolotu.

Powłoka samolotu (zwana też nadwoziem samonośnym lub płatowcem) zapewnia kształt i strukturę pojazdu. Zespoły takie jak siedzenia, składane stoliki i silniki są doczepiane do niej za pomocą różnych elementów mocujących. Wszystkie zespoły są zbudowane w taki sposób, aby działać możliwie niezależnie, ponieważ nikt nie lubi, gdy ciotka Milly rozkładając swój stolik, powoduje, że samolot gwałtownie przechyla się mocno na prawą stronę.

Moduł powłoki (ang. *shell module*) zapewnia kształt i strukturę naszej aplikacji. Moduły funkcji takie jak czat, logowanie oraz nawigacja są „doczepiane” do powłoki za pomocą interfejsów API. Wszystkie moduły funkcji są zbudowane w taki sposób, aby działały możliwie niezależnie, ponieważ nikt nie lubi, gdy ciotka Milly wpisując w polu wiadomości suwaka czatu "ROTFLMAO!!! PWNED, mam Cię!", powoduje, że aplikacja natychmiast zamyka okno przeglądarki.

Powłoka jest tylko jednym z elementów architektury, które udoskonaliśmy w trakcie pracy nad wieloma projektami komercyjnymi. Ta architektura i umiejscowienie w niej powłoki zostały przedstawione na rysunku 3.1. Lubimy pisać powłokę na początku, ponieważ jest centralna dla architektury. Koordynuje **moduły funkcji** (ang. *feature modules*) z logiką biznesową i uniwersalnymi interfejsami przeglądarek, takimi jak URI lub ciasteczka (*cookie*). Gdy użytkownik klika przycisk *Wstecz*, loguje się lub wykonuje dowolną inną czynność, która zmienia możliwy do zapisania w zakładkach stan aplikacji, powłoka koordynuje te zmiany.



Rysunek 3.1. Powłoka w naszej architekturze aplikacji SPA

Czytelnicy zaznajomieni z architekturą Model-Widok-Kontroler (ang. *Model-View-Controller* — MVC) mogą potraktować powłokę jak główny kontroler, ponieważ koordynuje ona kontrolery wszystkich podległych modułów funkcji.

Powłoka jest odpowiedzialna za następujące kwestie:

- renderowanie kontenerów funkcji i zarządzanie nimi;
- zarządzanie stanem aplikacji;
- koordynowanie modułów funkcji.

Koordinacja modułów funkcji zostanie omówiona szczegółowo w następnym rozdziale. W tym rozdziale opisano renderowanie kontenerów funkcji i zarządzanie stanem aplikacji. Najpierw przygotujemy nasze pliki i przestrzenie nazw.

3.2. Konfigurowanie plików i przestrzeni nazw

Skonfigurujemy nasze pliki i przestrzenie nazw zgodnie ze standardami kodowania opisanymi w dodatku A. W szczególności będziemy mieć jeden plik JavaScript dla każdej przestrzeni nazw JavaScript i będziemy używać samowykonujących się funkcji anonimowych, aby nie dopuścić do zanieczyszczenia globalnej przestrzeni nazw. Będziemy także konfigurować pliki CSS w strukturze równoległej. Taka konwencja przyspiesza projektowanie, poprawia jakość i ułatwia konserwację. Jej wartość wzrasta wraz z dodawaniem do projektu kolejnych modułów i kolejnych programistów.

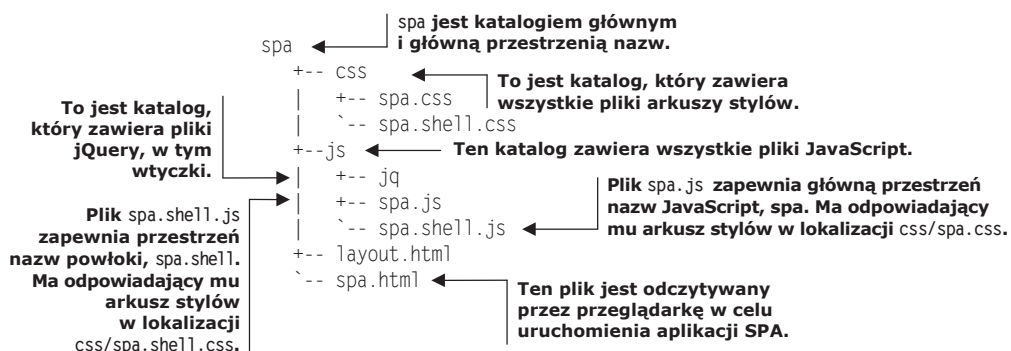
3.2.1. Tworzenie struktury plików

Wybraliśmy spa dla głównej przestrzeni nazw aplikacji. Synchronizujemy nazwy plików JavaScript i CSS, przestrzeń nazw JavaScript oraz nazwy selektorów CSS. Ułatwia to śledzenie, który plik JavaScript odpowiada któremu plikowi CSS.

PLANOWANIE KATALOGÓW I PLIKÓW

Programiści aplikacji internetowych często umieszczają swój plik HTML w pewnym katalogu, a następnie pliki CSS i JavaScript umieszczają w podkatalogach. Nie ma powodu, aby łamać tę konwencję. Utwórzmy katalogi i pliki według schematu przedstawionego w listingu 3.1.

Listing 3.1. Pliki i katalogi — pierwsze przejście



Mamy już podstawy, więc zainstalujemy bibliotekę jQuery.

INSTALACJA BIBLIOTEKI JQUERY ORAZ WTYCZEK

Biblioteka jQuery i jej wtyczki są często oferowane w postaci plików regularnych lub zminifikowanych. Prawie zawsze zainstalujemy pliki regularne, ponieważ pomagają to w debugowaniu, a minifikację i tak przeprowadzamy w ramach systemu budowania. Nie przejmuj się na razie funkcją tych plików — dojdziemy do tego w dalszej części rozdziału.

Biblioteka jQuery dostarcza użyteczną wieloplatformową manipulację DOM oraz inne narzędzia. Skorzystamy z wersji 1.9.1, która jest dostępna na stronie: http://docs.jquery.com/Downloading_jQuery. Umieścimy ją w naszym katalogu jQuery.

```
...
+-- js
| +-- jq
| | +-- jquery-1.9.1.js
...
```

Wtyczka `uriAnchor` biblioteki jQuery zapewnia narzędzia do zarządzania komponentem kotwicy identyfikatora URI. Można ją pobrać ze strony: <https://github.com/mmikowski/urianchor>. Umieścimy wtyczkę w tym samym katalogu jQuery.

```
...
+-- js
| +-- jq
| | +-- jquery.uriAnchor-1.1.3.js
...
```

Pliki i katalogi powinny wyglądać tak, jak przedstawiono w listingu 3.2.

Listing 3.2. Pliki i katalogi po dodaniu biblioteki jQuery i wtyczki

```
spa
+-- css
| +-- spa.css
| `-- spa.shell.css
+-- js
| +-- jq
| | +-- jquery-1.9.1.js
| | `-- jquery.uriAnchor-1.1.3.js
| +-- spa.js
| `-- spa.shell.js
+-- layout.html
`-- spa.html
```

Mamy już przygotowane wszystkie pliki, więc możemy zacząć pisać kody HTML, CSS i JavaScript.

3.2.2. Pisanie kodu HTML aplikacji

Kiedy otwieramy nasz dokument przeglądarkowy (`spa/spa.html`), możemy rozkoszować się wszystkimi dobrodziejstwami SPA, które dotąd osiągnęliśmy. Oczywiście jest to pusty plik, więc dobrodziejstwa ograniczają się do wolnej od błędów, bardzo bezpiecznej pustej strony, która nie robi absolutnie nic. Zróbmy coś z tą „pustą stroną”.

Dokument przeglądarkowy (*spa/spa.html*) zawsze pozostanie niewielki. Jego jedynym zadaniem jest załadowanie bibliotek i arkuszy stylów, a następnie uruchomienie aplikacji. Odpalmy ulubiony edytor tekstu i dodajmy cały kod, przez który musimy przejść w tym rozdziale, tak jak zostało to przedstawione w listingu 3.3.

Listing 3.3. Kod HTML aplikacji – *spa/spa.html*

```
<!doctype html>
<html>
<head>
  <title>Starter SPA</title>
  <!-- Arkusze stylów. -->
  <link rel="stylesheet" href="css/spa.css" type="text/css"/>
  <link rel="stylesheet" href="css/spa.shell.css" type="text/css"/>
  <!-- Zewnętrzny kod JavaScript. -->
  <script src="js/jq/jquery-1.9.1.js" ></script>
  <script src="js/jq/jquery.uriAnchor-1.1.3.js"></script>
  <!-- Nasz kod JavaScript. -->
  <script src="js/spa.js" ></script>
  <script src="js/spa.shell.js"></script>
  <script>
    $(function () { spa.initModule( $('#spa') ); });
  </script>
</head>
<body>
  <div id="spa"></div>
</body>
</html>
```

Najpierw ładujemy arkusze stylów. To optymalizuje wydajność. Jeśli dodamy zewnętrzne arkusze stylów, należy załadować je w pierwszej kolejności.

Następnie ładujemy zewnętrzny kod JavaScript. Obecnie jedynymi zewnętrznymi skryptami, które ładujemy, są biblioteka jQuery i wtyczka do manipulacji kotwicą.

Ładujemy biblioteki JavaScript. Powinny być uporządkowane według zagłębienia przestrzeni nazw. Jest to ważne, ponieważ obiekt przestrzeni nazw (*spa*) musi zostać zadeklarowany, zanim będziemy mogli zadeklarować jego obiekty potomne, np. *spa.shell*.

Inicjowanie aplikacji, kiedy gotowy jest model DOM. Czytelnicy zaznajomieni z biblioteką jQuery zauważą, że w naszym kodzie użyliśmy skrótu, ponieważ fragment `$(function (... mógł zostać zapisany jako $(document).ready(function (...`

Projektanci świadomi wydajności mogą zapytać: „Dlaczego nie umieszczamy skryptów na końcu kontenera *body*, jak w tradycyjnych stronach WWW?”. To dobre pytanie, bo zazwyczaj umożliwia to szybsze renderowanie stron, ponieważ statyczna zawartość HTML i CSS może być wyświetlana, zanim zakończy się ładowanie kodu JavaScript. Aplikacje SPA nie działają jednak w ten sposób. Generują kod HTML z JavaScript i dlatego umieszczanie skryptów poza nagłówkiem nie powoduje szybszego renderowania. Zamiast tego możemy zachować wszystkie zewnętrzne skrypty w sekcji *head*, aby poprawić organizację i czytelność.

3.2.3. Tworzenie głównej przestrzeni nazw CSS

Naszą główną przestrzenią nazw jest *spa* i według konwencji z dodatku A główny arkusz stylów należy nazwać *spa/css/spa.css*. Utworzyliśmy ten plik wcześniej, ale nadszedł czas, by go wypełnić. Ponieważ jest to główny arkusz stylów, będzie posiadał kilka sekcji więcej niż inne pliki CSS. Użyjmy ponownie ulubionego edytora tekstu, aby dodać reguły, których potrzebujemy, tak jak zostało to przedstawione w listingu 3.4.

Listing 3.4. Główna przestrzeń nazw CSS — spa/css/spa.css

```

/*
 * spa.css.
 * Style głównej przestrzeni nazw.
 */
/** Rozpoczęcie resetu. */ ←
 * {
   margin : 0;
   padding : 0;
   -webkit-box-sizing : border-box;
   -moz-box-sizing : border-box;
   box-sizing : border-box;
 }
 h1,h2,h3,h4,h5,h6,p { margin-bottom : 10px; }
 ol,ul,dl { list-style-position : inside;}
/** Zakończenie resetu. */
/** Rozpoczęcie standardowych selektorów. */ ←
 body {
   font : 13px 'Trebuchet MS', Verdana, Helvetica, Arial, sans-serif;
   color : #444;
   background-color : #888;
 }
 a { text-decoration : none; }
 a:link, a:visited { color : inherit; }
 :hover {text-decoration: underline; }
 strong {
   font-weight : 800;
   color : #000;
 }
/** Zakończenie standardowych selektorów. */
/** Rozpoczęcie selektorów przestrzeni nazw spa. */ ←
 #spa {
   position : absolute;
   top : 8px;
   left : 8px;
   bottom : 8px;
   right : 8px;

   min-height : 500px;
   min-width : 500px;
   overflow : hidden;

   background-color : #fff;
   border-radius : 0 8px 0 8px;
 }
/** Zakończenie selektorów przestrzeni nazw spa. */
/** Rozpoczęcie selektorów narzędziowych. */ ←
 .spa-x-select {}
 .spa-x-clearfloat {
   height : 0 !important;
   float : none !important;
   visibility : hidden !important;
   clear : both !important;
 }
/** Zakończenie selektorów narzędziowych. */

```

Resetowanie większości selektorów. Nie ufamy domyślnym ustawieniom przeglądarki. Autorzy kodów CSS rozpoznają to jako powszechną praktykę, choć nie bez kontrowersji.

Dostosowanie standardowych selektorów. Ponownie nie ufamy domyślnym ustawieniom przeglądarki, a także chcemy zapewnić wspólny wygląd w całej aplikacji dla określonego typu elementów. Można je dostosować (i tak zrobimy) przez bardziej określone selektory w innych plikach.

Definiowanie selektorów przestrzeni nazw. Zasadniczo jest to selektor dla elementu używającego głównej przestrzeni nazw, np. #spa.

Zapewnienie selektorów narzędziowych do stosowania we wszystkich innych modułach. Są one poprzedzone prefiksem spa-x-.

Zgodnie z naszymi standardami kodowania wszystkie identyfikatory CSS i nazwy klas w tym pliku zostały poprzedzone prefiksem `spa-`. Skoro utworzyliśmy już główny plik CSS aplikacji, możemy utworzyć odpowiadającą mu przestrzeń nazw JavaScript.

