Helion

CodeIgniter 4

Zaawansowane tworzenie stron WWW w PHP

Łukasz Sosna

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Jan Paluch
Grafika na okładce została wykorzystana za zgodą Shutterstock.com

Helion S.A.
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/codel4
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-7487-4

Printed in Poland.

Spis treści

Rozdział 3.

Rutowanie URL, odbieranie danych, zapisywanie logów oraz pasek debugera

Rutowanie w obecnych czasach jest elementem bezwzględnie potrzebnym ze względu na wyszukiwarki, które — analizując URL oraz treść strony — są w stanie podwyższyć jej ranking, a co za tym idzie, także jej pozycję w wynikach wyszukiwania. Adres zapisany za pomocą rutowania jest także łatwiejszy w zapamiętaniu przez użytkownika, może także mówić o tym czego dotyczy strona, czy też ułatwiać czytnikom RSS prostsze przechodzenie po łączach, gdyż nie trzeba będzie w nich używać znaków specjalnych. Przykładów zastosowania tego rozwiązania jest o wiele więcej.

Ustalenia sposobu przepisywania adresów URL można dokonać poddając edycji plik */app/config/Routes.php*.

Załóżmy, że chcielibyśmy stworzyć stronę z danymi kontaktowymi. Utwórz w tym celu plik */app/Controllers/Urladdresses.php*. Następnie wpisz do niego standardowe linie kodu występujące w każdym tworzonym dotychczas kontrolerze i utwórz metodę `contact`.

LISTING 3.1. *Kontroler z przykładem przepisywania adresu*

```
<?php

namespace App\Controllers;

class Urladdresses extends BaseController
{
    public function contact()
    {
        echo 'Informacje kontaktowe';
    }

}

?>
```

Teraz wywołaj kontroler (wraz z jego metodą) za pomocą przeglądarki, wpisując adres *http://codeigniter4.lukasz.sos.pl/urladdresses/contact/*. W efekcie otrzymasz stronę z informacją o danych kontaktowych. Wszystko działa. Jednak pewnym problemem wydaje się to, że adres, którym się posłużyłeś, jest zbyt długi i w konsekwencji trudny do zapamiętania. Można w bardzo łatwy sposób go skrócić, pozbywając się nazwy kontrolera z adresu i zostawiając jedynie metodę. W tym celu otwórz plik */app/config/Routes.php*. Następnie przejdź pod definicję domyślnego kontrolera `$routes->get('/', 'Home::index');` i wpisz treść zawartą na listingu.

LISTING 3.2. *Przykład przepisywania adresu URL*

```
$routes->add('contact', 'Urladdresses::contact');
```

Dzięki dodaniu tego wpisu korzystasz z klasy `routes`, w której za pomocą metody `add` dodajesz nową zasadę przepisywania adresu URL. Pierwszy parametr — `contact` — to nazwa, jaką wpiszesz po adresie URL strony, natomiast drugi stanowią połączone dwoma dwukropkami nazwy kontrolera oraz metody, która zostanie wywołana.

Teraz wpisz w polu adresu przeglądarki URL będący skróconą wersją poprzedniego (bez nazwy kontrolera): *http://codeigniter4.lukasz.sos.pl/contact/*. Wynik zapytania wysłanego do serwera w takiej postaci wygląda identycznie jak za pierwszym razem, kiedy wpisywałeś dane z nazwą kontrolera. Jednak teraz, dzięki temu, że adres jest krótszy, wygląda lepiej, łatwiej będzie go zapamiętać, a także — co najważniejsze — będzie lepiej indeksowany przez robota wyszukiwarki.

Parametry w adresie URL

Przepisywanie adresów oferuje także możliwość podawania parametrów, a ponadto pozwala na sprecyzowanie, jaką zawartość mogą one zawierać; każdy z nich może np. być liczbą, ciągiem znaków, haszem, a nawet wyrażeniem regularnym.

