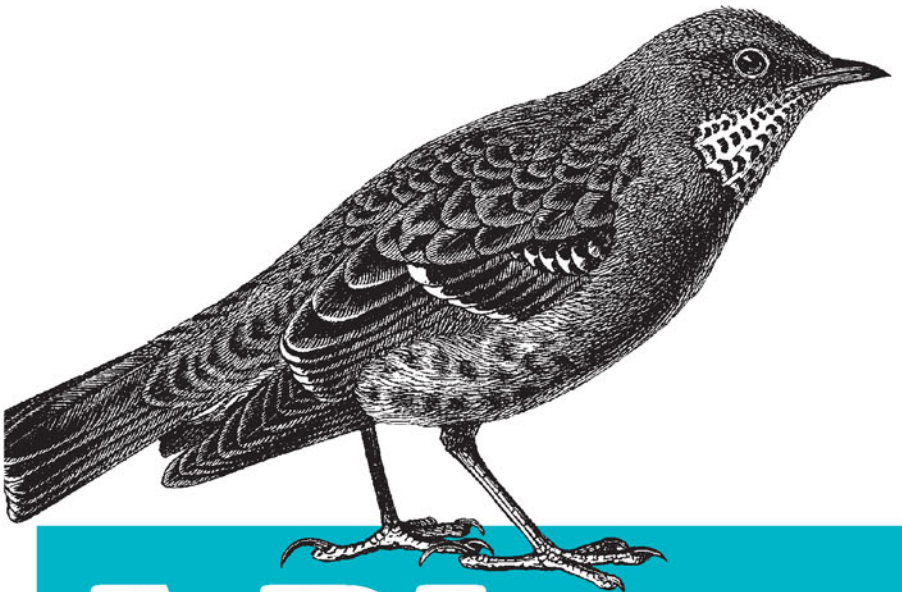
O'REILLY®



API

nowoczesnej strony WWW
Usługi sieciowe w PHP

TWÓJ KLUCZ DO DOSKONAŁEGO API!

Helion

Lorna Jane Mitchell

Tytuł oryginału: PHP Web Services

Tłumaczenie: Łukasz Piwko (wstęp, rozdz. 2 – 13, dodatki), Paweł Halladin (rozdz. 1)

ISBN: 978-83-283-0551-9

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

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

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

Printed in Poland.

Spis treści

ROZDZIAŁ 3.

Nagłówki

W poprzednich rozdziałach przedstawiłam różne prezentacje formatu HTTP i wyjaśniłam, że w żądaniach i odpowiedziach sieciowych jest przesyłanych znacznie więcej informacji niż to, co znajduje się w ich treści głównej. Oczywiście, treść jest najważniejsza i zazwyczaj istotna dla użytkownika, ale nagłówki również przenoszą kluczowe informacje, dzięki którym możliwa jest efektywna komunikacja między klientem i serwerem. Gdyby porównać treść żądania do listu zawierającego gotówkę, to nagłówki byłyby adresem, znaczkiem i np. instrukcją „Otworzyć dnia…" (rysunek 3.1).

Rysunek 3.1. Koperta ze znaczkiem, adresem i stemplem pocztowym

Te dodatkowe informacje sprawiają, że treść dotrze w odpowiednie miejsce, oraz stanowią dla adresata instrukcję, jak należy się z nimi obchodzić.

Nagłówki żądań i odpowiedzi

Wiele nagłówków HTTP może występować zarówno w żądaniach, jak i odpowiedziach. Ale niektóre są przydatne tylko *albo* w jednych, *albo* w drugich. Poniżej znajdują się przykłady prawdziwych nagłówków żądań i odpowiedzi z mojej strony internetowej (*http://www.lornajane.net/*) i przeglądarki Chrome.

Nagłówki żądań:

```
GET / HTTP/1.1
Host: www.lornajane.net
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.19 (KHTML, like Gecko)
Chrome/25.0.1323.1 Safari/537.19
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Accept-Charset: ISO-8859-1,utf-8;q=0.7,+;q=0.3
```

Nagłówki odpowiedzi:

```
HTTP/1.1 200 OK
Server: Apache/2.2.14 (Ubuntu)
X-Powered-By: PHP/5.3.2-1ubuntu4.11
X-Pingback: http://www.lornajane.net/xmlrpc.php
Last-Modified: Thu, 06 Dec 2012 14:46:05 GMT
Cache-Control: no-cache, must-revalidate, max-age=0
Content-Type: text/html; charset=UTF-8
Content-Length: 25279
Date: Thu, 06 Dec 2012 14:46:05 GMT
X-Varnish: 2051611642
Age: 0
Via: 1.1 varnish
Connection: keep-alive
```