3.2.4. Tworzenie głównej przestrzeni nazw JavaScript

Naszą główną przestrzenią nazw jest `spa`, więc zgodnie z konwencją przedstawioną w dodatku A główny plik JavaScript powinniśmy nazwać `spa/js/spa.js`. Minimalny wymagany kod JavaScript to `var spa = {};`. Chcemy jednak dodać metodę inicjowania aplikacji i upewnić się, że nasz kod przejdzie walidację JSLint. Możemy użyć szablonu z dodatku A i nieco go okroić, ponieważ nie potrzebujemy wszystkich sekcji. Otwórzmy plik za pomocą drugiego z ulubionych edytorów tekstu i wypełnijmy go, tak jak zostało to przedstawione w listingu 3.5.

Listing 3.5. Główna przestrzeń nazw JavaScript — `spa/js/spa.js`

```

/*
 * spa.js.
 * Moduł głównej przestrzeni nazw.
 */

/*jslint browser : true, continue : true, ← Ustawianie przełączników JSLint zgodnie
  devel : true, indent : 2, maxerr : 50,   z szablonem modułu z dodatku A.
  newcap : true, nomen : true, plusplus : true,
  regexp : true, sloppy : true, vars : false,
  white : true
 */
/*global $, spa */ ← Informacja dla narzędzia JSLint, że ma oczekiwać zmiennych
                    globalnych spa oraz $. Jeśli okaże się, że dodajemy do tej listy własne
                    zmienne po spa, prawdopodobnie robimy coś źle.

var spa = (function () {
  var initModule = function ( $container ) {
    $container.html(
      '<h1 style="display:inline-block; margin:25px;">'
      + 'witaj. świecie!'
      + '</h1>'
    );
  };

  return { initModule: initModule };
})();

```

← Użycie wzorca modułu z rozdziału 2. do utworzenia przestrzeni nazw spa. Moduł ten eksportuje jedną metodę `initModule`, która, jak wskazuje nazwa, jest funkcją inicjującą aplikację.

Chcemy się upewnić, że nasz kod nie będzie miał żadnych typowych błędów oraz złych praktyk. W dodatku A napisano, jak zainstalować i uruchomić cenne narzędzie JSLint, które właśnie do tego służy. Opisano w nim, co oznaczają wszystkie przełączniki `/*jslint...*/` umieszczane na początku naszych plików. Narzędzie JSLint zostanie również omówione szczegółowo w rozdziale 5.

Sprawdźmy kod, wpisując w wierszu poleceń `jslint spa/js/spa.js`. Nie powinniśmy zobaczyć żadnych ostrzeżeń ani błędów. Możemy teraz otworzyć dokument przeglądarkowy (`spa/spa.html`) i zobaczyć obowiązkowy komunikat *witaj, świecie!*, tak jak zostało to przedstawione na rysunku 3.2.



Rysunek 3.2. Obowiązkowy zrzut ekranu „witaj, świecie!”

Powitaliśmy już świat i jesteśmy ośmieleni smakiem sukcesu, przejdźmy więc do bardziej ambitnego zadania. W kolejnym podrozdziale rozpoczniemy budowę pierwszej „prawdziwej” aplikacji SPA.

3.3. Tworzenie kontenerów funkcji

Powłoka tworzy kontenery wykorzystywane przez moduły funkcji i zarządza nimi. Kontener naszego suwaka czatu będzie dostosowany do popularnej konwencji i zostanie zakotwiczony w prawym dolnym rogu okna przeglądarki. Powłoka jest odpowiedzialna za kontener suwaka, ale nie będzie zarządzać zachowaniem wewnątrz tego kontenera — jest to zarezerwowane dla modułu funkcji czatu, który zostanie omówiony w rozdziale 6.

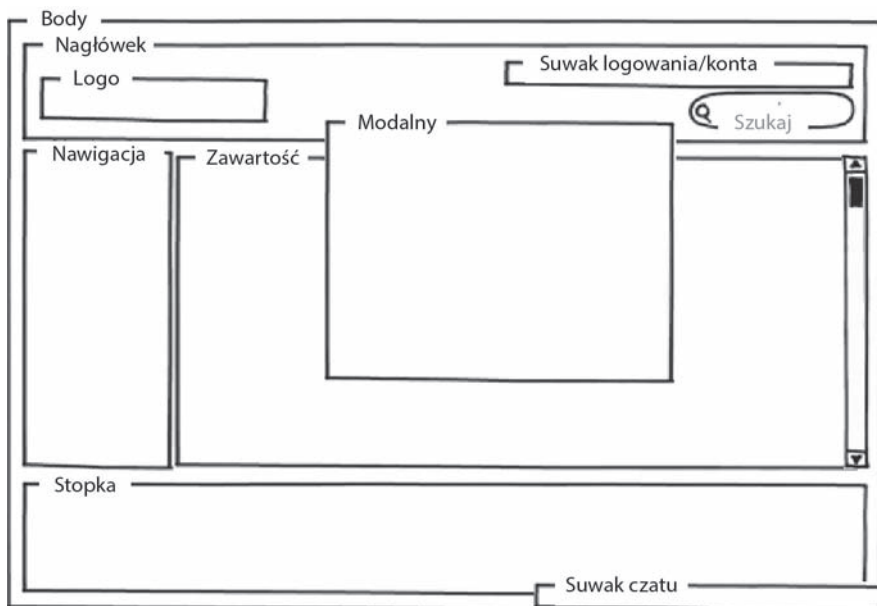
Umieścimy suwak czatu w układzie, który jest stosunkowo kompletny. Na rysunku 3.3 przedstawiony został schemat kontenerów, jaki chcielibyśmy otrzymać.

Oczywiście jest to tylko schemat. Musimy przekształcić go w kody HTML i CSS. Zobaczmy, jak możemy to zrobić.

3.3.1. Wybór strategii

Będziemy rozwijać kody HTML i CSS dla kontenerów funkcji w pliku dokumentu o pojedynczym układzie *spa/layout.html*. Dopiero po dostosowaniu kontenerów do naszych potrzeb przeniesimy kod do plików CSS i JavaScript powłoki. To podejście jest zwykle najszybszym i najbardziej efektywnym sposobem projektowania wstępnego układu, ponieważ nie musimy się przejmować interakcjami z większością pozostałego kodu.

Najpierw napiszemy kod HTML, a później dodamy style.



Rysunek 3.3. Schemat kontenerów aplikacji

3.3.2. Pisanie kodu HTML powłoki

Dużą zaletą HTML5 i CSS3 jest to, że naprawdę *można* oddzielić stylizację od zawartości. Schemat pokazuje wymagane kontenery i sposób ich zagnieżdżenia. To wszystko, czego potrzebujemy, aby bez obaw napisać kod HTML dla kontenerów. Otwórzmy dokument układu (*spa/layout.html*) i wpiszy kod HTML przedstawiony w listingu 3.6.

Listing 3.6. Tworzenie kodu HTML dla kontenerów — *spa/layout.html*

```

<!doctype html>
<html>
<head>
  <title>Układ HTML</title>
  <link rel="stylesheet" href="css/spa.css" type="text/css"/>
</head>
<body>
  <div id="spa">
    <div class="spa-shell-head">
      <div class="spa-shell-head-logo"></div>
      <div class="spa-shell-head-acct"></div>
      <div class="spa-shell-head-search"></div>
    </div>
    <div class="spa-shell-main">
      <div class="spa-shell-main-nav"></div>
      <div class="spa-shell-main-content"></div>
    </div>
    <div class="spa-shell-foot"></div>
    <div class="spa-shell-chat"></div>
  </div>

```

← Zagnieżdżenie logo, ustawienia konta (acct) oraz pola wyszukiwania (search) wewnątrz kontenera nagłówka.

← Umieszczenie kontenerów nawigacji (nav) i zawartości (content) wewnątrz głównego kontenera.

← Utworzenie kontenera stopki.

← Zakotwiczenie kontenera czatu (chat) na dole po prawej stronie kontenera zewnętrznego.

```

    <div class="spa-shell-modal"></div> ← Tworzenie kontenera modal,
  </div>                               który unosi się nad inną zawartością.
</body>
</html>

```

Należy sprawdzić poprawność kodu HTML, aby się upewnić, że nie ma błędów. W tym celu lubimy wykorzystywać wiekowe narzędzie Tidy, które pozwala znaleźć brakujące znaczniki i inne powszechne błędy HTML. Narzędzie Tidy jest dostępne online na stronie: <http://infohound.net/tidy/> lub w postaci kodu źródłowego na stronie: <http://tidy.sourceforge.net/>. Jeśli używasz dystrybucji Linuksa, takich jak Ubuntu czy Fedora, narzędzie Tidy będzie prawdopodobnie dostępne w standardowych repozytoriach. Nadajmy tym kontenerom nieco stylu.

3.3.3. Pisanie kodu CSS powłoki

Napiszemy kod CSS w celu dostarczenia **płynnego szablonu** (ang. *liquid layout*), w którym szerokość i wysokość zawartości będą się dostosowywać, aby wypełnić okno przeglądarki największymi możliwymi rozmiarami. Nadamy kontenerom funkcji kolory tła, dzięki czemu będą dobrze widoczne. Ponadto nie zastosujemy żadnych obramowań, ponieważ mogą one zmienić rozmiar pól CSS. Wprowadziłyby to niepotrzebną nudę do szybkiego procesu prototypowego. Gdy będziemy zadowoleni z prezentacji kontenerów, możemy do nich wrócić i w razie potrzeby dodać obramowania.

Płynne szablony

Gdy układ staje się coraz bardziej skomplikowany, może zajść potrzeba użycia JavaScript, aby zapewnić jego **płynność**. Często procedura obsługi zdarzeń zmiany rozmiaru okna jest stosowana do określenia rozmiaru okna przeglądarki, a następnie przeliczenia i zastosowania nowych wymiarów CSS. Ta technika zostanie zilustrowana w rozdziale 4.

Dodajmy kod CSS do sekcji `<head>` dokumentu układu (*spa/layout.html*). Możemy umieścić go tuż po linku arkusza stylów *spa.css*, tak jak zostało to przedstawione w listingu 3.7. Wszystkie zmiany zostały zaznaczone **pogrubioną czcionką**.

Listing 3.7. Tworzenie kodu CSS dla kontenerów — *spa/layout.html*

```

...
<head>
  <title>Układ HTML</title>
  <link rel="stylesheet" href="css/spa.css" type="text/css"/>
  <style>
    .spa-shell-head, .spa-shell-head-logo, .spa-shell-head-acct,
    .spa-shell-head-search, .spa-shell-main, .spa-shell-main-nav,
    .spa-shell-main-content, .spa-shell-foot, .spa-shell-chat,
    .spa-shell-modal {
      position : absolute;
    }
    .spa-shell-head {
      top : 0;
      left : 0;
      right : 0;

```

```
    height : 40px;
  }
  .spa-shell-head-logo {
    top : 4px;
    left : 4px;
    height : 32px;
    width : 128px;
    background : orange;
  }
  .spa-shell-head-acct {
    top : 4px;
    right : 0;
    width : 64px;
    height : 32px;
    background : green;
  }
  .spa-shell-head-search {
    top : 4px;
    right : 64px;
    width : 248px;
    height : 32px;
    background : blue;
  }
  .spa-shell-main {
    top : 40px;
    left : 0;
    bottom : 40px;
    right : 0;
  }
  .spa-shell-main-content,
  .spa-shell-main-nav {
    top : 0;
    bottom : 0;
  }
  .spa-shell-main-nav {
    width : 250px;
    background : #eee;
  }
  .spa-x-closed .spa-shell-main-nav {
    width : 0;
  }
  .spa-shell-main-content {
    left : 250px;
    right : 0;
    background : #ddd;
  }
  .spa-x-closed .spa-shell-main-content {
    left : 0;
  }
  .spa-shell-foot {
    bottom : 0;
    left : 0;
    right : 0;
    height : 40px;
  }
  .spa-shell-chat {
    bottom : 0;
```

```

    right : 0;
    width : 300px;
    height : 15px;
    background : red;
    z-index : 1;
  }
  .spa-shell-modal {
    margin-top : -200px;
    margin-left : -200px;
    top : 50%;
    left : 50%;
    width : 400px;
    height : 400px;
    background : #fff;
    border-radius : 3px;
    z-index : 2;
  }
</style>
</head>
...

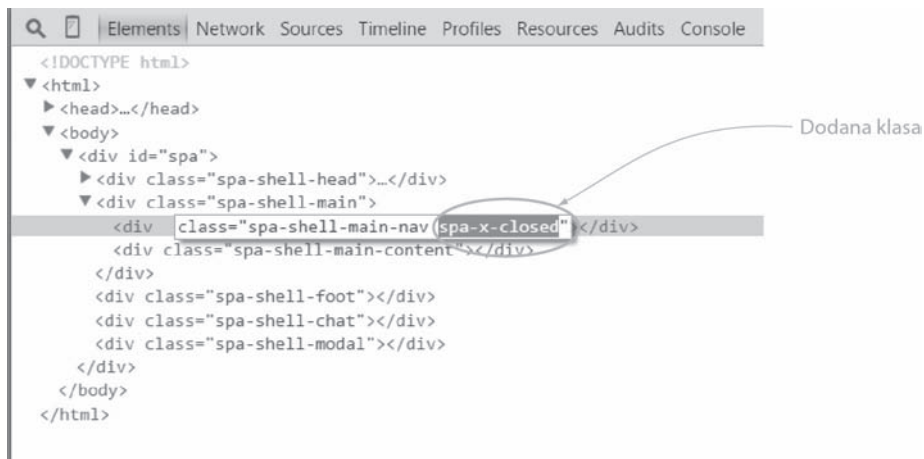
```

Po otwarciu dokumentu przeglądarkowego (*spa/layout.html*) powinniśmy zobaczyć stronę, która wygląda bardzo podobnie do naszego schematu, tak jak przedstawiono na rysunku 3.4. Kiedy zmienimy rozmiar okna przeglądarki, zobaczymy, że również kontenery funkcji zmieniają swoje rozmiary w razie potrzeby. Nasz płynny szablon ma jednak pewne ograniczenia. Jeśli szerokość lub wysokość będą mniejsze niż 500 pikseli, pojawią się paski przewijania. Dodajemy je, ponieważ nie jesteśmy w stanie ścisnąć naszej zawartości poniżej tego wymiaru.