Do kontrolera */app/Controllers/Urladdresses.php* dodaj kolejną metodę, tym razem o nazwie `product`. Będzie ona przyjmowała jeden parametr.

LISTING 3.3. *Kontroler z metodą przyjmującą parametr*

```
<?php

namespace App\Controllers;

class Urladdresses extends BaseController
{
    […]

    public function product($ProductNumber)
    {
```

```
        echo 'Informacje o produkcie: ' . $ProductNumber;
    }
}

?>
```

Następnie w pliku */app/config/Routes.php* zdefiniuj nową metodę przepisywania adresów.

LISTING 3.4. *Metoda przepisywania adresów wraz z parametrami*

```
$routes->add('product/(:any)', 'Urladdresses::product/$1');
```

Dzięki zamieszczeniu takiego wpisu przepisywany jest adres, w którym zostanie po domenie wpisane słowo `product`, a następnie dowolny ciąg znaków. Zostanie on zamieniony na kontroler `Urladdresses` i podany do metody `product`, do której przekaże parametr `$1`. Kolejne parametry oznacza się w podobny sposób: `$2`, `$3`, `$4` itd.

Sprawdź to, wpisując do przeglądarki adres *http://codeigniter4.lukasz.sos.pl/product/spodnie*. Zostanie wyświetlona strona z informacją o danym produkcie.

Filtrowanie parametrów

Parametry mogą zostać odpowiednio przefiltrowane już podczas przepisywania adresów URL. Służą do tego specjalne wieloznaczniki. Przykładem takiego wieloznacznika może być zastosowany w kodzie z listingu 3.4 zapis `(:any)`. Oznacza on każdą wartość, jaką będzie zawierał adres URL.

W poprzednim przykładzie wypisywałeś informacje o produkcie, przekazując do metody zawartość w ciągu URL. Możesz jednak znacznie bardziej uściślić rodzaj parametru. Służą do tego wieloznaczniki.

TABELA 3.1. *Filtrowanie parametrów*

Wieloznacznik	Opis
`(:any)`	Pasuje do każdego znaku podanego jako parametr
`(:segment)`	Pasuje do każdego — oprócz ukośnika (/) — znaku podanego jako parametr
`(:num)`	Pasuje do każdego ciągu składającego się z cyfr
`(:alpha)`	Pasuje do każdego ciągu składającego się z liter
`(:alphanum)`	Pasuje do każdego ciągu składającego się z liter, cyfr lub kombinacji liter i cyfr
`(:hash)`	Działa podobnie do `(:segment)`, jednak może służyć do przesyłania haszów różnych elementów, na przykład identyfikatora lub hasła

W kontrolerze */app/Controllers/Urladdresses.php* zaktualizuj teraz metodę `product`, w wyniku czego tym razem pokaże ona tylko numer produktu.

LISTING 3.5. *Kontroler z metodą przyjmującą argument*

```
<?php

namespace App\Controllers;

class Urladdresses extends BaseController
{
    […]

    public function product($ProductNumber)
    {
        echo 'Informacje o produkcie nr: ' . $ProductNumber;
    }
}

?>
```

Następną czynnością, którą należy wykonać, jest zmiana parametru wieloznacznika, który ma przyjmować metoda. W pliku */app/config/Routes.php* zmień w pierwszym parametrze metody `add` wieloznacznik `(:any)` na `(:num)`.

LISTING 3.6. *Metoda przepisywania adresów wraz z parametrem*

```
$routes->add('product/(:num)', 'Urladdresses::product/$1');
```

Teraz wpisz kolejno w oknie adresu przeglądarki dwa adresy URL. Pierwszy będzie zawierał liczbę w parametrze — na przykład 1: *http://codeigniter4.lukasz.sos.pl/product/1*. W wyniku zatwierdzenia tak wpisanego adresu, wyświetli się strona z informacją o produkcie posiadającym ID odpowiadający podanej liczbie. Następnie wpisz ten sam adres, zmieniając jedynie ostatni parametr z liczby na wyraz: *http://codeigniter4.lukasz.sos.pl/product/spodnie*. Otrzymasz komunikat o błędzie informujący, że taka strona nie istnieje.

