Oleg Krivtsov

USING ZEND FRAMEWORK 2

Learn how to create modern web applications with PHP and Zend Framework 2

- Model-View-Controller
- Using Twitter Bootstrap CSS Framework
- Managing database with Doctrine ORM
- Collecting user input with web forms
- and more...

Using Zend Framework 2

The book about Zend Framework 2 that is easy to read and understand for beginners

Oleg Krivtsov

This book is for sale at http://leanpub.com/using-zend-framework-2

This version was published on 2016-01-23

To my father who assembled my first computer and shown me how to write a simple program.
To my mother who shown me how to overcome life's obstacles and become a winner.

Contents

Preface

Why to Read this Book?

The "Using Zend Framework 2" is a book about programming web-sites with Zend Framework 2. With this e-Book, you can save your time and efforts learning ZF2.

The author strives to give material starting with simple things that a beginner should understand. Advanced things go last in a chapter. This makes this book the first book about Zend Framework that is easy to read and understand for a newbie.

Zend Framework Explained

The "Using Zend Framework 2" book is dedicated to web site development with PHP and Zend Framework 2 (ZF2). ZF2 is a modern PHP web development framework intended for building professionally looking, scalable and secure web-sites. Such web sites are easy to test and maintain. The framework utilizes the best practices and common design patterns, inspired by the evolution of web development industry. This includes *Model-View-Controller* pattern, allowing to organize the code in a consistent and standard way, making it easier to implement automatic code testing.

See ZF2 Wider

This e-book is not only about Zend Framework, but also about closely related libraries. Although Zend Framework 2 has dedicated component for accessing the database, in this book we use third-party library called Doctrine ORM — a de-facto standard object-oriented way to perform database management. In the sample applications we will create in chapters of this book, Twitter Bootstrap CSS Framework is used, allowing to produce nice looking visual appearance and layout of HTML elements on the web pages.

ZF2 Book for Beginners

This book is intended for web developers involved in the development of sites in PHP. The author strives to give material starting with simple things that a beginner should understand. Advanced things go last in a chapter. You do not need to be a guru in design patterns to understand most of the stuff.

To read and understand this book, you need to have a basic knowledge of PHP language. A good point for learning PHP is its official web site[1] and the online documentation[2]. It would be good

[1] http://php.net/

[2] http://php.net/docs.php

if you have some idea of what is HTTP request, GET and POST variables, namespaces, classes and interfaces.

Because PHP is closely related to other web technologies, it is also recommended that you have some basic experience in the following:

- HTML (Hyper Text Markup Language) – used for creating web pages that can be displayed in a web browser.
- CSS (Cascading Style Sheets) – used for defining the look and feel of a web page, like font size or background color.
- JavaScript – a client-side scripting language used for making a web page more interactive.

For learning HTML, CSS and JavaScript, a good starting point is W3Schools Tutorials[3].

Structure of the Book

This book is divided by chapters. A chapter is dedicated to a single topic. For example, Chapter 1 "Introduction to Zend Framework 2" is intended to make you familiar with fundamental concepts and main components of the framework; Chapter 2 "Zend Skeleton Application" is dedicated to giving you instructions to install the skeleton application, which can be used for creation of your own web sites, and so on.

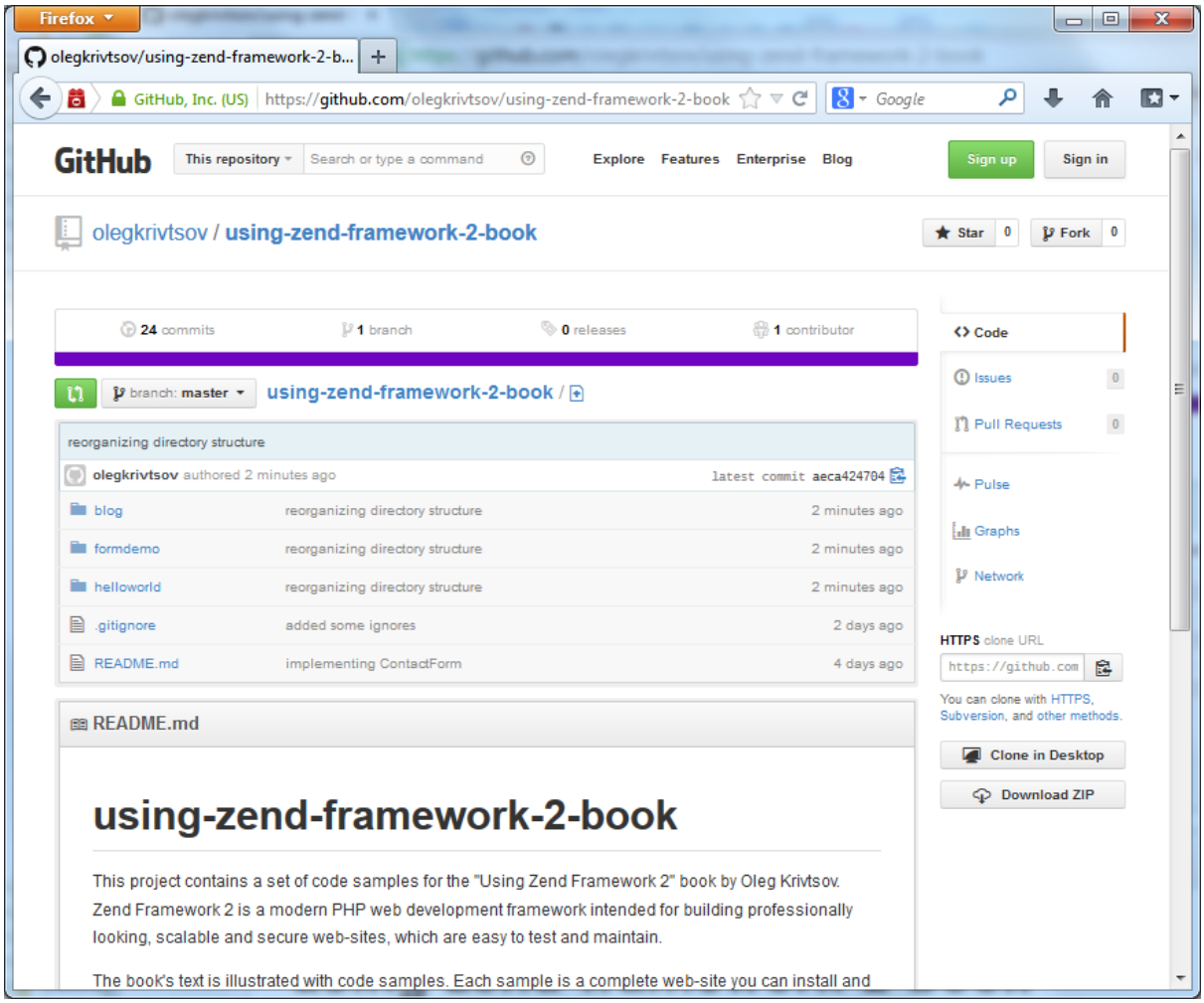
Learn ZF2 by Example

This ZF2 book's text is illustrated with code samples (the source code is published on GitHub). Each sample is a complete web-site you can install and run yourself to see Zend Framework 2 in action. You can even use the samples as a base for your own web sites.

All the source code is stored on GitHub code hosting. The code is publicly available, and you can download the entire code archive by visiting this page[4]. To download the archive, click the *Download ZIP* button that can be found on the page (see the figure below).

[3] http://www.w3schools.com/

[4] https://github.com/olegkrivtsov/using-zend-framework-2-book



Samples can be downloaded from GitHub

The structure of the code archive is presented below.

```
using-zend-framework-2-book
        blog
        helloworld
        formdemo
        ...
```

Book Reviews

Richard Holloway: "This will likely improve your overall understanding of modern PHP"

Richard Holloway is an organiser of PHP Hampshire[5], which is a recognised PHP user group:

"Many people struggle to get into Zend Framework 2 but this book does a good job of taking you over that initial steep learning curve and providing enough information to get you started on building websites."

[5] http://phphants.co.uk

The complete review is available by this link[6].

Testimonials

Below, there are some selected testimonials from satisfied readers of the book:

"I'm a very satisfied reader of your book (using zend framework 2": it details many important notions, but it never miss to give the big picture: great work!" ~Francesco

"I've recently bought your book "Using Zend Framework 2" and I think this is the best available resource to get started with ZF2." ~Janusz K

"I purchased your book on Zend framework 2 some days ago and I thought i should congratulate you for your amazing work. I tried another books and methods to learn zf2, but definitely your book is the only that works for me."

Zf2 is something complex to me and your book is making it easier. I really like the detailed explanations of the concepts and examples you use." ~Welington*

"I am one of (hopefully many) people who bought and read your 'using ZF2' book. [...] Your book taught me not only many new concepts, but also why these concepts came to be and (as a personal comfort to me) that almost half of these new features (or rather: ways of thinking) were things I was already doing, albeit in some other, non-object oriented way; I just never realised it. Having things explained by someone who obviously knows what he is talking about was a great help to me, and while I have yet to reach any important milestone, I feel I understand what I have to do much better now and I am much more confident that I will eventually successfully 'refresh' my hopelessly outdated projects." ~J.B.

Your Feedback

Thank you for reading this book and helping to make it better. You are encouraged to point out errors, make suggestions and critical remarks. You can write the author a message through the dedicated Forum[7]. Alternatively, you can contact the author through his E-mail address (`olegkrivtsov@gmail.com`). Your feedback is highly appreciated.

Acknowledgements

Thanks to Edu Torres, a 2D artist from Spain, for making the cover and illustrations for this book. Also thanks to Moriancumer Richard Uy and Charles Naylor for helping the author to find and fix the mistakes in the text.

The author would like to thank Richard Holloway (an organiser of PHP Hampshire[8], which is a recognised PHP user group in South England) for reviewing the book. Richard's review[9] is really useful for determining the proper development direction for this book.

[6] http://richardjh.org/blog/book-review-using-zend-framework-2/
[7] https://leanpub.com/using-zend-framework-2/feedback
[8] http://phphants.co.uk
[9] http://richardjh.org/blog/book-review-using-zend-framework-2/

1. Introduction to Zend Framework 2

In this chapter we'll learn about Zend Framework 2, its main principles and components. We'll also compare Zend Framework 2 with other PHP frameworks.

1.1 What is Zend Framework 2?

PHP is a popular web-site development language. However, it has been proven that writing web-sites in pure PHP is difficult. If you write a web application in PHP, you have to organize your code in some way, collect and validate user input, implement support of user access control, manage database, perform scheduled mail delivery, test your code and so on. As your site grows in size, it becomes more and more difficult to develop the code in such manner. Moreover, when you switch to the development of a new site, you will notice that a large portion of the code you have already written for the old site can be used again with small modifications. This code can be separated in a library. This is how frameworks appeared.

A framework is some kind of a library, a piece of software (also written in PHP) providing web developers with code base and consistent standardized ways of creating web applications. Imagine that your web-site is a house, then PHP language is its foundation, and the framework is the basement. The basement contains a lot of building blocks (components) and tools prepared for you to make it easier to build the upper floors of your house (see figure 1.1).

Zend Framework 2 is a free and open-source PHP framework. Its development is guided (and sponsored) by Zend, which is also known as the vendor of the PHP language itself. The first version (Zend Framework 1) was released in 2007 and since then it has become obsolete. Zend Framework 2 (or shortly ZF2) is the second version of this software, and it was released in September 2012. At the moment of writing this book, Zend Framework 2.3 is out.

Zend Framework 2 provides you with the following capabilities:

- Develop your web site much faster than when you write it in pure PHP. ZF2 provides many components that can be used as a code base for creating your site.
- Easier cooperation with other members of your site building team. Model-View-Controller pattern used by ZF2 allows to separate business logics and presentation layer of your web site, making its structure consistent and maintainable.
- Scale your web site with the concept of modules. ZF2 uses the term *module*, allowing to separate decoupled site parts, thus allowing to reuse models, views, controllers and assets of your web-site in other works.
- Accessing database in an object-oriented way. Instead of directly interacting with the database using SQL queries, you can use Doctrine Object-Relational Mapping (ORM) to manage the structure and relationships between your data. With Doctrine you map your database table to a PHP class (also called an *entity* class) and a row from that table is mapped to an instance of that class. Doctrine allows to abstract of database type and make code easier to understand.

- Write secure web sites with ZF2-provided components like form input filters and validators, HTML output escapers and cryptography algorithms, human check (Captcha) and Cross-Site Request Forgery (CSRF) form elements.

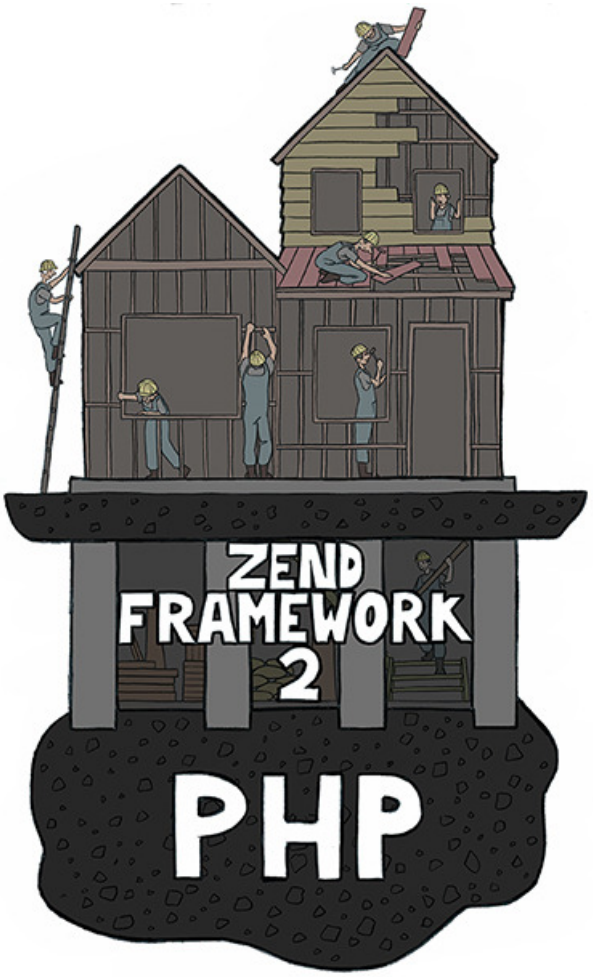


Figure 1.1. Zend Framework sits on top of PHP and contains reusable components for building your web site

1.2 License

Zend Framework 2 is licensed under BSD-like license, allowing you to use it in both commercial and free applications. You can even modify the library code and release it under another name. The only thing you cannot do is to remove the copyright notice from the code. If you use Zend Framework 2, it is also recommended that you mention about it in your site's documentation or on About page.

Below, the Zend Framework 2 license text is presented. As you can see, it is rather short.

1.3 What Companies Prefer Zend Framework 2?

Today, there are many companies who prefer Zend Framework 2 for building their powerful web sites. Some of them are listed on the main page of the framework.zend.com[1]. Among them:

- **BBC** The British Broadcasting Corporation (BBC) is a British public service broadcasting statutory corporation [2].

[1] http://framework.zend.com/

[2] BBC – From Wikipedia, the free encyclopedia http://en.wikipedia.org/wiki/BBC

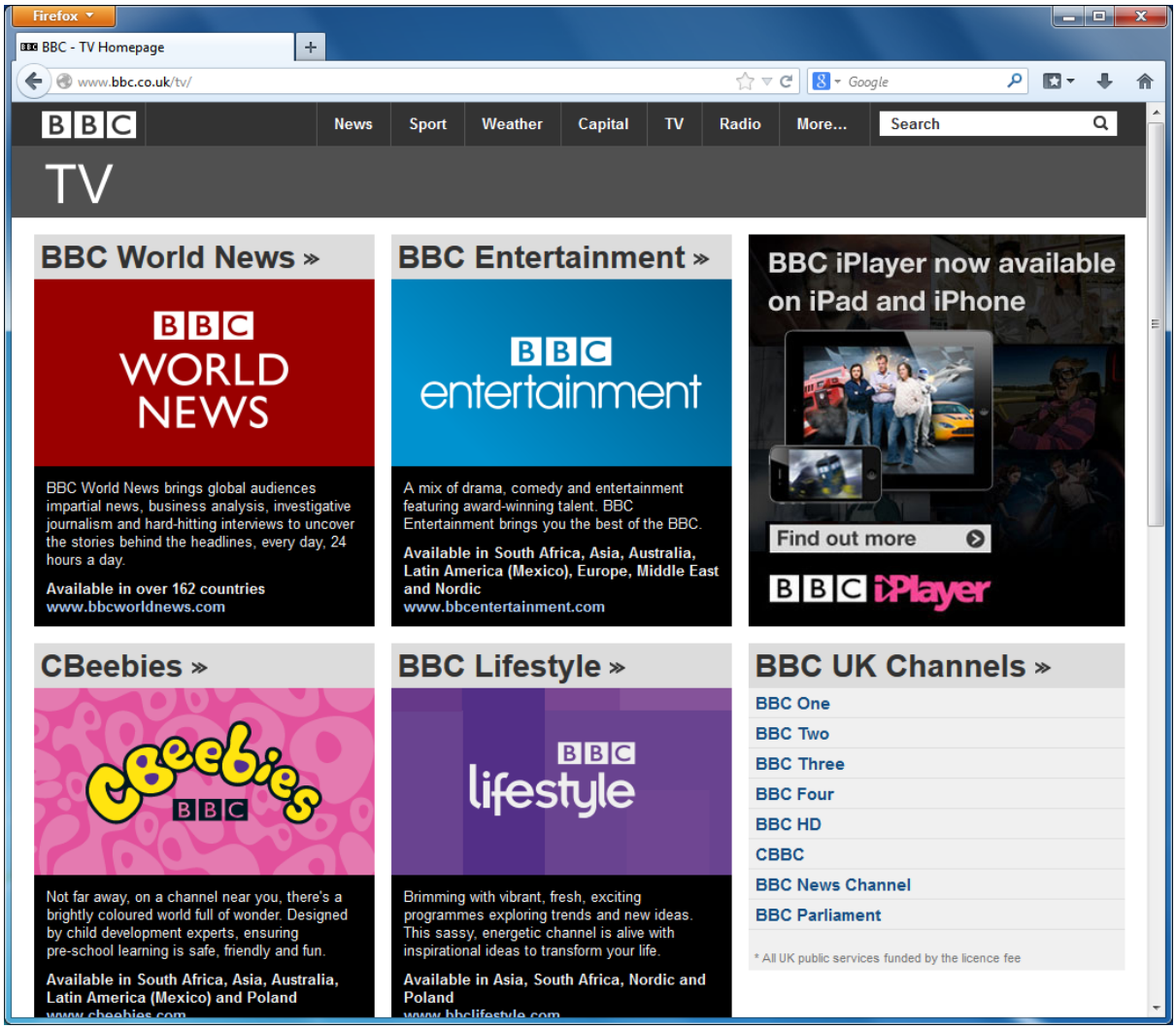


Figure 1.2. BBC web site is based on Zend Framework 2

- **BNP Paribas** Banque BNP Paribas is a French bank and financial services company, European leader in global banking and financial services and is one of the six strongest banks in the world according to the agency Standard & Poor's [3].