W odpowiedzi znajduje się nagłówek `Content-Type`, który byłby obecny także w żądaniu, gdyby było ono typu `POST`. Takie wielozadaniowe nagłówki nazywają się **nagłówkami jednostek** (ang. *entity header*) i dotyczą treści przesyłanej w żądaniu lub odpowiedzi HTTP. Do nagłówków wysyłanych tylko w żądaniach zaliczają się `User-Agent`, `Accept`, `Authorization` i `Cookie`. Natomiast nagłówek `Set-Cookie` jest właściwy tylko odpowiedziom.

Najczęściej używane nagłówki HTTP

W poprzednich przykładach przedstawiłam parę najczęściej używanych nagłówków HTTP. Teraz w kolejnych kilku podrozdziałach przyjrzymy się

tym nagłówkom, które najczęściej spotyka się podczas pracy z różnymi interfejsami API. Pokażę Ci, jak wysyłać i odbierać różne typy nagłówków z poziomu PHP, aby je poprawnie obsługiwać we własnych aplikacjach.

Nagłówek User-Agent

Nagłówek `User-Agent` przekazuje informacje o kliencie, który wysłał żądanie HTTP. Najczęściej jest to klient programowy. Spójrz na poniższy nagłówek:

```
User-Agent Mozilla/5.0 (Linux; U; Android 2.3.4; en-gb; SonyEricssonSK17i
Build/4.0.2.A.0.62) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile
Safari/533.1
```

Jak sądzisz, z jakiego urządzenia wysłano to żądanie? Pewnie myślisz, że wysłałam je z telefonu Sony Ericsson z Androidem… i jest to całkiem możliwe. Ale równie dobrze ktoś mógł użyć poniższego polecenia Curl:

```
curl -H "User-Agent: Mozilla/5.0 (Linux; U; Android 2.3.4; en-gb;
SonyEricssonSK17i Build/4.0.2.A.0.62) AppleWebKit/533.1 (KHTML, like Gecko)
Version/4.0 Mobile Safari/533.1" http://requestb.in/przyklad
```

Nie da się z całą pewnością stwierdzić, czy takie żądanie jak powyższe *rzeczywiście* pochodzi z telefonu z Androidem czy raczej tylko z czegoś, co się pod taki telefon *podszywa*. Na podstawie tych informacji można wysłać odpowiedź — w sumie, jeśli ktoś chce udawać mały telefon z systemem Android, to należy mu odpowiedzieć treścią przeznaczoną dla takiego urządzenia. Ale trzeba też sobie uświadomić, że nagłówek `User-Agent` nie może być używany do żadnych ważnych spraw, np. definiowania własnego nagłówka w celu uwierzytelnienia użytkownika. Jak wszystkie dane z zewnątrz, nagłówek ten może zostać zmanipulowany i należy traktować go podejrzliwie.

W PHP można zarówno przetwarzać, jak i wysyłać nagłówek `User-Agent`. Oto przykład wysłania go przy użyciu strumieni:

```
<?php

$url = 'http://localhost/book/user-agent.php';
$options = array(
    "http" => array(
        "header"  => "User-Agent: zaawansowany magiczny klient HTTP"
    )
);

$page = file_get_contents($url, false , stream_context_create($options));
echo $page;
```

W podobny sposób w żądaniu można ustawić dowolne nagłówki. Analogicznie też można je odebrać, stosując takie samo podejście. Wszystkie potrzebne dane znajdują się w tablicy `$_SERVER`. A w przedstawionym przykładzie zawartość nagłówka `User-Agent` można sprawdzić w elemencie `$_SERVER["HTTP_USER_AGENT"]`.

Oto prosty przykład:

```
<?php

echo "To żądanie zostało wysłane przez: "
    . filter_var($_SERVER['HTTP_USER_AGENT'], FILTER_SANITIZE_STRING);
```

W pracy z urządzeniami przenośnymi często używa się takich nagłówków jak `User-Agent` w połączeniu z biblioteką WURFL (*http://wurfl.sourceforge.net/*) w celu wykrywania możliwości sprzętu i odpowiedniego dostosowywania dla niego treści. Natomiast w przypadku interfejsów API lepiej jest oczekiwać, że klienty użyją różnych nagłówków, żądając odpowiednich dla siebie typów treści, zamiast decydować o tym w sposób centralny.

