Rozdział 1. Pobieranie frameworka oraz jego konfiguracja

Framework należy pobrać zanim zaczniemy z nim pracować. Znajduje się on na swojej stronie internetowej: *http://www.yiiframework.com/*. Na otwartej stronie znajdują się łącza w nagłówku. Najeżdżamy kursorem na menu downloads, a z niego wybieramy opcję framework.

Przewijamy stronę do nagłówka: **Install from an Archive File**. Z pod nagłówka wybieramy łącze: **Yii 2 with basic application template.** Po jego kliknięciu rozpocznie się pobieranie pliku **Tar** skompresowanego w **Gzipie**. Plik ten bez problemu można otworzyć za pomocą darmowego kompresora **7-Zip**[1].

Dlaczego wybraliśmy przy pobieraniu opcję: **Yii 2 with basic application template** zamiast: **Yii 2 with advanced application template**. Będziemy wszystkie elementy budować od podstaw, tak abyś mógł poznać zasady działania oraz programowania w frameoworku. Chcę ci pokazać co te oprogramowanie potrafi i dlatego wszystkie elementy wykonamy własnoręcznie. Będziemy powtarzać wszystkie czynności i dzięki temu nie będziesz musiał się zastanawiać nad tym jakiego konstruktora użyć. Właściwie nie obyłoby się bez modyfikacji w plikach z modelami oraz kontrolerami. Z naszego punktu widzenia lepiej napisać to w własny sposób i nauczyć się więcej o działalności frameworka.

Budowa frameworka jest na pierwszy rzut bardzo skomplikowany, jednak po szybkim zapoznaniu się z nim łatwo się zorientować gdzie znajdują się rzeczy odpowiedzialne za kontrolę nad aplikacją, modelami z dostępem do bazy danych oraz pliki widoku. Autorzy framweorka postarali się, aby można go było umieścić wraz z treścią poza katalogiem widocznym w internecie, więc nawet jeżeli popełnisz błąd, na przykład nie usuwając ważnego pliku, to i tak nie będzie widoczny w sieci. Jest to świetne rozwiązanie. Dlatego nasza strona znajduje się w katalogu **web.**

Wgrywanie frameworka na serwer

Framework ze względu na swoją budowę można wgrać w różnoraki sposób na serwer WWW przeznaczony do obsługi naszej strony WWW. Istnieją dwie możliwości.

Pierwszą z nich jest wgranie wszystkich plików programu na serwer w miejscu katalogu, którego zawartość jest widoczna publicznie, najczęściej jest to: **public_html**. Należy wówczas odpowiednio skonfigurować wszystkie ścieżki dostępu w głównym pliku **index.php** umieszczony w fameworku w katalogu **web.** Otwieramy plik i edytujemy w trzech miejscach poprawiając ścieżki dostępu z wskazania folderu wyżej **../**, na folder obecny: **/**

```
<?php

// comment out the following two lines when deployed to production
defined('YII_DEBUG') or define('YII_DEBUG', true);
defined('YII_ENV') or define('YII_ENV', 'dev');
```

[1] http://www.7-zip.org/

```
defined('YII_ENV_TEST') or define('YII_ENV_TEST', true);

require(__DIR__ . '/vendor/autoload.php');
require(__DIR__ . '/vendor/yiisoft/yii2/Yii.php');

$config = require(__DIR__ . '/config/web.php');

(new yii\web\Application($config))->run();
```

Drugą możliwością, polecaną przez programistów framewoarka, jest wgranie całego oprogramowania do katalogu o jeden poziom wyżej niż **public_html**. Następnie należy przenieść pliki z katalogu **web**[2] do głównego folderu **public_html** gdzie znajduje się nasza strona.

Osobiście w tej książce będę się skupiał na drugiej możliwości. Jest ona bezpieczniejsza niż pierwsza ze względu na fakt iż użytkownik końcowy nie będzie miał wpływu na zawartość poszczególnych plików i w razie jakiegoś błędu w framewoarku nie będzie mógł go użyć w tak prosty sposób jak w przypadku kiedy pliki będą dostępne dla każdego kto odwiedza naszą stronę.