Rysunek 3.4.
Kody HTML i CSS
dla kontenerów
— *spa/layout.html*

Możemy użyć narzędzi dla programistów przeglądarki Google Chrome, aby wypróbować kilka nowo zdefiniowanych stylów, które nie są wykorzystywane w ekranie początkowym. Dodajmy np. klasę *spa-x-closed* do kontenera *spa-shell-main*. Spowoduje to zamknięcie paska nawigacji znajdującego się po lewej stronie. Usunięcie tej klasy przywróci pasek nawigacji (rysunek 3.5).



Rysunek 3.5. W oknie narzędzi dla programistów kliknij dwukrotnie kod HTML, aby dodać klasę

3.4. Renderowanie kontenerów funkcji

Przygotowany dokument układu (*spa/layout.html*) jest ładnym fundamentem. Teraz użyjemy go w naszej aplikacji SPA. Najpierw musimy sprawić, aby powłoka renderowała kontenery zamiast używać statycznych zawartości HTML i CSS.

3.4.1. Konwersja HTML na JavaScript

Nasz kod JavaScript musi zarządzać wszystkimi zmianami dokumentów. W związku z tym musimy przekształcić opracowany wcześniej kod HTML na łańcuchach znaków JavaScript. Zachowamy wcięcia kodu HTML, aby zwiększyć czytelność i ułatwić utrzymywanie, tak jak zostało to przedstawione w listingu 3.8.

Listing 3.8. Konkatenowanie szablonu HTML

```
var main_html = String()
+ '<div class="spa-shell-head">'
+ '<div class="spa-shell-head-logo"></div>'
+ '<div class="spa-shell-head-acct"></div>'
+ '<div class="spa-shell-head-search"></div>'
+ '</div>'
+ '<div class="spa-shell-main">'
+ '<div class="spa-shell-main-nav"></div>'
+ '<div class="spa-shell-main-content"></div>'
+ '</div>'
+ '<div class="spa-shell-foot"></div>'
+ '<div class="spa-shell-chat"></div>'
+ '<div class="spa-shell-modal"></div>';
```

Nie przejmujemy się żadnym wpływem konkatenowanych łańcuchów znaków na wydajność. Kiedy przejdziemy do etapu produkcji, minifikator JavaScript sam połączy łańcuchy znaków.


```

+ '<div class="spa-shell-modal"></div>'
},
stateMap = { $container : null },
jqueryMap = {};

setjQueryMap, initModule;

//----- ZAKOŃCZENIE SEKCJI ZMIENNYCH ZAKRESU MODUŁU -----
//----- ROZPOCZĘCIE SEKCJI METOD NARZĘDZIOWYCH -----
//----- ZAKOŃCZENIE SEKCJI METOD NARZĘDZIOWYCH -----
//----- ROZPOCZĘCIE SEKCJI METOD DOM -----
// Rozpoczęcie metody DOM /setjQueryMap/.
setjQueryMap = function () {
  var $container = stateMap.$container;
  jqueryMap = { $container : $container };
};
// Zakończenie metody DOM /setjQueryMap/.
//----- ZAKOŃCZENIE SEKCJI METOD DOM -----
//----- ROZPOCZĘCIE SEKCJI PROCEDUR OBSŁUGI ZDARZEŃ -----
//----- ZAKOŃCZENIE SEKCJI PROCEDUR OBSŁUGI ZDARZEŃ -----
//----- ROZPOCZĘCIE SEKCJI METOD PUBLICZNYCH -----
// Rozpoczęcie metody publicznej /initModule/.
initModule = function ( $container ) {
  stateMap.$container = $container;
  $container.html( configMap.main_html );
  setjQueryMap();
};
// Zakończenie metod y publicznej /initModule/.
return { initModule : initModule };
//----- ZAKOŃCZENIE SEKCJI METOD PUBLICZNYCH -----
})();

```

Umieszczamy dynamiczne informacje udostępniane w całym module w obiekcie stateMap.

Buforowanie kolekcji jQuery w jqueryMap.

Umieszczamy wszystkie funkcje do tworzenia i modyfikowania elementów strony w sekcji metod DOM.

Deklarujemy wszystkie zmienne zakresu modułu w tej sekcji. Wiele z nich zostanie przypisanych później.

Rezerwujemy sekcję metod narzędziowych dla funkcji, które nie oddziałują z elementami strony.

Użyjemy setjQueryMap do buforowania kolekcji jQuery. Ta funkcja powinna się znaleźć w prawie każdym module powłoki i module funkcji, jakie napiszemy. Stosowanie bufora jqueryMap może znacznie zmniejszyć liczbę linii poprzecznych dokumentu jQuery i zwiększyć wydajność.

Rezerwujemy sekcję procedur obsługi zdarzeń dla funkcji procedur obsługi zdarzeń jQuery.

Umieszczamy publicznie dostępne metody w sekcji metod publicznych.

Tworzymy metodę publiczną initModule, która będzie używana do inicjowania modułu.

Eksportujemy publiczne metody bezpośrednio przez zwracanie ich w mapie. Obecnie dostępna jest tylko metoda initModule.

Mamy teraz moduł, który renderuje kontenery funkcji, ale nadal musimy zapełnić plik CSS i poinstruować moduł głównej przestrzeni nazw (*spa/js/spa.js*), aby korzystał z modułu powłoki (*spa/js/spa.shell.js*) zamiast prezentować tradycyjny komunikat *witaj, świecie!* Zabierzmy się za to.

3.4.3. Pisanie arkusza stylów powłoki

Jeśli zajrzemy do pomocnych konwencji przestrzeni nazw przedstawionych w dodatku A, dowiemy się, że selektory *spa-shell-** należy umieścić w pliku o nazwie *spa/css/spa.shell.css*. Możemy skopiować kod CSS opracowany w pliku *spa/layout.html* bezpośrednio do tego pliku, tak jak zostało to przedstawione w listingu 3.10.

Listing 3.10. Kod CSS powłoki, pierwsze podejście — *spa/css/spa/shell.css*

```

/*
 * spa.shell.css.
 * Style powłoki.
 */

```

```

.spac-shell-head, .spac-shell-head-logo, .spac-shell-head-acct,
.spac-shell-head-search, .spac-shell-main, .spac-shell-main-nav,
.spac-shell-main-content, .spac-shell-foot, .spac-shell-chat,
.spac-shell-modal {
  position : absolute;
}
.spac-shell-head {
  top : 0;
  left : 0;
  right : 0;
  height : 40px;
}
.spac-shell-head-logo {
  top : 4px;
  left : 4px;
  height : 32px;
  width : 128px;
  background : orange;
}
.spac-shell-head-acct {
  top : 4px;
  right : 0;
  width : 64px;
  height : 32px;
  background : green;
}
.spac-shell-head-search {
  top : 4px;
  right : 64px;
  width : 248px;
  height : 32px;
  background : blue;
}
.spac-shell-main {
  top : 40px;
  left : 0;
  bottom : 40px;
  right : 0;
}
.spac-shell-main-content, ← Zdefiniowanie udostępnionych reguł CSS.
.spac-shell-main-nav {
  top : 0;
  bottom : 0;
}
.spac-shell-main-nav {
  width : 250px;
  background : #eee;
}
.spac-x-closed .spac-shell-main-nav { ←
  width : 0;
}
.spac-shell-main-content {
  left : 250px;
  right : 0;
  background : #ddd;
}

```

Użyj klas nadrzędnych, aby wpływać na elementy potomne. Być może jest to jedna z najbardziej zaawansowanych możliwości CSS, która nie jest używana dość często.


```

.spas-x-closed .spa-shell-main-content {
  left : 0;
}
.spa-shell-foot {
  bottom : 0;
  left : 0;
  right : 0;
  height : 40px;
}
.spa-shell-chat {
  bottom : 0;
  right : 0;
  width : 300px;
  height : 15px;
  background : red;
  z-index : 1;
}
.spa-shell-modal {
  margin-top : -200px;
  margin-left : -200px;
  top : 50%;
  left : 50%;
  width : 400px;
  height : 400px;
  background : #fff;
  border-radius : 3px;
  z-index : 2;
}

```

Zrób wcięcie dla selektorów pochodnych i umieść je bezpośrednio pod selektorem nadrzędnym. Selektory pochodne są wyraźnie zależne od selektora nadrzędnego pod względem znaczenia.

Wszystkie selektory mają prefiks `spa-shell-`. Ma to wiele korzyści:

- wskazuje, że te klasy są kontrolowane przez moduł powłoki (*spa/js/spa.shell.js*);
- zapobiega kolidowaniu przestrzeni nazw z zewnętrznymi skryptami i innymi modułami;
- w trakcie debugowania i sprawdzania dokumentu HTML możemy od razu zobaczyć, które elementy są generowane i kontrolowane przez moduł powłoki.

Dzięki tym korzyściom nie spadniemy w ogniste czeluście piekła gulaszu nazw selektorów CSS. Każdy, kto kiedykolwiek zarządzał arkuszami stylów, nawet w umiarkowanej skali, powinien dokładnie wiedzieć, o czym mówimy.

3.4.4. Nakazanie aplikacji, aby korzystała z powłoki

Zmodyfikujmy teraz moduł głównej przestrzeni nazw (*spa/js/spa.js*), aby używał powłoki zamiast niewolniczo kopiować komunikat *witaj, świecie!* do modelu DOM. Poniższa poprawka zaznaczona **pogrubioną czcionką** powinna załatwić sprawę.

```

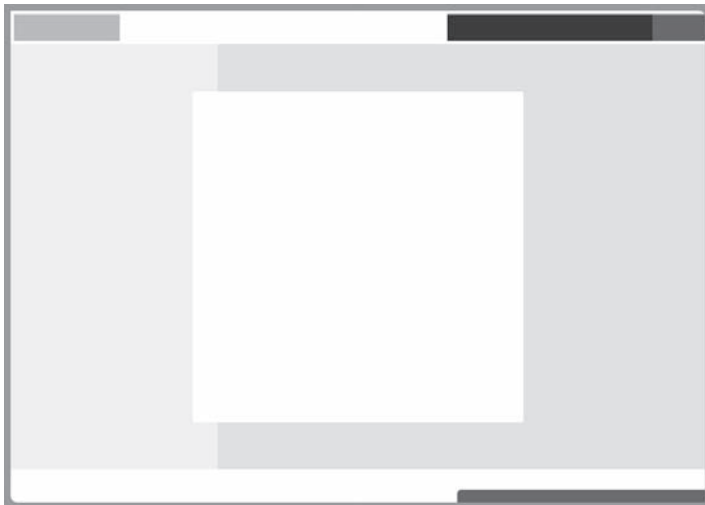
/*
 * spa.js.
 * Moduł głównej przestrzeni nazw.
 */
...
/*global $, spa */

var spa = (function () {

```

```
var initModule = function ( $container ) {  
    spa.shell.initModule( $container );  
};  
return { initModule: initModule };  
}());
```

Po otwarciu dokumentu przeglądarkowego (*spa/spa.html*) powinniśmy teraz zobaczyć ekran, taki jak przedstawiono na rysunku 3.6. Możemy użyć narzędzi dla programistów Google Chrome, aby potwierdzić, że dokument wygenerowany przez aplikację SPA (*spa/spa.html*) odpowiada dokumentowi układu (*spa/layout.html*).



Rysunek 3.6.
To jak déjà vu
— *spa/spa.html*

Z takim fundamentem zaczniemy pracę nad tym, aby powłoka zarządzała kontenerami funkcji. Może to być również dobry czas, aby zrobić sobie przerwę, ponieważ następny podrozdział jest dość wymagający.

3.5. Zarządzanie kontenerami funkcji

Powłoka renderuje i kontroluje **kontenery funkcji**. Są to kontenery „najwyższego poziomu” (zazwyczaj `DIV`), które przechowują zawartość funkcyjną. Powłoka inicjuje i koordynuje wszystkie moduły funkcji w aplikacji. Ponadto wskazuje modułom funkcji, aby tworzyły całą zawartość wewnątrz kontenerów funkcji i zarządzały nią. Moduły funkcji zostaną omówione szczegółowo w rozdziale 4.

W tym podrozdziale najpierw napiszemy metodę rozwijania i zwijania kontenera funkcji suwaka czatu. Następnie zbudujemy procedurę obsługi zdarzeń kliknięcia, aby suwak mógł być otwierany lub zamykany zgodnie z życzeniem użytkownika. Na koniec sprawdzimy nasze dokonania i omówimy kolejną ważną rzecz — zarządzanie stanem strony za pomocą identyfikatora fragmentu adresu URI.

3.5.1. Pisanie metody rozwijania i zwijania suwaka czatu

Będziemy umiarkowanie ambitni w kwestii funkcji suwaka czatu. Musimy doprowadzić ją do jakości produkcyjnej, ale nie musi być ekstrawagancka. Oto wymagania, którym powinniśmy sprostać:

- 1) umożliwienie programistom konfigurowania szybkości i wysokości ruchów suwaka;
- 2) utworzenie pojedynczej metody do rozwijania i zwijania suwaka czatu;
- 3) uniknięcie sytuacji wyścigu, w której suwak może być rozwijany i zwijany w tym samym czasie;
- 4) umożliwienie programistom przekazania opcjonalnego wywołania zwrotnego, które ma być wywoływane na zakończenie ruchu suwaka;
- 5) tworzenie kodu testowego w celu zapewnienia prawidłowego działania suwaka.

Dostosujmy powłokę do tych wymagań, tak jak przedstawiono w listingu 3.11¹. Wszystkie zmiany zostały zaznaczone **pogrubioną czcionką**. Zapoznaj się z adnotacjami, które szczegółowo opisują, w jaki sposób zmiany odnoszą się do wymagań.