Nagłówki do negocjacji treści

Nagłówek `Content-Type` służy do określania formatu danych przesyłanych w treści żądania. Dzięki temu odbiorca wie, jak rozszyfrować otrzymane informacje. Pokrewny nagłówek `Accept` służy do określania przez klienta, jaki rodzaj treści jest *dopuszczalny*. Innymi słowy, za pomocą tego nagłówka klient może poinformować serwer, jaką treść jest w stanie obsłużyć. Poniżej znajduje się pokazywany już wcześniej przykładowy nagłówek `Accept` zazwyczaj wysyłany przez przeglądarkę Google Chrome:

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

Przy odczytywaniu nagłówka `Accept` każdą z wartości oddzielonych przecinkami należy traktować jako osobną jednostkę. W powyższym nagłówku klient podał następujące preferencje dotyczące typów treści:

- `text/html`,
- `application/xhtml+xml`,
- `application/xml`,
- `*/*`.

Jeśli dostarczone dane będą w jednym z tych formatów, to nasz klient je rozpozna. Ale dwa ostatnie elementy zawierają dodatkowe informacje: wartość `q`. Określa ona, jak bardzo pożądany jest dany rodzaj treści. Wartość domyślna w tym przypadku to `q=1`.

Ostatni punkt zawiera typ treści */*. Gwiazdki to symbole wieloznaczne, co oznacza, że przeglądarka Chrome rzekomo potrafi obsłużyć każdy rodzaj treści — wydaje się to mało prawdopodobne. Jeśli ktoś wymyśli własny format rozpoznawany tylko przez jego klienta i serwer, to przeglądarka Chrome sobie z nim nie poradzi, więc zapis */* jest mylący.

Użycie nagłówków `Content` i `Content-Type` do informowania o tym, jakiego rodzaju treść rozpoznaje klient i co w rzeczywistości zostało wysłane, nazywa się **negocjowaniem treści**. Dzięki temu, że negocjacje formatów są prowadzone za pośrednictwem nagłówków, metadane nie mieszają się z rzeczywistymi informacjami, co miałoby miejsce, gdyby oba rodzaje parametrów były przesyłane w treści głównej lub adresie URL żądania. Ogólnie rzecz biorąc, nagłówki są dobrym rozwiązaniem.

Negocjować można nie tylko treść. W jednym z wcześniejszych przykładów znajdują się następujące nagłówki:

```
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Accept-Charset: ISO-8859-1,utf-8;q=0.7,+;q=0.3
```

Ilustrują one inne rodzaje negocjacji, np. dotyczące kodowania obsługiwanego przez klienta, preferowanych języków oraz zestawów znaków. Na podstawie tych informacji można odpowiednio sformatować odpowiedź, aby była optymalna dla danego rodzaju urządzeń.

Przetwarzanie nagłówka Accept

Zobaczmy, jak poprawnie powinno się przetwarzać nagłówek `Accept`. Każdy taki nagłówek zawiera listę oddzielonych przecinkami wartości; niektóre z nich zawierają dodatkowo parametr `q`, określający priorytetowość. Brak tego parametru odczytuje się tak, jakby był ustawiony na `1`. Wróćmy do przykładu nagłówka `Accept` z mojej przeglądarki. Można go rozbić na poszczególne części, sprawdzić ich priorytet, a następnie odpowiednio posortować. Poniżej znajduje się przykładowa funkcja zwracająca tablicę obsługiwanych formatów w kolejności od najbardziej pożądanego:

```
<?php

function parseAcceptHeader() {
    $hdr = $_SERVER['HTTP_ACCEPT' ];
    $accept = array();
    foreach (preg_split('/\s*,\s*/' , $hdr) as $i => $term) {
        $o = new \stdclass;
        $o->pos = $i;
        if (preg_match(",^(\S+)\s*;\s*(?:q|level)=([0-9\.]+),i" , $term, $M)) {
            $o->type = $M[1];
```

```
            $o->q = (double)$M[2];
        } else {
            $o->type = $term;
            $o->q = 1;
        }
        $accept[] = $o;
    }
    usort($accept, function ($a, $b) {
        /* pierwsza warstwa: wygrywa najwyższa wartość współczynnika q */
        $diff = $b->q - $a->q;
        if ($diff > 0) {
            $diff = 1;
        } else if ($diff < 0) {
            $diff = - 1;
        } else {
            /* jeśli wystąpi remis, wygrywa pierwszy na liście */
            $diff = $a->pos - $b->pos;
        }
        return $diff;
    });
    $accept_data = array();
    foreach ($accept as $a) {
        $accept_data[$a->type] = $a->type;
    }
    return $accept_data;
}
```

Przeglądarki mogą wysyłać różne nagłówki i wynik ich analizy za pomocą powyższego kodu może być inny niż przedstawiony tutaj.