Ścieżki dostępu będą podawane w formie **../controller** i oznacza to iż należy wyjść o jeden poziom wyżej z katalogu publicznego gdzie znajduje się katalog **controller**.

Na koniec musimy pamiętać jeszcze o jednej bardzo ważnej rzeczy. Kiedy nasz projekt ujrzy światło dzienne i będziemy chcieli zaprezentować stronę szerszej publiczności należy w pliku **index.php** zakomentować dwie pierwsze linie oraz wyłączyć tryb debugowania.

```
//defined('YII_DEBUG') or define('YII_DEBUG', true);
//defined('YII_ENV') or define('YII_ENV', 'dev');
```

Sprawdzenie wymagań

Framework jak każde oprogramowanie ma także swoje wymagania dotyczące serwera i obsłuchiwanych przez serwer oprogramowania. Po wgraniu plików na serwer należy skopiować plik **requirements.php** z głównego katalogu frameworka do folderu dostępnego publicznie: **public_html**.

Następnie trzeba taki plik przeedytować zmieniając ścieżkę dostępu do plików dzięki którym program sprawdzi opcje serwera. Należy dodać w ścieżkę **frameworkPath** odwołanie do katalogu powyżej obecnego. Zrobimy to dodając do niej frazę **../**

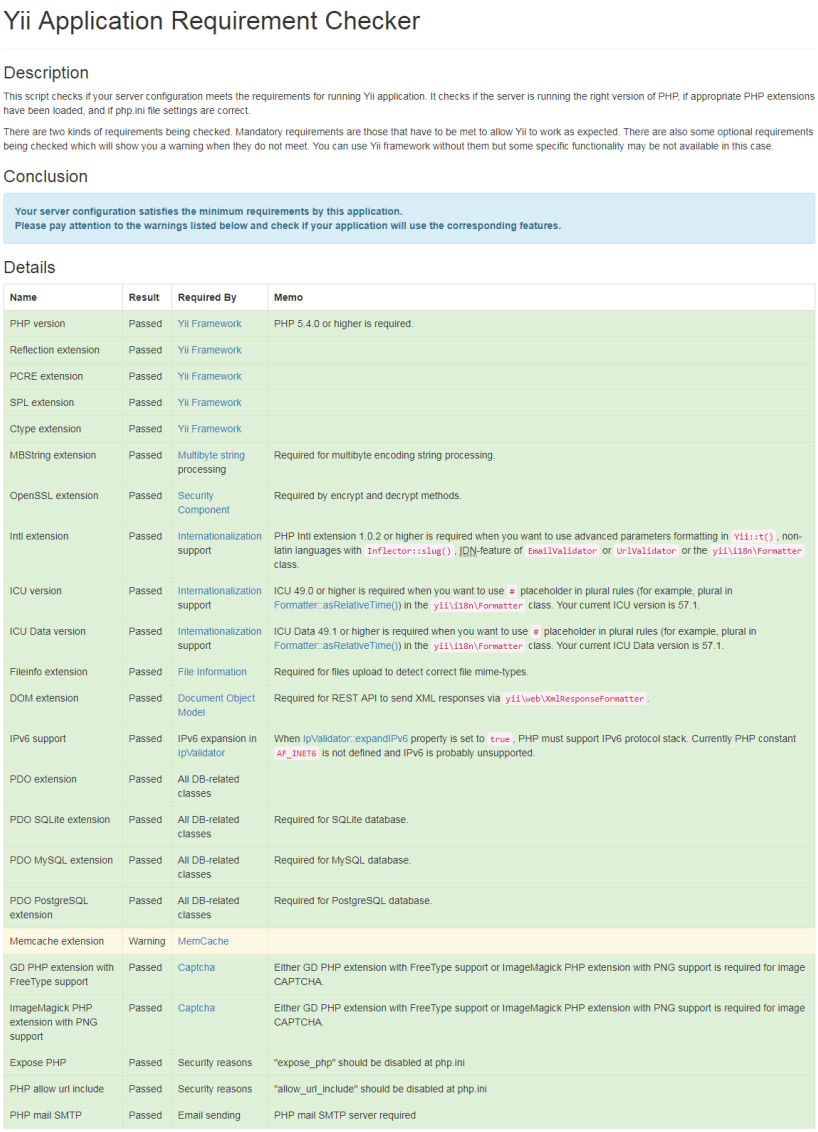
```
$frameworkPath = dirname(__FILE__) . '/../vendor/yiisoft/yii2';
```