Listing 3.11. Modyfikacja powłoki w celu zaimplementowania funkcji rozwijania i zwijania suwaka czatu — spa/js/spa.shell.js

```

...
spa.shell = (function () {
//----- ROZPOCZĘCIE SEKCJI ZMIENNYCH ZAKRESU MODUŁU -----
var
  configMap = {
    main_html : String()
    ...
    chat_extend_time      : 1000, ← Przechowujemy czasy i wysokości rozwijania i zwijania
    chat_retract_time     : 300,   suwaka czatu w mapie konfiguracji modułu. Spełnienie
    chat_extend_height    : 450,   wymagania 1.: „umożliwienie programistom
    chat_retract_height   : 15     konfigurowania szybkości i wysokości ruchów suwaka”.
  },
  stateMap = { $container : null },
  jqueryMap = {};

  setJqueryMap, toggleChat, initModule; ← Dodajemy metodę toggleChat do listy
//----- ZAKOŃCZENIE SEKCJI ZMIENNYCH ZAKRESU MODUŁU -----

//----- ROZPOCZĘCIE SEKCJI METOD NARZĘDZIOWYCH -----
//----- ZAKOŃCZENIE SEKCJI METOD NARZĘDZIOWYCH -----

//----- ROZPOCZĘCIE SEKCJI METOD DOM -----
// Rozpoczęcie metody DOM /setJqueryMap/.
setJqueryMap = function () {
  var $container = stateMap.$container;

  jqueryMap = { ← Buforowanie kolekcji jQuery suwaka czatu w mapie jqueryMap.
    $container : $container,
    $chat : $container.find( '.spa-shell-chat' )
  }
}

```

¹ To dobry moment, aby podziękować ulubionym ciałom niebieskich za bibliotekę jQuery, ponieważ bez niej byłoby to o wiele trudniejsze.

```

    };
  };
  // Zakończenie metody DOM /setJqueryMap/.
  // Rozpoczęcie metody DOM /toggleChat/.
  // Cel: wysuwanie i chowanie suwaka czatu.
  // Argumenty:
  // * do_extend — jeśli prawda (true), wysuwa suwak, jeśli fałsz (false), chowa;
  // * callback (wywołanie zwrotne) — opcjonalna funkcja do wykonywania na zakończenie animacji.
  // Ustawienia:
  // * chat_extend_time, chat_retract_time,
  // * chat_extend_height, chat_retract_height.
  // Zwraca wartość logiczną (boolean):
  // * true — animacja suwaka aktywowana;
  // * false — animacja suwaka nieaktywowana.
  //
  toggleChat = function ( do_extend, callback ) {
    var
      px_chat_ht = jqueryMap.$chat.height(),
      is_open = px_chat_ht === configMap.chat_extend_height,
      is_closed = px_chat_ht === configMap.chat_retract_height,
      is_sliding = !is_open && !is_closed;

    // Unikanie sytuacji wyścigu.
    if ( is_sliding ){ return false; }

    // Rozpoczęcie rozwijania suwaka czatu.
    if ( do_extend ) {
      jqueryMap.$chat.animate(
        { height : configMap.chat_extend_height },
        configMap.chat_extend_time,
        function () {
          if ( callback ){ callback( jqueryMap.$chat ); }
        }
      );
      return true;
    }

    // Zakończenie rozwijania suwaka czatu.

    // Rozpoczęcie zwijania suwaka czatu.
    jqueryMap.$chat.animate(
      { height : configMap.chat_retract_height },
      configMap.chat_retract_time,
      function () {
        if ( callback ){ callback( jqueryMap.$chat ); }
      }
    );
    return true;
  };
  // Zakończenie zwijania suwaka czatu.
};
// Zakończenie metody DOM /toggleChat/.
//----- ZAKOŃCZENIE SEKCJI METOD DOM -----

//----- ROZPOCZĘCIE SEKCJI PROCEDUR OBSŁUGI ZDARZEŃ -----
//----- ZAKOŃCZENIE SEKCJI PROCEDUR OBSŁUGI ZDARZEŃ -----

//----- ROZPOCZĘCIE SEKCJI METOD PUBLICZNYCH -----

```

Dodanie metody toggleChat. Spełnienie wymagania 2.: „utworzenie pojedynczej metody do rozwijania i zwijania suwaka czatu”.

Zapobieganie powstawaniu sytuacji wyścigu poprzez odmowę podjęcia działań, jeśli suwak jest już w ruchu. Spełnienie wymagania 3.: „uniknięcie sytuacji wyścigu, w której suwak może być rozwijany i zwijany w tym samym czasie”.

Wykonanie wywołania zwrotnego po zakończeniu animacji. Spełnienie wymagania 4.: „umożliwienie programistom przekazania opcjonalnego wywołania zwrotnego, które ma być wywoływane na zakończenie ruchu suwaka”.

Rozwinięcie suwaka po 3 sekundach od momentu załadowania strony i schowanie go po 8 sekundach. Spełnienie wymagania 5.: „tworzenie kodu testowego w celu zapewnienia prawidłowego działania suwaka”.

```
// Rozpoczęcie metody publicznej /initModule/.
initModule = function ( $container ){
  // Ładowanie HTML i mapowanie kolekcji jQuery.
  stateMap.$container = $container;
  $container.html( configMap.main_html );
  setJqueryMap();

  // Testowanie przełączania.
  setTimeout( function () {toggleChat( true ); }, 3000 );
  setTimeout( function () {toggleChat( false );}, 8000 );
};
// Zakończenie metody publicznej /initModule/.
return { initModule : initModule };
//----- ZAKOŃCZENIE SEKCJI METOD PUBLICZNYCH -----
}());
```

Jeśli bawisz się razem z nami w domu, sprawdźmy najpierw kod za pomocą narzędzia JSLint, wpisując w wierszu poleceń `jslint spa/js/spa.shell.js` — nie powinniśmy zobaczyć żadnych ostrzeżeń ani błędów. Następnie przeładujemy dokument przeglądarkowy (`spa/spa.html`) i sprawdźmy, czy suwak czatu rozwija się po trzech sekundach i chowa po ośmiu. Mamy już przesuwający się suwak, więc możemy wdrożyć kliknięcie myszką przez użytkownika w celu przełączania pozycji suwaka.

3.5.2. Dodanie procedury obsługi zdarzeń kliknięcia suwaka czatu

Większość użytkowników oczekuje, że kliknięcie suwaka czatu spowoduje jego rozwinięcie lub zwinięcie, i jest to powszechna konwencja. Oto wymagania, które chcemy spełnić:

- 1) ustawienie tekstu podpowiedzi dla użytkownika, np. *Kliknij, aby ukryć*;
- 2) dodanie procedury obsługi zdarzeń kliknięcia, aby wywoływać funkcję `toggleChat`;
- 3) powiązanie procedury obsługi zdarzeń kliknięcia ze zdarzeniem jQuery.

Dostosujemy powłokę do tych wymagań, tak jak przedstawiono w listingu 3.12. Wszystkie zmiany ponownie zostały zaznaczone **pogrubioną czcionką**, a adnotacje szczegółowo opisują, w jaki sposób zmiany odnoszą się do wymagań.

Listing 3.12. Modyfikacja powłoki w celu zaimplementowania obsługi zdarzeń kliknięcia suwaka czatu — `spa/js/spa.shell.js`

```
...
spa.shell = (function () {
  //----- ROZPOCZĘCIE SEKCJI ZMIENNYCH ZAKRESU MODUŁU -----
  var
    configMap = {
      ...
      chat_retract_height : 15,
      chat_extended_title : 'Kliknij, aby ukryć',
      chat_retracted_title : 'Kliknij, aby pokazać'
    },
    stateMap = {
      $container : null,
```

Dodanie tekstu podpowiedzi dla operacji zwijania i rozwijania suwaka do mapy configMap. Spełnienie wymagania 1.: „ustawienie tekstu podpowiedzi dla użytkownika...”.

```

    is_chat_retracted : true ← Dodanie is_chat_retracted do mapy stateMap. Dobrą praktyką
  },                               jest utworzenie listy wszystkich kluczy stosowanych
  jQueryMap = { },                 w stateMap, aby można było je łatwo znaleźć i sprawdzić.
                                  Ten jest używany przez metodę toggleChat.

  setjQueryMap, toggleChat, onClickChat, initModule: ← Dodanie metody onClickChat do listy
//----- ZAKOŃCZENIE SEKCJI ZMIENNYCH ZAKRESU MODUŁU ----- nazw funkcji zakresu modułu.
...
//----- ROZPOCZĘCIE SEKCJI METOD DOM -----
// Rozpoczęcie metody DOM /setjQueryMap/.
...
// Zakończenie metody DOM /setjQueryMap/.
// Rozpoczęcie metody DOM /toggleChat/.
// Cel: wysuwanie i chowanie suwaka czatu.
...
// Stan: konfiguruje stateMap.is_chat_retracted: ← Aktualizacja dokumentów API toggleChat, aby wskazać,
// * true — suwak jest zwinięty;                 w jaki sposób właściwość stateMap.is_chat_retracted
// * false — suwak jest rozwinięty.             jest ustawiana przez tę metodę.
//
toggleChat = function ( do_extend, callback ) {
  var
    px_chat_ht = jQueryMap.$chat.height(),
    is_open = px_chat_ht === configMap.chat_extend_height,
    is_closed = px_chat_ht === configMap.chat_retract_height,
    is_sliding = ! is_open && ! is_closed;

  // Unikanie sytuacji wyścigu.
  if ( is_sliding ) { return false; }

  // Rozpoczęcie rozwijania suwaka czatu.
  if ( do_extend ) {
    jQueryMap.$chat.animate(
      { height : configMap.chat_extend_height },
      configMap.chat_extend_time,
      function () {
        jQueryMap.$chat.attr(
          'tytuł', configMap.chat_extended_title
        );
        stateMap.is_chat_retracted = false;
        if ( callback ) { callback( jQueryMap.$chat ); }
      }
    );
    return true;
  }
  // Zakończenie rozwijania suwaka czatu.

  // Rozpoczęcie zwijania suwaka czatu.
  jQueryMap.$chat.animate(
    { height : configMap.chat_retract_height },
    configMap.chat_retract_time,
    function () {
      jQueryMap.$chat.attr(
        'tytuł', configMap.chat_retracted_title
      );
      stateMap.is_chat_retracted = true;
      if ( callback ) { callback( jQueryMap.$chat ); }
    }
  );
}

```

Dostosowanie funkcji toggleChat, aby kontrolowała tekst w dymku oraz wartość stateMap.is_chat_retracted. Spełnienie wymagania 1.: „ustawienie tekstu podpowiedzi dla użytkownika...”.

```

return true;
    // Zakończenie zwiżania suwaka czatu.
};
// Zakończenie metody DOM /toggleChat/.
//----- ZAKOŃCZENIE SEKCJI METOD DOM -----

//----- ROZPOCZĘCIE SEKCJI PROCEDUR OBSŁUGI ZDARZEŃ -----
onClickChat = function ( event ) {
    toggleChat( stateMap.is_chat_retracted );
    return false;
};
//----- ZAKOŃCZENIE PROCEDURY OBSŁUGI ZDARZEŃ -----

//----- ROZPOCZĘCIE SEKCJI METOD PUBLICZNYCH -----
// Rozpoczęcie metody publicznej /initModule/.
initModule = function ( $container ) {
    // Ładowanie HTML i mapowanie kolekcji jQuery.
    stateMap.$container = $container;
    $container.html( configMap.main_html );
    setJqueryMap();

    // Inicjowanie suwaka czatu i wiązanie procedury obsługi kliknięcia.
    stateMap.is_chat_retracted = true;
    jqueryMap.$chat
        .attr( 'tytuł', configMap.chat_retracted_title )
        .click( onClickChat );
};
// Zakończenie metody publicznej /initModule/.

return { initModule : initModule };
//----- ZAKOŃCZENIE SEKCJI METOD PUBLICZNYCH -----
}());

```

Dodanie procedury obsługi zdarzeń onClickChat. Spełnienie wymagania 2.: „dodanie procedury obsługi zdarzeń kliknięcia, aby wywoływać funkcję toggleChat”.

Zainicjowanie obsługi zdarzeń przez ustawienie wartości stateMap.is_chat_retracted i tekstu w dymku. Następnie powiązanie procedury obsługi ze zdarzeniem kliknięcia. Spełnienie wymagania 3.: „powiązanie procedury obsługi zdarzeń kliknięcia ze zdarzeniem jQuery”.

Jeśli bawisz się razem z nami w domu, sprawdźmy ponownie kod, wpisując w wierszu poleceń `jslint spa/js/spa.shell.js`. Tym razem również nie powinniśmy zobaczyć żadnych ostrzeżeń ani błędów.

Należy pamiętać o pewnym aspekcie procedur obsługi zdarzeń jQuery, który naszym zdaniem jest bardzo ważny: wartość zwracana jest interpretowana przez jQuery, aby określić dalszą obsługę zdarzenia. Zwykle zwracamy `false` z procedury obsługi zdarzeń jQuery. Ma to następujące skutki:

- Wskazanie bibliotece jQuery, aby nie wywoływać domyślnej akcji, np. otwarcia odnośnika lub zaznaczenia tekstu. Ten sam efekt można uzyskać poprzez wywołanie metody `event.preventDefault()` w procedurze obsługi zdarzeń.
- Wskazanie bibliotece jQuery, aby powstrzymać uruchomienie przez dane zdarzenie tego samego zdarzenia w elemencie nadrzędnym DOM (to zachowanie jest często nazywane **bąbelkowaniem**). Ten sam efekt można uzyskać poprzez wywołanie metody `event.stopPropagation()` w procedurze obsługi zdarzeń.
- Zakończenie wykonywania procedury obsługi. Jeśli kliknięty element ma powiązane inne procedury obsługi po tej procedurze, wykonywana jest następna w kolejce. (Jeśli nie chcemy wykonywać kolejnych procedur obsługi, możemy wywołać metodę `event.preventDefaultImmediatePropagation()`).

Zazwyczaj chcemy, aby procedury obsługi zdarzeń wykonywały te trzy akcje. Niedługo napiszemy procedury obsługi zdarzeń, w których nie chcemy wykonywać tych akcji. Takie procedury obsługi zdarzeń będą zwracały wartość `true`.

Powłoka nie musi koniecznie obsługiwać kliknięcia. Może zamiast tego zapewnić możliwość manipulowania suwakiem w postaci wywołania zwrótnego do modułu czatu — i zachęcamy do tego. Ponieważ jednak nie napisaliśmy jeszcze tego modułu, na razie obsługujemy zdarzenie kliknięcia w powłoce.