Dla nagłówka `Accept` wysłanego przez moją przeglądarkę otrzymałam następujący wynik:

```
array(4) {
  ["text/html" ] =>
  string(9) "text/html"
   ["application/xhtml+xml" ] =>
  string(21) "application/xhtml+xml"
   ["application/xml" ] =>
  string(15) "application/xml"
   ["*/*" ] =>
  string(3) "*/*"
}
```

Na podstawie tych informacji można wywnioskować, w jakim formacie najlepiej odesłać dane. Poniżej znajduje się prosty przykładowy skrypt wywołujący funkcję `parseAcceptHeader()`, przeglądający formaty w celu sprawdzenia, które potrafi obsłużyć, i wysyłający informacje w odpowiedniej formie:

```
<?php

$data = array ("greeting" => "Cześć, " , "name" => "Lorna" );

$accepted_formats = parseAcceptHeader();
$supported_formats = array("application/json" , "text/html" );
foreach($accepted_formats as $format) {
    if(in_array($format, $supported_formats)) {
        // użycie tego formatu
        break;
    }
}

switch($format) {
    case "application/json" :
        header("Content-Type: application/json" );
        $output = json_encode($data);
        break;
    case "text/html" :
    default:
        $output = "<p>" . implode(',' , $data) . "</p>" ;
        break;
    }

echo $output;
```

Nagłówek `Accept` można przetwarzać na bardzo wiele sposobów (i wszystkie te techniki można też stosować do przetwarzania nagłówków `Accept-Language`, `Accept-Encoding` oraz `Accept-Charset`), ale bardzo ważne jest, by robić to poprawnie. Jak duże znaczenie ma analiza nagłówka `Accept`, można przeczytać we wpisie *The Accept Header* (Nagłówek Accept) na blogu Chrisa Shifletta, pod adresem *http://shiflett.org/blog/2011/may/the-accept-header*. Funkcja `parseAcceptHeader()` została napisana głównie na podstawie komentarzy znajdujących się pod tym wpisem. Możesz użyć tej funkcji, biblioteki PHP typu mimeparse (*https://github.com/ramsey/mimeparse*), własnego rozwiązania lub narzędzia dostępnego w systemie szkieletowym. Najważniejsze jest to, aby nagłówek był przetwarzany poprawnie, a nie np. przy użyciu funkcji do porównywania łańcuchów.

Demonstracja nagłówków Accept przy użyciu Curl

Poniżej znajduje się kilka przykładów wywołania za pomocą polecenia Curl tego samego adresu URL, ale przy użyciu różnych nagłówków `Accept`, aby otrzymać różne odpowiedzi:

```
curl http://localhost/book/hello.php
Cześć, Lorna

curl -H "Accept: application/json" http://localhost/book/hello.php
```

```
{"greeting":"Cześć, ","name":"Lorna"}

curl -H "Accept: text/html;q=0.5,application/json"
http://localhost/book/hello.php
{"greeting":"Cześć, ","name":"Lorna"}
```

Jeśli chcesz wysłać te żądania z poziomu PHP, to możesz ustawić odpowiednie nagłówki przy tworzeniu żądania. Poniżej znajduje się przykład napisany przy użyciu rozszerzenia PHP `curl`, wysyłający takie same żądania jak poprzednio:

```
<?php

$url = "http://localhost/book/hello.php";

$ch = curl_init($url);
curl_setopt($ch, CURLOPT_HEADER, array(
    "Accept: text/html;q=0.5,application/json" ,
));
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
$response = curl_exec($ch);
echo $response;
curl_close($ch);
```

Liczba obsługiwanych nagłówków zależy od konkretnej aplikacji. Dobrym pomysłem jest oferowanie różnych typów treści, takich jak JSON i XML, a nawet czystego tekstu. Wszystko zależy od rodzaju programu i potrzeb jego użytkowników. Ale jeśli zdecydujesz się wprowadzić obsługę różnych typów treści, to wiesz już, jak to najlepiej zrobić.