Definiowanie własnych wieloznaczników

Oprócz wieloznaczników dostępnych w systemie można także zdefiniować własne, jeśli uznamy, że systemowe nam z jakiegoś powodu nie odpowiadają — na przykład ze względu na rozbudowaną treść oraz konieczność innego niż oferowane przez nie filtrowania zawartości adresu URL.

W kontrolerze */app/Controllers/Urladdresses.php* dodaj nową metodę odpowiadającą za wyświetlenie informacji o produkcie.

LISTING 3.7. *Kontroler z metodą przyjmującą argument*

```
<?php

namespace App\Controllers;

class Urladdresses extends BaseController
{
    […]

    public function productshow($ProductId)
    {
        echo 'Informacje o produkcie posiadającym identyfikator: ' . $ProductId;
    }
}

?>
```

Następną czynnością, którą należy wykonać, jest zdefiniowanie własnego wieloznacznika, który ma przyjmować jako parametr metoda. W pliku */app/config/Routes.php* dodaj nowy wieloznacznik. Posłuż się w tym celu metodą `addPlaceholder`, której parametrami są nazwa wieloznacznika oraz zawartość, jaką może przyjąć. Jest to dosyć uproszczona metoda, podobna nieco do wyrażeń regularnych. W naszym przypadku parametrem podanym do metody będzie ciąg znaków zawierających litery od *a* do *g*. Chcemy ponadto, żeby w zwróconym wyniku były dokładnie dwa znaki z tego zakresu.

LISTING 3.8. *Definiowanie własnego wieloznacznika*

```
$routes->addPlaceholder('productitem', '[a-g]{2}');
$routes->add('productshow/(:productitem)', 'Urladdresses::productshow/$1');
```

Wpisz poprawny adres do okna przeglądarki i zatwierdź go wciskając klawisz *Enter*: *http://codeigniter4.lukasz.sos.pl/productshow/ag*. W efekcie wyświetli się strona z informacjami o produkcie. Teraz wpisz ten sam adres, lecz z krótszym (zawierającym tylko jeden znak z wyznaczonego zakresu) drugim parametrem: *http://codeigniter4.lukasz.sos.pl/productshow/a*. W takim przypadku wystąpi błąd i otrzymasz komunikat z informacją, iż strona nie została odnaleziona. Wpisz jeszcze jeden adres: *http://codeigniter4.lukasz.sos.pl/productshow/az*. Także tym razem wystąpi błąd sygnalizowany komunikatem o nieodnalezieniu strony. Wszystko ze względu na parametr. Pierwsza litera (*a*) jest poprawna, jednak druga (*z*) wykracza poza zakres dozwolonych znaków, kończący się na literze *g*.

Wyrażenia regularne

Wyrażenia regularne to element, który pozwoli Ci na dokładne dobranie wszystkich parametrów pojawiających się w adresie URL. Możesz dostosować je do czego tylko będziesz chciał.

W pliku zawierającym kontroler */app/Controllers/Urladdresses.php* dodaj nową metodę, służącą do wyświetlenia produktu posiadającego numer.

LISTING 3.9. *Kontroler z metodą przyjmującą argument*

```
<?php

namespace App\Controllers;

class Urladdresses extends BaseController
{
    […]

    public function productnumber($ProductNumber)
    {
        echo 'Numer produktu: ' . $ProductNumber;
    }
}

?>
```

W pliku */app/config/Routes.php* dodaj nowy wiersz, w którym umieścisz wyrażenie regularne pozwalające na wybranie jedynie liczb ([0-9]) z ciągu podanego jako parametr. Liczb może być jedna lub więcej, co określa zastosowany w wyrażeniu znak +.