Dodajmy teraz nieco polotu w arkuszu stylów powłoki. Zmiany zostały przedstawione w listingu 3.13.

Listing 3.13. Dodanie szlifów powłoce — `spa/css/spa.shell.css`

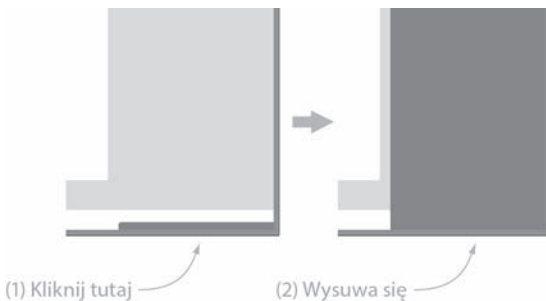
```
...
.sp-shell-foot {
  ...
}
.sp-shell-chat {
  bottom      : 0;
  right       : 0;
  width       : 300px;
  height      : 15px;
  cursor      : pointer;
  background  : red;
  border-radius : 5px 0 0 0;
  z-index     : 1;
}
.sp-shell-chat:hover {
  background : #a00;
}
.sp-shell-modal { ... }
...
```

Zmiana kursora na wskaźnik w momencie przesunięcia go nad obszar suwaka. Jest to informacja dla użytkownika, że coś się stanie, jeśli kliknie.

Zaokrąglenie narożnika, aby suwak wyglądał ładniej.

Zmiana koloru suwaka, gdy kursor zostanie przesunięty nad obszar suwaka. Jest to wzmocnienie komunikatu do użytkownika, że po kliknięciu wykonana zostanie jakaś akcja.

Kiedy przeglądamy dokument przeglądarkowy (`spa/spa.html`), możemy kliknąć suwak i zobaczyć, jak się rozwija (rysunek 3.7).



Rysunek 3.7. Rozwijanie suwaka czatu — `spa/spa.html`

Suwak rozwija się znacznie wolniej niż chowa. Możemy zmienić prędkość suwaka, zmieniając konfigurację w powłoce (`spa/js/spa.shell.js`), np.:

```
...
configMap = {
  main_html : String()
```



```

...
    chat_extend_time : 250,
    chat_retract_time : 300,
    ...
  },
...

```

W następnym podrozdziale dostosujemy aplikację w taki sposób, aby lepiej zarządzać jej stanem. Gdy skończymy, wszystkie funkcje historii przeglądarki (zakładki, przyciski *Dalej* i *Wstecz*) będą działać w przypadku suwaka czatu tak, jak oczekuje tego użytkownik.

3.6. Zarządzanie stanem aplikacji

W informatyce **stan** (ang. *state*) jest unikatową konfiguracją informacji w aplikacji. Aplikacje desktopowe i internetowe zasadniczo starają się utrzymać pewien stan między sesjami. Gdy zapisujemy np. dokument edytora tekstu, a następnie otwieramy go ponownie w późniejszym terminie, dokument zostaje przywrócony. Aplikacja może również przywrócić rozmiar okna, preferencje użytkownika oraz lokalizację kursora i strony. Nasza aplikacja SPA także musi zarządzać stanem, ponieważ osoby korzystające z przeglądarek oczekują pewnych zachowań.

3.6.1. Zachowania przeglądarki oczekiwane przez użytkowników

Aplikacje desktopowe i internetowe różnią się znacznie pod względem aspektu stanu, który utrzymują. Aplikacja desktopowa może pominąć przycisk *Poprzedni*, jeśli nie oferuje możliwości „powrotu”. W aplikacji internetowej mamy jednak przycisk *Wstecz* (jedna z najczęściej używanych kontroltek przeglądarki), który rzuca się w oczy użytkownika, prosząc się o kliknięcie — i nie możemy go usunąć.

To samo dotyczy przycisków *Dalej*, *Zakładki* oraz *Historia*. Użytkownicy oczekują, że kontrolki *historii* będą działać. Jeśli nie, nasi użytkownicy będą marudzić, a nasza aplikacja nigdy nie wygra Webby Award. W tabeli 3.1 przedstawiono przybliżone odpowiedniki tych kontroltek historii w aplikacji desktopowej.

Tabela 3.1. Porównanie kontroltek przeglądarki i aplikacji desktopowej

Kontrolki przeglądarki	Kontrolki aplikacji desktopowej	Komentarz
<i>Wstecz</i>	<i>Cofnij</i>	Cofnięcie do poprzedniego stanu
<i>Dalej</i>	<i>Powtórz</i>	Przywrócenie stanu sprzed ostatniego cofnięcia
<i>Zakładki</i>	<i>Zapisz jako</i>	Zachowanie stanu aplikacji do późniejszego wykorzystania lub odwołania się
<i>Historia</i>	<i>Cofnij historię</i>	Przeglądanie kroków z sekwencji <i>Wstecz/Dalej</i>

Ponieważ mamy ambicję wygrać nagrodę Webby, musimy się upewnić, że te kontrolki historii będą działać tak, jak nasi użytkownicy oczekują. Omówmy strategię zapewnienia zachowań oczekiwanych przez użytkowników.

3.6.2. Wybór strategii zarządzania kontrolkami historii

Optymalna strategia zapewnienia kontrolki historii powinna spełniać następujące wymagania:

1. Kontrolki historii powinny działać według oczekiwań użytkownika (zob. tabela 3.1).
2. Projektowanie z uwzględnieniem obsługi kontrolki historii powinno generować rozsądne koszty. Nie powinno wymagać znacznie więcej czasu i większej złożoności w porównaniu z projektowaniem bez uwzględniania kontrolki historii.
3. Aplikacja powinna działać z odpowiednią wydajnością. Czas reakcji na akcje użytkownika nie powinien być dłuższy, a interfejs użytkownika nie powinien być bardziej skomplikowany.

Rozważmy kilka strategii, wykorzystując jako przykład suwak czatu i następujące działania użytkownika:

1. Zuzia odwiedza aplikację SPA i klika suwak czatu, aby go otworzyć.
2. Dodaje aplikację SPA do zakładek, a następnie przechodzi do przeglądania innych stron.
3. Później decyduje się powrócić do naszej aplikacji i klika jej zakładkę.

Weźmy pod uwagę trzy strategie zapewniające oczekiwane działanie zakładki Zuzi. Nie staraj się ich zapamiętać. Chcemy tylko zilustrować ich względne zalety²:

Strategia 1. Po kliknięciu procedura obsługi zdarzeń bezpośrednio wywołuje procedurę `toggleChat` i ignoruje URI. Gdy Zuzia wraca do swojej zakładki, suwak zostaje wyświetlony w pozycji domyślnej, czyli zamknięty. Zuzia nie jest zadowolona, ponieważ zakładka nie działa tak, jak tego oczekiwała. Programista Kuba też nie jest zadowolony, ponieważ jego menedżer produktu uznał użyteczność aplikacji za nieakceptowalną i wierci mu dziurę w brzuchu w tej kwestii.

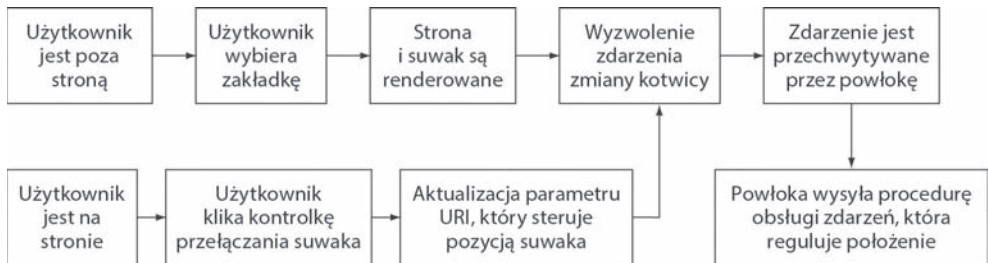
Strategia 2. Po kliknięciu procedura obsługi zdarzeń bezpośrednio wywołuje procedurę `toggleChat`, a następnie modyfikuje URI, aby zapisać ten stan. Gdy Zuzia wraca do swojej zakładki, aplikacja musi rozpoznawać parametr w identyfikatorze URI i podjąć odpowiednią akcję. Zuzia jest zadowolona. Programista Kuba **nie** jest zadowolony, ponieważ musi teraz obsługiwać dwa warunki otwierające suwak: zdarzenie kliknięcia w czasie wykonywania oraz parametr URI podczas ładowania. Menedżer produktu Kuby też nie jest zbyt szczęśliwy, ponieważ obsługa tego dualnego podejścia jest wolniejsza oraz podatna na błędy i nieścisłości.

Strategia 3. Po kliknięciu procedura obsługi zdarzeń zmienia URI, a następnie szybko zwraca wartość. Procedura obsługi zdarzeń `hashchange` powłoki pobiera zmianę i wysyła do procedury `toggleChat`. Gdy Zuzia wraca do swojej zakładki, identyfikator URI jest parsowany przez tę samą procedurę i przywrócony zostaje otwarty suwak. Zuzia jest zadowolona, ponieważ zakładka działa zgodnie z jej oczekiwaniami. Programista Kuba też jest zadowolony, ponieważ może użyć *jednej ścieżki kodu do zaim-*

² Istnieją też inne strategie, takie jak wykorzystanie trwałych ciasteczek lub ramki `IFrame`, ale są zbyt ograniczone i zawile, aby zasługiwać na uwagę.

plementowania wszystkich zapisywalnych w zakładkach stanów. Menedżer produktu Kuby również jest zadowolony, ponieważ rozwój aplikacji jest szybki i stosunkowo wolny od błędów.

Naszym preferowanym rozwiązaniem jest **strategia 3.**, ponieważ obsługuje wszystkie kontrolki historii (wymaganie 1.), uwzględnia i minimalizuje obawy dotyczące projektowania (wymaganie 2.) oraz zapewnia wydajność aplikacji poprzez dostosowywanie tylko tych części strony, które trzeba zmienić, gdy używana jest kontrolka historii (wymaganie 3.). To rozwiązanie, w którym URI zawsze steruje stanem strony, nazywamy **wzorcem interfejsu kotwicy** (rysunek 3.8).



Rysunek 3.8. Wzorec interfejsu kotwicy

Wrócimy do tego wzorca w rozdziale 4. Skoro wybraliśmy już strategię, teraz ją zaimplementujemy.

3.6.3. Zmiana kotwicy przy wystąpieniu zdarzenia historii

Komponent kotwicy adresu URI wskazuje przeglądarce, którą część strony wyświetlić. Inne popularne nazwy kotwicy to **komponent zakładki** (ang. *bookmark component*) lub **identyfikator fragmentu** (ang. *hash fragment*). Kotwica zawsze zaczyna się od znaku # i w poniższym kodzie została zaznaczona **pogrubioną czcionką**.

`http://localhost/spa.html#!chat=open`

Tradycyjnie programiści wykorzystują mechanizm kotwicy, aby umożliwić użytkownikom łatwe „przeskakiwanie” pomiędzy sekcjami długiego dokumentu. Strona internetowa, która np. na początku posiada spis treści, może linkować wszystkie tytuły punktów do odpowiadających im sekcji w dokumencie. Ponadto każda z sekcji może mieć na końcu link „powrót do początku”. Blogi i fora nadal szeroko wykorzystują ten mechanizm.

Jedną z wyjątkowych cech komponentu kotwicy jest to, że przeglądarka *nie* przeładowuje strony, gdy zostaje on zmieniony. Komponent kotwicy jest kontrolką tylko po stronie klienta, co czyni go idealnym miejscem do przechowywania stanu aplikacji. Ta technika jest wykorzystywana przez wiele aplikacji SPA.

Odnosimy się do zmiany stanu aplikacji, który chcemy zachować w historii przeglądarki jako **zdarzenie historii**. Ponieważ zdecydowaliśmy, że otwarcie lub zamknięcie czatu jest zdarzeniem historii (przegapiłeś zebranie), nasza procedura obsługi zdarzeń kliknięcia może zmieniać kotwicę, aby wyrazić stan suwaka czatu. Możemy użyć wtyczki

uriAnchor jQuery do odwołania najcięższej roboty. Zmodyfikujmy powłokę, aby kliknięcie użytkownika zmieniało URI tak, jak przedstawiono w listingu 3.14. Wszystkie zmiany zostały zaznaczone **pogrubioną czcionką**.

Listing 3.14. Zastosowanie wtyczki uriAnchor jQuery — spa/js/spa.shell.js

```

...
//----- ROZPOCZĘCIE SEKCJI PROCEDUR OBSŁUGI ZDARZEŃ -----
onClickChat = function ( event ) {
  if ( toggleChat( stateMap.is_chat_retracted ) ) {
    $.uriAnchor.setAnchor({
      chat : ( stateMap.is_chat_retracted ? 'open' : 'closed' )
    });
  }
  return false;
};
//----- ZAKOŃCZENIE SEKCJI PROCEDUR OBSŁUGI ZDARZEŃ -----
...

```

Gdy teraz klikniemy suwak, zobaczymy, że zmieni się kotwica w adresie URI, ale tylko wtedy, gdy wykonanie funkcji toggleChat powiedzie się i zwróci wartość true. Kiedy np. klikniemy, aby otworzyć, a następnie zamknąć suwak czatu, zobaczymy adres:

http://localhost/spa.html#!chat=closed

Na temat wykrzyknika

Wykrzyknik po symbolu kratki (!) w naszym przykładowym adresie URI służy do poinformowania Google oraz innych wyszukiwarek, że ten adres URI może być indeksowany dla operacji wyszukiwania. Więcej informacji na temat optymalizacji wyszukiwarek znajdziesz w rozdziale 9.

Musimy się upewnić, że po zmianie kotwicy zmieniona zostanie tylko ta część aplikacji, która wymaga dostosowania. Przyspiesza to znacznie aplikację i pozwala uniknąć niepokojącego „migotania”, które występuje, gdy części strony są niepotrzebnie czyszczone i ponownie renderowane. Załóżmy np., że Zuzia przegląda listę profili tysięcy użytkowników, kiedy otwiera suwak czatu. Jeśli kliknie przycisk *Wstecz*, aplikacja powinna po prostu zamknąć suwak — profile nie powinny być ponownie renderowane.