Zabezpieczanie żądań za pomocą nagłówka Authorization

Nagłówki mogą dostarczać informacje pozwalające na zidentyfikowanie użytkownika. Oddzielenie ich od danych aplikacji wszystko upraszcza i często sprawia, że aplikacja jest bezpieczniejsza. Jeśli chodzi o bezpieczeństwo użytkowników interfejsów API, to *do usług sieciowych* mają zastosowanie wszystkie techniki stosowane przy zabezpieczaniu serwisów internetowych. Nie trzeba wymyślać niczego nowego, choć nieraz widziałam, jak ktoś wynajdywał koło na nowo, zamiast używać już gotowych standardów.

Podstawowe uwierzytelnianie HTTP

Jednym z najprostszych sposobów na zabezpieczenie strony internetowej jest użycie podstawowego uwierzytelniania HTTP. Polega ono na przesyłaniu w nagłówku `Authorization` każdego żądania zaszyfrowanych danych

poświadczających użytkownika. Zasada działania tej techniki jest bardzo prosta: klient otrzymuje nazwę użytkownika i hasło i wykonuje następujące czynności:

1. Łączy nazwę użytkownika i hasło w łańcuch *nazwaużytkownika*:*hasło*.
2. Koduje wynik w formacie Base64.
3. Wysyła dane w nagłówku, np. `Authorization: Basic` *łańcuch w formacie base64*.
4. Jako że tokeny są przesyłane w postaci tekstowej, powinno się używać protokołu HTTPS.

W opisany sposób można utworzyć nagłówek ręcznie albo użyć specjalnych wbudowanych narzędzi. Poniżej znajduje się przykład napisany przy użyciu rozszerzenia `curl`, wysyłający do strony żądanie pod ochroną podstawowego uwierzytelniania:

```
<?php

$url = "http://localhost/book/basic-auth.php";

$ch = curl_init($url);
curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC ) ;
curl_setopt($ch, CURLOPT_USERPWD, "user:pass" );
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
$response = curl_exec($ch);
echo $response;
curl_close($ch);
```

W języku PHP wszystkie potrzebne informacje można znaleźć w zmiennej superglobalnej `$_SERVER`. Jeśli używane jest podstawowe uwierzytelnianie, nazwę użytkownika i hasło można znaleźć odpowiednio w elementach `$_SERVER["PHP_AUTH_USER"]` i `$_SERVER["PHP_AUTH_PASSWORD"]`. Jeżeli ktoś wyśle żądanie bez danych poświadczających lub z niepoprawnymi informacjami, serwer zamiast żądanej treści może zwrócić kod statusu 401 Unauthorized.

OAuth

Inną metodą zabezpieczania serwisów internetowych, przydatną szczególnie wtedy, gdy jakiś zewnętrzny odbiorca pobiera dane należące do użytkownika, jest OAuth (*http://oauth.net/*). OAuth to standardowa technika umożliwiająca konsumentowi udostępnienie innemu użytkownikowi swoich danych, które są przechowywane przez dostawcę, z którym użytkownik ten jest w jakiś sposób powiązany, bez podawania hasła. Użytkownik wchodzi na stronę dostawcy w celu weryfikacji swojej tożsamości i przydzielenia praw dostępu odbiorcy i może cofnąć te uprawnienia w dowolnym

momencie. W metodzie tej dostawca może odróżnić żądania wysyłane przez użytkownika od żądań wysyłanych przez coś lub kogoś innego w imieniu tego użytkownika.

Szczegółowy opis protokołu OAuth wykracza poza tematykę tej książki (jeśli chcesz dowiedzieć się więcej na jego temat, możesz sięgnąć np. po książkę *Getting Started with OAuth 2.0* wydawnictwa O'Reilly). Wspominam o nim, ponieważ wykorzystuje on nagłówek Authorization i jest powszechnie używany w różnych API.

Nagłówki niestandardowe

Jak prawie każdy aspekt protokołu HTTP, zestaw dostępnych nagłówków nie jest stały. Jeśli ktoś chce przesłać informacje, dla których nie ma odpowiedniego nagłówka, to może wymyślić własny. Jedynym warunkiem jest, aby nazwy takich niestandardowych nagłówków poprzedzić znakami `X-`.