LISTING 3.10. *Definiowanie własnego wyrażenia regularnego*

```
$routes->add('productnumber/([0-9]+)', 'Urladdresses::productnumber/$1');
```

Wpisz do przeglądarki internetowej adres *http://codeigniter4.lukasz.sos.pl/productnumber/1234567890*. Strona, która się otworzyła, pokazuje informacje o produkcie mającym numer 1234567890. Teraz spróbuj dodać do adresu literę: *http://codeigniter4.lukasz.sos.pl/productnumber/1234567890a*. Obecnie strona informuje, że strona o podanym adresie URL nie istnieje. W taki oto sposób za pomocą wyrażeń regularnych można uzyskiwać przepisywanie adresów na odpowiednią metodę w kontrolerze.

Grupowe przepisywanie adresów

CodeIgniter został wyposażony w możliwość grupowego przepisywania adresów URL, dzięki czemu nie trzeba się już męczyć z definiowaniem każdego adresu z osobna — wystarczy zastosować metodę `group`.

W pliku */app/config/Routes.php* dodaj definicję grupowego przepisywania adresów URL. Będzie ona polegała na prostej zasadzie. W przypadku zwykłego przepisywania musiałbyś zamieszczać trzy wersje metody `add` z definicjami całego adresu do przepisania. W tym przypadku, po użyciu metody `group`, jako pierwszy parametr podaj ciąg, który ma znaleźć się

w adresie URL, następnie zdefiniuj funkcję, w której parametrem jest `$routes`. Teraz określ — tak jak poprzednio, za pomocą metody `add` — reguły podrzędne. Będę one polegały na przepisywaniu całych adresów: *http://[...]/user/register*, *http://[...]/user/login*, *http://[...]/user/logout*. Dzięki temu można łatwiej zapanować nad przepisywanymi adresami i podzielić je na grupy.

LISTING 3.11. *Grupowanie przepisywania adresów URL*

```
$routes->group('user', function($routes)
{
    $routes->add('register', 'Users::register');
    $routes->add('login', 'Users::login');
    $routes->add('logout', 'Users::logout');
});
```

Własna strona dla błędu 404

W CI można zdefiniować własną stronę dla błędu 404. W pliku */app/config/Routes.php* musisz w tym celu dodać wpis, który zastąpi domyślną stronę z kodem błędu http 404 inną stroną.

W metodzie `set4040verride` napisz funkcję, która wyświetli wybrany przez Ciebie plik błędu.

LISTING 3.12. *Definiowanie nowej strony błędu*

```
$routes->set4040verride(function()
{
    echo view('errors/error404.html');
});
```

Pobieranie danych wysłanych do programu

Dane mogą być wysyłane w różne sposoby. Najpopularniejszy polega na użyciu metod typu GET i POST, jednak istnieje wiele innych możliwości dodania własnych danych za pomocą programu uruchomionego w konsoli lub działającego na serwerze WWW.

Włączanie klasy

Klasa `$request` jest automatycznie ładowana do klasy `Controller`, więc nie trzeba jej do niej jawnie dodawać. Elementy obiektu żądania są dostępne od razu, możesz więc ich używać w swoich kontrolerach, kiedy tylko będziesz tego chciał. Jednak, jeżeli trzeba będzie użyć tej klasy w innych miejscach niż kontrolery, należy ową klasę jawnie załadować.

LISTING 3.13. *Włączanie obiektu żądania request za pomocą usług*

```
$request = \Config\Services::request();
```

Klasę można dodać w konstruktorze klasy, tak aby była dostępna w każdej metodzie, w której będziesz chciał ją zastosować.