Zadajmy sobie trzy pytania, które pomogą określić, czy zmiana pochodząca ze zdarzenia jest warta obsłużenia w historii:

- Jak bardzo użytkownik może chcieć dodania do zakładek zmiany, która nastąpiła?
- Jak bardzo użytkownik może chcieć przywrócenia stanu strony sprzed zmiany?
- Jak bardzo kosztowne to będzie?

Chociaż przyrostowe koszty związane z utrzymywaniem stanu są zwykle niewielkie przy zastosowaniu wzorca interfejsu kotwicy, istnieją pewne sytuacje, w których może to być kosztowne lub niemożliwe. Dokonany zakup online byłby np. bardzo trudny do odwrócenia, gdy użytkownik kliknie przycisk *Wstecz*. W takich sytuacjach należy całkowicie unikać wpisów w historii. Na szczęście obsługuje to wtyczka uriAnchor.

3.6.4. Użycie kotwicy do sterowania stanem aplikacji

Chcemy, aby komponent kotwicy zawsze sterował możliwym do zapisania w zakładkach stanem aplikacji. Dzięki temu funkcje historii zawsze będą działać zgodnie z oczekiwaniami. Poniższy pseudokod ilustruje, w jaki sposób chcemy obsługiwać zdarzenie historii:

- Gdy wystąpi zdarzenie historii, zmień komponent kotwicy URI, aby odzwierciedlić zmianę stanu:
 - procedura obsługi, która odebrała zdarzenie, wywołuje narzędzie powłoki, aby zmienić kotwicę;
 - następnie procedura obsługi zdarzeń kończy działanie.
- Procedura obsługi zdarzeń powłoki hashchange zauważy zmianę URI i działa zgodnie z nią:
 - porównuje bieżący stan ze stanem proponowanym przez nową kotwicę;
 - próbuje zmienić te sekcje aplikacji, które wymagają dostosowania na podstawie wykonanego porównania;
 - jeśli nie może dokonać żądanych zmian, utrzymuje bieżący stan i przywraca pasującą do niego kotwicę.

Naszukowaliśmy już nasz pseudokod, więc przejdźmy do pracy nad przekształceniem go w prawdziwy.

MODYFIKACJA POWŁOKI W CELU WYKORZYSTANIA KOMPONENTU KOTWICY

Zmodyfikujmy powłokę, aby używać komponentu kotwicy do sterowania stanem aplikacji, tak jak przedstawiono w listingu 3.15. Jest tu całkiem sporo nowego kodu, ale nie zniechęcaj się — wszystko zostanie wyjaśnione w odpowiednim czasie.

Listing 3.15. Wykorzystanie kotwicy do sterowania stanem aplikacji
— `spa/js/spa.shell.js`

```

...
spa.shell = (function () {
  //----- ROZPOCZĘCIE SEKCJI ZMIENNYCH ZAKRESU MODUŁU -----
  var
    configMap = {
      anchor_schema_map : {
        chat : { open : true, closed : true }
      },
      main_html : String()
      ...
    },
    stateMap = {
      $container : null,
      anchor_map : {},
      is_chat_retracted : true
    },
    jqueryMap = {},
    copyAnchorMap, setJqueryMap, toggleChat,
    changeAnchorPart, onHashchange,
    onClickChat, initModule;

```

Definiowanie mapy używanej przez uriAnchor dla celów walidacji.

Zapisanie bieżącej wartości kotwicy w mapie w stanie modułu stateMap.anchor_map.

Deklarowanie trzech dodatkowych metod: copyAnchorMap, changeAnchorPart i onHashchange.

```
//----- ZAKOŃCZENIE SEKCJI ZMIENNYCH ZAKRESU MODUŁU -----
```

```
//----- ROZPOCZĘCIE SEKCJI METOD NARZĘDZIOWYCH -----
```

```
// Zwraca kopię przechowywanej mapy kotwicy; minimalizuje narzut.
```

```
copyAnchorMap = function () {
    return $.extend( true, {}, stateMap.anchor_map );
};
```

Użyj narzędzia `extend()` jQuery, aby skopiować obiekt. Jest to wymagane, ponieważ wszystkie obiekty JavaScript są przekazywane przez referencję, a poprawne skopiowanie takiego obiektu nie jest łatwe.

```
//----- ZAKOŃCZENIE SEKCJI METOD NARZĘDZIOWYCH -----
```

```
//----- ROZPOCZĘCIE SEKCJI METOD DOM -----
```

```
...
```

```
// Rozpoczęcie metody DOM /changeAnchorPart/.
```

```
// Cel: zmiana części komponentu kotwicy URL.
```

```
// Argumenty:
```

```
// * arg_map — mapa opisująca, którą część kotwicy URI chcemy zmienić.
```

```
// Zwraca: wartość logiczną (boolean):
```

```
// * true — część kotwicy adresu URI została zaktualizowana;
```

```
// * false — część kotwicy adresu URI nie mogła zostać zaktualizowana.
```

```
// Akcja:
```

```
// bieżąca reprezentacja kotwicy przechowana w stateMap.anchor_map;
```

```
// sposób kodowania przez wtyczkę uriAnchor został omówiony w kolejnym podpunkcie.
```

```
// Ta metoda:
```

```
// * tworzy kopię tej mapy za pomocą metody copyAnchorMap();
```

```
// * modyfikuje wartości kluczy za pomocą arg_map;
```

```
// * zarządza rozróżnianiem pomiędzy niezależnymi
```

```
// i zależnymi wartościami w kodowaniu;
```

```
// * próbuje zmienić adres URI za pomocą uriAnchor;
```

```
// * zwraca true w przypadku powodzenia oraz false, jeśli operacja się nie powiedzie.
```

```
//
```

```
changeAnchorPart = function ( arg_map ) {
```

```
    var
```

```
        anchor_map_revise = copyAnchorMap(),
```

```
        bool_return = true,
```

```
        key_name, key_name_dep;
```

```
// Rozpoczęcie wprowadzania zmian w mapie kotwicy.
```

```
KEYVAL:
```

```
for ( key_name in arg_map ) {
```

```
    if ( arg_map.hasOwnProperty( key_name ) ) {
```

```
// Przeskakiwanie zależnych kluczy podczas iteracji.
```

```
    if ( key_name.indexOf( '_' ) === 0 ) { continue KEYVAL; }
```

```
// Aktualizacja niezależnej wartości klucza.
```

```
    anchor_map_revise[key_name] = arg_map[key_name];
```

```
// Aktualizacja odpowiadającego klucza zależnego.
```

```
key_name_dep = '_' + key_name;
```

```
if ( arg_map[key_name_dep] ) {
```

```
    anchor_map_revise[key_name_dep] = arg_map[key_name_dep];
```

```
}
```

```
else {
```

```
    delete anchor_map_revise[key_name_dep];
```

```
    delete anchor_map_revise['s' + key_name_dep];
```

```
}
```

Dodanie narzędzia `changeAnchor-Part`, aby aktualizować kotwicę w sposób niepodzielny. Narzędzie wykorzystuje mapę tego, co chcemy zmienić (np. {chat: 'open'}), i aktualizuje tylko określoną parę klucz-wartość w komponencie kotwicy.

```

    }
  }
// Zakończenie wprowadzania zmian w mapie kotwicy.

// Rozpoczęcie próby aktualizacji URI;
// przywrócenie poprzedniego w przypadku niepowodzenia.
try {
  $.uriAnchor.setAnchor( anchor_map_revise );
}
catch ( error ) {
// Zastąpienie adresu URI stanem obecnym.
  $.uriAnchor.setAnchor( stateMap.anchor_map,null,true );
  bool_return = false;
}
// Zakończenie próby aktualizacji adresu URI...
return bool_return;
};
// Zakończenie metody DOM /changeAnchorPart/.
//----- ZAKOŃCZENIE SEKCJI METOD DOM -----

//----- ROZPOCZĘCIE SEKCJI PROCEDUR OBSŁUGI ZDARZEŃ -----
// Rozpoczęcie procedury obsługi zdarzeń /onHashchange/.
// Cel: obsługa zdarzenia hashchange.
// Argumenty:
// * event — obiekt zdarzeń jQuery.
// Ustawienia: brak.
// Zwraca: false.
// Akcja:
// * parsuje komponent kotwicy adresu URI;
// * porównuje zaproponowany stan aplikacji ze stanem bieżącym;
// * dostosowuje aplikację tylko wtedy, kiedy proponowany stan
//   różni się od istniejącego.
//
onHashchange = function ( event ) {
  var
    anchor_map_previous = copyAnchorMap(),
    anchor_map_proposed,
    _s_chat_previous, _s_chat_proposed,
    s_chat_proposed;

// Próba parsowania kotwicy.
  try { anchor_map_proposed = $.uriAnchor.makeAnchorMap(); }
  catch ( error ) {
    $.uriAnchor.setAnchor( anchor_map_previous, null, true );
    return false;
  }
  stateMap.anchor_map = anchor_map_proposed;

// Zmienne złożone.
  _s_chat_previous = anchor_map_previous._s_chat;
  _s_chat_proposed = anchor_map_proposed._s_chat;

// Rozpoczęcie dostosowywania komponentu czatu, jeśli został zmieniony.
  if ( ! anchor_map_previous
    || _s_chat_previous !== _s_chat_proposed
  ) {
    s_chat_proposed = anchor_map_proposed.chat;

```

Nie ustawiaj kotwicy, jeśli nie będzie zgodna ze schematem (wtyczka uriAnchor wygeneruje wyjątek). Jeśli tak się stanie, przywróć komponent kotwicy do poprzedniego stanu.

Dodanie procedury obsługi zdarzeń onHashchange do obsługi zmian kotwicy adresu URI. Użyj wtyczki uriAnchor, aby przekonwertować kotwicę na mapę i porównać z poprzednim stanem w celu określenia akcji. Jeśli proponowana zmiana kotwicy jest nieprawidłowa, kotwica jest resetowana do poprzedniej wartości.

```

switch ( s_chat_proposed ) {
  case 'open' :
    toggleChat( true );
    break;
  case 'closed' :
    toggleChat( false );
    break;
  default :
    toggleChat( false );
    delete anchor_map_proposed.chat;
    $.uriAnchor.setAnchor( anchor_map_proposed, null, true );
}
}
}
// Zakończenie dostosowywania komponentu czatu, jeśli został zmieniony.

return false;
};
// Zakończenie procedury obsługi zdarzeń /onHashchange/.

// Rozpoczęcie procedury obsługi zdarzeń /onClickChat/.
onClickChat = function ( event ) {
  changeAnchorPart({
    chat: ( stateMap.is_chat_retracted ? 'open' : 'closed' )
  });
  return false;
};
// Zakończenie procedury obsługi zdarzeń /onClickChat/.
//----- ZAKOŃCZENIE SEKCJI PROCEDUR OBSŁUGI ZDARZEŃ -----
//----- ROZPOCZĘCIE SEKCJI METOD PUBLICZNYCH -----
// Rozpoczęcie metody publicznej /initModule/.
initModule = function ( $container ) {
  ...
// Skonfigurowanie wtyczki uriAnchor do stosowania schematu.
  $.uriAnchor.configModule({
    schema_map : configMap.anchor_schema_map
  });
}

// Obsługa zdarzeń zmiany kotwicy URL.
// Robi się to po tym, jak wszystkie moduły funkcji zostaną skonfigurowane
// i zainicjowane. W przeciwnym razie nie będą one gotowe do obsługi
// zdarzenia wyzwalającego, które jest stosowane, aby się upewnić, że kotwica
// jest uznana za załadowaną.
//
$(window)
  .bind( 'hashchange', onHashchange )
  .trigger( 'hashchange' );
}
// Zakończenie metody publicznej /initModule/.

return { initModule : initModule };
//----- ZAKOŃCZENIE SEKCJI METOD PUBLICZNYCH -----
}());

```

Zmodyfikowanie procedury obsługi zdarzeń `onClickChat`, aby zmieniała tylko parametr `chat` kotwicy.

Skonfigurowanie wtyczki `uriAnchor`, aby przeprowadzała test względem schematu.

Wiązanie procedury obsługi zdarzeń `hashchange` i natychmiastowe wywołanie jej, aby moduł wziął pod uwagę zakładkę przy początkowym ładowaniu.

Dostosowaliśmy kod, więc wszystkie kontrolki historii (przyciski *Wstecz* i *Dalej*, zakładki oraz przeglądanie historii) powinny działać zgodnie z oczekiwaniami. Ponadto kotwica

powinna „naprawiać się”, jeśli ręcznie zmienimy ją, wpisując nieobsługiwane wartości lub parametry — spróbuj np. zmienić kotwicę w pasku adresu przeglądarki, wpisując `#!chat=barney` i wciskając *Enter*.

Działają już kontrolki historii, omówmy więc sposób używania kotwicy do sterowania stanem aplikacji. Najpierw pokażemy, w jaki sposób korzystać z wtyczki `uriAnchor` do kodowania i dekodowania kotwicy.

SPOSÓB KODOWANIA I DEKODOWANIA KOTWICY PRZEZ WTYCZKĘ URIANCHOR

Używamy zdarzenia `hashchange` jQuery do rozpoznawania zmian w komponencie kotwicy. Stan aplikacji jest kodowany z wykorzystaniem koncepcji **niezależnych** i **zależnych** par klucz-wartość. Za przykład może posłużyć nam poniższa kotwica zaznaczona **pogrubioną czcionką**:

```
http://localhost/spa.html#!chat=profile:on:uid,suzie|status,green
```

W tym przykładzie **niezależnym** kluczem jest `profile`, a jego wartość to `on`. Kluczami, które dalej definiują stan `profile`, są klucze **zależne** umieszczane po separatorze w postaci dwukropka (`:`). Należą do nich klucze `uid` o wartości `suzie` oraz `status` o wartości `green`.