[3] BNP Paribas – From Wikipedia, the free encyclopedia http://en.wikipedia.org/wiki/BNP_Paribas

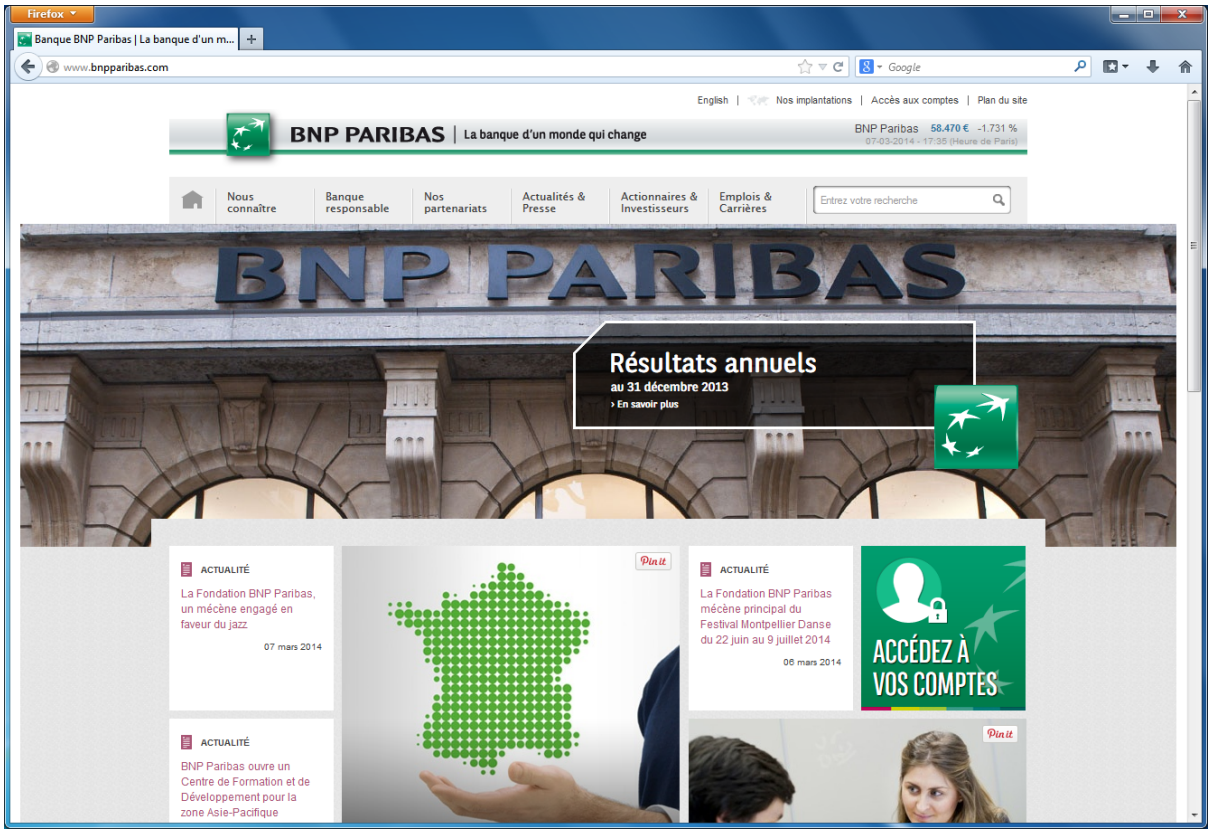


Figure 1.3. BNP Paribas web site is based on Zend Framework 2

1.4 Distributions

You can download the source code of Zend Framework 2 from the official site[4] (presented in figure 1.4) to become familiar with its structure and components.

ZF2 can be downloaded in two types: *full* and *minimum*. A full-size archive contains a complete set of components plus demos; its size is about 3 Mb. Mimimum-size distribution contains library components only, and its size is 3 Mb (also !).

In most cases you won't need to download the code of Zend Framework 2 manually. Instead, you will install it with Composer dependency manager. We will become familiar with Composer later in Chapter 2.

1.5 User Support

Support is an important thing to consider when deciding whether to use the framework as the base for your web site or not. Support includes well written documentation, webinars, community forums and (optionally) commercial support services, like trainings and certification programs.

[4] http://framework.zend.com/

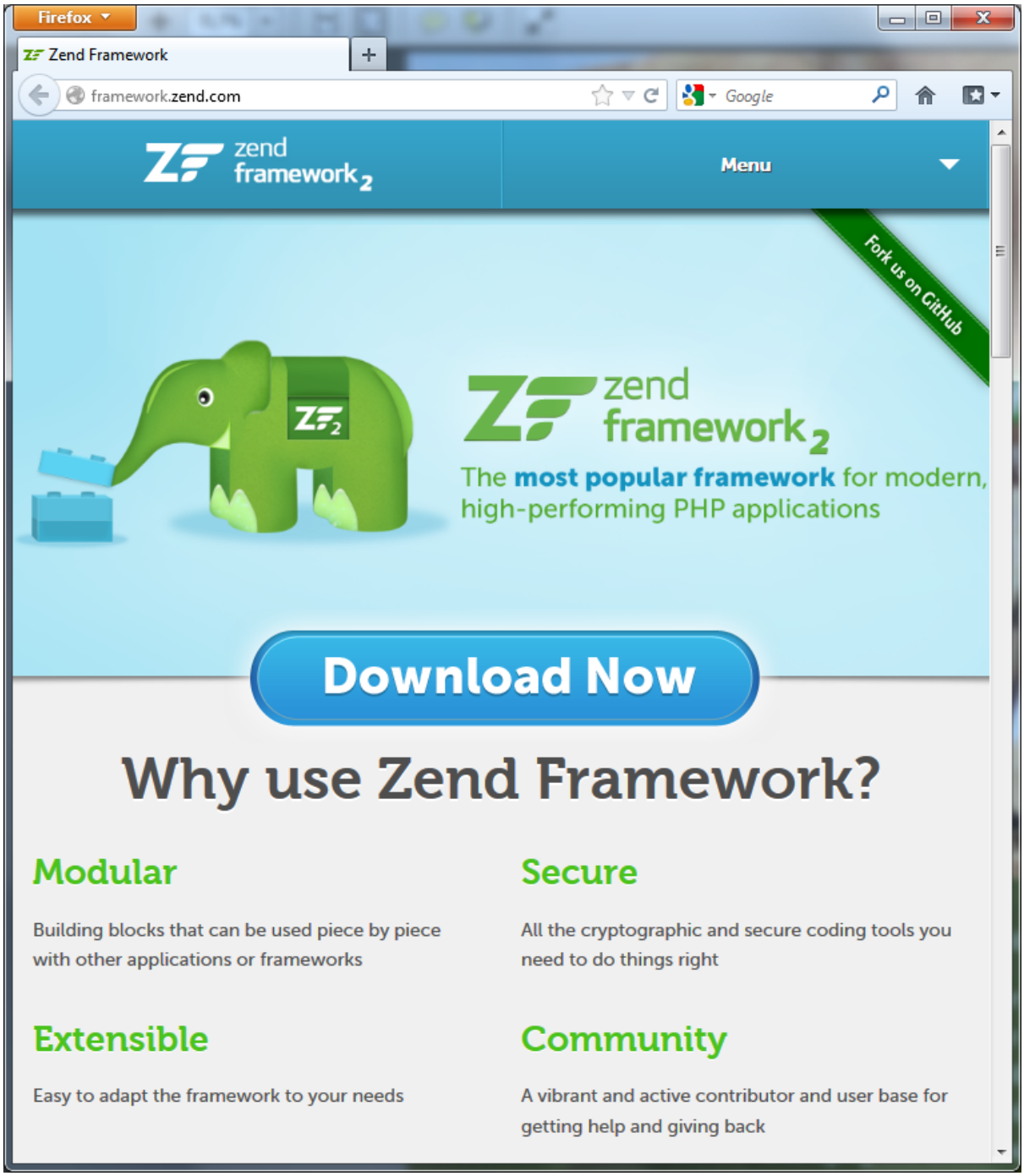


Figure 1.4. Zend Framework official project web site

Documentation. Documentation for ZF2 is located by this address[5]. It includes beginner's tutorial, programmers manual, and API reference (API stands for Application Programming Interface).

Community Forums. Zend Framework 2 has dedicated user groups all over the world. The list of groups can be found on this page[6].

Webinars are video tutorials covering various ZF2 features. Complete list of webinars can be

[5] http://framework.zend.com/learn/

[6] http://framework.zend.com/participate/user-groups

found by this link[7]. Among webinar topics, there are:

- *Zend Framework 2 Patterns.* Tells about what's new in ZF2 compared to the first version of the framework. It also shows how namespaces, class autoloading, and exceptions are used in ZF2.
- *Getting started with ZF2.* Teaches you the basics of developing ZF2-based applications, like creating controllers and views, manipulating services and listening to events.
- *The MVC architecture of ZF2.* Teaches the MVC (Model View Controller) architecture of Zend Framework 2.

Training Classes with live instructors can be accessed by this link[8]. Here you can learn ZF2 by doing exercises, mini-projects and developing real code.

Certification Program. Allows you to become a Zend Certified Engineer (ZCE), thus making it easier to improve your skills as an architect and to find a job in a competitive PHP job market.

1.6 Supported Operating Systems

As any PHP web-site, ZF2-based web application can work on a Windows server, on a Linux server and on any other operating systems PHP can run in. For instance, for creating samples for this book, the author used Ubuntu Linux operating system.

If you do not know yet what OS to use for your web development, it is recommended for you to use Linux, because most server software operates on Linux servers. You can refer to Appendix A for some instructions on configuring your development environment.

1.7 Server Requirements

Zend Framework 2 requires that your server has PHP version 5.3.3 (or later) installed. Note that this is a rather strict requirement. Not all cheap shared hostings and not all private servers have such a modern PHP version.

Moreover, the recommended way of installing ZF2 (and other components your app depends on) is using Composer[9]. This forces the need of shell access (SSH) to be able to execute Composer command-line tool. Some shared hostings provide FTP access only, so you won't be able to install a ZF2-based web app on such servers the usual way.

What do I do if I don't have shell access to server?

If your hosting allows you to upload files through FTP protocol, you can prepare all files on your local machine and then upload the files to the server as an archive.

[7] http://www.zend.com/en/resources/webinars/framework

[8] http://www.zend.com/en/services/training/course-catalog/zend-framework-2

[9] http://getcomposer.org/

ZF2 utilizes URL rewriting extension for redirecting web-users to entry script of your site (you have to enable Apache's `mod_rewrite` module.) You may also need to install some PHP extensions, like memory caching extension, to improve ZF2 performance. This can be a difficulty when using a shared hosting and requires that you have admin rights on your server.

So, if you are planning to use ZF2 on a shared web hosting, think twice. The best server to install ZF2 on is a server with the latest version of PHP and with shell access to be able to execute Composer, install PHP extensions and provide an ability to schedule console PHP scripts by `cron`.

If your company manages its own server infrastructure and can afford upgrading PHP version to the latest one, you can install ZF2 on your private server.

An acceptable alternative is installing a ZF2-based web application to a cloud-based hosting service, like Amazon Web Services[10]. Amazon provides Linux server instances as a part of EC2 service. EC2 is rather cheap, and it provides a free usage tier[11] letting you try it for free for one year.

1.8 Security

Zend Framework 2 follows the best practices to provide you with a secure code base for your web sites. ZF2 creators release security bug fixes on a regular basis. You can incorporate those fixes with a single command through Composer dependency manager.

ZF2 provides many tools allowing to make your web site secure:

- *Routing* allows to define strict rules on how an acceptable page URL should look like. If a site user enters an invalid URL in a web browser's navigation bar, he is automatically redirected to an error page.
- *Access control lists and Role-Based Access Control (RBAC)* allow to define flexible rules for granting or denying access to certain resources of your web site. For example, an anonymous user would have access to your index page only, authenticated users would have access to their profile page, and the administrator user would have access to site management panel.
- *Form validators and filters* ensure that no unwanted data is collected through web forms. Filters, for example, allow to trim strings or strip HTML tags. Validators are used to check that the data that had been submitted through a form conforms to certain rules. For example, E-mail validator checks that an E-mail field contains valid E-mail address, and if not, raises an error forcing the site user to correct the input error.
- *Captcha* and *CSRF* (Cross-Site Request Forgery) form elements are used for human checks and hacker attack prevention, respectively.
- *Escaper* component allows to strip unwanted HTML tags from data outputted to site pages.
- *Cryptography support* allows you to store your sensitive data (e.g. credentials) encrypted.

[10] http://aws.amazon.com/

[11] http://aws.amazon.com/free/

1.9 Performance

ZF2 creators have claimed to do a great job to improve performance of the ZF2 comparing to the first version of the framework.

Lazy class autoloading. Classes are loaded once needed. You don't have to write `require_once` for each class you want to load. Instead, the framework automatically discovers your classes using the *autoloader* feature. Autoloader uses either class map or class naming conventions to find and load the needed class.

Efficient plugin loading. In ZF2, plugin classes are instantiated only when they really need to. This is achieved through service manager (the central repository for services of your application).

Support of caching. PHP has several caching extensions (like APC or Memcache) that can be used to speed-up ZF2-based web sites. Caching saves frequently used data to memory to speed-up data retrieval. For example, a Zend Framework 2 application consists of many files which require time for PHP interpreter to process the files each time you open the page in the browser. You can use APC extension to cache precompiled PHP opcodes to speed up your site. Additionally, you can use the ZF2's event manager to listen to dispatching events, and cache HTML response data with Memcache extension.

Are there any benchmark tests of ZF2 performance?

As per the author's knowledge, currently, there are no reliable benchmark tests that would allow to compare ZF2 performance with performance of other frameworks.

1.10 Design Patterns

Zend Framework 2 creators are big fans of various design patterns. Although you don't have to understand patterns to read this book, this section is intended to give you an idea of what design patterns ZF2 is based on.

- *Model-View-Controller (MVC) pattern.* Model-View-Controller pattern is used in all modern PHP frameworks. In an MVC-application you separate your code into three categories: models (your business logics go here), views (your presentation goes here) and controllers (code responsible for interaction with user goes here). This is also called *the separation of concerns*. With MVC, you can *reuse* your components in a different project. It is also easy to substitute any part of this triad. For example, you can easily replace a view with another one, without changing your business logics.
- *Domain Driven Design (DDD) pattern* In Zend Framework 2, by convention, you will have model layer further splitted into *entities* (classes mapped on database tables), *repositories* (classes used to retrieve entities), *value objects* (model classes not having identity), *services* (classes responsible for business logics).
  Additionally, you will have *forms* (classes responsible for collecting user input), view helpers (reusable plugin classes intended for rendering stuff) and others.

- *Aspect Oriented Design pattern.* In ZF2, everything is event-driven. When a site user requests a page, an *event* is generated (triggered). A listener (or observer) can catch event and do something with it. For example, a router service parses the URL and determines what controller class to call. When the event finally reaches the page renderer, an HTTP response is generated and the user sees the web page.
- *Singleton pattern.* In ZF2, there is the service manager object which is the centralized registry of all services available in the application. Each service exists in a single instance only.
- *Strategy pattern.* While browsing ZF2 documentation or source code, you'll encounter the word *strategy* for sure. A strategy is just a class encapsulating some algorithm. And you can use different algorithms based on some condition. For example, the page renderer has several rendering strategies, making it possible to render web pages differently based on `Accept` HTTP header (the renderer can generate an HTML page, a JSON response, an RSS feed etc.)
- *Adapter pattern.* Adapters allow to adapt a generic class to concrete use case. For example, `Zend\Db` component provides access to database in a generic way. Internally, it uses adapters for each supported database (SQLite, MySQL, PostgreSQL and so on.)
- *Factory pattern.* You can create an instance of a class using the `new` operator. Or you can create it with a factory. A factory is just a class encapsulating creation of other objects. Factories are useful, because they simplify dependency injection - you can provide a generic factory interface instead of hard-coding the concrete class name. This simplifies the testing of your model and controller classes.

1.11 Components

ZF2 developers believe that the framework should be a set of decoupled components with minimum dependencies on each other. This is how ZF2 is organized.

The idea was to let you use some selected ZF2 components alone, even if you write your site with another framework. This becomes even easier, keeping in mind that each component of ZF2 is a Composer-installable package, so you can easily install any ZF2-component together with its dependencies through a single command.

The table 1.2 lists ZF2 components with their brief description. As you can see from the table, we can divide all ZF2 components into the following groups [12]:

- *Core Components.* These components are used (either explicitly or implicitly) in almost any web application. They provide the base functionality for class auto-loading, for triggering events and listening to them, for loading modules, for organizing the code within a module in conformance to the Model-View-Controller pattern, for creating console commands and more.
- *Web Forms.* Forms are the way of collecting user-entered data on web pages. A form usually consists of elements (input fields, labels, etc). For checking and filtering the user-entered data, filters and validators are utilized.

[12] These component groups are not an official classification, but the author's personal point of view.