LISTING 3.14. *Włączanie klasy w konstruktorze klasy*

```
<?php

use CodeIgniter\HTTP\RequestInterface;

class SomeClass
{
    protected $request;

    public function __construct(RequestInterface $request)
    {
        $this->request = $request;
    }
}
?>
```

Metoda isCLI()

`isCLI()`

Metoda `isCLI()` sprawdza, czy kod został uruchomiony z linii poleceń. Jeżeli tak, wówczas zwróci wartość `True`; w przeciwnym wypadku zwrócona zostanie wartość `False`.

Metoda isAJAX()

`isAJAX()`

Metoda `isAJAX()` pozwala na sprawdzenie, czy kontroler został wywołany przez procedurę AJAX. W takim przypadku zwróci wartość `True`, a w innym — `False`.

Metoda isSecure()

`isSecure()`

Metoda `isSecure()` sprawdza, czy żądanie zostało wywołane poprzez protokół HTTPS. Jeśli tak, otrzymamy wartość `True`, jeśli nie — `False`.

Metoda getVar()

```
getVar([$index = null[, $filter = null[, $flags = null]]])
```

Metoda ta zwraca wartość zmiennej lub klucza, jeżeli istnieją, lub wartość `null` w przypadku ich braku.

Parametry:

- `$index` — nazwa zmiennej lub klucza,
- `$filter` — typ filtra dla zmiennej lub klucza[1],
- `$flags` — flaga dla zmiennej lub klucza[2].

Metoda getGet()

```
getGet([$index = null[, $filter = null[, $flags = null]]])
```

Ta metoda pobiera wartość zmiennej z tablicy GET zdefiniowanej jako pierwszy parametr. Jeżeli nie znajdzie takiej zmiennej, zwraca wartość `null`. Pierwszy parametr nie jest obowiązkowy i w przypadku niepodania go zostanie zwrócona tablica wraz z wszystkimi zmiennymi wysłanymi metodą GET.

Parametry:

- `$index` — nazwa zmiennej lub klucza,
- `$filter` — typy filtrów, które mają zostać zastosowane,
- `$flags` — typy flag, które mają zostać zastosowane przy przetwarzaniu danych.

Metoda getPost()

```
getPost([$index = null[, $filter = null[, $flags = null]]])
```

Jest to metoda przeznaczona do zwracania zmiennych wysłanych metodą POST. Nazwę zmiennej podajemy w pierwszym argumencie. Jeżeli zmienna istnieje, zostanie zwrócona jej wartość, jeżeli nie istnieje — zostanie zwrócona wartość `null`. Pierwszy argument jest wartością niewymaganą, więc można go nie podawać. W takim przypadku zostanie zwrócona cała tablica POST, która została wysłana do kontrolera.

[1] Zestawienie i opis typów filtrów można znaleźć pod adresem: *https://www.php.net/manual/en/filter.filters.php*.

[2] Zestawienie i opis typów flag można znaleźć pod adresem: *https://www.php.net/manual/en/filter.filters.flags.php*.

Parametry:

- `$index` — nazwa zmiennej, którą chcemy pozyskać,
- `$filter` — filtr do zastosowania,
- `$flags` — flaga do zastosowania.

Metoda getPostGet()

```
getPostGet([$index = null[, $filter = null[, $flags = null]]])
```

Metoda ta sprawdza, czy zmienna zdeklarowana w pierwszym parametrze została przesłana metodą POST. W przypadku niewykrycia takiego przesyłania następuje sprawdzenie, czy zmienna została przesłana metodą GET. Jeżeli zmiennej nie ma i w tej tablicy, zwracana jest wartość `null`.

Parametry:

- `$index` — nazwa zmiennej, której poszukujemy,
- `$filter` — filtr do zastosowania do danej zmiennej,
- `$flags` — flaga zastosowana do zmiennej.

Metoda getGetPost()

```
getGetPost([$index = null[, $filter = null[, $flags = null]]])
```

Metoda sprawdza, czy zmienna zdeklarowana w pierwszym parametrze została przesłana metodą GET. W przypadku braku zmiennej w tablicy GET przystępuje do sprawdzenia, czy została ona przesłana metodą POST. Jeżeli zmienna nie istnieje także w tej tablicy, zwracana jest wartość `null`.