Następnie w pasku adresu przeglądarki wpisujemy adres URL strony wraz z nazwą pliku dzięki któremu dowiemy się czy nasz serwer spełnia wymagania: **http://yii.phpbluedragon.eu/requirements.php**.

W przeglądarce zostanie uruchomiony skrypt, który wykryje oprogramowanie wchodzące w skład serwera i pozwala sprawdzić czy chociażby minimalne wymagania zostaną spełnione. Pojawi się wówczas informacja z szczegółami jakie wersje i programy znajdują się na serwerze. W przypadku braku możliwości uruchomienia frameworka na serwerze będzie zaprezentowane jakich funkcji brakuje. Można wówczas napisać do administratora naszego serwera aby włączył odpowiednie elementy.

---

[2] Najczęściej spotykany na serwerach jest folder public_html, jednak w niektórych przypadkach może o inaczej się nazywać



Yii Application Requirement Checker

Description

This script checks if your server configuration meets the requirements for running Yii application. It checks if the server is running the right version of PHP, if appropriate PHP extensions have been loaded, and if php.ini file settings are correct.

There are two kinds of requirements being checked. Mandatory requirements are those that have to be met to allow Yii to work as expected. There are also some optional requirements being checked which will show you a warning when they do not meet. You can use Yii framework without them but some specific functionality may be not available in this case.

Conclusion

Your server configuration satisfies the minimum requirements by this application.
Please pay attention to the warnings listed below and check if your application will use the corresponding features.

Details

Name	Result	Required By	Memo
PHP version	Passed	Yii Framework	PHP 5.4.0 or higher is required.
Reflection extension	Passed	Yii Framework	
PCRE extension	Passed	Yii Framework	
SPL extension	Passed	Yii Framework	
Ctype extension	Passed	Yii Framework	
MBString extension	Passed	Multibyte string processing	Required for multibyte encoding string processing.
OpenSSL extension	Passed	Security Component	Required by encrypt and decrypt methods.
Intl extension	Passed	Internationalization support	PHP Intl extension 1.0.2 or higher is required when you want to use advanced parameters formatting in Yii::t(), non-latin languages with Inflector::slug(), IDN-feature of EmailValidator or UrlValidator or the yii\i18n\Formatter class.
ICU version	Passed	Internationalization support	ICU 49.0 or higher is required when you want to use # placeholder in plural rules (for example, plural in Formatter::asRelativeTime()) in the yii\i18n\Formatter class. Your current ICU version is 57.1.
ICU Data version	Passed	Internationalization support	ICU Data 49.1 or higher is required when you want to use # placeholder in plural rules (for example, plural in Formatter::asRelativeTime()) in the yii\i18n\Formatter class. Your current ICU Data version is 57.1.
Fileinfo extension	Passed	File Information	Required for files upload to detect correct file mime-types.
DOM extension	Passed	Document Object Model	Required for REST API to send XML responses via yii\web\XmlResponseFormatter.
IPv6 support	Passed	IPv6 expansion in IpValidator	When IpValidator::expandIPv6 property is set to true, PHP must support IPv6 protocol stack. Currently PHP constant AF_INET6 is not defined and IPv6 is probably unsupported.
PDO extension	Passed	All DB-related classes	
PDO SQLite extension	Passed	All DB-related classes	Required for SQLite database.
PDO MySQL extension	Passed	All DB-related classes	Required for MySQL database.
PDO PostgreSQL extension	Passed	All DB-related classes	Required for PostgreSQL database.
Memcache extension	Warning	MemCache	
GD PHP extension with FreeType support	Passed	Captcha	Either GD PHP extension with FreeType support or ImageMagick PHP extension with PNG support is required for image CAPTCHA.
ImageMagick PHP extension with PNG support	Passed	Captcha	Either GD PHP extension with FreeType support or ImageMagick PHP extension with PNG support is required for image CAPTCHA.
Expose PHP	Passed	Security reasons	"expose_php" should be disabled at php.ini
PHP allow url include	Passed	Security reasons	"allow_url_include" should be disabled at php.ini
PHP mail SMTP	Passed	Email sending	PHP mail SMTP server required

