

Twórz strony dostępne dla każdej przeglądarki i dowolnego połączenia internetowego!

- Najlepsze praktyki pisanie kodu HTML, CSS i JavaScript
- Konkretny kod znaczników stylów
- Eliminacja złych nawyków programistycznych
- Tworzenie ogólnodostępnych witryn

tworzenie stron metodą stopniowego ulepszania

Witryny dostępne dla każdego

Todd Parker

Patty Toland

Scott Jehl

Maggie Costello Wachs

» Idź do

- Spis treści
- Przykładowy rozdział

» Katalog książek

- Katalog online
- Zamów drukowany katalog

» Twój koszyk

- Dodaj do koszyka

» Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

» Czytelnia

- Fragmenty książek online

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl
© Helion 1991–2010

Tworzenie stron metodą stopniowego ulepszania. Witryny dostępne dla każdego

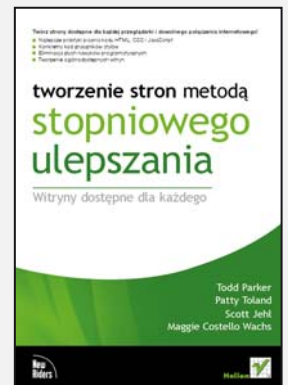
Autorzy: Todd Parker, Scott Jehl, Maggie Costello Wachs, Patty Toland

Tłumaczenie: Łukasz Piwko

ISBN: 978-83-246-2797-4

Tytuł oryginału: [Designing with Progressive Enhancement: Building the Web that Works for Everyone](#)

Format: 168×237, stron: 416



Twórz strony dostępne dla każdej przeglądarki i dowolnego połączenia internetowego!

Obecnie tempo powstawania nowych stron i aplikacji jest zawrotne. Korzystanie z sieci jest coraz łatwiejsze, szybsze i bardziej dynamiczne. Co więcej, uwalniamy się od kabli dzięki rozwojowi telefonów komórkowych z dostępem do internetu i kompaktowych netbooków, zapewniających nam połączenie ze światem niezależnie od miejsca, w którym się znajdujemy. Jednak to bogactwo internetowych zasobów ma również swoje wady. Zaawansowane techniki znakomicie działają w najnowszych przeglądarkach, obsługujących technologie oparte na CSS i języku JavaScript. Istnieje jednak mnóstwo urządzeń, które w najlepszym wypadku obsługują je tylko w ograniczonym stopniu, przez co wysiłki programisty mogą spełznąć na niczym. Projektanci i programiści stron internetowych muszą zatem pogodzić trzy wzajemnie wykluczające się cele. Chcą wykorzystać wszystkie dostępne i ekscytujące technologie, aby móc tworzyć przykuwające uwagę interaktywne strony, nie tracąc jednocześnie na ich dostępności. A wszystko to powinni osiągnąć, tworząc czysty i łatwy do utrzymania kod.

Stopniowe ulepszanie to technika tworzenia stron internetowych zapewniająca wszystkim użytkownikom dostęp do podstawowych treści i opcji witryny – przy użyciu dowolnej przeglądarki i połączenia. Jednocześnie posiadaczom szybszych połączeń i bardziej zaawansowanych przeglądarek oferuje ono ulepszoną wersję strony.

„Tworzenie stron metodą stopniowego ulepszania. Witryny dostępne dla każdego” to praktyczny przewodnik po zasadach tworzenia stron za pomocą nowej metodologii. Dzięki niemu dowiesz się, dlaczego standardowe techniki tworzenia stron wykluczają niektórych użytkowników z grona odbiorców, i jak analizować projekty interfejsów, aby były funkcjonalne na każdym poziomie zaawansowania przeglądarki. Poznasz pakiet testów możliwości przeglądarek, a także najlepsze metody pisania kodu HTML, CSS i JavaScript w technice stopniowego ulepszania.

- Planowanie struktury i organizacji witryny
- Pisanie semantycznego kodu
- Stosowanie stylów
- Elementy interaktywne
- Testowanie możliwości przeglądarek
- Karty
- Okna dialogowe i nakładki
- Przyciski, pola wyboru, suwaki i menu
- Konstruktor list

Dostarczaj pełnię wrażeń wszystkim odbiorcom Twoich stron internetowych!