- *User Management.* This important group includes components for providing user authentication, authorization and access control. Internally, these are based on the PHP feature called sessions.
- *Presentation Utilities.* In this group, we can put components implementing useful web page elements: navigation bars, progress bars, etc.
- *Persistence.* This group contains components whose purpose is to convert in-memory data into formats storable on a disk media (XML, JSON, a database, etc.), and vice-versa.
- *Testing and Debugging.* In this (small) group, there are several components for logging, testing and debugging your web site.
- *Web Services.* This group contains components that implement various protocols for accessing your web site programmatically (e.g. RSS, XML RPC, SOAP and so on).
- *Mail.* Useful components for composing E-mail messages and sending them with different transports.
- *Miscellaneous.* Various components that cannot be put in any other group.

Table 1.2. Zend Framework 2 Components

Component Name	*Description*
Core Components	
`Zend\Cache`	Provides a generic way to cache any data. Caching is used to save frequently used data to memory (or another storage media) to speed-up data retrieval.
`Zend\Code`	Provides facilities to generate arbitrary code using an object oriented interface. Also includes annotation parsing.
`Zend\Console`	Provides an ability to create applications runnable from shell command line. Console can be used, for example, for executing scheduled actions, like mail delivery.
`Zend\Di`	Dependency injection. Can be used to easily substitute and replace dependent classes.
`Zend\EventManager`	Allows to send events and register listeners to react to them. This component is covered in Chapter 3.
`Zend\Http`	Provides an easy interface for performing Hyper-Text Transfer Protocol (HTTP) requests. This component is covered in Chapter 4.
`Zend\Loader`	PHP class discovery and autoloading support. Autoloading is a more efficient replacement for `require_once`. This component is covered in Chapter 3.
`Zend\ModuleManager`	Zend Framework 2 module manager. In ZF2, every application consists of modules. This component is covered in Chapter 3.
`Zend\Mvc`	Support of Model-View-Controller pattern. Separation of business logic from presentation. This component is covered in Chapter 4.
`Zend\ServiceManager`	Service manager. This is the registry of all services available in the application, making it possible to access services from

Table 1.2. Zend Framework 2 Components

Component Name	*Description*
	any point of the web site. This component is covered in Chapter 3.
`Zend\Stdlib`	Miscellaneous utility classes: string utils, array utils, serializable queues, etc.
`Zend\View`	Provides a system of helpers, output filters, and variable escaping. Used in presentation layer. This component is covered in Chapter 4.
`Zend\Uri`	A component that aids in manipulating and validating Uniform Resource Identifiers (URIs).
Persistence	
`Zend\Dom`	Provides tools for working with DOM documents and structures. This includes querying DOM trees with CSS selectors and XPath.
`Zend\Db`	Provides database access in cross-database style.
`Zend\Json`	Provides convenience methods for serializing native PHP to JSON and decoding JSON to native PHP. Used for object serialization.
`Zend\Serializer`	Provides an adapter based interface to simply generate storable representation of PHP types by different facilities, and recover them.
User Management	
`Zend\Authentication`	Provides an API for user authentication. Users are typically authenticated by providing a username and password which are compared against a database table or Apache password file.
`Zend\Permissions`	Access control lists (ACLs) and role-based access control (RBAC).
`Zend\Session`	Manage and preserve session data, a logical complement of cookie data, across multiple page requests by the same client.
Presentation Utilities	
`Zend\Barcode`	Provides a generic way to generate barcodes. A barcode is a small bar containing stripes of various width and is optically readable by a machine. You may have seen barcodes when purchasing goods in a supermarket. This component is covered in Chapter 5.
`Zend\Captcha`	Human input check. Generates a random image ensuring that your site's user is not a robot. This component is covered in Chapter 10.
`Zend\Navigation`	Sitemaps, breadcrumbs and site navigation trees.
`Zend\Paginator`	Breaking large tabular data results into pages.
`Zend\ProgressBar`	Component to create and update progress bars in different environments.
`Zend\Escaper`	Smart class for escaping text output. Used to secure web site views.
`Zend\Tag`	A component suite which provides a facility to work with taggable items.
Testing and Debugging	
`Zend\Debug`	A small component containing a debugging utility class.

Table 1.2. Zend Framework 2 Components

Component Name	*Description*
`Zend\Log`	Component for general purpose logging. Logging site operations is used to troubleshoot possible errors with your site in development and production environments.
`Zend\Test`	Base classes for unit testing and test bootstrapping.
Web Forms	
`Zend\Filter`	Provides a set of commonly needed data filters, like string trimmer. This component is covered in Chapter 8.
`Zend\Form`	Web form data collection, filtering, validation and rendering. This component is covered in Chapter 7 and Chapter 10.
`Zend\InputFilter`	Provides an ability to define form data validation rules. This component is covered in Chapter 7.
`Zend\Validator`	Provides a set of commonly needed validators. This component is covered in Chapter 9.
Web Services	
`Zend\Feed`	Provides functionality for consuming RSS and Atom feeds.
`Zend\Ldap`	Provides support for Lightweight Directory Access Protocol (LDAP) operations including but not limited to binding, searching, and modifying entries in an LDAP directory.
`Zend\Server`	Client-server generic class interfaces.
`Zend\Soap`	Implementation of Simple Object Transfer Protocol (SOAP).
`Zend\XmlRpc`	Used for creation of web-services utilizing XML Remote Procedure Call (RPC) protocol.
Mail	
`Zend\Mail`	Provides generalized functionality to compose and send both text and MIME-compliant multi-part E-mail messages. This component is covered in Chapter 7.
`Zend\Mime`	Support class for Multipurpose Internet Mail Extensions (MIME) messages.
Miscellaneous	
`Zend\Config`	Provides a nested object property based user interface for accessing the configuration data within application code.
`Zend\Crypt`	Contains implementation of symmetric and asymmetric cryptographic algorithms.
`Zend\File`	PHP class file discovery.
`Zend\I18n`	Support of multi-lingual web sites.
`Zend\Math`	Big integer support and some auxiliary math functionality.
`Zend\Memory`	This class encapsulates memory management operations, when PHP

Table 1.2. Zend Framework 2 Components

Component Name	*Description*
	works in limited memory mode.
`Zend\Text`	Various text utilities like character tables and FIGlets.
`Zend\Version`	Allows to retrieve the version of Zend Framework. This component is covered in Chapter 4.

1.12 ZF2 Service Components

In addition to standard Zend Framework 2 components described in the previous section, there are so called *Services for Zend Framework 2* components. Those components provide implementations of API for accessing various popular web resources (e.g. Flickr, Twitter, SlideShare, reCaptcha and so on) programmatically. Table 1.3 contains the list of (currently available) service components together with their brief descriptions:

Component Name	*Description*
`ZendService\Akismet`	Provides API for accessing Akismet[13] (a spam filtering service for your blog).
`ZendService\Amazon`	Provides API for using Amazon[14] web services. Amazon provides a number of web services, among them EC2 (web hosting in the cloud), S3 (storage in the cloud) and others.
`ZendService\AppleApns`	Provides a client for the Apple Push Notification Service (APNs for short), which is a service for propagating information to iOS and Mac OS X devices.
`ZendService\Audioscrobbler`	API for using the Audioscrobbler[15] service, which is a database that tracks listening habits.
`ZendService\Delicious`	API for using del.icio.us[16] services. Provides access to posts at *del.icio.us* and read-only access to public data of all users.
`ZendService\DeveloperGarden`	Provides API for accessing services of Deutsche Telekom, such as voice connections or sending SMS messages.
`ZendService\Flickr`	API for using the Flickr[17] photo sharing service.
`ZendService\Google\Gcm`	Provides a client for the Google Cloud Messaging API.
`ZendService\LiveDocx`	Provides API to LiveDocx service that allows to generate PDF, DOCX, DOC, HTML or RTF files.
`ZendService\Nirvanix`	API for using Nirvanix service which provides an Internet Media File System (IMFS), a storage service that allows applications to upload, store and access files.

[13] http://akismet.com/
[14] http://aws.amazon.com/
[15] http://www.audioscrobbler.net/
[16] https://delicious.com/
[17] http://www.flickr.com/

Component Name	*Description*
`ZendService\Rackspace`	API to manage the Rackspace services Cloud Files and Cloud Servers.
`ZendService\ReCaptcha`	Provides API for the reCAPTCHA[18] service, used to digitize books and (as a side product) generate CAPTCHA images.
`ZendService\SlideShare`	Access to the SlideShare[19] services for hosting slide shows online.
`ZendService\StrikeIron`	API for accessing the StrikeIron[20] web services – Cloud-Based Data Quality & Enhancement Solutions.
`ZendService\Technorati`	Provides interface for using Technorati[21], which is a place storing individual reviews, essays, interviews, and news stories.
`ZendService\Twitter`	Provides API to Twitter[22] microblogging service.
`ZendService\Windows Azure`	Provides API for accessing Microsoft Windows Asure[23] cloud web hosting platform.

Because the API to above mentioned web resources may be changed by their vendors without a notice, those components are not part of the "core" Zend Framework 2 distribution. By the same reason, those components are not discussed deeply in this book.

1.13 Differences with Zend Framework 1

For readers who have an experience in Zend Framework 1, in this section we'll give some information on what has changed in Zend Framework 2.

Below, the main technical differences between ZF1 and ZF2 are presented:

1.13.1 Backwards Compatibility

ZF1's architecture passed an evolutionary path, preserving backwards compatibility and accumulating many solutions which were not as efficient as they could be. ZF2 has been rewritten from scratch to implement the best features of ZF1 in a better, more efficient and scalable way. Because of these breaking changes, ZF2 is not backwards-compatible with ZF1.

1.13.2 ZFTool

In Zend Framework 1, you used ZFTool for creating the application, adding layouts and controllers. In ZF2, you create your new applications by downloading *Zend Skeleton Application* available on GitHub. By the way, in ZF2 you can install a component called ZFTool, and it can also create the skeleton application or a module for you.

[18] http://www.google.com/recaptcha
[19] http://www.slideshare.net/
[20] http://www.strikeiron.com/
[21] http://technorati.com/
[22] http://twitter.com
[23] http://www.windowsazure.com/

1.13.3 Modules

In Zend Framework 1, your application was monolithic (although there was a concept of module). In ZF2, everything is a module. The skeleton application has single `Application` module by default. Each module may contain configuration, models, views, controllers and the assets (e.g. database tables, files etc.) A module can call classes from other modules. You can install third-party modules and reuse your own modules across applications.

1.13.4 Aspect Oriented Design

In ZF2, *events* are used to make it possible to decouple modules. You can install a module, and it will just work by listening to events occurring in the application without knowing about other modules. Events include bootstrapping, routing, dispatching and rendering.

1.13.5 Namespaces

In ZF1, you worked with long underscore-separated class names like `Zend_Controller_Action`. In ZF2, PHP namespaces are used, so instead you'll have something like `Zend\MVC\Controller\AbstractActionC` which can be easier to type with auto-completion feature and easier to understand. Namespaces allow to define short class names (aliases) and use them instead of full names. By convention, namespaces are mapped to directory structure, making it easier to perform class autoloading.

1.13.6 Configuration

In ZF1, you had an application-level INI config file. In ZF2, each module has its configuration file in a form of PHP array. At application level, module configurations are finally merged into a single large nested PHP array.

1.13.7 Service Manager

In ZF1, you had an application registry of classes, which allowed you to access application services and even put your own class to the registry and use it later. In ZF2, we have the *service manager*, which is a more complex version of the registry, implementing lazy loading and dependency injection. With service manager, you can register a *service* class and use it across your modules. For example, Doctrine ORM library registers the Entity Manager service which you can use to access the database across the module controllers.

1.14 Competing Frameworks

Zend Framework is not the only web development framework. There are others, like Symfony[24], Cake PHP[25], CodeIgniter[26] and Yii Framework[27]. To estimate the average popularity of these

[24] http://symfony.com/
[25] http://cakephp.org/
[26] http://ellislab.com/codeigniter
[27] http://www.yiiframework.com/

frameworks in some way, we can use Google Trends[28] site, which allows to track count of a keyword search queries over time. For example, if you enter "Zend Framework, CakePHP, Yii, CodeIgniter, Symfony" into the query field, you will get the graph as shown in figure 1.5.

As you can see from the graph, Zend Framework (blue line) has reached its popularity peak by 2010, and since then it has slowly lost its popularity. However, ZF is still one of the strong players on the market. On the other hand, Cake PHP, Symfony, CodeIgniter and Yii framework are becoming highly popular nowadays.

Let's also look at the relative popularity of ZF1 and ZF2 by typing "Zend Framework, Zend Framework 2" into the search query field. The result is shown in figure 1.6.

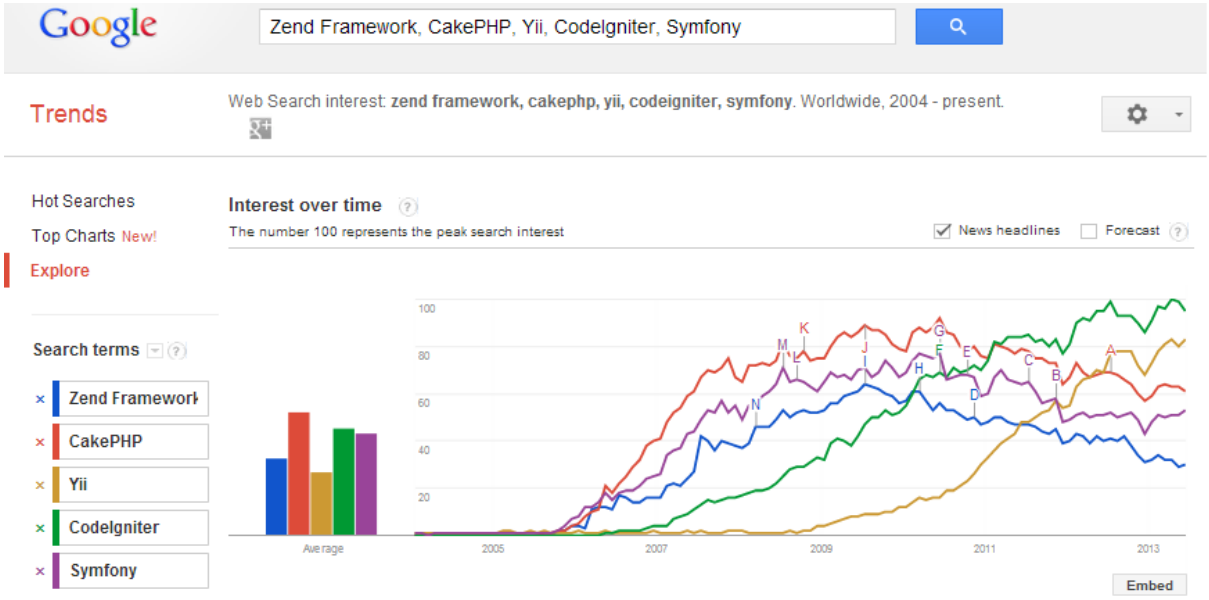


Figure 1.5. Popularity of PHP frameworks. Powered by Google Trends

As we can see, Zend Framework 2 (the red line) was released not so long ago, and has yet to become popular. The author believes that ZF2 has all the necessary qualities to become popular over time.

[28] http://www.google.ru/trends/

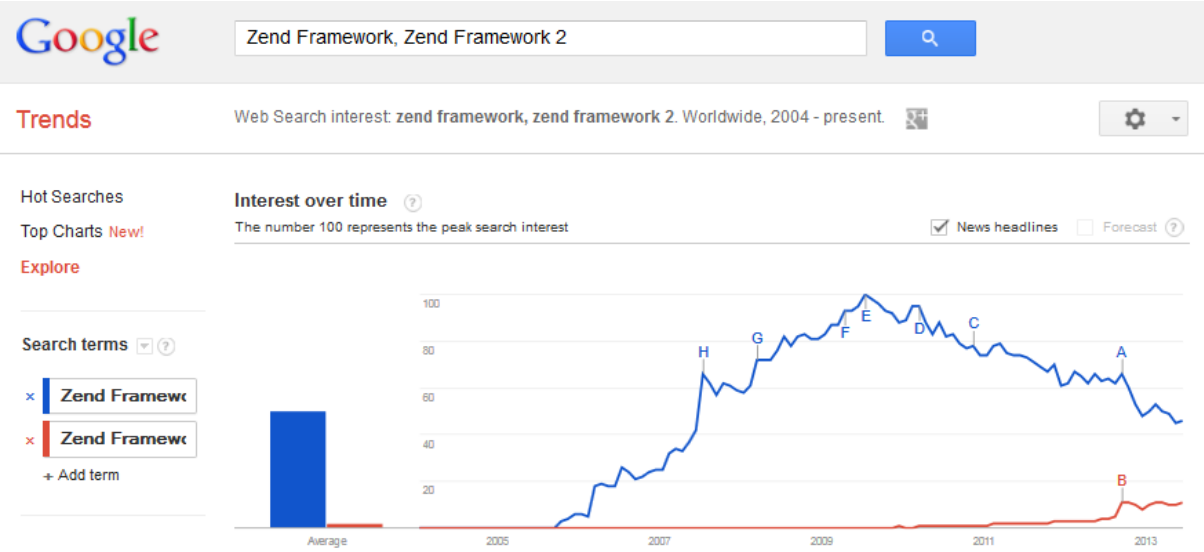


Figure 1.6. Popularity of Zend Framework 2 comparing to the first version. Powered by Google Trends

If you are familiar with one of the above mentioned frameworks, in table 1.4 you can find the comparison of features provided by those PHP frameworks. Capabilities of Zend Framework 2 are marked with **bold**.