Rysunek 1.1. Skutek uruchomienia skryptu sprawdzającego wymagania.

Używanie ładnych adresów URL

Pierwszym elementem bez którego nie ma sensu cokolwiek robić w frameworku będzie używanie przez system ładnych URLi. Oznacza to iż zamiast podawać pod znaku zapytania parametry do pliku **index.php** będziemy korzystali ze ścieżek w adresie. Ułatwi to zarówno pracę tobie jak i korzystanie ze strony użytkownikom końcowym. To jednak nie wszystko. Dzisiaj strony internetowe tworzy się zważając na zachowania botów wyszukiwarek na twojej witrynie. Najważniejsza wyszukiwarką jest **Google** i dodaje ona punkty za czytelne adresy. Dzięki temu sposobowi podniesiemy naszą stronę w liczbie punktów przyznawanych przez wyszukiwarkę. Niestety, ale nadeszły czasy w których witryn internetowych nie pisze się dla osób, ale dla robotów wyszukiwarek.

W katalogu **public_html** lub innym w którym znajduje się twoja strona internetowa należy zamieścić plik o nazwie **.htaccess**. W zależności czy strona znajduje się pod głównym adresem domeny, wówczas do pliku wpisujemy treść.

```
Options +FollowSymLinks
IndexIgnore */*
```

```
RewriteEngine on
RewriteBase /
# if a directory or a file exists, use it directly
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d

# otherwise forward it to index.php
RewriteRule . index.php
```

Dzięki plikowi wszystkie adresy wprowadzane przez użytkownika po stwierdzeniu braku katalogu o nazwie jaką wpisaliśmy, nasze żądanie zostanie przekazane do pliku **index.php** i tam odpowiednio rozbite i przekazane do realizacji kontrolerowi.

Następną rzeczą jest skonfigurowanie systemu. Przechodzimy do edycji pliku **../config/web.php**. Szukamy linii **urlManager**, a po jej znalezieniu usuwamy z niej komentarz.

Definiujemy ścieżkę do naszej strony w parametrze **baseURL**. Dodajemy wartość **true** w **enablePrettyUrl** co spowoduje, że ładnie wyglądające ścieżki zostaną włączone oraz ustawiamy **showScriptName** na wartość **false**, aby plik **index.php** nie był pokazywany w adresach URL.

```
        'urlManager' => [
            'baseUrl' => 'http://yii.phpbluedragon.eu/',
            'enablePrettyUrl' => true,
            'showScriptName' => false,
            'enableStrictParsing' => true,
```

Następnie w **rules** deklarujemy nasze ścieżki.

```
'rules' => [
```

W przypadku kiedy adres nie będzie zawierał żadnych parametrów czyli strona zostanie otwarta wpisując adres URL do przeglądarki lub poprzez wyszukiwarkę internetową trafimy na stronę główną, należy zdeklarować jej zawartość. Tworzymy ją poprzez wpisanie pustej wartości i przekierowania w takim wypadku do kontrolera **page** oraz metody **home**.

```
'' => 'page/home',
```

Użytkownicy muszą mieć możliwość logowania, rejestracji oraz wylogowania.

```
'login' => 'user/login',
'register' => 'user/register',
'logout' => 'user/logout',
```