Spis treści

Podziękowania	11
Wstęp	13

CZĘŚĆ I METODA STOPNIOWEGO ULEPSZANIA OPARTA NA TESTACH

Rozdział 1. Nasze podejście	33
Sprawdzanie możliwości przeglądarek	34
Planowanie stopniowego ulepszania — prześwietlanie	36
Od prześwietlania do działania — technika stopniowego ulepszania	38
Praktyka	40
Rozdział 2. Stopniowe ulepszanie w akcji — prześwietlanie	41
Prześwietlanie — informacje podstawowe	42
Definiowanie hierarchii treści i znajdowanie odpowiedników HTML dla komponentów	43
Tworzenie bazowego kodu znaczników i bezpiecznych stylów	45
Dodawanie rozszerzeń	46
Przypadek 1. — Planowanie struktury i układu witryny z wiadomościami	48
Ocenianie organizacji treści i nadawanie nazw	48
Organizowanie treści za pomocą standardowych elementów HTML	50
Struktura nawigacji	50
Dodawanie treści warstwowej i animowanej	51
Dynamiczne filtrowanie i sortowanie	52
Przypadek 2. — Kolejność czynności oraz sprawdzanie i wysyłanie danych z formularza składania zamówienia	53
Rozkładanie na czynniki formularza składania zamówienia	54
Stosowanie znaczników zapewniających dostępność	60
Definiowanie ograniczeń i walidacja	62
Składanie wersji podstawowej i rozszerzonej	63
Przypadek 3. — Interaktywne wizualizowanie danych w kalkulatorze budżetu	63
Wybór podstawowych elementów dla suwaków	64
Tworzenie dostępnych suwaków z kodu bazowego	66
Tworzenie wykresu	67

Przypadek 4. — Funkcjonalność zwykłych programów w oknie przeglądarki:	
menedżer zdjęć.....	69
Znakowanie elementów nawigacji globalnej.....	70
Skomplikowane interakcje.....	71
Tworzenie niestandardowych formularzy i nakładek	75
Obsługa przycisku Wstecz.....	76
Lista kontrolna prześwietlania	77

Rozdział 3. Pisanie semantycznego kodu **79**

Znakowanie tekstu i obrazów.....	80
Znakowanie tekstu.....	81
Listy.....	86
Dane tabelaryczne.....	87
Obrazy	89
Wstawianie plików multimedialnych	90
Osadzanie treści ze stron zewnętrznych	92
Znakowanie treści interaktywnej.....	93
Łącza	93
Struktura formularza.....	94
Kontrolki formularza	96
Tworzenie kontekstu na stronie.....	101
Elementy blokowe i śródliniowe.....	102
Przypisywanie elementom identyfikatorów i klas.....	103
Oznaczaj główne części strony za pomocą ról WAI-ARIA	104
Kolejność elementów.....	105
Stosuj atrybut title.....	107
Struktura dokumentu HTML.....	108
Definiowanie typu dokumentu.....	109
Nagłówek dokumentu.....	110
Dostępność.....	115
Wytyczne dotyczące dostępności i regulacje prawne	115
Wytyczne WCAG	117

Rozdział 4. Efektywne stosowanie stylów **119**

Sposoby wstawiania arkuszy stylów na stronę.....	120
Zapisywanie stylów w plikach zewnętrznych.....	120
Dołączanie zewnętrznych arkuszy stylów	121
Konwencje nazewnictwa	123

Style dla wersji podstawowej i rozszerzonej witryny	124
Bezpieczny arkusz stylów dla wersji podstawowej	124
Style w wersji rozszerzonej	125
Dostępność	127
Błędy przeglądarek i różne sposoby interpretowania stylów	129
Komentarze warunkowe	129
Często spotykane problemy i ich rozwiązania	130
Rozdział 5. Rozszerzenia skryptowe i elementy interaktywne	135
Dodawanie skryptów do strony	136
Nie wpisuj skryptów bezpośrednio w kodzie strony	136
Dołączanie zewnętrznych skryptów	136
Rola JavaScriptu w kodzie podstawowym	137
Najlepsze praktyki pisania rozszerzeń	137
Uruchamianie skryptów, gdy treść jest gotowa	137
Dodawanie elementów interaktywności	138
Budowanie rozszerzonego kodu znacznikowego przy użyciu JavaScriptu	140
Zapewnianie widoczności treści	143
Stosowanie rozszerzeń stylistycznych	144
Dostępność i właściwości użytkowe	145
Implementacja dostępu za pomocą klawiatury	146
Definiowanie atrybutów WAI-ARIA	147
Testowanie dostępności	148
Dostępność i właściwości użytkowe	149
Rozdział 6. Testowanie możliwości przeglądarek	151
EnhanceJS — pakiet testów możliwości przeglądarek	152
Sposób działania testów	153
Stosowanie rozszerzeń przy użyciu EnhanceJS	155
Konfigurowanie skryptu EnhanceJS	157
Wczytywanie dodatkowych arkuszy stylów	158
Wczytywanie dodatkowych skryptów	160
Modyfikowanie odnośnika do przełączania wersji	161
Wymuszanie określonego wyniku testu EnhanceJS	162
Rozszerzanie zestawu testów EnhanceJS	163
Modyfikowanie pakietu testów za pomocą opcji EnhanceJS	163
Tworzenie nowych lub dodatkowych egzemplarzy EnhanceJS	164
Wyświetlanie powiadomień o błędach	165
Optymalizowanie skryptu EnhanceJS na serwerze	165