Wtyczka `uriAnchor` (*js/jq/jquery.uriAnchor-1.1.3.js*) zajmuje się kodowaniem i dekodowaniem wartości zależnych i niezależnych. Możemy użyć metody `$.uriAnchor.setAnchor()` do zmiany adresu URI przeglądarki w taki sposób, aby pasował do wcześniejszego przykładu:

```
var anchorMap = {
  profile : 'on',
  _profile : {
    uid : 'suzie',
    status : 'green'
  }
};
$.uriAnchor.setAnchor( anchorMap );
```

Metoda `makeAnchorMap` może być używana do odczytywania kotwicy i parsowania jej do postaci mapy:

```
var anchorMap = $.uriAnchor.makeAnchorMap();
console.log( anchorMap );

// Jeśli komponentem kotwicy adresu URI w przeglądarce jest
// http://localhost/spa.html#!chat=profile:on:uid,suzie|status,green,
//
// to metoda console.log( anchorMap ) powinna wyświetlić:
//
// { profile : 'on',
//   _profile : {
//     uid : 'suzie',
//     status : 'green'
//   }
// };
//
```

Mamy nadzieję, że już lepiej rozumiesz, w jaki sposób wtyczka `uriAnchor` może być używana do kodowania i dekodowania stanu aplikacji wyrażonego w komponencie kotwicy adresu URI. Zobaczmy teraz, w jaki sposób użyć komponentu kotwicy adresu URI do sterowania stanem aplikacji.

STEROWANIE STANEM APLIKACJI ZA POMOCĄ ZMIAN KOTWICY

Nasza strategia kontroli historii polega na tym, że każde zdarzenie zmieniające możliwe do zapisania w zakładkach stan powinno wykonywać dwie czynności:

- 1) zmienić kotwicę;
- 2) niezwłocznie powrócić.

Dodaliśmy do powłoki metodę `changeAnchorPart`, która pozwoliła nam aktualizować tylko część kotwicy, gwarantując jednocześnie, że niezależne i zależne klucze i wartości są obsługiwane prawidłowo. Ujednoliciło to logikę zarządzania kotwicą i *jest jedynym sposobem modyfikowania kotwicy przez aplikację*.

Kiedy mówimy „niezwłocznie powrócić”, mamy na myśli, że po zmianieniu kotwicy praca procedury obsługi zdarzeń jest zakończona. Nie zmienia ona elementów strony i nie aktualizuje zmiennych lub flag. Po prostu wraca bezpośrednio do zdarzenia wywołującego. Zostało to zilustrowane w naszej procedurze obsługi zdarzeń `onClickChat`:

```
onClickChat = function ( event ) {
  changeAnchorPart({
    chat: ( stateMap.is_chat_retracted ? 'open' : 'closed' )
  });
  return false;
};
```

Ta procedura obsługi zdarzeń korzysta z metody `changeAnchorPart`, aby zmienić parametr `chat` kotwicy, a następnie szybko powraca. Ponieważ komponent kotwicy został zmieniony, inicjowane jest zdarzenie przeglądarki `hashchange`. Powłoka nasłuchuje zdarzeń `hashchange` i podejmuje działania na podstawie zawartości kotwicy. Jeśli powłoka zauważy np., że wartość parametru `chat` zmieniła się z `open` na `closed`, zamyka suwak czatu.

Możesz potraktować kotwicę (modyfikowaną przez metodę `changeAnchorPart`) jako interfejs *API* dla stanów możliwych do zapisania w zakładkach. Piękno tego podejścia polega na tym, że nie ma znaczenia, *dłaczego* kotwica została zmieniona — zmodyfikowała ją aplikacja, użytkownik kliknął zakładkę, użył przycisków *Wstecz* lub *Dalej* albo bezpośrednio wpisał adres w pasku adresu przeglądarki. W każdym przypadku wszystko działa poprawnie i wykorzystywana jest tylko pojedyncza ścieżka wykonywania.

3.7. Podsumowanie

Zakończyliśmy implementację dwóch podstawowych odpowiedzialności powłoki. Utworzyliśmy dwa kontenery funkcji i przeprowadziliśmy ich stylizację. Utworzyliśmy także framework do sterowania stanem aplikacji za pomocą kotwicy adresu URI. Aby lepiej zilustrować te koncepcje, zaktualizowaliśmy suwak czatu.

Nasza praca nad powłoką jeszcze się nie skończyła, ponieważ musimy zająć się jej trzecią podstawową odpowiedzialnością: koordynowaniem modułów funkcji. W kolejnym rozdziale omówimy, jak budować moduły funkcji, konfigurować i inicjować je za pomocą powłoki oraz jak je wywoływać. Izolowanie funkcji w osobnych modułach znacznie poprawia niezawodność, utrzymywalność, skalowalność oraz przepływ pracy. Zachęca także do wykorzystania i projektowania modułów zewnętrznych. Zostań z nami — to jest chwila prawdy.

Skorowidz

A

AddThis, 121
adres
 URI, 83, 109, 120, 133
 część, 281
 indeksowanie, 110, 350
 URL, 133, 277
Airbrake, 356
AJAX, 26, 75
Akamai, 358
algorytm
 LRU, 365
 round-robin, 300
Amazon, 358
anchor interface, *Patrz:* interfejs kotwicy
anonymous function, *Patrz:* funkcja anonimowa
aplikacja
 desktopowa, 107
 internetowa, 107
 jednostronicowa, *Patrz:* SPA
 Socket.IO, 290
 stan, 107, 115, 116
 zakończenie, 441
arkusz stylów, 126, 139, 161, 201
aktualizacja, 237
atak
 duży wektor, 289
 wstrzyknięcia SQL, 261
authentication, *Patrz:* uwierzytelnianie
authorization, *Patrz:* autoryzacja
autoryzacja, 260, 288, 289

B

basic authentication, *Patrz:* uwierzytelnianie podstawowe
baza danych, 299
 buforowanie, *Patrz:* buforowanie bazy danych grafowa, 300

MongoDB, *Patrz:* MongoDB
NoSQL, 29, 300
relacyjna, 300
 serwer, *Patrz:* serwer bazy danych
bąbelkowanie, 105, 357
bezpieczeństwo, 25
biblioteka
 frameworków, 252
 jQuery, *Patrz:* jQuery
 kolejność załączania, 130
 PhoneGap, 27
 Socket.IO, *Patrz:* Socket.IO
 TaffyDB2, *Patrz:* TaffyDB2
błąd, 356, 357
bookmark component, *Patrz:* komponent zakładki
Bringham Robert, 379
buforowanie, 359
 bazy danych, 359, 360
 HTTP, 359, 362, 363, 364, 365
 po stronie klienta, 359
 przetworzonych odpowiedzi serwera, 359
 serwera, 359, 365, 366
 zapytań, 360, 371
Bugsense, 356

C

caching, *Patrz:* buforowanie
callback, *Patrz:* funkcja wywołanie zwrotne
Cappuccino, 27
Cassandra, 300
Catalyst, 123
CDN, 358
Chrome, 31
 konsola, 39
 narzędzia dla programistów, 38, 353
ciasteczka, 84
CloudFront, 358
CoffeeScript, 27
Connect, 266, 267, 271

Content Delivery Network, *Patrz:* CDN
 cookie, *Patrz:* ciasteczka
 Crockford Douglas, 33
 CRUD, *Patrz:* operacja CRUD
 CSS, 85, 87, 90, 92, 97, 99, 126, 137, 139, 404
 czat, 163, 165, 206, 229
 okno, 165
 rozmówca, 207
 wiadomość, *Patrz:* wiadomość

D

dane, 165
 baza, *Patrz:* baza danych
 moduł, *Patrz:* moduł danych
 POST, 276
 prawdziwe, 182
 przekazywanie przez referencję, 134
 symulowanie, 183
 synchronizowanie, 261
 transformacja, 300
 typ, 134
 walidacja, 315, 316
 wiązanie, *Patrz:* wiązanie danych
 data binding, *Patrz:* wiązanie danych
 definicja typu dokumentu, *Patrz:* DTD
 Disqus, 121, 124
 Distribution Cloud, 359
 Django, 123
 Document Object Model, *Patrz:* DOM
 Document Type Definition, *Patrz:* DTD
 DOM, 33, 86, 403
 magazyn, 360
 DOM storage, *Patrz:* DOM magazyn
 domain-specific template language, *Patrz:* język
 szablonów
 domknięcie, 73, 74, 75, 76, 78
 DoubleClick, 121
 DSL, *Patrz:* język szablonów
 DTD, 317
 Dust, 237
 dziedziczenie, 67, 398
 dziennik zdarzeń, 267, 356

E

Edgecast, 358
 edytor tekstu, 96
 vim, *Patrz:* vim
 ekran, 140
 Errorception, 356

execution context, *Patrz:* kontekst wykonywania
 execution context object, *Patrz:* obiekt
 kontekstu wykonywania
 Express, 268, 269, 270, 271, 272, 273, 278, 280,
 290, 363

F

feature module, *Patrz:* moduł funkcji
 Firefox, 31
 Flash, 25
 FlashSocket, 289, 338
 FMVC, 123
 format JSON, 300, 302, 303, 312
 Fractal Model-View-Controller, *Patrz:* FMVC
 fraktal, 123
 framework
 Connect, *Patrz:* Connect
 Express, *Patrz:* Express
 pośredniczący, 266
 testowy, 420
 WWW, 268
 funkcja, 47, 52, 67, 68, 400
 anonimowa, 68, 69, 70, 71, 72
 basicAuth, 288
 connect.logger, 268
 czatu, *Patrz:* czat
 deklarowanie, 402
 eval, 408
 fabryki, 61
 globalna, 402
 kontener, 83, 90, 100
 renderowanie, 85, 95
 moduł, *Patrz:* moduł funkcji
 nazwa, 389, 390, 391, 396
 Passport, 289
 polimorficzna, 397
 pośrednicząca, 266, 267, 288
 przypisanie wartości, 402
 setInterval, 357, 408
 setTimeout, 357, 408
 walidacji, 321
 wynoszenie, 51
 wywołanie zwrotne, 134
 wywoływanie, 54, 402

G

garbage collectors, *Patrz:* pamięć mechanizm
 odzyskiwania
 Google Analytics, 354, 356

Google Chrome, *Patrz:* Chrome
 Google Web Toolkit, *Patrz:* GWT
 Googlebot, 350, 352, 353
 GWT, 27, 28

H

halting problem, *Patrz:* problem stopu
 Handlebars, 237
 hash fragment, *Patrz:* identyfikator fragmentu
 HTML
 konwersja na JavaScri, 95
 walidator, *Patrz:* walidator HTML
 HTTP buforowanie, *Patrz:* buforowanie HTTP

I

identyfikator fragmentu, 109, *Patrz też:* kotwica
 IE, 31
 inferencja typów, 392
 instrukcja
 continue, 405
 do, 404, 405
 etykieta, 404
 if, 406
 let, 48
 return, 406
 switch, 404, 406
 try, 407
 var, 407
 with, 407
 interfejs
 API, 120, 122, 124, 132, 134, 135, 136, 178, 181
 implementacja, 139, 144
 kanał, *Patrz:* kanał
 specyfikacja, 209
 testowanie, 195
 kotwicy, 83, 133
 użytkownika, 163, 205, 241
 testowanie, 167
 Internet Explorer, 31
 inżynieria wsteczna, 260

J

jasmine-jquery, 420, 421
 Java, 25
 JavaScript, 24, 25, 33, 45, 46, 300
 akapit, 380
 generator kodu, 27
 implementacja, 26

 kod
 generowany, 27
 K&R, 383
 składnia, 46, 404
 standard kodowania, 377, 378
 walidator, 33
 wiersz
 długość, 379
 łamanie, 382
 znak niedrukowalny, 385
 jednostka
 px, *Patrz:* piksel
 względna, 140
 język
 dynamiczny, 46
 JavaScript, *Patrz:* JavaScript
 luźno typowany, 46
 Perl, 263
 Python, 263
 Ruby, 263
 szablonów, 237
 XML, 317
 jQuery, 26, 29, 30, 33, 37, 86, 179, 403, 421
 instalacja, 86
 JSLint, 33, 37, 89, 103, 323, 408, 410
 instalacja, 408
 konfiguracja, 409
 JSON, 26, 29
 JSON schema, *Patrz:* schemat JSON
 JSONovich, 312
 JSONView, 312
 JSV, 318

K

kanal
 asynchroniczny, 207, 208
 synchroniczny, 207, 208
 klasa, 59, 60, 62
 klient aplikacji, 81
 klucz, 115
 klucz-wartość, 115, 300, 365
 komentarz, 37, 390
 dokumentujący, 386, 388
 objaśniający, 386
 TODO, 388
 komponent zakładki, *Patrz:* kotwica
 komunikat, 189, 222, 224
 disconnect, 343
 emulowanie, 220
 leavechat, 343

komunikat
 listchange, 224
 updateavatar, 224, 225, 345
 updatechat, 342
 konferencja, 224
 konstruktor, 60, 401
 Function, 408
 kontekst wykonywania, 53, 74, 78
 kontener procesów aplikacji, 261
 kotwica, 109, 111, 116
 adresu URI, 83, 109, 120, 133
 interfejs, *Patrz:* interfejs kotwicy

L

link checker, 167
 liquid layout, *Patrz:* szablon płynny
 literal obiektu, *Patrz:* obiekt literal
 LiveFyre, 121, 124
 logika
 aplikacji, 301
 biznesowa, 163, 164, 165, 166, 237, 259
 routingu, 274
 long-polling, *Patrz:* odpytywanie długie