Użytkownicy potrzebują także akcji przypominania hasła. Dodajemy przekierowania na kontroler **user** z akcją **rempass**, a w drugiej ustawiamy wartość zmiennej **UserId** na liczbę poprzez **\d+**, oznacza to iż może wystąpić więcej niż jedna cyfra. Następnie **UserHash1** i **UserHash2** które będą ciągami znaków składających się z liter i cyfr: **\w** oraz znaku myślnika: **\-**. Obie określenia wartości podajemy w nawiasie za którym wstawiamy znak **+**, który oznacza iż znaków może wystąpić dowolna ilość. Żądanie przekierujemy do kontrolera **user**, a w nim do metody **rempasset**. Zmienne, które będziemy ustawiać w metodzie musza być **IDENTYCZNE** pod względem nazwy. W innym przypadku otrzymamy błąd z informacją o tym iż strona nie została znaleziona.

```
'remind-password' => 'user/rempass',
'remind-password-set/<UserId:\d+>/<UserHash1:[\w\-]+>/<UserHash2:[\w\-]+>' =>
'user/rempassset',
```

Definiujemy dostęp do profilu oraz opcji zmiany hasła.

```
'profile' => 'user/profile',
'change-password' => 'user/changepassword',
```

Użytkownik po zarejestrowaniu swojego konta musi je aktywować odbierając na swoja skrzynkę e-mail wiadomość wraz ze specjalnym linkiem do aktywacji. Będzie on zawierał identyfikator użytkownika **UserId**, który może być jedynie liczbą oraz **UserKey** - specjalny klucz wygenerowany dla użytkownika, który może zawierać litery i cyfry oraz znak **-**.

```
'activate/<UserId:\d+>/<UserKey:[\w\-]+>' => 'user/activate',
```

Specjalna akcja dla użytkownika, który nie ma praw dostępu do wydzielonej części serwisu WWW.

```
'right' => 'user/right',
```

Dostęp do stron będzie odbywał sie poprzez wywołanie adresu URL wraz ze słowem **page**, następnie specjalnie spreparowanym adresem **PageURL** zawierającym słowa, liczby i myślnik. Następną ważnym elementem jest identyfikator strony w bazie danych **PageId** składający się jedynie z liczb. W przypadku braku jakiegokolwiek parametru użytkownik zostanie przekierowany do metody **index**.

```
'page/<PageUrl:[\w\-]+>/<PageId:\d+>' => 'page/showone',
'page' => 'page/index',
```

Artykuły podobnie jak w przypadku stron. Wywołanie wraz z parametrem **ArticleUrl** - specjalnie skonstruowanym tytułem artykuły oraz **ArticleId** - zawierający identyfikator artykułu składającym się jedynie z liczb będzie odwoływało się do odpowiedniej treści z bazy danych. W innym przypadku zostanie wyświetlona zawartość metody **index**.

```
'article/<ArticleUrl:[\w\-]+>/<ArticleId:\d+>' => 'article/showone',
'article' => 'article/index',
```

Wpis z bloga będzie zawierał w swoim adresie **BlogUrl** - specjalnie skonstruowany tytuł przyjazdy dla przeglądarek oraz **BlogId** składający się wyłącznie z liczb. Następnie w przypadku kliknięcia na kategorię także będzie przekazywany **CategoryUrl** - tytuł kategorii przyjazny przeglądarkom internetowym oraz jej identyfikator **CategoryId**. Na samym końcu jest wczytywania tagów gdzie będzie on się mógł składać z liter, liczb oraz znaku **-**.

```
'blog/<BlogUrl:[\w\-]+>/<BlogId:\d+>' => 'blog/showone',
'blog' => 'blog/index',
'category/<CategoryUrl:[\w\-]+>/<CategoryId:\d+>' => 'blog/category',
'hash/<Hash:[\w\-]+>' => 'blog/hash',
```

Kontakt z odwiedzającymi naszą stronę dokonamy poprzez ustawienie przechodzenia po wpisaniu po adresie słowa **contact**.

```
'contact' => 'contact/index',
```