CZĘŚĆ II STOPNIOWE ULEPSZANIE — ZAGADNIENIA PRAKTYCZNE

Rozdział 7. Tworzenie komponentów techniką stopniowego ulepszenia	169
Sposób pisania kodu komponentów	170
Organizacja tej części książki.....	171
Przykładowy kod do pobrania.....	173
Rozdział 8. Treść składana	175
Prześwietlanie	176
Tworzenie dostępnej treści zwijanej.....	178
Kod bazowy	178
Wersja rozszerzona.....	179
Skrypt obsługujący zwijanie i rozwijanie treści	182
Korzystanie ze skryptu	184
Rozdział 9. Karty	187
Prześwietlanie	187
Tworzenie kart.....	190
Kod bazowy	190
Wersja rozszerzona.....	192
Skrypt kart.....	196
Dalsze rozszerzanie kart	199
Zakładki i historia	199
Automatyczne obracanie kart	202
Wstawianie na karty treści zewnętrznej.....	203
Wyświetlanie kart jako harmonijki	203
Korzystanie ze skryptu kart	204
Rozdział 10. Chmurki	207
Prześwietlanie	208
Tworzenie chmurek za pomocą atrybutu title	211
Kod bazowy	211
Wersja rozszerzona.....	212
Skrypt.....	214
Tworzenie chmurki z kotwicy	216
Tworzenie chmurki przy użyciu treści zewnętrznej.....	218
Korzystanie ze skryptu chmurek.....	219

Rozdział 11. Kontrolki drzewiaste	221
Prześwietlanie	222
Tworzenie kontrolki drzewiastej.....	224
Kod bazowy	225
Wersja rozszerzona.....	226
Skrypt rozszerzeń.....	230
Korzystanie ze skryptu	236
Rozdział 12. Tworzenie wykresów przy użyciu elementu canvas	239
Prześwietlanie	240
Kod bazowy.....	242
Tworzenie dostępnych wykresów	244
Pobieranie danych z tabeli.....	244
Prezentowanie danych w elemencie canvas	249
Zaawansowana stylizacja tabeli.....	256
Dostępność danych tabeli.....	258
Techniki zaawansowane — wtyczka visualize.js.....	259
Rozdział 13. Okna dialogowe i nakładki	263
Prześwietlanie	264
Tworzenie okna dialogowego.....	266
Kod bazowy	267
Wersja rozszerzona.....	268
Skrypt okna dialogowego.....	273
Dalsza praca nad oknem dialogowym.....	278
Korzystanie ze skryptu	278
Rozdział 14. Przyciski	283
Prześwietlanie	284
Przyciski z elementu input.....	285
Kod bazowy	285
Wersja rozszerzona.....	287
Skrypt zmieniający stany.....	291
Przyciski z zaawansowanym formatowaniem.....	291
Kod bazowy	292
Wersja rozszerzona.....	292
Skrypt zamieniający elementy input na button	294

Korzystanie ze skryptu	296
Dalsze rozszerzanie funkcjonalności przycisków	297
Rozdział 15. Pola wyboru, przyciski radiowe i rankingi gwiazdkowe	299
Prześwietlanie	300
Tworzenie niestandardowego pola wyboru	303
Kod bazowy	303
Wersja rozszerzona.....	305
Skrypt pól wyboru.....	307
Niestandardowe przyciski radiowe.....	309
Kod bazowy	310
Wersja rozszerzona.....	310
Skrypt przycisków radiowych	312
Dalsze modyfikowanie elementów input.....	315
Kod bazowy	315
Kod rozszerzony.....	316
Skrypt komponentu rankingowego.....	318
Korzystanie ze skryptów	321
Rozdział 16. Suwaki	323
Prześwietlanie	324
Tworzenie suwaka.....	327
Kod bazowy	327
Wersja rozszerzona.....	329
Skrypt suwaków	335
Korzystanie ze skryptu suwaka	340
Rozdział 17. Menu wyboru	343
Prześwietlanie	344
Tworzenie dostępnego menu wyboru	346
Kod bazowy	346
Wersja rozszerzona.....	347
Skrypt rozszerzający	355
Zaawansowane stylizowanie menu wyboru	362
Korzystanie ze skryptu	364