Table 1.4. Comparison of Features provided by PHP frameworks

Feature	*ZF2*	*Symfony 2*	*Cake PHP*	*CodeIgniter*	*Yii*
Current version	**2.2.1**	2.3.1	2.3.6	2.1.3	1.1.13
Distribution archive size	**3 Mb**	4.4 Mb	2.0 Mb	2.2 Mb	3.9 Mb
Installation	**Composer**	Composer	Archive	Archive	Archive
Compatibility with shared hostings	**Bad (requires SSH and vhosts)**	Bad (requires SSH and vhosts)	Good	Good	Good
Monolithic or component-based?	**Components**	Components	Components	Components	Monolithic
Prefer conventions or configs?	**Configuration**	Configuration	Conventions	Conventions	Conventions
Database access pattern	**Data Mapper (Doctrine/ORM), Table Gateway, Row Gateway**	Data Mapper (Doctrine/ORM)	Active Record	Traditional or Active Record	Active Record, PDO

Table 1.4. Comparison of Features provided by PHP frameworks

Feature	*ZF2*	*Symfony 2*	*Cake PHP*	*CodeIgniter*	*Yii*
Database migrations	**Yes (Doctrine-provided)**	Yes (Doctrine-provided)	Yes	Yes	Yes
CSS Framework	**Twitter Bootstrap**	Twitter Bootstrap	Any	Any	Blueprint CSS
View Template Language	**Any you want, or none at all**	Twig (recommended)	Smarty/Twig	Any you want, or none	None or Prado
Unit testing support	**Yes (PHPUnit)**	Yes (PHPUnit)	Yes (PHPUnit)	Yes (PHPUnit)	Yes (PHPUnit-based)
Functional testing	**Yes (PHPUnit)**	Yes (PHPUnit)	Yes	No	Yes (Selenium)
Database fixtures	**Yes (Doctrine-provided)**	Yes (Doctrine bundle)	Yes	No	Yes

Summarizing the table above, we can say that:

- Zend Framework 2 can be considered as one of the most mature and established PHP frameworks on the market. This allows to be sure that ZF2 creators won't stop to update and support it unexpectedly.
- The major way for installing ZF2 is through Composer dependency manager. Symfony 2 is similar to ZF2 in this sence. Other PHP frameworks utilize the conventional installation from an archive.
- ZF2 (as Symfony 2) has bad compatibility with shared hostings because of the Composer-based installation method and strict PHP version requirements. So, if you need to install your website to a shared hosting, you probably need to contact the hosting's support for installation instructions.
- ZF2 provides the sophisticated configuration methods, so you can fine-tune and override any aspect of its work. Some other PHP frameworks prefer the "conventions over configuration" way, making it easier for newbies to start developing websites.
- For the presentation layer, ZF2 suggests the use of Twitter Bootstrap CSS Framework by default. But this does not limit you on using any other CSS frameworks.
- In ZF2, you can use several database access methods. And like in most PHP frameworks, you can benefit from using an object-oriented way of managing the database (with Doctrine ORM). Additionally, you can use Doctrine migrations mechanism for managing the database schema in a standardized way.
- Comparing to other frameworks, ZF2 provides good capabilities for unit- and functional testing (based on PHPUnit framework). This makes it possible to automate the testing of

the code you write. For testing the database functionality, you can use Doctrine-provided fixture mechanism.

1.15 Summary

A PHP framework is a library, giving you the code base and defining consistent ways of creating web applications. Zend Framework 2 is a modern web development framework created by the Zend Company, the vendor of PHP language. It provides the developers with outstanding capabilities for building scalable and secure web sites. ZF2 is licensed under BSD-like license and can be used for free in both commercial and open-source applications.

ZF2 is updated frequently, making your sites more resistant to vulnerabilities and security holes.

On its official site, ZF2 provides the documentation (tutorials and API reference), webinars, community forums and commercial support services, like trainings and certification program.

ZF2 creators strive to improve the performance of ZF2 in comparison to the first version of the framework. Among the features that contribute into the performance of ZF2, there are lazy class autoloading and support of caching.

On the market, Zend Framework is not the only web development framework. ZF2 is positioned as a good framework for corporate applications because of its pattern-based design and scalability. However, you can use ZF2 in any-sized web application with great success.

I've found a mistake in this chapter/have a suggestion. What do I do?

Please contact the author using the dedicated Forum[29]. Alternatively, you can contact the author through his E-mail address (`olegkrivtsov@gmail.com`). The author appreciates your feedback and will be happy to answer you and improve this book.

[29] https://leanpub.com/using-zend-framework-2/feedback

2. Zend Skeleton Application

Zend Framework 2 provides you with the so called "skeleton application" to make it easier to create your new applications from scratch. In this chapter, we will download and install the skeleton application which can be used as a base for creating your web sites. It is recommended that you refer to Appendix A before reading this chapter to get your development environment configured.

2.1 Getting Zend Skeleton Application

The Skeleton Application is a simple ZF2-based application that contains most necessary things for creating your own simple web site. The skeleton application's code is stored on GitHub code hosting and can be publicly accessed by this link[1]. To download the source code of the skeleton application as a ZIP archive, click the *Download ZIP* button (see the figure 2.1 below).

To download the code archive on a Linux machine without graphical interface, you can use the `wget` command, like this:

```
wget https://github.com/zendframework/ZendSkeletonApplication/archive/master.zip
```

Unpack the downloaded ZIP archive to some directory. If you are programming in Linux, it is recommended to unpack it in your home directory:

```
cp /path/to/downloaded/archive/ ZendSkeletonApplication-master.zip ~
cd ~
unzip ZendSkeletonApplication-master.zip
```

The commands above will copy the file `ZendSkeletonApplication-master.zip` archive that you've downloaded to your home directory, then unpack the archive.

If you are using Windows, you can place the skeleton app directory anywhere in your system, but ensure that file and directory access rights are sufficient for your web server to read and write the directory and its files. Actually, if you don't put your files to `C:\Program Files`, everything should be OK.

[1] https://github.com/zendframework/ZendSkeletonApplication

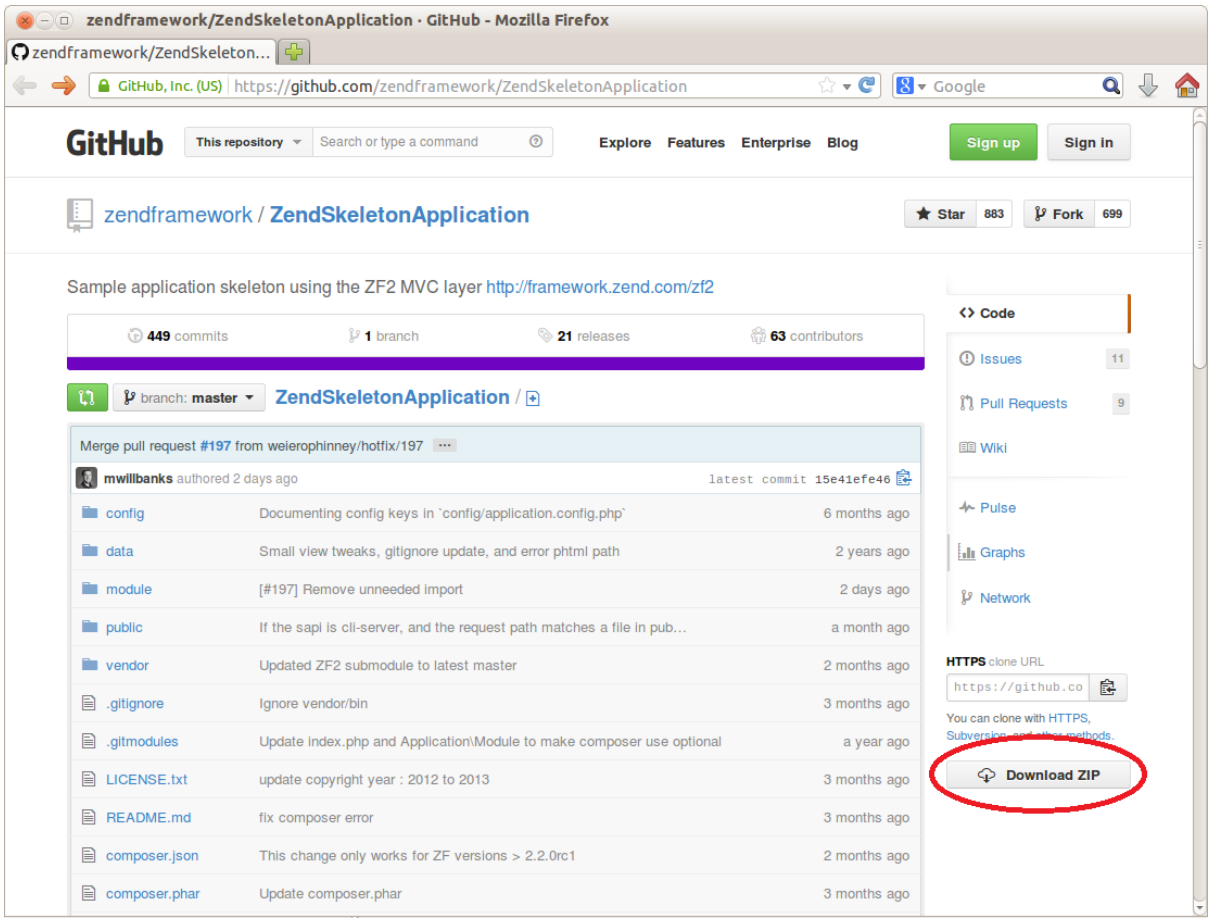


Figure 2.1. Zend Skeleton Application's code is stored on GitHub

2.2 Typical Directory Structure

Every ZF2-based web-site (including the skeleton app) is organized in the same recommended way. Of course, you can configure your application to use a different directory layout, but this may make it difficult to support your web-site by other people who are not familiar with such a directory structure.

Let's have a brief look at the typical directory structure (see figure 2.2):

As you can see, in the top-level directory (we will denote it `APP_DIR` from now on), there are several files:

- `README.md` is a text file containing a brief description of the skeleton application;
- `LICENSE.txt` is a text file containing ZF2 license (you had a chance to read it in Chapter 1 of this book);
- `composer.phar` is an executable PHP archive containing the code of Composer dependency management tool; we will use it later;
- `composer.json` is a JSON configuration file for Composer.

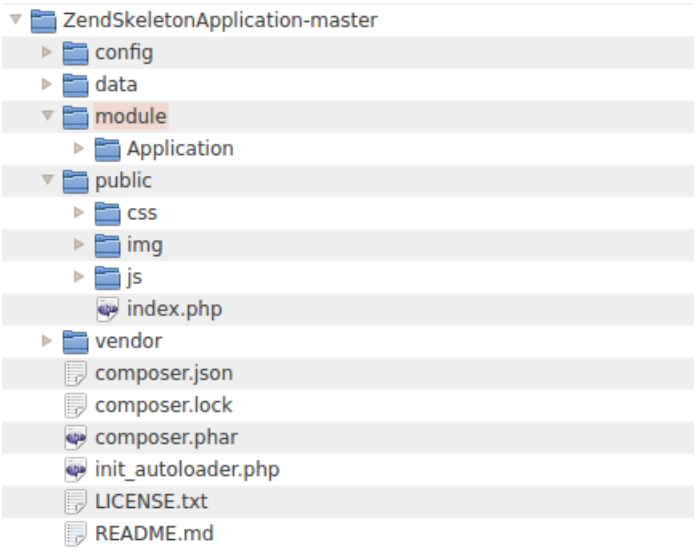


Figure 2.2. Typical Directory Structure

And we also have several subdirectories:

The `config` directory contains application-level configuration files.

The `data` directory contains the data your application might create; it may also contain cache used to speed-up Zend Framework.

The `module` directory contains all application modules. Currently there is a single module called `Application`. The `Application` is the main module of your web-site. You can also put other modules here, if you wish. We will talk about the modules a little bit later.

The `vendor` directory's purpose is to contain third-party library files, including Zend Framework 2 library files. Currently this directory is almost empty, but we will install all required libraries later.

The `public` directory contains data publicly accessible by the web-user. As you can see, web-users will mainly communicate with the `index.php`, which is also called the *entry point* of your web site.

Your web site will have a single entry point, *index.php*, because this is more secure than allowing anyone to access all your PHP files.

Inside of the `public` directory, you can also find `.htaccess` file. Its main purpose is to define URL rewriting rules, but you also can use it to define access rules for your web-site. For example, with `.htaccess` you can grant access to your web-site from your own IP address only, or use HTTP authorization to request users for username and password.

The `public` directory contains several subdirectories also publicly accessible by web-users:

- `css` subdirectory contains all publicly accessible CSS files;
- `img` subdirectory contains publicly accessible images (*.JPG, *.PNG, *.GIF, *.ICO, etc.);

- and `js` subdirectory stores publicly accessible JavaScript files used by your web-pages. Typically, files of jQuery[2] library are placed here, but you can put your own JavaScript files here, too.

What is jQuery library?

jQuery is a JavaScript library which was created to simplify the client-side scripting of HTML pages. jQuery's selector mechanism allows to easily attach event handlers to certain HTML elements, making it really simple to make your HTML pages interactive.

Because the Zend Skeleton Application is stored on GitHub, inside of the directory structure, you can find hidden `.gitignore` and `.gitmodules` files. These are GIT[3] version control system's files. You can ignore them (or even remove them if you do not plan to store your code in a GIT repository).

Because we will later use the skeleton as the base for our Hello World application, let's rename the `ZendSkeletonApplication-master` directory into `helloworld`, which also sounds shorter. In Linux, you can do that with the following command:

```
mv ZendSkeletonApplication-master helloworld
```

2.3 Installing Dependencies with Composer

When writing a ZF2-based web-site, you are recommended to use Composer[4] for installation of your application's dependencies. A dependence is some third-party code your app uses. For example Zend Framework 2 is the dependence for your web-site.

In Composer, any library is called *a package*. All packages installable by Composer are registered on Packagist.org[5] site. With Composer, you can identify the packages that your app requires and have Composer to download and install them automatically.

The dependencies of the skeleton application are declared in `APP_DIR/composer.json` file (see below):

[2] http://jquery.com/
[3] http://git-scm.com/
[4] http://getcomposer.org/
[5] https://packagist.org/

Contents of composer.json file

```
{
    "name": "zendframework/skeleton-application",
    "description": "Skeleton Application for ZF2",
    "license": "BSD-3-Clause",
    "keywords": [
        "framework",
        "zf2"
    ],
    "homepage": "http://framework.zend.com/",
    "require": {
        "php": ">=5.3.3",
        "zendframework/zendframework": ">2.2.0rc1"
    }
}
```

What is JSON?

JSON (JavaScript Object Notation), is a text-based file format used for human-readable representation of simple structures and nested associative arrays. Although JSON originates from Java, it is used in PHP and in other languages, because it is convenient for storing configuration data.

In that file, we see some basic info on the skeleton application (its name, description, license, keywords and home page). You will typically change this info for your future web-sites. This information is optional, so you can even safely remove it, if you do not plan to publish your web application on `Packagist` catalog.

What is interesting for us now is the `require` key. The `require` key contains the dependencies declarations for our application. We see that we require PHP engine version 5.3.3 or later and Zend Framework 2.2.0 Release Candidate 1 or later.

The information contained in `composer.json` file is enough to locate the dependencies, download and install them into the `vendor` subdirectory. Let's finally do that by typing the following commands from your command shell (replace `APP_DIR` placeholder with your actual directory name):

```
cd APP_DIR
php composer.phar self-update
php composer.phar install
```

The commands above will change your current working directory to `APP_DIR`, then self-update the Composer to the latest available version, and then install your dependencies. By the way, Composer does not install PHP for you, it just ensures PHP has an appropriate version, and if not, it will warn you.

If you look inside the `vendor` subdirectory, you can see that it now contains a lot of files. Zend Framework 2 files can be found inside the `APP_DIR/vendor/zendframework/zendframework/library` directory (figure 2.3). Here you can encounter all the components that we described in Chapter 1 (Authentication, Barcode, etc.)

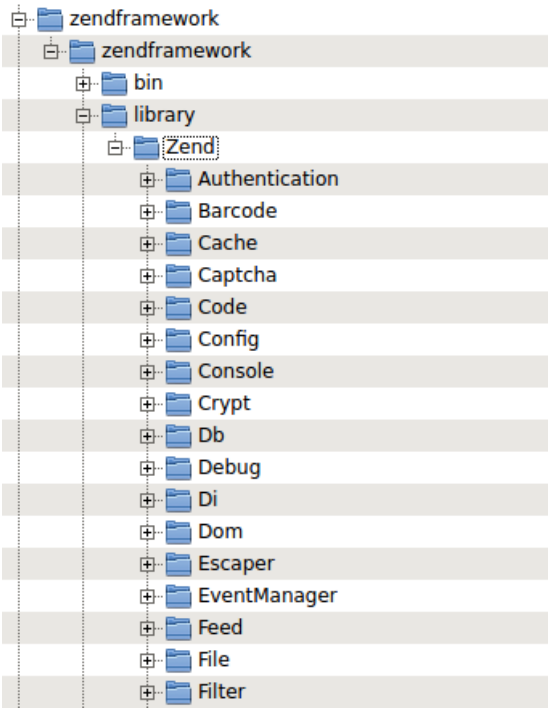


Figure 2.3. Vendor directory

In some other frameworks, another (conventional) way of dependency installation is used. You just download the dependency library as an archive, unpack it and put it somewhere inside of your directory structure (typically, to `vendor` directory). This approach was used in Zend Framework 1.

2.4 Apache Virtual Host

Now we are almost ready to get our skeleton web-site live! The last thing we are going to do is configure an Apache virtual host. A virtual host term means that you can run several web-sites on the same machine. The virtual sites are differentiated by domain name (like `site.mydomain.com` and `site2.mydomain.com`) or by port number (like `localhost` and `localhost:8080`). Virtual hosts work transparently for site users, that means users have no idea whether the sites are working on the same machine or on different ones.

Can I install the web-site to /var/www directory without virtual hosts?

With ZF2-based web sites, it would be more convenient to use Apache virtual hosts instead of putting the files inside of `/var/www`. This is because the `public` subdirectory needs to be the document root of your site. If you put an entire application in `/var/www`, which is the document root by default, you would have to additionally configure the `.htaccess` file to forbid access to everything except the `public` subdirectory. With virtual host configuration this is a bit easier to do.

Currently, we have the skeleton application inside of home folder. To let Apache know about it, we need to edit the virtual host file.