Pobieranie programów będzie polegało na podaniu identyfikatora składającego się wyłącznie z liczb **ProgramId** oraz parametru **ProgramName** składającego się z liter, liczb oraz znaku **-**. W przypadku braku jakichkolwiek parametrów zostanie wywołana metoda **index** z kontrolera **download**.

```
'download/<ProgramId:\d+>/<ProgramName:[\w\-\.]+>' => 'download/getprogram',
'download' => 'download/index',
```

Panel administratora nie będzie posiadał przepisywania adresów URL więc dodajmy tylko fragment, który pomoże we wpisywaniu go w przeglądarce. Dodajemy frazę **admin**, która spowoduje uruchomienie kontrolera **configadmin** z akcją **index**.

```
'admin => 'configadmin/index',
```

Kończąc metody przepisywania adresów URL należy jeszcze dodać możliwość obsługi błędnych adresów lub takich których nie będziemy specjalnie ustawiać. Robimy to dodając standardowe polecenia.

```
'<controller:\w+>/<id:\d+>' => '<controller>/view',
'<controller:\w+>/<action:\w+>/<id:\w+>' => '<controller>/<action>',
'<controller:\w+>/<action:\w+>' => '<controller>/<action>',
],
],
```

Musimy pamiętać o bardzo ważnej rzeczy, którą jest wpisywanie w odpowiedniej kolejności adresów w pliku konfiguracyjnym.

Przedstawię to na przykładzie działu z pobieraniem plików. Po wpisani formatowania adresów w sposób jak na listingu poniżej zostanie zawsze zrealizowana metoda **index** z kontrolera **download**.

```
'download' => 'download/index',
'download/<ProgramId:\d+>/<ProgramName:[\w\-\.]+>' => 'download/getprogram',
```

Właśnie dlatego należy rozpoczynać od definicji użycia kontrolera z największą ilością parametrów. Dzięki temu program sprawdzi wszystkie pozostałe możliwości, zanim dojdzie do wyświetlenia metody **index**.

```
'download/<ProgramId:\d+>/<ProgramName:[\w\-\.]+>' => 'download/getprogram',
'download' => 'download/index',
```

Ustawianie klucza walidacji ciasteczek

Nasza aplikacja nie będzie działać bez podania klucza przeznaczonego do obsługi ciasteczek ustawianych przez framework Yii. Bez tego klucza niestety, ale nie uruchomimy naszej strony WWW. Należy w pliku

../config/web.php odnaleźć linię z wyrazem będącym kluczem tabeli **request**, a następnie do klucza **cookieValidationKey** wpisać dowolną losową wartość. Dzięki temu ciasteczka będą lepiej zabezpieczone przed próbą ich przejęcia.

```
'request' => [
            'cookieValidationKey' => 'sT+W=/8@qGL`kARLE.;Z(toGJ?zHb0',
        ],
```

Dodawanie komponentu

Tworzenie menu, formatowanie jego wyglądu oraz logi systemu nie musi być realizowane w każdym kontrolerze. Yii umożliwia stworzenie własnego komponentu w którym zawrzemy wszystkie elementy potrzebne do realizacji działań, które występują w więcej niż jednym miejscu. Taki komponent należy załadować i zrobimy to w pliku **../config/web.php**. Podajemy nazwę po której będziemy wywoływali funkcje zgodne z nazwą pliku, następnie wpisujemy słowo **class** i do niego ładujemy komponent używając jako ścieżki dostępu słowa **app**, które oznacza główny folder oprogramowania, następnie podajemy katalog **components** oraz plik bez rozszerzenia.

```
'components' => [
        'OtherFunctionsComponent' => [
            'class' => 'app\components\OtherFunctionsComponent',
        ],
```

Obsługi błędów aplikacji

Najczęściej kiedy występują błędy w adresie URL lub braku, niewłaściwych parametrów uruchamiany jest specjalny kontroler, którego zadaniem jest poinformowanie odbiorcy końcowego o wystąpieniu błędu. Otwieramy plik **../config/web.php** i w linii zawierającej klucz **errorHandler** wymazujemy wszystkie pary kluczy i wartości. Następnie wpisujemy aby wszelkie błędy były kierowane do kontrolera **user** oraz metody **error**.

```
'errorHandler' => [
            'errorAction' => 'user/error',
        ],
```

Konfiguracja serwera poczty

Podczas rejestracji użytkownika czy też kiedy będziemy chcieli aby użytkownicy skontaktowali się z serwisem poprzez wysłane listu e-mail będzie nam potrzebne konto poczty elektronicznej, dzięki której można uruchomić połączenie z serwerem. Niestety ale funkcja **mail()** coraz bardziej traci na znaczeniu i na niektórych serwerach zupełnie z niej zrezygnowano.