Parametry:

- `$index` — nazwa szukanej przez nas zmiennej,
- `$filter` — typ filtra do zastosowania,
- `$flags` — typ flagi do zastosowania.

Metoda getCookie()

```
getCookie([$index = null[, $filter = null[, $flags = null]]])
```

Metoda przeznaczona do sprawdzania wartości ciasteczka (pliku *cookie*). Jeśli plik nie zostanie odnaleziony, zostanie zwrócona wartość `null`, w przeciwnym przypadku metoda zwróci

jego wartość. Można także nie podawać pierwszego argumentu; w takim razie zostanie zwrócona tablica wraz z wszystkimi ciasteczkami wysłanymi do strony.

Parametry:

- `$index` — nazwa pliku ciasteczka,
- `$filter` — typ filtra,
- `$flags` — typ flagi.

Metoda getServer()

`getServer([$index = null[, $filter = null[, $flags = null]]])`

Metoda zwraca zawartość zmiennej opisującej konfigurację kluczem bądź tablicą wskazaną w pierwszym parametrze, którego poszukujemy. W przypadku znalezienia klucza zostanie zwrócona jego zawartość, jeśli zaś metoda go nie odnajdzie, otrzymamy wartość `null`.

Parametry:

- `$index` — nazwa klucza tablicy,
- `$filter` — filtr dotyczący zawartości,
- `$flags` — flaga dla zawartości.

Metoda getUserAgent()

`getUserAgent([$filter = null])`[MK1]

Metoda ta zwraca ciąg znaków reprezentujących przeglądarkę internetową, z której korzysta klient serwisu.

Parametry:

- `$filter` — filtr do zastosowania do zawartości.

Zapisywanie błędów występujących podczas obsługi witryny

Funkcja `log_message()` przekazuje komunikaty o danym zdarzeniu do pliku dziennika systemowego. Pierwszym parametrem tej funkcji jest określenie rodzaju zdarzenia: `debug`, `info`, `notice`, `warning`, `error`, `critical`, `alert` lub `emergency`. Natomiast w drugim parametrze powinien się znaleźć komunikat stanowiący opis danego zdarzenia. Na listingu 2.19

pokazany jest kontroler, którego zadaniem będzie pobranie danych zapisanych w pliku o nazwie *data.txt*. Jeżeli taki plik nie istnieje, na ekranie zostanie wyświetlony komunikat informujący o błędzie; odpowiednia informacja zostanie ponadto zapisana do pliku dziennika.

LISTING 3.15. *Zapisywanie komunikatu o zdarzeniu. W tym przypadku wystąpił błąd*

```
<?php

namespace App\Controllers;

class Getfile extends BaseController
{
    public function getfilecontent()
    {
        if (file_exists('data.txt'))
        {
            $ArrayFile = file('data.txt');
        }
        else
        {
            echo 'Błąd. Brak pliku.';
            log_message('error', 'Plik o nazwie data.txt nie istnieje!');
        }
    }
}

?>
```

Kolejną czynnością, którą należy wykonać, jest konfiguracja systemu logowania błędów. W tym celu otwórz plik */app/Config/Logger.php*. Znajduje się tam zmienna: `public $threshold = 3;`. Jej wartość decyduje o tym, jakie błędy są zapisywane. Zmień wpisaną w niej liczbę 3 na 4, ponieważ zgłoszony w przykładzie błąd ma poziom ważności 4. Od tej pory zapisywane będą wszystkie błędy tego poziomu — oczywiście także te o wyższych wartościach.

Jeśli już to zrobiłeś, wpisz w przeglądarce adres *http://codeigniter4.lukasz.sos.pl/getfile/getfilecontent*. Teraz sprawdź plik logu zapisany w lokalizacji */writeable/logs/[data].log*. Na samym dole pliku znajduje się uzyskany przed chwilą błąd.