Rozdział 18. Konstruktor list	367
Prześwietlanie	368
Tworzenie konstruktora list	370
Kod bazowy	370
Wersja rozszerzona.....	371
Skrypt konstruktora list	375
Wzbogacanie funkcjonalności konstruktora list — zaznaczanie wielu elementów, sortowanie, automatyczne uzupełnianie i menu kontekstowe.....	380
Zaznaczanie wielu elementów.....	380
Sortowanie za pomocą przeciągania i upuszczania.....	380
Automatyczne uzupełnianie.....	380
Menu kontekstowe	381
Korzystanie ze skryptu	381
Rozdział 19. Pole plikowe	383
Prześwietlanie	384
Tworzenie niestandardowego pola plikowego.....	386
Kod bazowy	386
Kod rozszerzony.....	387
Skrypt rozszerzający	390
Korzystanie ze skryptu	393
Podsumowanie	395
Skorowidz	397

Rozdział pierwszy

Nasze podejście

Najnowsze innowacje w zakresie technologii internetowych — głównie wprowadzane za pomocą zaawansowanych technik CSS, działających po stronie klienta JavaScriptu i Ajaksa, oraz wtyczek typu Flash — zamieniły nowoczesne witryny internetowe w atrakcyjne wizualnie aplikacje o dużych możliwościach interakcyjnych. Jest jednak jeden problem — różne przeglądarki i urządzenia na różne sposoby obsługują wymienione technologie. Mimo iż nowoczesne przeglądarki i najnowsze urządzenia przenośne potrafią wyświetlić nawet najbardziej skomplikowane interfejsy, stanowią one tylko część wszystkich odbiorców stron internetowych. A jak już wspominaliśmy wcześniej, tworząc witrynę lub aplikację dostępną tylko wąskiej grupie najnowszych urządzeń i przeglądarek, znacznie utrudniamy sobie dotarcie do jak najszerzego grona odbiorców.

Chcemy sprawić, aby treść, przesłanie i funkcjonalność naszych klientów były dostępne dla każdego — nie tylko dla tych, którzy mają nowoczesne przeglądarki obsługujące najnowocześniejsze technologie, lecz każdego, kto ma urządzenie z dostępem do sieci. Dlatego kilka lat temu postanowiliśmy zastosować filozofię stopniowego ulepszania w projektach naszych klientów.

Teoretycznie idea stopniowego ulepszania jest prosta — twórz strony zgodne ze standardami i przy użyciu czystego HTML-a, aby mieć pewność, że praktycznie każde urządzenie wyświetli ich treść w jakiś użyteczny sposób. Następnie nałóż na ten szkielet warstwy CSS i JavaScript, z których będą mogły skorzystać te przeglądarki, które obsługują te technologie.

Gdy zaczęliśmy stosować to podejście do tworzenia stron i przetestowaliśmy nasz produkt, dokonaliśmy ważnego odkrycia — zapomnieliśmy, że wiele starych przeglądarek i nowszych urządzeń przenośnych tylko *częściowo* obsługuje CSS i JavaScript oraz że użytkownicy mogą wyłączać ich obsługę, aby zwiększyć szybkość, podnieść poziom bezpieczeństwa lub polepszyć jakieś właściwości użytkowe. Obecnie skomplikowane interfejsy aplikacji i widgety są tworzone przy użyciu ściśle współpracującego kodu CSS i JavaScript — wyobraź sobie korzystanie z kalendarza lub paska przewijania w przeglądarce, która obsługuje JavaScript, ale niepoprawnie obsługuje CSS (rezultat byłby opłakany).

Podczas testowania naszych witryn utworzonych przy zastosowaniu technik stopniowego ulepszania znaleźliśmy kilka przeglądarek, które doskonale strony zamieniały w bezużyteczny bałagan, ponieważ uruchamiały skrypty i stosowały style, które nie w pełni były przez nie obsługiwane. Jak więc dowiedzieć się, które przeglądarki poprawnie zastosują nasze ulepszenia?

Doszlśmy do wniosku, że aby mieć pewność, iż stopniowe ulepszanie pozwoli opracować produkt dostępny dla każdego, musimy zrobić trzy rzeczy:

- ▶ Krytycznie przejrzeć projekt i upewnić się, że każdy jego element — nawet najsprytniejszy komponent Web 2.0 lub Ajax — powstał w oparciu o semantyczny kod HTML o prawidłowej strukturze, który nawet w przeglądarkach nieobsługujących CSS i JavaScriptu zachowuje przynajmniej swoje podstawowe funkcje.
- ▶ Przetestować poziom obsługi CSS i JavaScriptu w przeglądarce *przed* zastosowaniem ulepszeń, aby zdecydować, czy w przeglądarce tej poprzestać na podstawowej funkcjonalności, czy dodać dla niej ulepszenia.
- ▶ W przypadku przeglądarek obsługujących rozszerzone funkcje należy zapewnić jak najwyższy poziom dostępności do strony za pomocą klawiatury oraz dodać elementy wspomagające pracę czytelników ekranu.