Dobrym przykładem, który często można spotkać w sieci, jest narzędzie Varnish (*https://www.varnish-cache.org/*), dodające do odpowiedzi własne nagłówki. Zainstalowałam je także w swoim serwisie, dzięki czemu w żądaniach znajduję następujące informacje:

```
HTTP/1.1 302 Found
Server: Apache/2.2.14 (Ubuntu)
Location: http://www.lornajane.net/
Content-Type: text/html; charset=iso-8859-1
Content-Length: 288
Date: Tue, 11 Dec 2012 15:53:46 GMT
X-Varnish: 119643096 119643059
Age: 5
Via: 1.1 varnish
Connection: keep-alive
```

Ten dodatkowy nagłówek `X-Varnish` stanowi dowód, że żądanie zostało obsłużone przez narzędzie Varnish. Nie jest to oficjalny nagłówek i dlatego przed jego nazwą znajdują się znaki `X-`. W interfejsach API dostępnych w sieci można znaleźć wiele różnych nagłówków tego typu. Innym doskonałym przykładem jest serwis GitHub (*http://developer.github.com*). Oto, jaką odpowiedź otrzymuję, gdy wyślę żądanie listy repozytoriów związanych z moim kontem (*http://api.github.com/users/lornajane/repos*):

```
HTTP/1.1 200 OK
Server: nginx
Date: Tue, 11 Dec 2012 16:01:00 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
```

```
Status: 200 OK
X-Content-Type-Options: nosniff
Cache-Control: public, max-age=60, s-maxage=60
X-GitHub-Media-Type: github.beta
X-RateLimit-Limit: 60
Content-Length: 106586
Last-Modified: Sat, 01 Dec 2012 11:23:32 GMT
Vary: Accept
X-RateLimit-Remaining: 59
ETag: "8c0bde8e577f52c7f68de5d7099e041b"
```

W przykładzie tym znajduje się kilka niestandardowych nagłówków, ale na szczególną uwagę zasługują te zbudowane wg wzoru `X-RateLimit-*`, które sprawdzają, czy nie jest wysyłanych zbyt wiele żądań. Przy użyciu takich niestandardowych nagłówków między klientem i serwerem można przesłać dodatkowe dane, które nie należą do treści głównej. Dzięki temu żadne informacje nie mieszają się ze sobą.

Skorowidz

L

M

N

O

P

R

S, Ś

T

U

W

X

Z

Ż

O'REILLY®

API nowoczesnej strony WWW
Usługi sieciowe w PHP

Za sukcesem i siłą najpopularniejszych serwisów w sieci często stoi decyzja o szerokim udostępnieniu API (ang. Application Programming Interface). Dzięki temu programiści z całego świata mogą tworzyć rozwiązania oparte na istniejących usługach, integrować różne systemy oraz tworzyć rozszerzenia. Sięgnij po tę książkę, naucz się korzystać z udostępnionego API i zbuduj własne.

Poznaj tajniki budowania przyjaznego interfejsu API. Ta książka pomoże Ci zorientować się, jak działa protokół HTTP, jakie metody udostępnia oraz jakie informacje możesz znaleźć w nagłówkach. Po opanowaniu podstaw przejdziesz do najpopularniejszych formatów wymiany informacji pomiędzy systemami – JSON oraz XML. Zaznajomisz się też z usługami SOAP, RPC i REST. Na sam koniec dowiesz się, jak projektować niezawodne usługi oraz obsługiwać błędy. Jeżeli chcesz, żeby Twoje API zainteresowało innych programistów, musisz zadbać o jego dobrą dokumentację. Zdobądź niezbędną wiedzę o PHP!

Przekonaj się:

- jak działa protokół HTTP
- czym są usługi RPC, SOAP oraz REST
- jak stworzyć dobrą dokumentację
- w jaki sposób obsługiwać błędy
- jak łatwo możesz zbudować własne API

Zbuduj API zgodne z oczekiwaniami innych programistów!

Lorna Jane Mitchell – jest konsultantką i programistką specjalizującą się w języku PHP i tworzeniu API. Prowadzi szkolenia oraz ratuje projekty. Jest autorką popularnych książek oraz materiałów wideo, a także blogerką i organizatorką konferencji PHPNW.

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-283-0551-9

9 788328 305519

Informatyka w najlepszym wydaniu

cena 32,90 zł