LISTING 3.16. *Informacje o błędzie w pliku logu systemowego*

```
ERROR - 2020-08-29 09:11:51 --> Plik o nazwie data.txt nie istnieje!
```

Zapisywanie logów można również zdefiniować posługując się tablicą z numerami błędów, jakie będziemy chcieli zapisywać do plików.

LISTING 3.17. *Logowanie różnych poziomów błędów*

```
public $threshold = [5, 8];
```

TABELA 3.2. *Błędy w aplikacji wraz z ich numerami oraz opisem*

Numer	Poziom	Opis
1	`debug`	Szczegółowe informacje na temat błędów występujących podczas pracy aplikacji
2	`info`	Informacje, które przydadzą się podczas sprawdzania działania aplikacji
3	`notice`	Informacje mające związek z działaniem aplikacji, które doprowadziły do błędu w samej aplikacji
4	`warning`	Ostrzeżenia generowane podczas działania aplikacji, informujące o takich zdarzeniach jak korzystanie z API
5	`error`	Błędy w aplikacji, które wymagają natychmiastowego naprawienia ze względu na to, że powodują jej złe działanie
6	`critical`	Błąd krytyczny, na przykład brak dostępu do komponentu lub zapytania SQL
7	`alert`	Informacja o błędzie, który musi zostać usunięty natychmiast, na przykład z powodu tego, że uniemożliwia uruchomienie strony
8	`emergency`	Najwyższy poziom możliwych problemów — zgłasza błędy powodujące, że strona całkowicie nie nadaje się do użytku

Włączanie paska debugera

Pasek debugera to bardzo ważny dodatek do systemu; powinno się go używać bardzo często podczas pisania i testowania aplikacji. Zawiera mnóstwo danych dotyczących tego, ile trwało wykonywanie kodu strony, ile pamięci RAM zostało przy tym zużyte. Pasek pokaże także zapytania do bazy danych oraz poinformuje, gdzie w hierarchii plików strony są zawarte pliki widoku, ile plików zostało użytych do wygenerowania obecnej strony, wyświetli ścieżkę dostępu w przypadku użycia przepisywania adresu URL. Pokaże ponadto zdarzenia, historię dostępu do strony czy używane zmienne.

W celu włączenia paska debugera musisz przełączyć framework w stan pracy właściwy dla tworzenia aplikacji. W tym celu przejdź do pliku *.env*, który znajduje się domyślnie katalog niżej niż opublikowana strona WWW. Znajdź w nim linię z napisem `CI_ENVIRONMENT`. Należy ją odkomentować poprzez usunięcie znaku hasz (#) z jej początku; od tej pory ta linia będzie uwzględniana podczas wykonywania programu. Teraz musisz jeszcze ustawić tę opcję na wartość `development`.

LISTING 3.18. *Ustawienie stanu pracy pozwalającego na debugowanie kodu*

```
#--------------------------------------------------------------------
# ENVIRONMENT
#--------------------------------------------------------------------

CI_ENVIRONMENT = development
```

RYSUNEK 3.1. *Przykładowy pasek debugera*

Możesz wybrać, jakie elementy mają się pojawić na pasku. Ustawienia te znajdują się w pliku konfiguracyjnym *app/Config/Toolbar.php*.

LISTING 3.19. *Konfiguracja paska debugera*

```
public $collectors = [
        \CodeIgniter\Debug\Toolbar\Collectors\Timers::class,
        \CodeIgniter\Debug\Toolbar\Collectors\Database::class,
        \CodeIgniter\Debug\Toolbar\Collectors\Logs::class,
        \CodeIgniter\Debug\Toolbar\Collectors\Views::class,
        \CodeIgniter\Debug\Toolbar\Collectors\Cache::class,
        \CodeIgniter\Debug\Toolbar\Collectors\Files::class,
        \CodeIgniter\Debug\Toolbar\Collectors\Routes::class,
        \CodeIgniter\Debug\Toolbar\Collectors\Events::class,
];
```

Można dowolnie zmieniać konfigurację paska poprzez zmianę pozycji klas; możliwe jest także wyłączanie wskazanych danych z paska debugera dzięki usunięciu odpowiadającej im linii.