Virtual host file may be located at a different path, depending on your operating system type. For example, in Linux Ubuntu it is located in `/etc/apache2/sites-available/000-default` file. Moreover, virtual host file name and content may look differently depending on Apache HTTP Server's version. For OS- and server-specific information about virtual hosts, please refer to Appendix A.

Let's now edit the default virtual host file to make it look like below (this example is applicable to Apache v.2.4):

Virtual host file

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost

    DocumentRoot /home/username/helloworld/public
    <Directory />
        Options FollowSymLinks
        AllowOverride None
    </Directory>
    <Directory /home/username/helloworld/public/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride All
        Order allow,deny
        allow from all
        Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log

    # Possible values include: debug, info, notice, warn, error, crit,
    # alert, emerg.
    LogLevel warn

</VirtualHost>
```

Line 1 of the file makes Apache to listen to all (*) IP addresses on port 80.

Line 2 defines the web master's E-mail address. If something bad happens to the site, Apache sends an alert E-mail to this address. You can enter your E-mail here.

Line 4 defines the document root directory (`APP_DIR/public`). All files and directories under the document root will be accessible by web-users. You should set this to be the absolute path to skeleton application's `public` directory. So, the directories and files inside `public` (like `index.php`, `css`, `js`, etc.) will be accessible, while directories and files above `public` directory (like `config`, `module`, etc.) wont' be accessible by web users, which enhances the security of the web site.

Lines 5-8 define default access rules for directories. These rules are rather strict. The `Options FollowSymLinks` directive allows Apache to follow symbolic links (in Linux, a symbolic links is an analog of a shortcut in Windows). The `AllowOverride None` directive forbids overriding any rules using `.htaccess` files.

Lines 9-14 define rules for the document root directory (`APP_DIR/public`). These rules override the default rules mentioned above. For example, the `AllowOverride All` directive allows to define any rules in `.htaccess` files. The `Order allow,deny` and `allow from all` control a three-pass access control system, effectively allowing everyone to visit the site.

Line 16 defines the path to `error.log` file, which can be used to troubleshoot possible errors occurring in your site code. Line 23 defines the logging level to use (the `warn` means that warnings and errors will be written to log).

Lines 18-19 are comments and ignored by Apache. You mark comments with the hash (#) character.

Zend Framework 2 utilizes Apache's URL rewriting module for redirecting web-users to entry script of your web-site. Please ensure that your web-server has `mod_rewrite` module enabled. For instructions on how to enable the module, please refer to Appendix A.

After editing the config file, do not forget to restart Apache to apply your changes.

2.5 Opening the Web Site in Your Browser

To open the web site, type "http://localhost" in your browser's navigation bar and press Enter. Figure 2.3 shows the site in action.

On the page that appears, you can see the navigation menu at the top. The navigation bar currently contains the single link named *Home*. Under the navigation bar, you can see the "Welcome to Zend Framework 2" caption. Below the caption, you can find some advices for beginners on how to develop new ZF2-based applications.

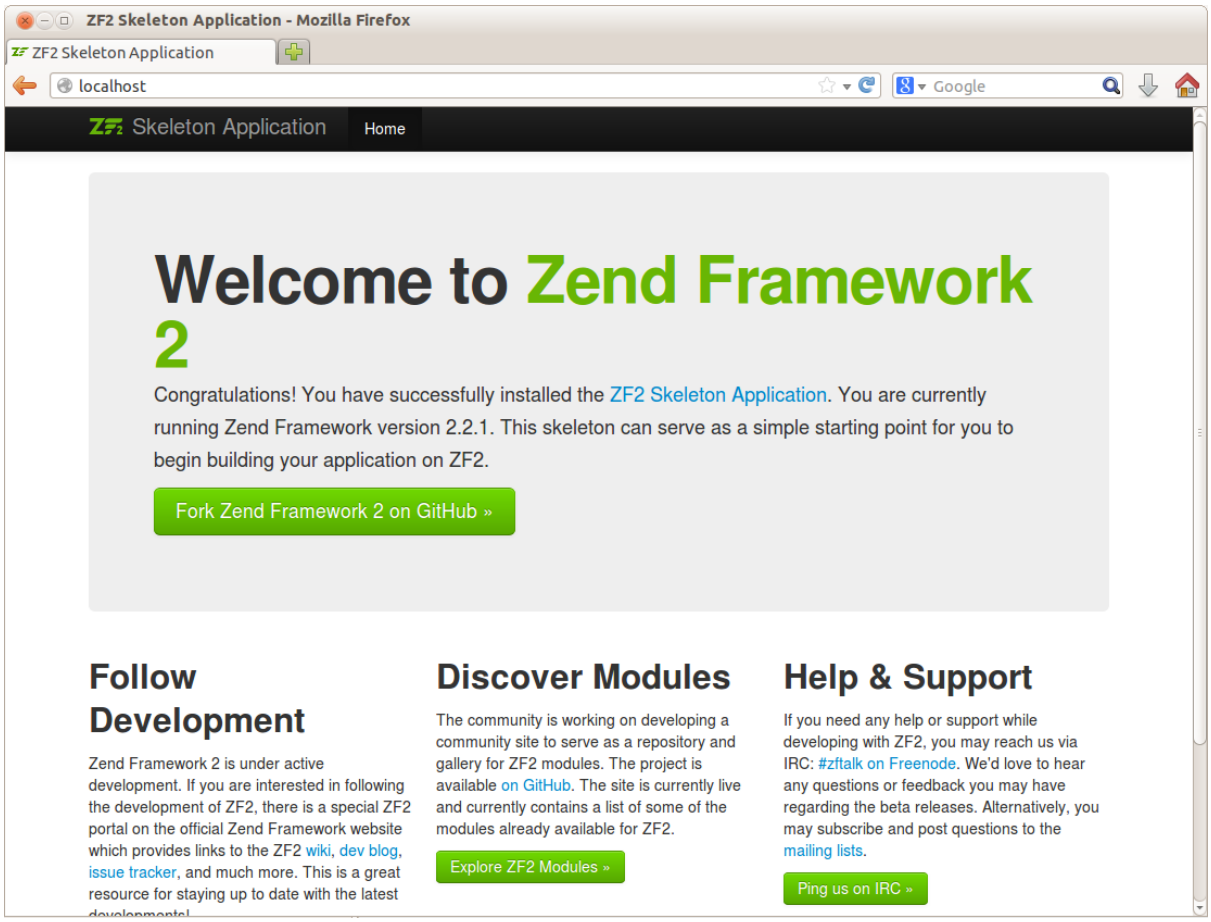


Figure 2.3. Zend Skeleton Application

2.6 Creating NetBeans Project

Now that we have the skeleton application set up and working, we will want to change something with it in the future. To easily navigate the directory structure, edit files and debug the web site, the common practice is to use an IDE (Integrated Development Environment). In this book, we use NetBeans IDE (see Appendix A for more information on how to install NetBeans).

To create NetBeans project for our skeleton application, run NetBeans and open menu *File->New Project...*. The *New Project* dialog appears (see figure 2.4).

In the *Choose Project* page that appears, you should choose PHP project type and in the right list select *Application with Existing Source* (because we already have the skeleton application's code). Then click the *Next* button to go to the next page (shown in figure 2.5).

In the *Name and Location* dialog page, you should enter the path to the code (like */home/username/helloworld*), the name for the project (for example, `helloworld`) and specify the version of PHP your code uses (PHP 5.3 or later). The PHP version is needed for the NetBeans PHP syntax checker which will scan your PHP code for errors and highlight them. Press the *Next* button to go to the next dialog page (shown in figure 2.6).

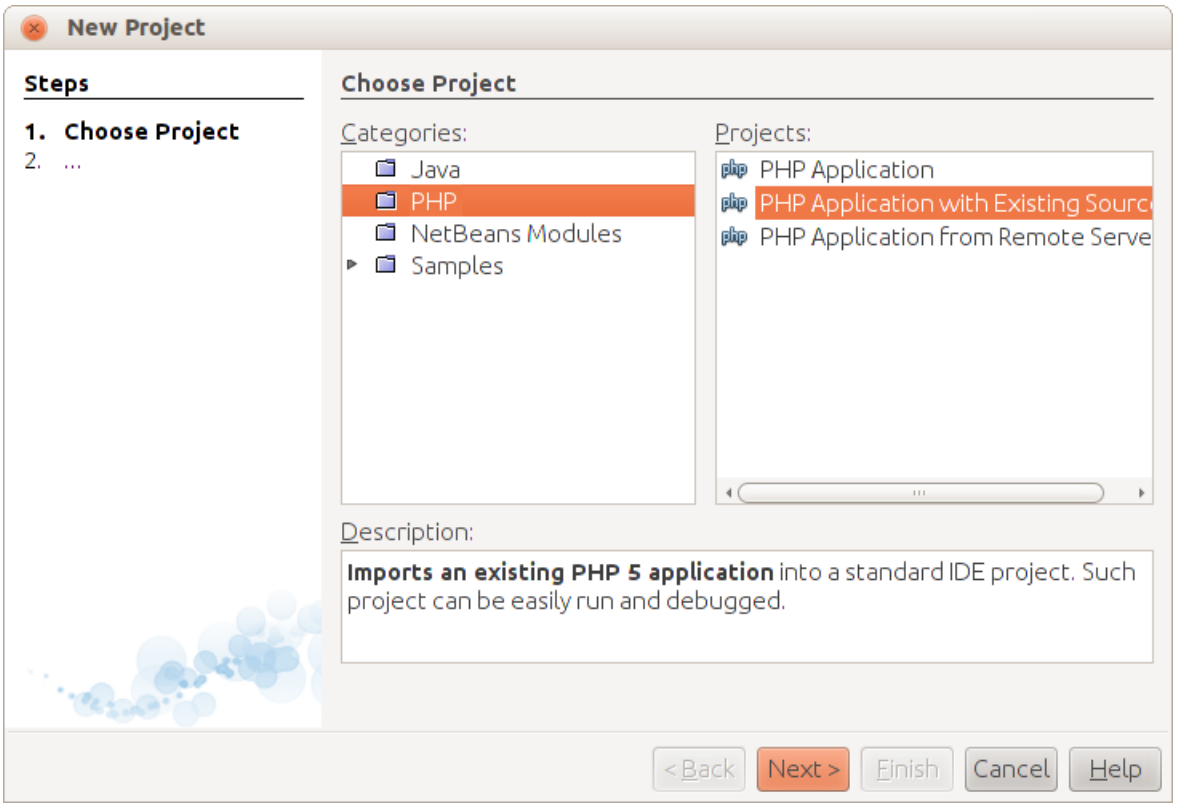


Figure 2.4. Creating NetBeans Project - Choose Project Page

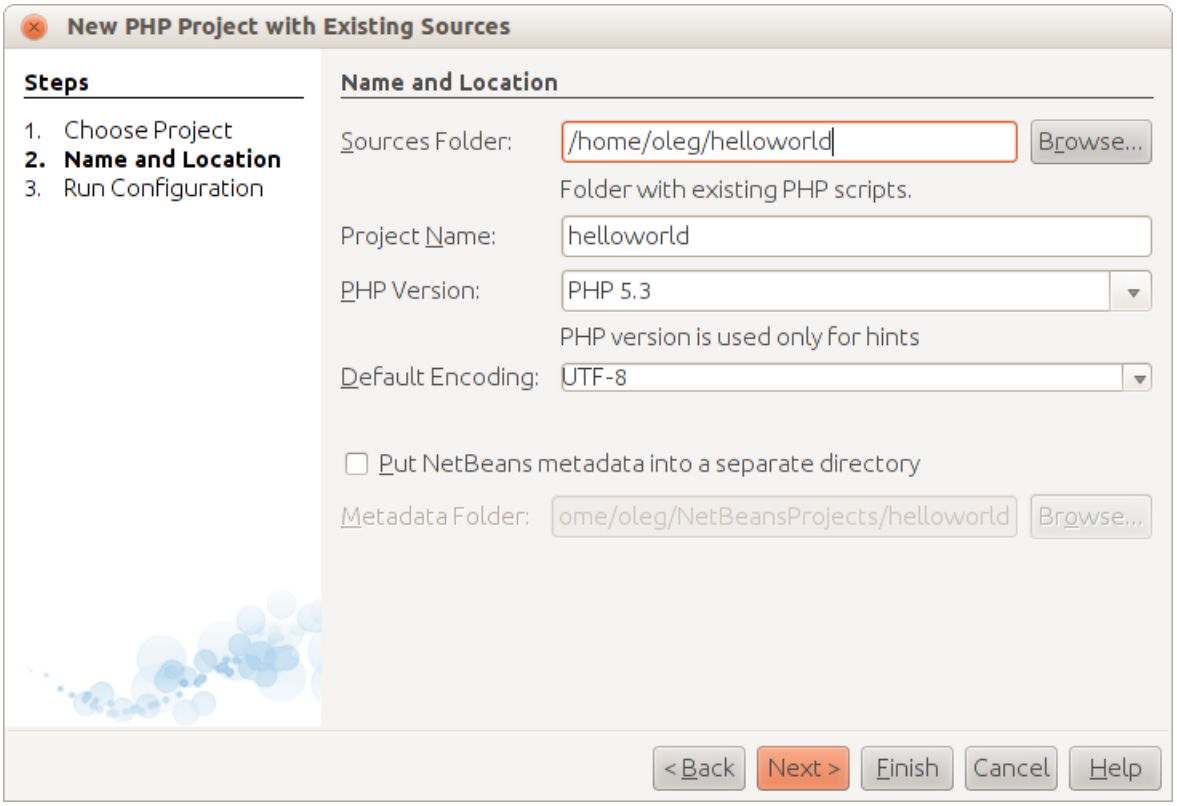


Figure 2.5. Creating NetBeans Project - Name and Location Page

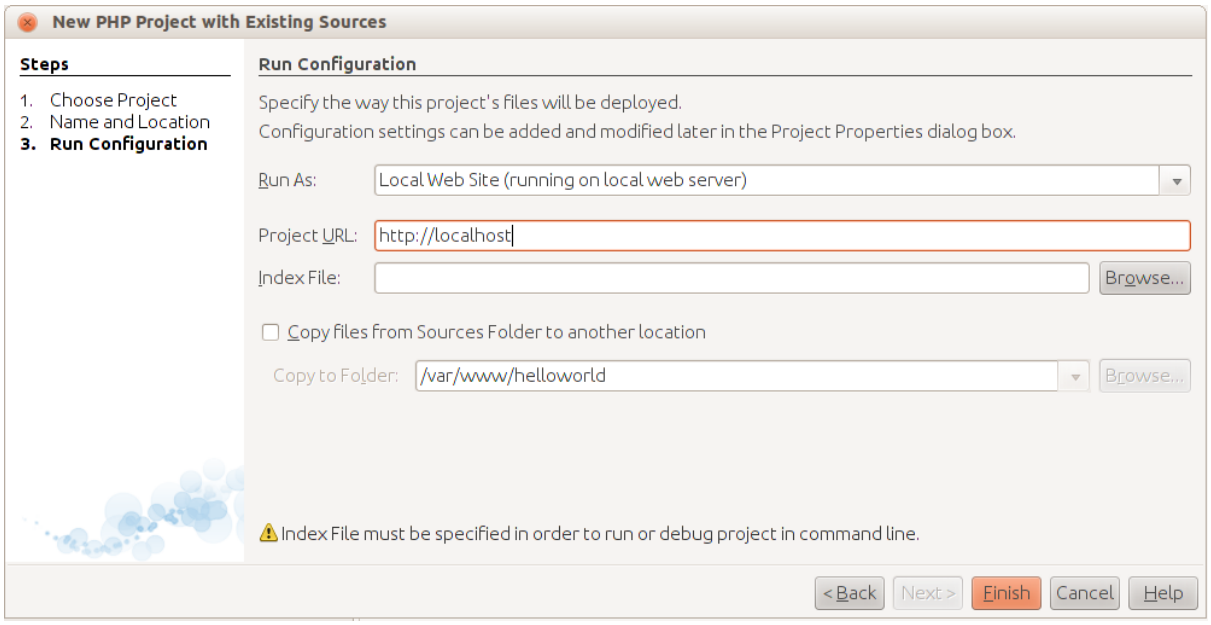


Figure 2.6. Creating NetBeans Project - Choosing Configuration Page

In the *Run Configuration* page, it is recommended that you specify the way you run the web site (Local Web Site) and web site URL (`http://localhost`). Keep the *Index File* field empty (because we are using `mod_rewrite`, the actual path to your `index.php` file is hidden by Apache). If you are seeing the warning message like "Index File must be specified in order to run or debug project in command line", just ignore it.

Click the *Finish* button to create the project. When the *helloworld* project has been successfully created, you should see the project window (see the figure 2.7).

In the project window, you can see the menu bar, the tool bar, the *Projects* pane where your project files are listed, and, in the right part of the window, you can see the code of the `index.php` entry file.

Please refer to Appendix B for more NetBeans usage tips, including launching and interactively debugging ZF2-based web sites.

It's time for some advanced stuff...

Congratulations! We've done the hard work of installing and running the Zend Skeleton Application, and now it's time to have a rest and read about some advanced things in the last part of this chapter.

The rest of this chapter is skipped in this free sample.

3. Web Site Operation

In this chapter we will provide some theory on how a typical Zend Framework 2 based application works. You'll learn how PHP namespaces are used for avoiding name collisions, what class autoloading is, how to define application configuration parameters and the stages present in an application's life-cycle. You will also become familiar with such an important ZF2 components as `Zend\EventManager`, `Zend\ModuleManager` and `Zend\ServiceManager`. If instead of learning the theory, you want to have some practical examples, skip this chapter and refer directly to Chapter 4.

ZF2 components covered in this chapter:

Component	*Description*
`Zend\Mvc`	Support of Model-View-Controller pattern. Separation of business logic from presentation.
`Zend\Loader`	Implements the PHP class autoloading support.
`Zend\ModuleManager`	This component is responsible for loading and initializing modules of the web application.
`Zend\EventManager`	This component implements functionality for triggering events and event handling.
`Zend\ServiceManager`	Implements the registry of all services available in the web application.

3.1 PHP Namespaces

When you use classes from different libraries (or even classes from different components of a single library) in your program, the class names may conflict. This means you can encounter two classes having the same name, resulting in PHP interpreter error. If you've ever programmed web sites with Zend Framework 1, you might remember those *extra* long class names like `Zend_Controller_Abstract`. The idea with long names was utilized to avoid name collisions between different components. Each component defined its own name prefix, like `Zend_` or `My_`.

To achieve the same goal, Zend Framework 2 uses the PHP 5.3 language feature called *namespaces*. The namespaces allow to solve the name collisions between code components, and provide you with the ability to make the long names shorter.

A namespace is a container for a group of names. You can nest namespaces into each other. If a class or function does not define a namespace, it lives inside of the *global* namespace (for example, PHP classes `Exception` and `DateTime` belong to global namespace).

A real-world example of a namespace definition (taken from *ZendMvc* component) is presented below:

```
1  <?php
2  namespace Zend\Mvc;
3
4  // ...
5
6  /**
7   * Main application class for invoking applications.
8   */
9  class Application
10 {
11   // ... class members were omitted for simplicity ...
12 }
```

You may notice that in example above we have the opening `<?php` tag which tells the PHP engine that the text after the tag is a PHP code. In example above, when the file contains only the PHP code (without mixing PHP and HTML tags), you don't need to insert the closing `?>` tag after the end of the code. Moreover, this is not recommended and may cause undesired effects, if you occasionally add some character after the closing `?>` tag.

In Zend Framework 2, all classes belong to top-level *Zend* namespace. The line 2 defines the namespace *Mvc*, which is nested into *Zend* namespace, and all classes of this component (including the `Application` class shown in this example on lines 9-12) belong to this namespace. You separate nested namespace names with the back-slash character ('\').

In other parts of code, you reference the `Application` class using its full name:

```
<?php
$application = new \Zend\Mvc\Application;
```

Please note the leading back-slash in `\Zend\Mvc\Application` name. It represents the global namespace.

It is also possible to use the *alias* (short name for the class) with the help of PHP's `use` statement:

```
<?php
// Define the alias in the beginning of the file.
use Zend\Mvc\Application;

// Later in your code, use the short class name.
$application = new Application;
```

Although the alias allows to use a short class name instead of the full name, its usage is optional. You are not required to always use aliases, and can refer the class by its full name.

Every PHP file of your application typically defines the namespace (except *index.php* entry script and config files, which typically do not). For example, the main module of your site, the *Application* module, defines its own namespace whose name equals to the module name:

```
<?php
namespace Application;

// ...

class Module
{
    // ... class members were omitted for simplicity ...
}
```

The rest of this chapter is skipped in this free sample.

4. Model-View-Controller

In this chapter, you will learn about the models, views and controllers (MVC). The web application uses the MVC pattern to separate business logic from presentation. The goal of this is to allow for code reusability and separation of concerns.

ZF2 components covered in this chapter:

Component	*Description*
`Zend\Mvc`	Support of MVC pattern. Implements base controller classes, controller plugins, etc.
`Zend\View`	Implements the functionality for variable containers, rendering a web page and common view helpers.
`Zend\Http`	Implements a wrapper around HTTP request and response.
`Zend\Version`	A small auxiliary component, which can be used for checking the version of Zend Framework.

4.1 Get the Hello World Example from GitHub

In this and in the next chapters, we will provide some code examples that you may want to reproduce yourself. It may be difficult for a novice to write code without mistakes. If you are stuck or can not understand why your code does not work, you can download the complete *Hello World* web application from GitHub code hosting. The examples from this chapter are mostly the part of this sample application.

To download the Hello World application, visit this page[1] and click the *Download ZIP* button to download the code as a ZIP archive (see figure 4.1). When download is complete, unpack the archive to some directory.

Then navigate to the `helloworld` directory containing the complete source code of the *Hello World* example:

```
/using-zend-framework-2-book
   /helloworld
     ...
```

The Hello World is a complete web site which can be installed on your machine. To install the example, you can either edit your default Apache virtual host file or create a new one. After editing the file, restart the Apache HTTP Server and open the web site in your web browser.

[1] https://github.com/olegkrivtsov/using-zend-framework-2-book

4.2 Separating Business Logic from Presentation

A typical web site has three kinds of functionality: code implementing business logic, code implementing user interaction and code rendering HTML pages (presentation). Before PHP frameworks, programmers usually merged these three types of code in a single big PHP script file, which made it a pain to test and maintain such a code, especially when you write a large web site.

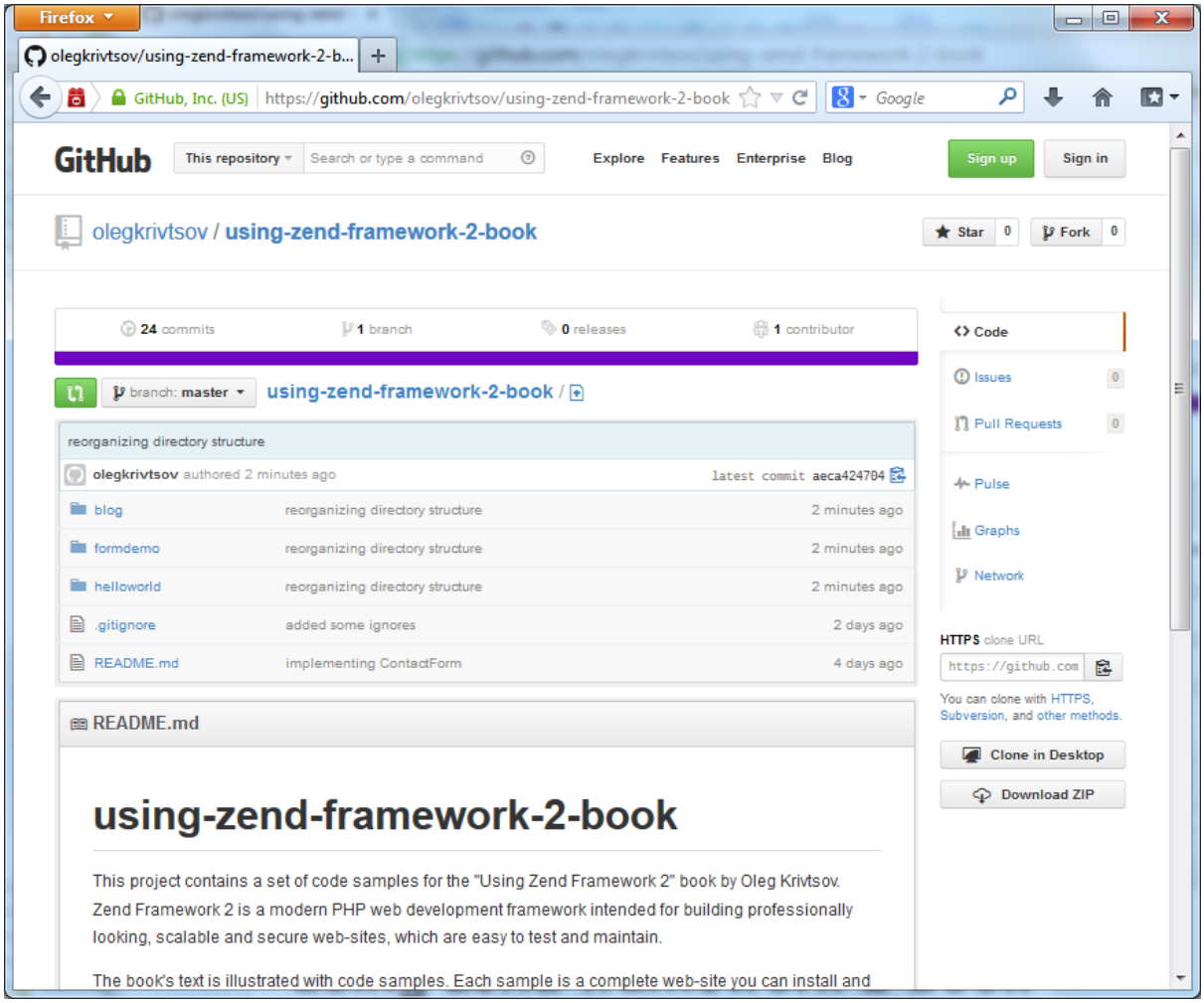


Figure 4.1. The Hello World sample can be downloaded from GitHub

Since that time, PHP became object-oriented, and now you can organize your code into classes. The *Model-View-Controller* (MVC) pattern is just a set of advices telling you how to organize your classes in a better manner, to make them easy to maintain.

In MVC, classes implementing your business logic are called *models*, code snippets rendering HTML pages are called *views*, and the classes responsible for interacting with user are called *controllers*.

Views are implemented as *code snippets*, not as classes. This is because views are typically very simple and contain only the mixture of HTML and inline PHP code.

The main objective of the MVC concept is to separate the business logic (models) from its visualization (views). This is also called the *separation of concerns*, when each layer does its specific tasks only.

By separating your models from views, you reduce the number of dependencies between them. Therefore, changes made to one of the layers have the lowest possible impact on other layers. This separation also improves the *code reusability*. For example, you can create multiple visual representations for the same models.

To better understand how this works, lets remember that any web site is just a PHP program receiving an HTTP request from the web server, and producing an HTTP response. Figure 4.2 shows how an HTTP request is processed by the MVC application and how the response is generated:

- First, the site visitor enters an URL in his web browser, for example *http://localhost*, and the web browser sends the request to the web server over the Internet.
- Web server's PHP engine runs the *index.php* entry script. The only thing the entry script does is creating the `Zend\Mvc\Application` class instance.
- The application uses its *router* component for parsing the URL and determining to which controller to pass the request. If the route match is found, the controller is instantiated and its appropriate *action method* is called.
- In the controller's action method, parameters are retrieved from GET and POST variables. To process the incoming data, the controller instantiates appropriate model classes and calls their methods.
- Model classes use business logic algorithms to process the input data and return the output data. The business logic algorithms are application-specific, and typically include retrieving data from database, managing files, interacting with external systems and so on.
- The result of calling the models are passed to the corresponding view script for the rendering of the HTML page.
- View script uses the model-provided data for rendering the HTML page.
- Controller passes the resulting HTTP response to application.
- Web server returns the resulting HTML web page to the user's web browser.
- The user sees the page in browser window.

Now you might have some idea how models, views and controllers cooperate to generate HTML output. In the next sections, we describe them in more details.

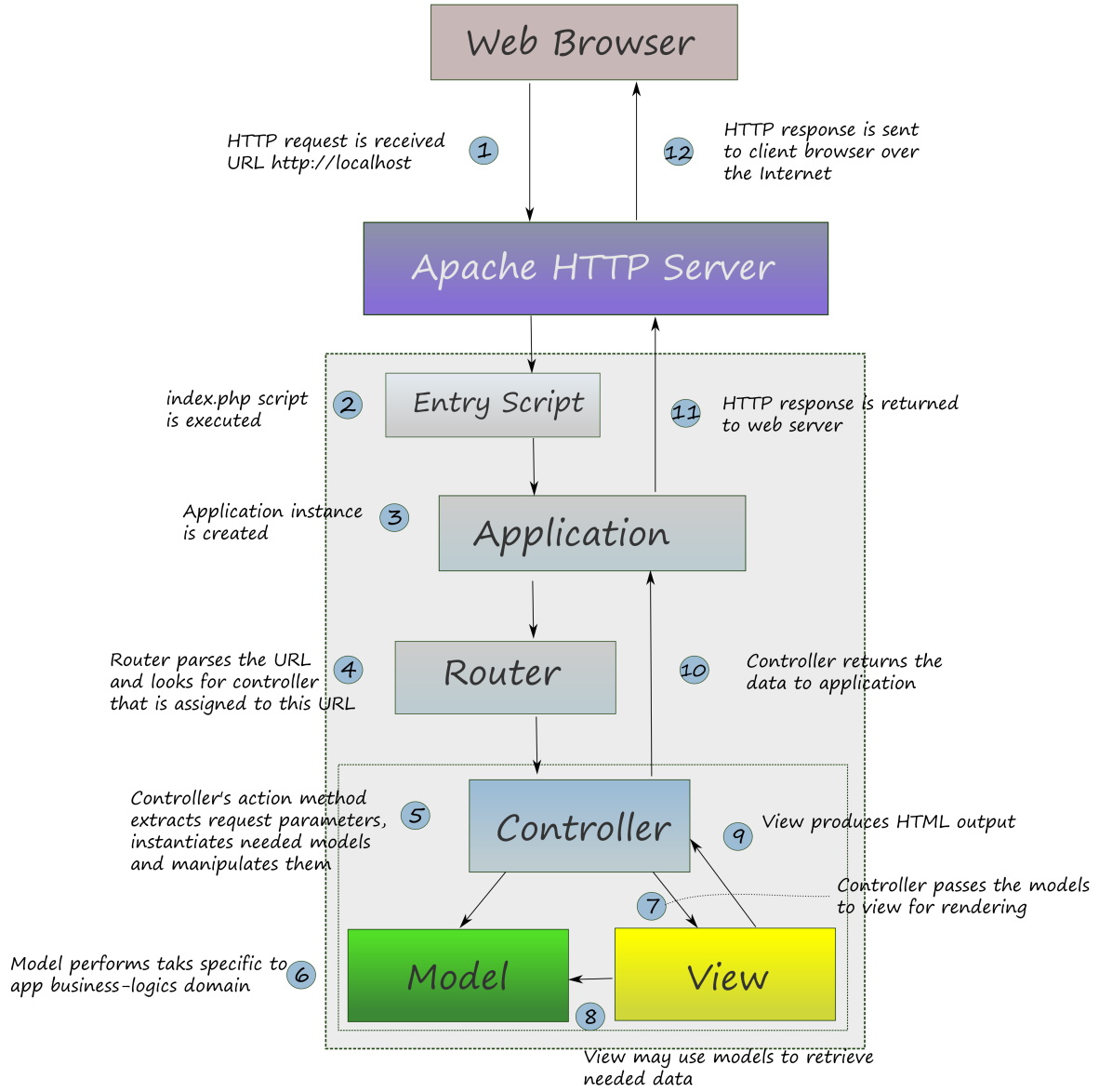


Figure 4.2. HTTP request processing in an MVC web application

The rest of this chapter is skipped in this free sample.

5. URL Routing

When a site user enters a URL in a web browser, the request is finally dispatched to controller's action. In this chapter, we will learn about how ZF2-based application maps page URLs to controllers and their actions. This mapping is accomplished with the help of routing. Routing is implemented as a part of `Zend\Mvc` component.

ZF2 components covered in this chapter:

Component	*Description*
`Zend\Mvc`	Implements support of MVC and routing.
`Zend\Barcode`	Auxiliary component implementing barcodes.

5.1 URL Structure

To better understand routing, we first need to look at the URL structure. A typical URL from an HTTP request consists of *segments*. The segments are URL parts delimited by slash characters ('/'): there are scheme, host name, path and query segments.

For example, let's look at the URL "http://site1.yourserver.com/path/to/page?query=Search" (figure 5.1).

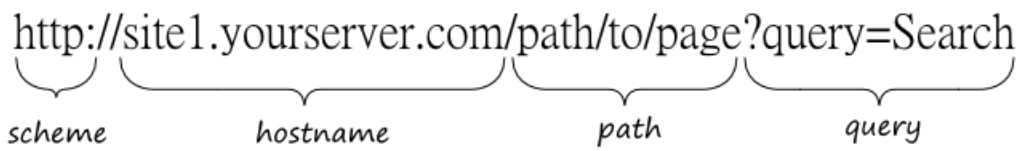


Figure 5.1. Typical URL structure

This URL begins with a scheme segment (the scheme typically looks like *http* or *https*). Then, the host name segment follows which is the domain name of your web server (like *site1.yourserver.com*). Optional path segments follow the host name. So if you have the path part "/path/to/page" then "path", "to", and "page" would each be a URL segment. Next, after the question mark, the optional query part follows. It consists of one or several "name=value" parameters separated from each other by an ampersand character ('&').

Each segment in a URL uses special character encoding, which is named the *URL encoding*. This encoding ensures that the URL contains only "safe" characters from the ASCII [1] table. If a URL contains unsafe characters, they are replaced with a percentage character ('%') followed by two hexadecimal digits (for example, the space character will be replaced by '%20').

[1] ASCII (American Standard Code for Information Interchange) is a character set which can be used to encode characters from the English alphabet. It encodes 128 characters: digits, letters, punctuation marks and several control codes inherited from Teletype machines.

5.2 Route Types

Routing is a mechanism which allows to map HTTP request to the controller. With routing, ZF2 knows which of the controller's action method to execute as the result of the request. For example, you can map "http://localhost/" URL to `IndexController::indexAction()` method or "http://localhost/about" URL to `IndexController::aboutAction()` method.

A typical routing rule has the *name*, *type* and *options*. The name is used to uniquely identify the rule. The type defines the name of the PHP class which implements the algorithm used for comparing the URL string. The options is an array that includes the *route* string which should be compared against the URL string, and several parameters called the *defaults*.

In general, the routing algorithm may use any data from HTTP request for matching the route. However, typically, it takes only the URL string (or its substring) as input. The algorithm then compares the URL with the route, and if the URL string matches the route, returns several parameters, including the controller's name and action method's name, and possibly others. These parameters may be either hard-coded in a configuration file or grabbed from the URL string. If a certain parameter cannot be retrieved from the URL, its default value is returned.

There are several standard route types provided by Zend Framework 2 (shown in table 5.1). These route types are implemented as classes living in the `Zend\Mvc\Router\Http` namespace.

Table 5.1. Route Types

Route Type	*Description*
Literal	Exact matching against a path part of a URL.
Segment	Matching against a path segment (or several segments) of a URL.
Regex	Matching the path part of a URL against a regular expression template.
Wildcard	Matching the path part of a URL against a key/value pattern.
Hostname	Matching the host name against some criteria.
Scheme	Matching URL scheme against some criteria.
Method	Matching an HTTP method (e.g. GET, POST, etc.) against some criteria.

Each route type in the table above (except the *Method* type) may be matched against a specific segment (or several segments) of a URL. The *Method* route type is matched against the HTTP method (either GET or POST) retrieved from HTTP request.

There is also the *Query* route type, which is declared deprecated and is not recommended to use. This route type is actually not needed, because you can retrieve query parameters from your URL with the `Params` controller plugin (see the *Retrieving Data from HTTP Request* section in Chapter 4).

5.3 Combining Route Types

Routes may be combined with the help of "aggregate" route types (shown in table 5.2). The compound route types allow to define arbitrarily complex URL mapping rules.

Table 5.2. Aggregate Route Types

Route Type	*Description*
SimpleRouteStack	Aggregates different route types in a list with priorities.
TreeRouteStack	Aggregates different route types in a tree-like structure.
Part	Aggregates different route types in a subtree.
Chain	Aggregates different route types in a chain (degenerated subtree).

The `TreeRouteStack` and `SimpleRouteStack` are used as the "top-level" route types. The *SimpleRouteStack* allows to organize different routing rules in a priority list. The *TreeRouteStack* allows to *nest* different routing rules, forming a "tree".

Figure 5.2 shows the route class inheritance diagram.

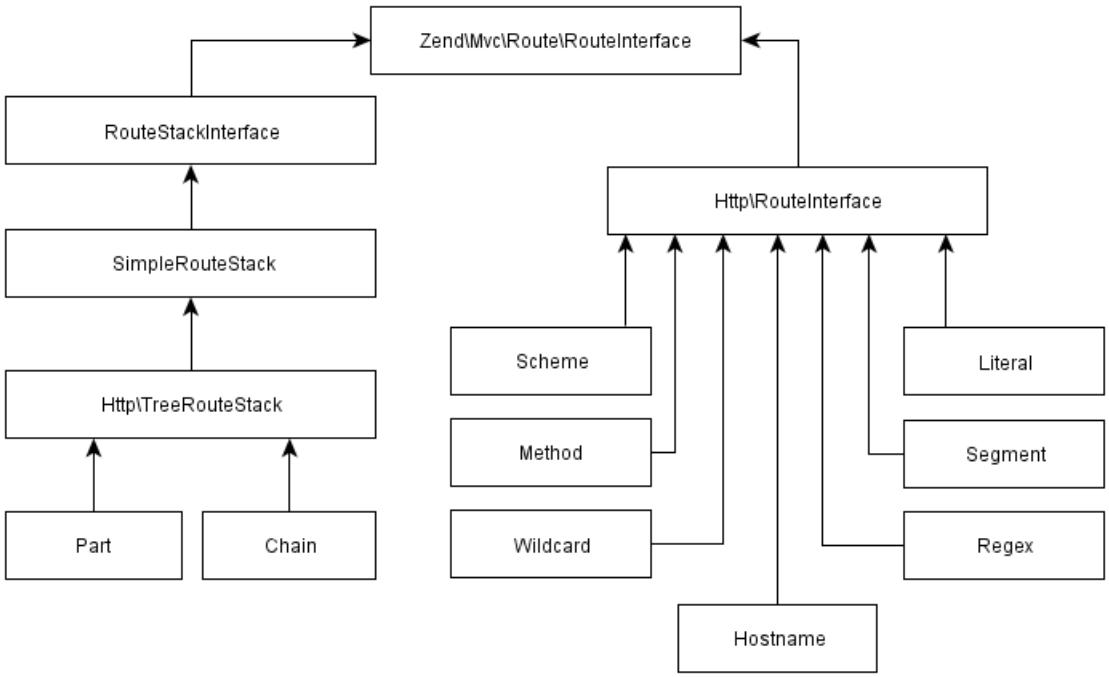


Figure 5.2. Route class inheritance diagram

As you can see from the image, all route classes are inherited from `RouteInterface` interface (we will learn this interface in details in the *Writing Own Route Type* section later in this chapter). The `SimpleRouteStack` is a parent class for `TreeRouteStack` class, which inherits the behavior of the simple route stack (allows to organize routes in priority list) and extends it (allows to organize routes in subtrees). The `Part` and `Chain` classes are derived from `TreeRouteStack` class and are used internally by the `TreeRouteStack` for building subtrees and chains of child routes.

You may notice that the `SimpleRouteStack` class lives in the `Zend\Mvc\Router` namespace, while other route classes live in its sub-namespace `Zend\Mvc\Router\Http`. This is because routing is also used for mapping shell commands to controllers in console applications. Thus, console route classes will live in `Zend\Mvc\Router\Console`, while the `SimpleRouteStack` compound route type will be used for both HTTP routing and console routing.

The rest of this chapter is skipped in this free sample.

6. Page Appearance and Layout

In this chapter you will learn how to make your web pages attractive and professionally looking with the help of Twitter Bootstrap CSS Framework and how to position elements on a page using ZF2 layout mechanism. You'll also become familiar with common view helpers allowing for composing web pages of reusable parts. If you are new to Twitter Bootstrap, it is also recommended that you refer to Appendix C for advanced description of Bootstrap capabilities.

ZF2 components covered in this chapter:

Component	*Description*
`Zend\Mvc`	Support of MVC pattern. Implements base controller classes, controller plugins, etc.
`Zend\View`	Implements the functionality for variable containers, rendering a web page and common view helpers.

6.1 About CSS Stylesheets and Twitter Bootstrap

In a ZF2-based web site, for defining the visual appearance and style of the web pages, CSS stylesheets are utilized. These CSS [1] files are typically stored in *APP_DIR/public/css* directory.

Because the CSS rules may be rather complex and require laborious adjustment and the skills of a designer, they can be separated in a "library" (framework). Analogous to PHP frameworks, CSS frameworks allow for code reusability.

Today, several CSS frameworks exist on the market, and one of them is Twitter Bootstrap[2] (or shortly, the Bootstrap). Originally designed at Twitter to unify the appearance of their own web tools, Bootstrap has became a popular CSS framework, allowing to make your web site professionally looking and visually appealing, even if you don't have advanced designer skills and without the need of creating basic CSS rules (but, of course you can define your own custom CSS rules on top of Bootstrap to customise your site's appearance). Bootstrap is freely distributed under the Apache License v.2.0[3].

Twitter Bootstrap is shipped with Zend Skeleton Application, so you can use it out of the box. Alternatively, you can download the newest version of Bootstrap from the project's official page[4]. At the moment of writing this book, the latest version is v.3.0.

Generally, the Bootstrap does the following things:

[1] If you are new to CSS, please refer to the excellent W3Schools CSS tutorial by visiting this link.
[2] http://twitter.github.io/bootstrap/
[3] http://www.apache.org/licenses/LICENSE-2.0.html
[4] http://getbootstrap.com/

- It provides the *CSS reset* that is a style sheet defining styles for all possible HTML elements. This ensures your web site will look the same way in all web browsers.
- It provides the *base CSS rules* that define style of typography (headings and text), tables, forms, buttons, images and so on.
- It defines the *grid system*. The grid system allows to arrange elements on your web page in a grid-like structure. For example, look at the Skeleton Application's main page (figure 6.1), where we have the grid consisting of three columns.
- It defines useful *web interface components* like dropdown menus, navigation bars, breadcrumbs, pagination and so on. For example, on the skeleton app's main page, there is the navigation bar component at the top, and the header (also called the Hero Unit or Jumbotron) component below the navbar. These components are very handy on any web site.
- In includes the *JavaScript extensions* that allow to make Bootstrap-provided interface components more interactive. For example, JavaScript is used to animate dropdown menus and display "modal dialogs".

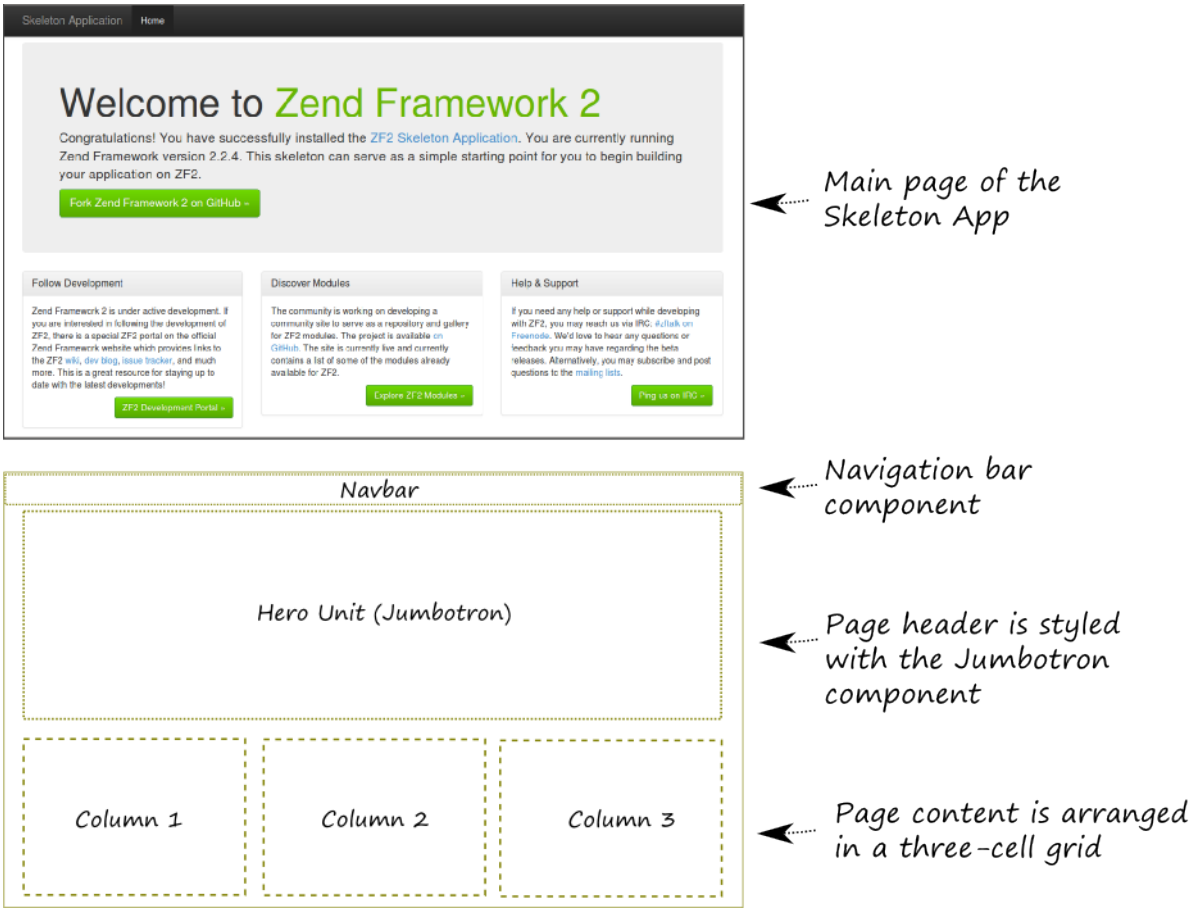


Figure 6.1. Main page of the skeleton app and its layout

If you are new to Twitter Bootstrap, it is recommended that you refer to Appendix C, where you can find more information about using Twitter Bootstrap and its components.

6.2 Page Layout in Zend Framework 2

Pages of your web site typically have some common structure that can be shared among them. For example, a typical page has the `<!DOCTYPE>` declaration to identify the HTML document, and the `<head>` and `<body>` elements:

Typical page structure

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Welcome</title>
    <!-- Include metas, stylesheets and scripts here -->
  </head>
  <body>
    <!-- Include page content here -->
  </body>
</html>
```