W tym rozdziale opisany jest test sprawności przeglądarek, za pomocą którego określamy, czy danej przeglądarce można zaserwować rozszerzoną wersję witryny. Następnie przejdziemy do szczegółowego przedstawienia techniki stopniowego ulepszania, którą stosujemy w naszych projektach na co dzień. Tę część zaczniemy od opisu czynności nazywanej przez nas „prześwietlaniem” (ang. *x-ray perspective*). Polega ona na przeanalizowaniu skomplikowanego projektu interfejsu, wyodrębnieniu semantycznego kodu HTML, na bazie którego oparta będzie podstawowa funkcjonalność, oraz zaplanowaniu zaawansowanych elementów CSS i JavaScript przeznaczonych dla nowoczesnych przeglądarek przy jednoczesnym zachowaniu pełnej zgodności z czytnikami ekranu.

Sprawdzanie możliwości przeglądarek

Z naszych wstępnych badań stopniowego ulepszania wynika, że większość programistów rozszerza funkcjonalność swoich stron na jeden z dwóch sposobów — dostarczając rozszerzoną funkcjonalność wszystkim przeglądarkom, które mają włączoną obsługę JavaScriptu lub kierując rozszerzenia do wybranych przeglądarek za pomocą techniki wykrywania typu przeglądarki (ang. *browser sniffing*).

Wykrywanie typu przeglądarki wykluczaliśmy od razu z kilku powodów:

- ▶ Efektywne wykrywanie przeglądarek wymaga drobiazgowej wiedzy na temat ich zachowań (oraz wszystkich ich wersji), przez co utrzymanie odpowiadających za to skryptów jest bardzo trudnym i nigdy niekończącym się zadaniem.

- ▶ Z definicji metoda ta nie jest przyszłościowa. Można wykrywać tylko te przeglądarki, które są dostępne obecnie, a wszystkie te, które pojawią się w przyszłości, będą blokowane, dopóki nie doda się ich do listy.
- ▶ Nawet najpełniejsza lista dopuszczonych przeglądarek może nie spełniać swojego zadania, ponieważ niektóre przeglądarki są skonfigurowane w taki sposób, aby zgłaszały się jako inna aplikacja, aby obejść właśnie te techniki (nazywa się to czasami „robieniem w balona” — ang. *browser spoofing*).

Zatem pierwsze z wymienionych podejść — dostarczanie ulepszeń przeglądarkom z włączoną obsługą JavaScriptu — wydaje się lepszym rozwiązaniem, ponieważ większość przeglądarek obsługujących JavaScript jest w stanie poradzić sobie z rozszerzeniami. Przypomnijmy jednak, że napotkaliśmy zaskakująco dużo przypadków tylko częściowego obsługiwania pewnych ulepszeń przez niektóre przeglądarki, co powodowało błędy JavaScript i układu stron.

Po uważnym przeanalizowaniu tych sytuacji odkryliśmy dwa główne powody ich występowania: niektóre przeglądarki źle interpretowały kod CSS, ponieważ obsługiwały niezgodnie ze standardami model polowy (ang. *box model*), pozycjonowanie, pływanie elementów i wiele innych często używanych własności CSS. Inne natomiast niepoprawnie wykonywały funkcje JavaScript, jak operowanie na modelu DOM, obsługa zdarzeń, zmienianie rozmiaru okna oraz wykonywanie żądań Ajax.

Zastosowanie techniki stopniowego ulepszania byłoby znacznie łatwiejsze, gdybyśmy mieli jakiś niezawodny sposób na uzyskanie przekroju *możliwości* przeglądarek. Wówczas moglibyśmy dostarczać rozszerzenia tylko tym aplikacjom, dla których mielibyśmy pewność, że CSS i JavaScript są obsługiwane poprawnie.

Mając ten cel w pamięci, metodą prób i błędów opracowaliśmy szkielet testowy funkcjonalności przeglądarek. Podstawowa lista funkcji do sprawdzenia w zakresie obsługi JavaScriptu była w zasadzie oczywista — zastosowaliśmy metodę o nazwie „wykrywanie obiektów” (ang. *object detection*), która polegała na pytaniu przeglądarki, czy rozpoznaje rodzime obiekty, takie jak funkcja `document.getElementById` i otrzymywaniu jednoznacznej odpowiedzi „prawda” lub „fałsz”. Każdy test został napisany w taki sposób, aby przeglądarki, które nie rozpoznawały jakiegoś kodu JavaScript, nie zgłaszały żadnych błędów.

Znacznie trudniej było sprawdzić, czy dana przeglądarka poprawnie obsługiwała zaawansowane techniki CSS. Nie da się po prostu sprawdzić, czy przeglądarka poprawnie obsługuje określone własności CSS, takie jak elementy pływające, pozycjonowanie, pionowe ułożenie elementów czy przezroczystość, za pomocą metody wykrywania obiektów.