Poniżej opisano przeznaczenie klas zawartych w pliku konfiguracyjnym paska debugera.

- *Timers* — zbiera dane dotyczące czasu aplikacji potrzebnego na wygenerowanie całej strony, a także czasu wykonywania kodu pomiędzy założonymi przez nas wskaźnikami.
- *Database* — wyświetla zapytania do bazy danych oraz czas poświęcony na ich wykonanie.
- *Logs* — wyświetla wszystkie logi dotyczące systemu. Opcja ta bardzo obciąża pamięć i powinna zostać wyłączona.
- *Views* — pokazuje pliki widoków wraz z czasem, który został przeznaczony na ich wygenerowanie.
- *Cache* — wyświetla informacje na temat przechwytywania danych z plików zamiast (na przykład) z bazy danych.
- *Files* — wyświetla listę plików użytych podczas generowania strony.
- *Routes* — pokazuje informacje na temat obecnego przepisywania adresów wraz z wszystkimi zmiennymi oraz innymi zdefiniowanymi przez nas opcjami.
- *Events* — wypisuje wszystkie wywołania wykonywane podczas generowania strony.

Twórz aplikacje PHP z CodeIgniterem

- Poznaj użyteczne klasy frameworka
- Naucz się wykorzystywać je w praktyce
- Rozwiń swoje umiejętności programistyczne

CodeIgniter to niewielki, lecz potężny framework, który ułatwia tworzenie aplikacji zarówno początkującym, jak i zaawansowanym programistom PHP. Dzięki implementacji wzorca model-view-controller wspiera podział kodu na warstwy odpowiedzialne za operacje związane z zarządzaniem danymi, wyświetlanie strony oraz obsługę logiki biznesowej, co upraszcza nie tylko pisanie, lecz również utrzymywanie i rozwój wykorzystujących go aplikacji.

Szerokie możliwości, doskonała wydajność, znakomite wsparcie programisty, zgodność z najnowszymi wersjami PHP, łatwość użycia i logiczna struktura bibliotek – wszystko to sprawia, że CodeIgniter jest wybierany przez kolejne generacje deweloperów, którym pozwala twórczo skupić się na projekcie oraz uniknąć wysiłku związanego z implementacją powtarzalnego kodu odpowiedzialnego za typowe operacje.

Jeśli chcesz szybko zacząć przygodę z frameworkiem, sięgnij po tę książkę! Bezboleśnie wprowadzi Cię ona w świat programowania z wykorzystaniem popularnego frameworka, przedstawi zasady zastosowania wzorca MVC, zapozna ze sposobami obsługi baz danych oraz zaprezentuje klasy systemowe i pomocnicze. Nabytą wiedzę możesz ugruntować i sprawdzić w praktyce dzięki ćwiczeniom, w ramach których krok po kroku nauczysz się walidować dane wprowadzane przez użytkownika oraz dzielić je na strony z wykorzystaniem własnego sortowania.

- Instalacja, konfiguracja i struktura frameworka CodeIgniter
- Implementacja wzorca model-view-controller
- Podstawy tworzenia aplikacji WWW z wykorzystaniem CodeIgnitera
- Metody umożliwiające obsługę baz danych i tworzenie zapytań
- Zastosowanie przydatnych klas systemowych i pomocniczych
- Praktyczne ćwiczenia z programowania wraz z rozwiązaniami

CodeIgniter – szybko, wydajnie, prosto do celu!

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-7487-4

Cena: 59,00 zł