The `<head>` element contains the page title text, meta information and references to included stylesheets and scripts. The `<body>` element contains the content of the page, like the logo image, the navigation bar, the page text, and the footer with copyright information.

In Zend Framework 2, you define this common structure with the "master" view template called the *layout*. The layout "decorates" other view templates.

The layout template typically has a *placeholder* in which ZF2 puts the content specific to a particular page (see figure 6.2 for example).

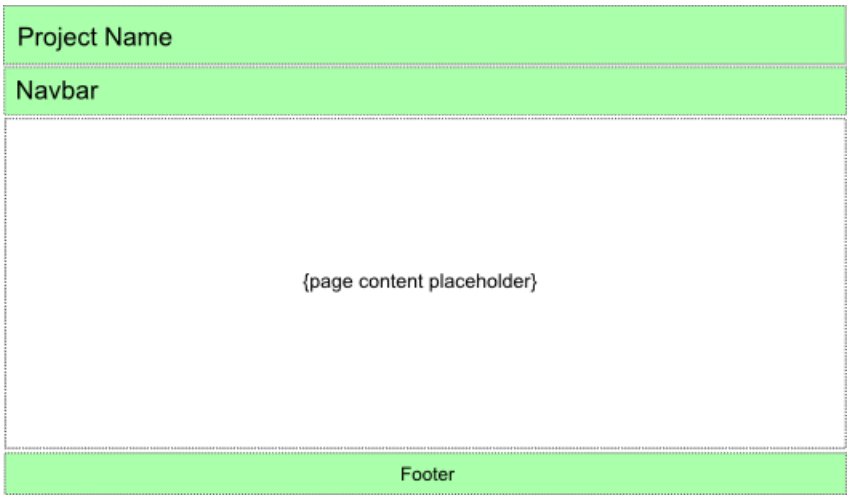


Figure 6.2. Content placeholder in layout template

In the Skeleton Application, the default layout template file is called *layout.phtml* and is located inside of the *view/layout* directory in *Application* module's directory (see figure 6.3 for example).

Let's look at the *layout.phtml* template file in more details. Below, the complete contents of the file is presented (because some lines of the file are too long for a book page, line breaks are inserted where necessary):

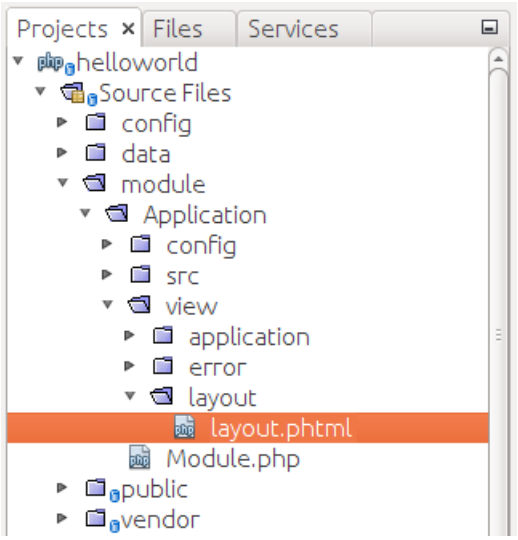


Figure 6.3. Layout directory

```
1  <?php echo $this->doctype(); ?>
2
3  <html lang="en">
4    <head>
5      <meta charset="utf-8">
6      <?php echo $this->headTitle('ZF2 '.
7                  $this->translate('Skeleton Application'))
8                       ->setSeparator(' - ')
9                       ->setAutoEscape(false)
10     ?>
11
12     <?php echo $this->headMeta()
13        ->appendName('viewport', 'width=device-width, initial-scale=1.0')
14        ->appendHttpEquiv('X-UA-Compatible', 'IE=edge')
15     ?>
16
17     <!-- Le styles -->
18     <?php
19       echo $this->headLink(array('rel' => 'shortcut icon',
20                'type' => 'image/vnd.microsoft.icon',
21                'href' => $this->basePath().'/img/favicon.ico'))
22        ->prependStylesheet($this->basePath().'/css/style.css')
23        ->prependStylesheet($this->basePath().'/css/bootstrap-theme.min.css')
24        ->prependStylesheet($this->basePath().'/css/bootstrap.min.css') ?>
25
26     <!-- Scripts -->
```

```
27      <?php echo $this->headScript()
28        ->prependFile($this->basePath().'/js/bootstrap.min.js')
29        ->prependFile($this->basePath().'/js/jquery.min.js')
30        ->prependFile($this->basePath().'/js/respond.min.js', 'text/javascript'\
31  ,
32                        array('conditional' => 'lt IE 9',))
33        ->prependFile($this->basePath().'/js/html5shiv.js', 'text/javascript',
34                        array('conditional' => 'lt IE 9',));
35      ?>
36
37    </head>
38    <body>
39      <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">
40        <div class="container">
41          <div class="navbar-header">
42            <button type="button" class="navbar-toggle" data-toggle="collapse"
43                    data-target=".navbar-collapse">
44              <span class="icon-bar"></span>
45              <span class="icon-bar"></span>
46              <span class="icon-bar"></span>
47            </button>
48            <a class="navbar-brand" href="<?php echo $this->url('home') ?>">
49              <img src="<?php echo $this->basePath('img/zf2-logo.png') ?>"
50                   alt="Zend Framework 2"/> 
51              <?php echo $this->translate('Skeleton Application') ?>
52            </a>
53          </div>
54          <div class="collapse navbar-collapse">
55            <ul class="nav navbar-nav">
56              <li class="active"><a href="<?php echo $this->url('home') ?>">
57                <?php echo $this->translate('Home') ?></a>
58              </li>
59            </ul>
60          </div><!--/.nav-collapse -->
61        </div>
62      </nav>
63      <div class="container">
64        <?php echo $this->content; ?>
65        <hr>
66        <footer>
67          <p>
68              &copy; 2005 - <?php echo date('Y') ?> by Zend Technologies Ltd.
69              <?php echo $this->translate('All rights reserved.') ?>
70          </p>
71        </footer>
72      </div> <!-- /container -->
```

```
73      <?php echo $this->inlineScript() ?>
74    </body>
75  </html>
```

You can see that the *layout.phtml* file (as a usual view template) consists of HTML tags mixed with PHP code fragments. When the template is rendered, ZF2 evaluates the inline PHP fragments and generates resulting HTML page visible to site users.

Line 1 above generates the `<!DOCTYPE>` [5] declaration of the HTML page with the `Doctype` view helper.

Line 3 defines the `<html>` element representing the root of the HTML document. The `<html>` tag is followed by the `<head>` tag (line 4), which typically contains a title for the document, and can include other information like scripts, CSS styles and meta information.

In line 5, the `<meta>` tag provides the browser with a hint that the document is encoded using UTF-8 [6] character encoding.

In line 6, we have the `HeadTitle` view helper that allows to define the title for the page ("ZF2 Skeleton Application"). The title will be displayed in the web browser's caption. The `setSeparator()` method is used to define the separator character for the compound page titles[7]; the `setAutoEscape()` method enhances the security by escaping unsafe characters from the page title. The `Translate` view helper is used for localizing your web site's strings into different languages.