Dlatego opracowaliśmy zestaw testów CSS. Każdy z nich wstawia na stronę za pomocą JavaScriptu niewidoczne elementy HTML, stosuje zestaw zaawansowanych reguł CSS, a następnie uruchamia funkcję JavaScript sprawdzającą wynik. Aby na przykład dowiedzieć się,

czy przeglądarka poprawnie obsługuje pozycjonowanie, test umieszcza za pomocą CSS w określonym miejscu element `div`, a następnie uruchamia skrypt porównujący uzyskane współrzędne tego elementu z danymi wzorcowymi.

Jeśli przeglądarka przejdzie wszystkie testy z powodzeniem, mamy pewność, że obsługuje CSS i JavaScript w zgodny ze standardami i spójny sposób, a więc powinna poradzić sobie z rozszerzoną funkcjonalnością witryny. W tym momencie test dynamicznie ładuje zaawansowane arkusze stylów i skrypty przekształcające podstawowy kod HTML we wzbogacone środowisko oraz dodaje odnośnik, za pomocą którego można wrócić do wersji podstawowej. Dodatkowo dodaje plik cookie umożliwiający pominięcie wykonywania testów po raz drugi, dzięki czemu następne otwarcie strony będzie szybsze.

Gdy zaczęliśmy stosować nasz pakiet testów, zauważyliśmy wiele korzyści płynących z jego używania w naszych implementacjach stopniowego ulepszania. Przede wszystkim, pozwolił nam skutecznie odróżnić przeglądarki mogące poprawnie zastosować rozszerzone funkcje od przeglądarek, które tego nie potrafiły. Dzięki temu znacznie zmniejszyliśmy ryzyko zrujnowania podstawowej wersji strony przez źle obsłużone rozszerzenia. Ponadto dzięki temu, że rozszerzenia były ładowane tylko w przeglądarkach, które pomyślnie przeszły testy, pozostałym przeglądarkom mogliśmy dostarczyć znacznie prostszy i lżejszy kod — żadne wielkie ilości kodu znaczników, style ani skrypty nie są ładowane z góry. W ten sposób znacznie skróciliśmy czas pobierania stron i ograniczyliśmy liczbę niepotrzebnych odwołań do serwera.

Nasz pakiet testów ma budowę modułarną i jest bardzo elastyczny. Można go dostosować do sprawdzania wybranych funkcji CSS i JavaScript wymaganych w konkretnym projekcie. Jeśli na przykład w tworzonej witrynie nie są wykorzystywane skomplikowane struktury elementów płynących albo skrypty Ajax, możemy odpowiadające im testy wyłączyć.

W swoich projektach nasz pakiet testów dzieli przeglądarki tylko na „podstawowe” i „rozszerzone”, co pozwala nam zapanować nad kodem i ułatwia jego utrzymanie. To binarne rozdzielanie jest dla nas sposobem na dotarcie do możliwie jak największej liczby urządzeń przy jak najmniejszym wysiłku. Na tej podstawie można oczywiście opracować bardziej zaawansowane skrypty, dzielące przeglądarki na więcej bardziej szczegółowych grup albo dostarczające funkcje zoptymalizowane pod kątem konkretnych urządzeń, np. iPhone lub Kindle.

Strukturę i działanie tego pakietu testów szczegółowo omówimy w rozdziale 6., w którym również przedstawimy kilka sytuacji, w których podejście modułarne może być korzystne.

Planowanie stopniowego ulepszania — prześwietlanie

Dostrajając nasz pakiet testów możliwości przeglądarek, zaczęliśmy rozwijać technikę rozkładania skomplikowanych projektów interfejsów sieciowych w celu przygotowania ich do zaprojektowania zgodnie z techniką stopniowego ulepszania.

Czasami udawało się bez problemu znaleźć odpowiedni element HTML do obsługi zaawansowanych składników projektu — indywidualnie wystylizowane menu rozwijane wygląda i zachowuje się tak podobnie do elementu `select`, że oczywiste było, iż należy go użyć. Analogicznie stylizowanie pól wyboru może stanowić nie lada wyzwanie, ale od samego początku wiadomo, że to jest tylko pole wyboru.

Niestety, nie zawsze było tak łatwo. Który z podstawowych elementów HTML powinien zostać użyty do reprezentowania ajaxowego komponentu do oceniania za pomocą gwiazdek w stylu Netflix? To samo dotyczy widocznych na wielu stronach internetowych z wiadomościami komponentów z kartami „most popular/emailed/commented” czy suwaków zakresu daty i ceny służących do filtrowania wyników w witrynie Kayak i innych witrynach e-commerce. Nie wspomnimy już o jeszcze bardziej skomplikowanych, ale bardzo popularnych aplikacjach wykorzystujących Ajax, w których stosowane są techniki przeciągania i upuszczania oraz wiele innych zaawansowanych rodzajów interakcji, takich jak np. Gmail. Żaden z elementów podstawowego języka HTML nie pasuje w naturalny sposób do tych obiektów.