Otwieramy plik **../config/web.php** i odkomentujmy linię z kluczem **mailer**. Teraz ustawiamy opcje. Pierwszym elementem jest klasa potrzebna do wysyłania listów e-mail. Na szczęście wszystkie pliki znajdują sie już w systemie. Ustawiamy ją na **SwiftMailer**. Następnie opcja **useFileTransport** bardzo przydatna w przypadku chęci przetestowania naszej strony internetowej. Ustawienie w tym kluczu wartości **true** spowoduje iż nasze e-maile nie będą wysyłane, a zapisywane w plikach. Natomiast kiedy nasza witryna trafi na serwer w internecie ustawiamy w tym kluczu wartość **false**, spowoduje to iż od tej pory e-maile będą wysyłane za pomocą serwera pocztowego. Klucz **transport** zawiera szczegółowe dane do konfiguracji serwera. Użyjemy klasy **SMTP** dla poczty wychodzącej. Następnie w kluczu host deklarujemy adres URL serwera, klucz **username** zawiera nazwę użytkownika, klucz **password** hasło dostępu do serwera SMTP, **port** zawiera numer portu na którym jest

dostępna usługa połączenia z serwerem **SMTP** oraz na końcu **encryption** ustawiane na tryb szyfrowania naszego serwera. Po zakończeniu wpisywania ustawień zapisujemy plik.

```
'mailer' => [
            'class' => 'yii\swiftmailer\Mailer',
            'useFileTransport' => false,
            'transport' => [
                  'class' => 'Swift_SmtpTransport',
                  'host' => 'mail.phpbluedragon.eu',
                  'username' => 'yii@phpbluedragon.eu',
                  'password' => 'HASŁO',
                  'port' => '587',
                  'encryption' => 'tls',
              ],
        ],
```

Uwaga! Twoje dane będą inne niż w moim przypadku. Musisz sprawdzić adres serwera poczty, port, użytkownika oraz hasła. Najważniejsze jest jednak to czy obsługuje on połączenie szyfrowane. Jeżeli nie znasz odpowiedzi na którekolwiek pytanie, napisz do administratora serwera.

Włączenie debugera aplikacji

Framework dostarcza nam specjalne narzędzie przeznaczone do znajdowania oraz namierzania błędów w naszej witrynie WWW. Wyświetla on między innymi konfiguracje serwera i **PHP**, status obecnej strony, zmienne dostępne poprzez między innymi **GET**, **POST**, **COOKIE** i **SESSION**, logi aplikacji, ilość potrzebnego czasu oraz poszczególnych metod na generowanie strony, ilość zużytej pamięci **RAM**, zapytania do bazy danych, wysłane przez nas listy e-mail oraz wiele innych cennych danych. Nie raz przekonałem się jak przydatne jest korzystanie z danych pochodzących z narzędzia.

Możemy aktywować wyświetlanie paska w pliku **../config/web.php**. Na samym dole znajduje się konfiguracja w tablicy **$config** o indeksie **bootstrap**. Należy go odkomentować i najlepiej jest dodać własny adres IP. Bez podania adresu dane dotyczące szczegółów działania strony będą dostępne publicznie co będzie bardzo niebezpieczne. Dlatego należy dodać dodatkowy klucz pod nazwą **allowedIPs**. Będzie on zawierał tablice wraz ze zdefiniowanymi adresami **IP** dla których framework może wyświetlić pasek programisty.

```
    $config['bootstrap'][] = 'debug';
    $config['modules']['debug'] = [
        'class' => 'yii\debug\Module',
        'allowedIPs' => ['83.4.106.220'],
    ];
```