Ł

łańcuch
 prototypów, 56, 59, 62
 zakresów, 56
 znaków, 385, 393
 konwersja, 408

M

mapa, 395, 398
 maper
 obiektowo-dokumentowy, *Patrz:* ODM
 obiektowo-relacyjny, *Patrz:* ORM
 mapowanie, 427
 Memcached, 300, 359, 365, 366
 menedżer paczek, *Patrz:* Node Package
 Manager
 metoda
 _trackEvent, 355
 _trackPageView, 354
 app.all, 280
 app.configure, 271
 app.get, 270, 280
 app.post, 276, 280
 asynchroniczna, 208

bind, 293
 chat.connect, 337
 checkType, 329
 configModule, 134
 get, 270, 277, 280
 handleResize, 156, 158
 http.createServer, 264, 265
 initModule, 134, 137
 io.connect, 292
 listen, 264
 makeMongoId, 329
 object.call, 407
 Object.create, 60, 61, 62
 on, 293
 post, 280, 324
 removeSlider, 156, 157, 158
 response.end, 264
 setSliderPosition, 134
 synchroniczna, 208
 test.done, 426
 testowanie, 215
 try/catch, 357
 wywołania zwrotnego, 341
 MicroMVC, 123
 middleware, *Patrz:* framework pośredniczący
 mocha, 420, 421
 module pattern, *Patrz:* wzorzec modułu
 moduł, 411
 analityczny, 121
 CRUD, 325, 327, 329
 czatu, 334
 danych, 163, 164
 express, 270
 funkcji, 84, 85, 119, 132, 164
 projektowanie, 122
 http, 264, 270
 komentowania na blogach, 121, 124
 powłoki, 84
 Socket.IO, 290, 292
 szablon, 96
 udostępniania, 121
 usług społecznościowych, 121
 zewnętrzny, 121, 122
 MongoDB, 300, 302, 366, 372
 polecenie, 305
 powłoka, 304
 sterownik, 305, 307
 struktura dokumentów, 303
 uruchamianie, 304
 wady, 304
 zalety, 302, 303

Mustache, 237
 MVC, 85, 123
 fraktalowy, *Patrz:* FMVC
 Kontroler, 123
 Model, 128, 163, 164, 166, 167, 189
 konfigurowanie, 168
 Modelem, 205
 Widok, 123
 MySQL, 300

N

namiastka, 126
 narzędzie
 curl, 277
 wget, 277, 324
 Neo4J, 300
 New Relic, 354, 357
 Node Package Manager, 263
 Node.js, 27, 29, 58, 167, 259, 261, 262, 311, 366,
 420, 421, 422
 nodeunit, 420, 421, 424, 426, 428
 konfiguracja, 421

O

obiekt, 176
 anonimowy, 185
 deferred, 426
 global, 58
 instancja, 60
 kontekstu wykonywania, 52, 53, 72, 76, 77
 kopiowanie, 398
 literal, 46, 47
 nazwa, 395, 397
 oparty na prototypach, 59, 60, 66
 szablon, 60
 tworzenie, 182, 185, 206, 210
 walidacja, 317, 318
 window, 58
 obietnica, 341
 Object Document Mapper, *Patrz:* ODM
 Object Relational Mapper, *Patrz:* ORM
 ODM, 307
 odpytywanie długie, 289, 338
 Openfire, 293
 operacja CRUD, 274, 275, 279, 281, 308, 309,
 311, 313, 315, 325
 operator
 !=, 408
 !==, 408

==, 408
 ===, 408
 new, 60
 przecinkowy, 407
 ORM, 307
 Overture, 121

P

pamięć
 mechanizm odzyskiwania, 73, 74, 78
 podręczna, 356, 359, 360, 361, 362, 363,
 364, 366
 pomijanie, 363, 366
 ustawianie, 367
 usuwanie klucza, 367, 371
 system automatycznego zwalniania, 73
 parsera, 408
 patr, 420, 421
 pętla
 for, 404, 405, 407
 licznik, 48
 while, 404, 407
 piksel, 140
 plik, 126
 app.js, 269
 cookie, *Patrz:* ciasteczka
 manifestu, 268
 nazwa, 403
 package.json, 268, 290, 307
 regularny, 86
 routes.js, 325
 spa.html, 30, 35, 137
 statyczny, 273, 358, 403
 zminifikowany, 86
 powłoka, 83, 84, 85, 138, 149, 164
 moduł, *Patrz:* moduł powłoki
 MongoDB, *Patrz:* MongoDB powłoka
 tworzenie, 96
 problem stopu, 441
 procedura
 hashchange, 111
 obsługi zdarzeń
 window.onerror, 357
 projektowanie
 sterowane testami, 227
 promises, *Patrz:* obietnica
 protokół
 Jabber, 293
 XMPP, 293

prototyp, 59, 62
 mutacja, 66
 prototype chain, *Patrz:* łańcuch prototypów
 prototype object, *Patrz:* obiekt prototypowy
 przeglądarka internetowa, 26, 171
 kontrolka, 107
 dotykowa, 173
 historii, 108, 109
 przestrzeń nazw, 166, 402
 CSS, 87, 139
 główna, 137
 JavaScript, 89
 spa.js, 137

R

Rackspace, 359
 Redis, 359, 365, 366
 renderowanie, 42
 responsywność, 42
 REST, 275
 Rhino, 167
 robot internetowy, 349, 350
 żądanie, 351, 352
 routing, 270, 274, 275, 278, 280, 281, 325
 przeniesienie do modułu, 285
 w osobnym module Node.js, 284
 routowanie komunikatów, 293
 rozdzielczość, 140
 Ruby on Rails, 123

S

Safari, 31
 schemat JSON, 318, 319
 scope chain, *Patrz:* łańcuch zakresów
 script kiddie, *Patrz:* skryptowy dzieciak
 Search Engine Optimization, *Patrz:* SEO
 sekcja head, 87, 92
 selektor natywny, 26
 self-executing anonymous function, *Patrz:* funkcja
 anonimowa samowystępująca się
 SEO, 350, 351
 serwer, 260, 261, 262
 Apache, 293, 358
 bazy danych, 300
 buforowanie, *Patrz:* buforowanie serwera
 HTTP, 261
 http.Server, 337
 mod_perl, 261
 mongod, 304

Node.js, 270
 Tornado, 261
 Twisted, 261
 wymiany komunikatów, 293
 ShareThis, 121
 shell, *Patrz:* powłoka
 shell module, *Patrz:* moduł powłoki
 silnik Google V8 JavaScript, 29
 Sinatra, 268
 single page web application, *Patrz:* SPA
 skrypt, 87
 skryptowy dzieciak, 316
 słowo kluczowe, 385
 new, 401
 this, 75
 var, 46, 48, 399
 Socket.IO, 29
 SPA, 22, 23, 24, 28, 42, 86
 historia, 24
 natywna, 27, 45
 przeszukiwanie, 350
 tworzenie, 30
 Spring MVC, 123
 stub, *Patrz:* namiastka
 suwak czatu, 31, 83, 90, 101, 108, 119, 154, 205
 rozwinięcie, 103
 system
 CDN, *Patrz:* CDN
 kontroli wersji, 388
 szablon, 230, 404, 411
 płynny, 92
 system, 237
 szyfrowanie, 289

Ś

środowisko
 development, 272, 273
 production, 272, 273, 363
 przełączanie, 272
 staging, 272
 testing, 272

T

tablica, 398
 asocjacyjna, 395
 kopiowanie, 398
 nazwa, 394
 TaffyDB, 29, 185
 TDD, *Patrz:* projektowanie sterowane testami

test

- automatyczny, 227
- awatarów, 437
- dla transakcji wiadomości, 433
- integracyjny, 416
- jednostkowy, 167
- kontrolny, 430
- mapowanie, *Patrz:* mapowanie testów
- nodeunit, 425
- regresyjny, 167
- stanu wylogowania, 437
- ustawienia, 441
- walidacyjny, 430
- test setting, *Patrz:* ustawienia testowe
- test-driven development, *Patrz:* projektowanie sterowane testami
- testowanie, 427
 - interfejsu użytkownika
 - przy użyciu atrapy danych, 416
 - Modelu bez przeglądarki
 - przy użyciu atrapy danych, 416, 420
 - z wykorzystaniem prawdziwych danych, 416
 - Modelu oraz interfejsu użytkownika
 - z wykorzystaniem prawdziwych danych, 416
 - tryb, 416
 - zestaw testowy, 422
- Tidy, 92
- trasa, *Patrz:* routing
- typ
 - logiczny, 393
 - numeryczny, 394

U

- usługa zewnętrzna, 356
- ustawienia testowe, 441
- uwierzytelnianie, 260, 289
 - po stronie klienta, 260
 - podstawowe, 288, 289
 - za pomocą usługi zewnętrznej, 260, 289
- użytkownik, 177, 337
 - bieżący, 165, 177
 - hasło, 288
 - identyfikator, 277
 - logowanie, 178, 193, 195, 197, 202, 210, 227, 341
 - projektowanie wrażeń, 198, 202
 - uprawnienia, 165
 - wylogowanie, 343

V

- ValueClick, 121
- vim, 96
- vows, 420, 421

W

- walidacja, 37, 260, 261, 317, 318, 321, 322, 408, 410
 - po stronie klienta, 261
 - po stronie serwera, 261
- walidator HTML, 167
- warstwa prezentacji, 237
- Web Storage, 359, 360
- WebKit, 420
- WebSocket, 289, 325
 - gniazdo, 290
 - opóźnienie, 338
 - zalety, 338
- wiadomość, 216, 241
 - w czasie rzeczywistym, 289
 - wymiana na czacie, 341
- wiązanie danych, 250, 252
- właściwość, 62, 63
- wtyczka, 25, 69, 86
 - uriAnchor, 110, 115
- wyrażenie regularne, 96, 394
- wyszukiwarka internetowa, 350
 - optymalizacja, *Patrz:* SEO
- wywołanie
 - DELETE, 275
 - GET, 275
 - PATCH, 275
 - POST, 275
 - PUT, 275
 - zwrotne, 341, 386
- wzorzec
 - modułu, 69, 71
 - REST, *Patrz:* REST

Z

- zasoby, 25
- zdarzenie, 208
 - globalne, 209
 - niestandardowe, 179
- Google, 354
- handleResize, 159
- hashchange, 115, 133, 155
- historii, 109, 111

zdarzenie

- jQuery, 179
- kolejka, 262
- mapowanie, *Patrz*: mapowanie zdarzeń
- onHashchange, 149, 155
- procedura obsługi, 230, 231
- rozgłaszanie, 179
- Socket.IO, 293
- subskrybowanie, 179
- window.onerror, 357, 358
- window.onload, 37
- window.resize, 160

zmienna

- deklarowanie, 46, 398, 399
- globalna, 47, 48, 56, 58, 402
 - nadpisanie, 70
 - Node.js, 422
- lokalna, 47, 56
- nazwa, 387, 389, 390, 391, 392, 395, 397
- przechowywanie, 52
- przypisanie wartości, 399
- statyczna, 66
- typ, 392, 393
 - nieznany, 397
- wyciekanie, 69
- wynoszenie, 49, 51
- zakres, 47, 49, 56, 57, 69, 71, 399

znacznik

- head, 87, 92
- script, 264, 294

znak

- !=, 408
- !==, 408
- #, 109
- #!, 110, 351
- //, 388
- [], 398
- {}, 398
- +, 408
- ++, 408
- ==, 408
- ===, 408
- cudzysłów
 - podwójny, 385
 - pojedynczy, 385
- łańcuch, *Patrz*: łańcuch znaków
- niedrukowalny, 385
- przecinka, 407

zombie, 420

Ż

żądanie

- GET, 270
- HTTP, 264

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**

Gdy uruchamiasz aplikację na swoim komputerze, nie wiesz, czy jest to tradycyjna aplikacja działająca na Twoim sprzęcie, czy może aplikacja internetowa. Nie widać przejść między stronami, a atrakcyjny interfejs użytkownika pozwala Ci błyskawicznie wykonać konieczne zadania. Być może za wyświetlanie interfejsu odpowiedzialna jest wyłącznie przeglądarka, która z serwerem komunikuje się tylko w celu wymiany danych. Tak właśnie działają jednostronicowe aplikacje internetowe (ang. Single Page Applications).

Jeżeli masz pomysł na taką aplikację, a nie wiesz, od czego zacząć, sięgnij po tę książkę. Dzięki niej zbudujesz swoją pierwszą jednostronicową aplikację internetową oraz poznasz możliwości współczesnego języka JavaScript. W kolejnych rozdziałach będziesz budować aplikację, jednocześnie zdobywając wiedzę na temat wzorca MVC, modelu danych, modułów oraz interfejsu. Kiedy już opanujesz tworzenie klienta, przyjdzie czas na implementację serwera. Jest on odpowiedzialny za przetwarzanie danych otrzymywanych z przeglądarki. W trakcie lektury poznasz możliwości Node.js oraz skonfigurujesz bazę danych. Na sam koniec przygotujesz Twoją aplikację do pracy w środowisku produkcyjnym.

Dzięki tej książce:

- poznasz możliwości języka JavaScript
- skorzystasz ze wzorca MVC
- zbudujesz jednostronicową aplikację internetową
- skonfigurujesz serwer współdziałający z Twoją aplikacją
- wykorzystasz potencjał Node.js

Zbuduj swoją pierwszą jednostronicową aplikację internetową!

Michael Mikowski — ma ponad 13-letnie doświadczenie w tworzeniu aplikacji internetowych. Budował systemy obsługujące setki milionów żądań każdego dnia. Wspiera liczne projekty open source, głównie TaffyDB. Jest autorem wielu wtyczek dla jQuery i prelegentem na licznych konferencjach.

Josh Powell — jego doświadczenia z siecią Internet sięgają czasów przeglądarki IE6. Pracuje jako projektant i architekt stron WWW, ma ponad 13-letnie doświadczenie w branży. Jest prelegentem i autorem artykułów dotyczących współczesnej sieci, a także pasjonatem SPA (ang. *Single Page Applications*).

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

Helion	
32145	numer katalogowy
księgarnia internetowa	
	http://helion.pl
zamówienia telefoniczne	
	0 801 339900
	0 601 339900
Sprawdź najnowsze promocje: • http://helion.pl/promocje Książki najchętniej czytane: • http://helion.pl/bestsellery Zamów informacje o nowościach: • http://helion.pl/nowosci	
Helion SA ul. Kościuszki 1c, 44-100 Gliwice tel.: 32 230 98 63 e-mail: helion@helion.pl http://helion.pl	
ISBN 978-83-283-0521-2	
9 788328 305212	
Informatyka w najlepszym wydaniu	
cena: 79,90 zł	