Jednakże, mimo iż wszystkie przedstawione przykłady opisują wysoce zindywidualizowane i interaktywne elementy, oferowane przez nie funkcje — wybieranie opcji na skali, przechodzenie z jednego bloku treści do innego, sortowanie, wyszukiwanie i pobieranie danych — można bez wątpliwości uzyskać również za pomocą standardowego HTML-a. Trzeba tylko trochę pomyśleć i rozłożyć je na czynniki w celu określenia elementów HTML, które również mogą zostać użyte do wykonania tego samego zadania.

Najtrudniejsze w tym wszystkim jest zajrzeć pod style, animacje i inne zachowania, aby zobaczyć podstawowe komponenty leżące u samego podłoża. Ten proces analizowania komponentów nazwalibyśmy prześwietlaniem. Np. kod CSS i skrypty decydujące o interaktywności suwaka były skórą i ubraniem. Natomiast w podstawowej wersji szkieletem było tekstowe pole wejściowe służące do ustawiania wysokich i niskich wartości liczbowych albo przyciski radiowe reprezentujące niewielki zestaw opcji do wyboru albo nawet menu wyboru zawierające długą listę opcji.

Prześwietlanie staje się bardziej skomplikowane i interesujące, gdy prześwietla się skomplikowane projekty, np. witryny zawierające różne rodzaje treści i interaktywne komponenty, aplikacje sieciowe zawierające treść dynamiczną oraz skomplikowane układy i schematy pracy, które dostarczają treść w zależności od działań użytkownika. I to tylko kilka z wielu możliwych przykładów.

W skali makro przeprowadzamy proces prześwietlania, którego celem jest określenie, jak większe części strony (lub wzorce na wielu stronach witryny) współpracują ze sobą, znalezienie wzorców zachowań, które mogłyby stanowić wskazówkę na temat kluczowych elementów treści i funkcjonalności, oraz ocenienie, jak można zoptymalizować te zasoby, aby działały jak najlepiej zarówno w podstawowej, jak i rozszerzonej wersji.

Podczas tej wysokopoziomowej analizy podstawowe zasady na poziomie komponentów i elementów pozostają niezmienione — w każdym przypadku chodzi o zidentyfikowanie wszystkich najważniejszych informacji i funkcji potrzebnych użytkownikowi, znalezienie elementów HTML, które można wykorzystać w określonej sytuacji (na podstawie formatów treści, wymagań w zakresie danych lub zasad biznesowych, ogólnego przepływu), oraz zdecydowanie się na standardowe elementy HTML, które najlepiej spełnią stawiane zadanie.

Bardziej szczegółowo o prześwietlaniu napiszemy w rozdziale 2., w którym dodatkowo przedstawimy kilka procesów dekonstrukcji na kilku skomplikowanych projektach. Natomiast prześwietlaniem pojedynczych komponentów interfejsu zajmiemy się w rozdziałach 18. i 19. Zobaczymy, jak techniki stopniowego ulepszania pozwalają uzyskać cel utworzenia jak najpełniejszych, najlepiej dostępnych i najbardziej użytecznych wersji rozszerzonej i podstawowej witryny.

Zanim jednak przejdziemy do szczegółowego omówienia praktycznego zastosowania techniki prześwietlania, przedstawimy proces, który opracowaliśmy w celu zastosowania techniki stopniowego ulepszania w naszych projektach.

Od prześwietlania do działania — technika stopniowego ulepszania

W miarę udoskonalania naszego pakietu testowego i stosowania techniki prześwietlania po jakimś czasie wypracowaliśmy swoistą terminologię, za pomocą której opisywaliśmy nasz proces stopniowego ulepszania. Dostarczamy dwie wersje produktów — podstawową, która działa we wszystkich urządzeniach z dostępem do internetu (na ile to możliwe), oraz rozszerzoną, która jest dostarczana tylko odpowiednio zaawansowanym przeglądarkom. Kod pierwszej z nich stanowi podstawę, na której opiera się wszystko pozostałe, dlatego nazwaliśmy go „kodem bazowym”. Natomiast kod HTML, CSS i skrypty odpowiadające za zaawansowane funkcje prezentacyjne i zachowania zyskały miano zasobów zaawansowanych lub rozszerzonych.

Aby z powodzeniem dostarczyć te dwie wersje użytkownikom, musimy przestrzegać trzech podstawowych zasad stopniowego ulepszania:

- ▶ Zacząć od czystej treści i kodu HTML o poprawnej strukturze.
- ▶ Precyzyjnie oddzielić warstwę prezentacyjną od układu.
- ▶ Nieinwazyjnie dołączyć zaawansowane zachowania i style po skrupulatnym przemyśleniu zagadnień związanych z dostępnością.