Ustawianie języka oprogramowania

System działa domyślnie w języku angielskim i w takim będą wyświetlane wszystkie komunikaty na przykład podczas walidowania formularza. Dlatego jeżeli chcemy ustawić dostępność strony w innym języku należy ustawić język. Otwórzmy plik **../config/web.php** i wpiszmy w jego tablice konfiguracyjną klucz **language** i ustawiamy jego wartość na **pl**. Ponieważ większość języków jest już domyślnie dostępnych w oprogramowaniu nie trzeba ściągać dodatkowych plików z tłumaczeniem.

```
'language' => 'pl',
```

Tworzenie dodatkowych danych konfiguracyjnych

Przy budowie strony będą nam jeszcze potrzebne dodatkowe dane do konfiguracji systemu. Stworzymy je w pliku **../config/params.php** w postaci tablicy, która jest włączana do głównej tablicy konfiguracyjnej. Podamy takie dane jak **adminEmail** - adres e-mail administratora, **adminName** - imię i nazwisko czy też nick administratora, **pageTitle** - będzie zawierało tytuł strony wytworzonej za pomocą framewoarka, **pageUrl** wraz z adresem internetowym strony oraz najważniejsza z opcji czyli **saltPassword** - jest to element wstawiany do każdego hasła przez jego zaszyfrowaniem. Pamiętaj że zmiana tego parametru będzie wymagała resetu hasła przez wszystkich użytkowników.

```
return [
    'adminEmail' => 'yii@phpbluedragon.eu',
    'adminName' => 'Łukasz Sosna',
    'pageTitle' => 'Strona oparta o framework Yii',
    'pageUrl' => 'http://yii.phpbluedragon.eu/',
    'saltPassword' => 'Z{+;r3hWP;',
];
```

Dodawanie połączenia z bazą danych

Dane prezentowane na stronie będą umieszczone w bazie danych tak aby mieć do nich dostęp przez zarówno panel administratora jak i bezpośrednio na przykład za pomocą narzędzia **phpMyAdmin**. Pierwszą rzeczą będzie podanie danych, które pomogą nam połączyć się z bazą danych. Otwieramy plik **../config/db.php** w którym znajduje się tablica do konfiguracji połączenia.

Pierwszym elementem jest **class**, który odpowiada za klasę dotyczącą połączenia z bazą danych. Drugim kluczem jest **DSN**, który zawiera rodzaj bazy danych w naszym przypadku będzie to **mysql**, następnie **host** to adres URL bazy oraz **dbname** czyli nazwę bazy danych. Kolejny klucz **username** służy zdefiniowaniu nazwy użytkownika mającego prawa do połączenia, klucz **password** do zdefiniowania hasła danego użytkownika. Następnie deklarujemy **charset** czyli zestaw znaków do porozumiewania się z bazą danych, a także **tablePrefix** czyli prefiks wszystkich tabel w bazie danych dotyczących naszego systemu. Dzięki dodawaniu prefiksów można w jednej bazie danych zainstalować teoretycznie niezliczoną ilość różnego rodzaju systemów. Na samym końcu zapisujemy plik.

```
return [
    'class' => 'yii\db\Connection',
    'dsn' => 'mysql:host=localhost;dbname=yii',
    'username' => 'yiiuser',
    'password' => 'yiipassword',
    'charset' => 'utf8',
    'tablePrefix' => 'yii_',
];
```

Usuwanie niepotrzebnych elementów

Zapewne odwiedziłeś stronę domową i wiesz że system zawiera kontrolery, modele oraz widoki do obsługi tejże strony. Najlepiej jest je usunąć, gdyż później możesz się zastanawiać skąd dany plik znalazł się w folderze. Dlatego kolejno usuwamy:

- ../controllers/SiteController.php
- ../models/ContactForm.php
- ../models/LoginForm.php
- ../models/User.php
- ../views/site - cały katalog

Z tak wstępnie przygotowanym frameworkiem możemy zacząć budować naszą stronę.