Ponieważ podstawowa wersja jest tworzona w celu umożliwienia dostępu do niej każdemu, pracę musimy *zacząć od klarownej treści i poprawnie skonstruowanego pod względem semantycznym kodu HTML*. Taki kod HTML służy jako funkcjonalna podstawa, na bazie

której można zbudować dodatki rozszerzające. Jest prawdopodobne, że będzie ona działać w większej liczbie różnych urządzeń, a ponadto stanowi klarowną strukturę do poruszania się dla technik wspomagających.

Konstrukcja kodu znaczników w treści strony ma ogromny wpływ na to, jak można będzie zastosować rozszerzenia, np. CSS i JavaScript, oraz to, w jakim stopniu strona będzie dostępna dla niewidomych użytkowników. Najlepiej do stylizowania i dodawania elementów interaktywnych nadają się strony o jasnej i precyzyjnie zorganizowanej strukturze. Ponadto taki kod łatwiej wykorzystać też w innych projektach.

Warunkiem, od którego spełnienia zależy to, czy uda się osiągnąć solidną podstawę służącą do opracowania zarówno podstawowej, jak i rozszerzonej wersji strony, jest zrozumienie właściwości, możliwości i ograniczeń semantyki języka HTML oraz znajomość elementów i atrybutów dostępnych w aktualnie obowiązującej wersji tego języka (nie wspominając o nowych właściwościach, będących jeszcze w fazie powstawania specyfikacji). Dostępne możliwości i najlepsze praktyki, które zalecamy, są opisane w rozdziale 3.

Drugą kluczową zasadą, której ściśle się trzymamy w naszym podejściu stopniowego ulepszania, to *oddzielenie układu od treści*. Najpierw budujemy szkielet strony, a dopiero potem wypełniamy go treścią. To pozwala nam znacznie uprościć proces tworzenia spójnego systemu szablonów dla całego projektu. Ponadto oddzielamy od treści cały kod CSS dotyczący prezentacji i stylu.

Gdy struktura i styl strony powstają niezależnie od jej treści, łatwiej jest tworzyć różne wersje tego samego układu i dostarczać wersje działające optymalnie w różnych przeglądarkach i urządzeniach, ponieważ żaden strukturalny kod CSS nie wpływa na styl treści. To z kolei pozwala nam wziąć pod uwagę rozszerzone typy mediów — standardowe ekrany komputerowe, wyświetlacze telefonów komórkowych i urządzenia wspomagające dla niepełnosprawnych — i selektywnie dostarczać podstawowe i zaawansowane style do wybranych z nich. W rozdziale 4. pokazujemy, które rodzaje stylów można bezpiecznie stosować w większości środowisk, jak zachowują się i wzajemnie oddziałują między sobą skomplikowane reguły CSS oraz jak można scentralizować i zoptymalizować style, aby otrzymać najczystsze i najbardziej niezawodne wersje podstawową i rozszerzoną.

Zaawansowane zachowania i elementy prezentacyjne uzyskane za pomocą JavaScriptu mogą znacznie wzbogacić doznania użytkownika. Jeśli jednak zastosuje się je niewłaściwie lub bezmyślnie, mogą one również sprawić, że witryna będzie całkowicie bezużyteczna dla wielu użytkowników. Istnieją bardzo dobre wytyczne i jasno określone najlepsze praktyki tworzenia i dodawania skryptów w taki sposób, aby *nieinwazyjnie wzbogacały strony* i bezpiecznie je ulepszały w przeglądarkach, które potrafią z nich skorzystać, oraz zapewniające, że podstawowa funkcjonalność nie zostanie zatracona. Te zasady i techniki oraz związane z nimi zagadnienia dotyczące dostępności są opisane w rozdziale 5.

Wyposażeni w dodatkową wiedzę na temat współpracy między kodem HTML, CSS i nieinwazyjnym JavaScriptem, szczególnie przyjrzymy się naszemu pakietowi testów w rozdziale 6. Dowiemy się, jak wykorzystywane są w nim opisane powyżej zasady i techniki w celu umożliwienia zastosowania bardziej niezawodnego podejścia stopniowego ulepszania.

Praktyka

W kilku następnych rozdziałach zwięźle opisujemy najlepsze praktyki, które należy znać, aby móc zaimplementować stopniowe ulepszanie w realnych projektach dla klientów. Natomiast w dalszych rozdziałach nauczymy się pisać poprawny kod HTML, efektywnie stosować CSS oraz dodawać rozszerzenia za pomocą JavaScriptu. Na zakończenie szczególnie opiszemy nasz pakiet testów.