In line 12, the `HeadMeta` view helper allows to define the `<meta name="viewport">` tag containing meta information for the web browser to control layout on different display devices, including mobile browsers. The `width` property controls the size of the viewport, while the `initial-scale` property controls the zoom level when the page is loaded. This makes the web page layout "responsive" to device viewport size.

In line 19, the `HeadLink` view helper allows to define the `<link>` tags. With the `<link>` tags, you typically define the "favicon" for the page (located in `APP_DATA\public\img\favicon.ico` file) and the CSS stylesheets.

In lines 22-24, the stylesheets common to all site pages are included by the `prependStylesheet()` method of the `HeadLink` view helper. Any page in our web site will load three CSS stylesheet files: *bootstrap.min.css* (the minified version of Twitter Bootstrap CSS Framework), *bootstrap-theme.min.css* (the minified Bootstrap theme stylesheet) and *style.css* (CSS file allowing us to define our own CSS rules overriding Bootstrap CSS rules).

Lines 27-35 include the JavaScript files that all your web pages will load. The scripts are executed by the client's web browser, allowing to introduce some interactive features for your pages. We use the the `bootstrap.min.js` (minified version of Twitter Bootstrap) and `jquery.min.js` (minified version of jQuery library) scripts. All scripts are located in *APP_DIR/public/js* directory.

[5] The `<!DOCTYPE>` declaration goes first in the HTML document, before the `<html>` tag. The declaration provides an instruction to the web browser about what version of HTML the page is written in (in our web site, we use HTML5-conformant document type declaration).

[6] The UTF-8 allows to encode any character in any alphabet around the world, that's why it is recommended for encoding the web pages.

[7] A "compound" page title consists of two parts: the first part ("Zend Skeleton Application") is defined by the layout, and the second part - defined by a particular page - is prepended to the first one. For example, for the *About* page of your site you will have the "About - Zend Skeleton Application", and for the *Documentation* page you will have something like "Documentation - Zend Skeleton Application".

Line 38 defines the `<body>` tag, the document's body which contains all the contents of the document, such as the navigation bar, text, hyperlinks, images, tables, lists, etc.

In lines 39-63, you can recognize the Bootstrap navigation bar definition. The skeleton application uses the collapsible navbar with dark inverse theme. The navbar contains the single link *Home*.

If you look at lines 63-72, you should notice the `<div>` element with `container` class which denotes the container element for the grid system. So, you can use the Bootstrap grid system to arrange the contents of your pages.

Line 64 is very important, because this line defines the inline PHP code that represents the page content placeholder we talked about in the beginning of this section. When the ZF2 page renderer evaluates the layout template, it echoes the actual page content here.

Lines 65-71 define the page footer area. The footer contains the copyright information like "2013 by Zend Technologies Ltd. All rights reserved." You can replace this information with you own company name.

Line 73 is the placeholder for JavaScript scripts loaded by the concrete page. The `InlineScript` view helper will substitute here all the scripts you register (about registering JavaScript scripts, you will see it later in this chapter).

And finally, lines 74-75 contain the closing tags for the body and the HTML document.

The rest of this chapter is skipped in this free sample.

7. Collecting User Input with Forms

In this chapter, you will become familiar with using web forms for gathering data entered by site users. In Zend Framework 2, functionality for working with forms is mainly spread across four components: the `Zend\Form` component, which allows you to build forms and contains the view helpers for rendering form elements; the `Zend\Filter`, `Zend\Validator` and `Zend\InputFilter` components which allow you to filter and validate user input:

Component	*Description*
`Zend\Form`	Contains base form model classes.
`Zend\Filter`	Contains various filters classes.
`Zend\Validator`	Implements various validator classes.
`Zend\InputFilter`	Implements a container for filters/validators.

7.1 Get the Form Demo Sample from GitHub

We will demonstrate form usage on the *Form Demo* sample web application bundled with the book. This sample is a complete web site you can install and see the working forms in action.

To download the *Form Demo* application, visit this page[1] and click the *Download ZIP* button to download the code as a ZIP archive. When the download is complete, unpack the archive to a directory of your choosing.

Then navigate to the `formdemo` directory which contains the complete source code of the *Form Demo* web application:

```
/using-zend-framework-2-book
  /formdemo
  ...
```

To install the example, you can either edit your default virtual host file or create a new one. After editing the file, restart the Apache HTTP Server and open the web site in your web browser. For additional information on Apache virtual hosts, you can refer to Appendix A.

[1] https://github.com/olegkrivtsov/using-zend-framework-2-book

7.2 About HTML Forms

Form functionality provided by Zend Framework 2 internally uses HTML forms. Because of that, we start with a brief introduction to HTML forms topic.

In HTML, forms are enclosed with `<form>` and `</form>` tags. A form typically consists of fields: text input fields, check boxes, radio buttons, submit buttons, hidden fields and so on. HTML provides several tags intended for defining form fields:

- `<input>` - specifies an input field where the user can enter some data (field appearance and behavior depends on the field type);
- `<textarea>` - multi-line text area which can contain an unlimited number of characters;
- `<button>` - a clickable button[2];
- `<select>` - a dropdown list;
- `<option>` - used inside the `<select>` element for defining the available options in a dropdown list.

In table 7.1, you can find examples of HTML form field definitions. Figure 7.1 contains corresponding field visualizations (except the "hidden" field type, which has no visual representation).

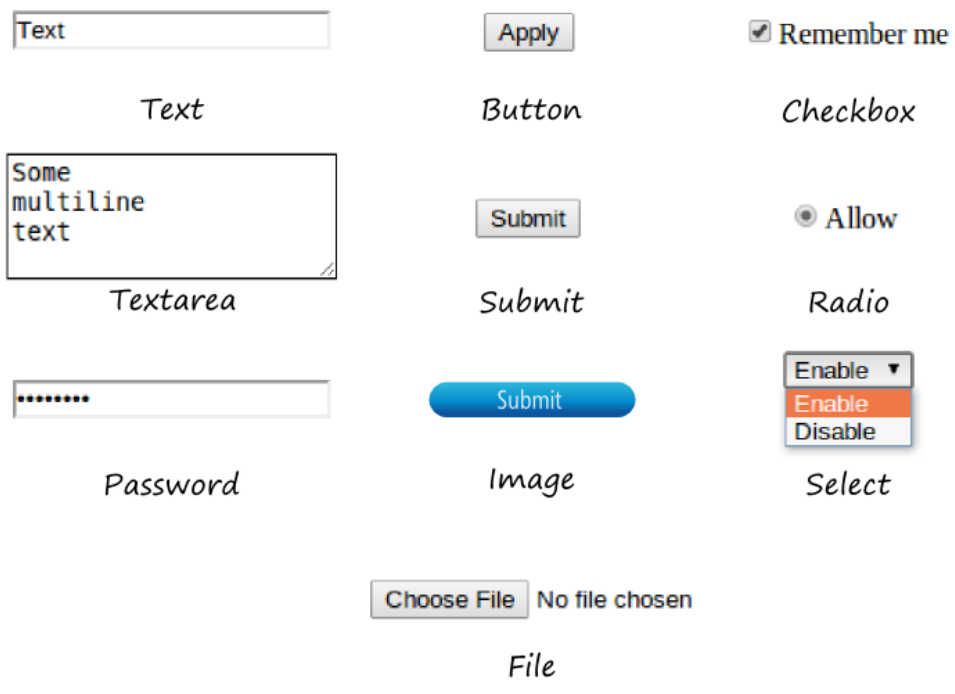


Figure 7.1. Standard HTML form fields

[2]The `<button>` field is analogous to `<input type="button">`, however it provides additional capabilities, like specifying a graphical icon on the button.

Table 7.1. Standard HTML form fields

Field	*Definition*
Text input field	`<input type="text" />`
Text area	`<textarea rows=4></textarea>`
Password	`<input type="password" />`
Button	`<input type="button" value="Apply"/>` or `<button type="button">Apply</button>`
Submit button	`<input type="submit" value="Submit" />`
Image (graphical submit button)	`<input type="image" src="button.jpg" />`
Reset button	`<input type="reset" value="Reset"/>`
Checkbox	`<input type="checkbox">Remember me</input>`
Radio	`<input type="radio" value="Radio">Allow</input>`
Select	`<select><option>Enable</option><option>Disable</option></select>`
File	`<input type="file" />`
Hidden field	`<input type="hidden" />`

HTML5 introduced several new form field types (listed in table 7.2); figure 7.2 contains corresponding field visualizations.

HTML5 fields provide more convenient ways for entering the most frequently used data types: numbers, dates, E-mails, URLs, etc. Additionally, on form submit, the web browser validates that the user entered data is in a correct format, and if not the browser will prevent form submission and ask the user to correct the input error.

Table 7.2. HTML5 form fields

Field	*Definition*
Color picker	`<input type="color" />`
Date	`<input type="date" />`
Date-time (with time zone)	`<input type="datetime" />`
Date-time (without time zone)	`<input type="datetime-local" />`
E-mail address	`<input type="email" />`
Number	`<input type="number" />`
Time	`<input type="time" />`
Month	`<input type="month" />`
Week	`<input type="week" />`
URL	`<input type="url" />`
Range (slider)	`<input type="range" />`
Search field	`<input type="search" name="googlesearch" />`
Telephone number	`<input type="tel" />`

At the moment of writing this chapter, not all modern web browsers completely support HTML5 form fields.

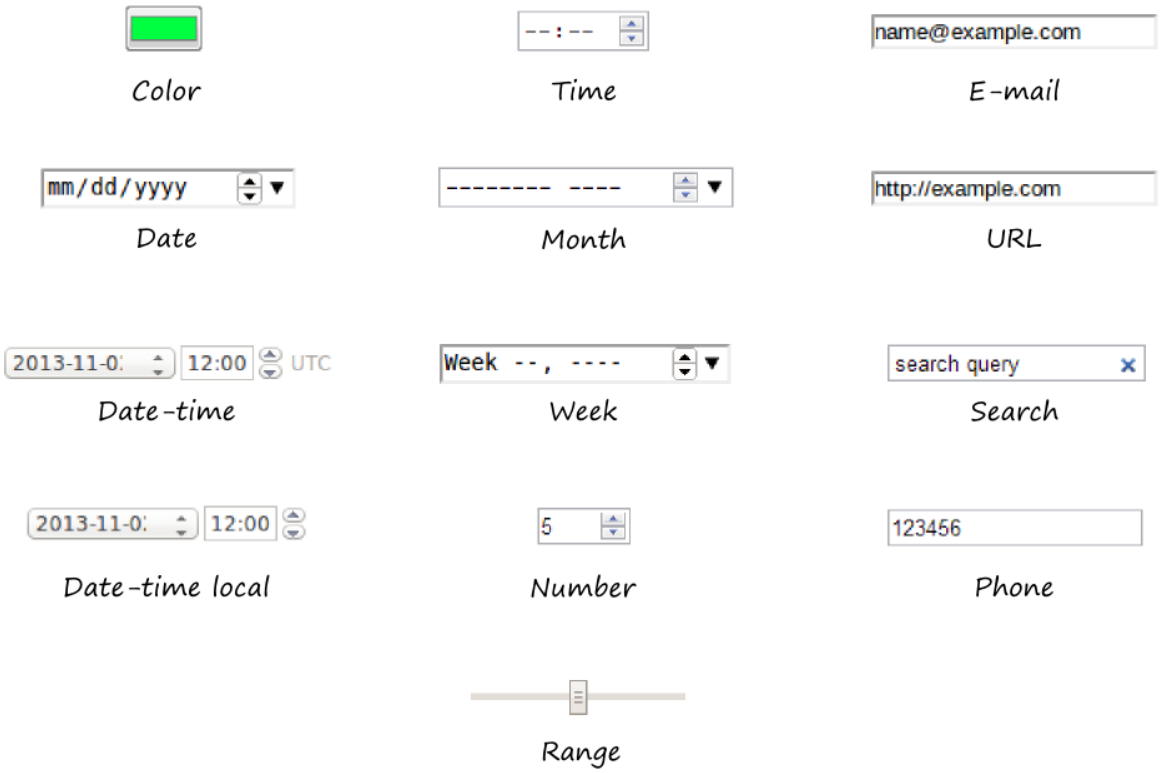


Figure 7.2. HTML5 form fields

7.2.1 Fieldsets

You can group related form fields with the help of the `<fieldset>` tag, as shown in the example below. The optional `<legend>` tag allows you to define the caption for the group.

```
<fieldset>
  <legend>Choose a payment method:</legend>
  <input type="radio" name="payment" value="paypal">PayPal</input>
  <input type="radio" name="payment" value="card">Credit Card</input>
</fieldset>
```

The HTML markup presented above will generate the group as in figure 7.3:

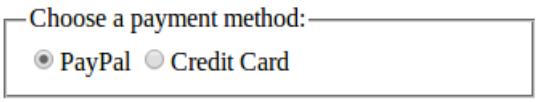


Figure 7.3. Fieldset

7.2.2 Example: "Contact Us" Form

An example of a typical HTML form is presented below:

```
1  <form name="contact-form" action="/contactus" method="post">
2    <label for="email">E-mail</label>
3    <input name="email" type="text">
4    <br>
5    <label for="subject">Subject</label>
6    <input name="subject" type="text">
7    <br>
8    <label for="body">Message</label>
9    <textarea name="body" class="form-control" rows="6"></textarea>
10   <br>
11   <input name="submit" type="submit" value="Submit">
12 </form>
```

In the example above, we have the feedback form which allows the user to enter his E-mail address, message subject, text, and then submit them to the server. The form definition begins with the `<form>` tag (line 1).

The `<form>` tag contains several important attributes:

- the `name` attribute specifies the name of the form ("contact-form").
- the `action` attribute defines the URL of the server-side script which is responsible for processing the submitted form ("/contactus").
- the `method` attribute defines the method (either GET or POST) to use for delivering form data. In this example, we use the POST method (recommended).

In line 3, we define a text input field with the help of the `<input>` element. The `name` attribute specifies the name of the field ("email"). The `type` attribute specifies the purpose of the element (the type "text" means the input field is intended for entering text).

In line 2, we have the `<label>` element which represents the label for the E-mail text input field (the corresponding input field's name is determined by the `for` attribute of the `<label>` element).

In lines 5-6, by analogy, we have the "Subject" input field and its label.

In line 9, we have the text area field which is suited well for entering multi-line text. The height of the text area (6 rows) is defined by the `rows` attribute.

In line 11, we have the submit button (input element with "submit" type). The `value` attribute allows you to set the title text for the button ("Submit"). By clicking this button, the user will send the form data to the server.

Line break `<br>` elements are used in lines 4, 7 and 10 to position form controls one below another (otherwise they would be positioned in one line).

To see what this form looks like, you can put its HTML markup code in a `.html` file and open the file in your browser. You will see the form visualization as in figure 7.4.

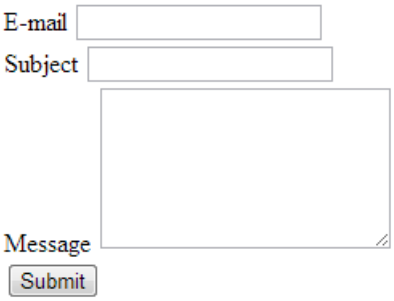


Figure 7.4. Visualization of the feedback form

If you enter some data in the feedback form and click the *Submit* button, the web browser will send an HTTP request to the URL you specified in the `action` attribute of the form. The HTTP request will contain the data you entered.

The rest of this chapter is skipped in this free sample.

8. Transforming Input Data with Filters

In this chapter, we will provide an overview of standard filters that can be used with your forms. A filter is a class which takes some input data, processes it, and produces some output data.

In general, you can even use filters *outside* forms to process an arbitrary data. For example, filters may be used in a controller action to transform the data passed as GET and/or POST variables to certain format.

ZF2 components covered in this chapter:

Component	*Description*
`Zend\Filter`	Contains various filters classes.
`Zend\InputFilter`	Implements a container for filters/validators.

8.1 About Filters

Filters are designed to take some input data, process it, and produce some output data. Zend Framework 2 provides a lot of standard filters that can be used for creating filtering rules of your forms (or, if you wish, to filter an arbitrary data outside of forms).

8.1.1 FilterInterface

Technically, a *filter* is a PHP class implementing the `FilterInterface` interface (it belongs to `Zend\Filter` namespace). The interface definition is presented below:

```
1  <?php
2  namespace Zend\Filter;
3
4  interface FilterInterface
5  {
6      // Returns the result of filtering $value.
7      public function filter($value);
8  }
```

As you can see, the `FilterInterface` interface has the single method `filter()` (line 7), which takes the single parameter `$value`. The method transforms the input data and finally returns the resulting (filtered) value.

A concrete filter class implementing the `FilterInterface` interface may have additional methods. For example, many filter classes have methods allowing configuration of the filter (set filtering options).

8.2 Standard Filters Overview

Standard filters implementing the `FilterInterface` interface belong to `Zend\Filter` component [1]. A filter class inheritance diagram is shown in figure 8.1. From that figure, you can see that base concrete class for most standard filters is the `AbstractFilter` class, which implements the `FilterInterface` interface [2].

You may notice that there is a strange filter called `StaticFilter` which does not inherit from `AbstractFilter` base class. This is because the `StaticFilter` class is actually a "wrapper" (it is designed to be a proxy to another filter without explicit instantiation of that filter).

Standard filters provided by the `Zend\Filter` component, along with a brief description of each, are listed in table 8.1.

As you can see from the table, the standard filters can be roughly divided into the following groups:

- filters casting input data to a specified type (integer, boolean, date-time, etc.);
- filters performing manipulations on a file path (getting the base name, parent directory name, etc.);
- filters performing compression and encryption of input data;
- filters manipulating string data (case conversion, trimming, character replacement and removal, URL normalizing, etc.); and
- proxy filters wrapping other filters (`Callback`, `FilterChain` and `StaticFilter`).

[1] In this section, we only consider the standard filters belonging to the `Zend\Filter` namespace, although there are other filters that can also be considered standard. For example, the `Zend\Filter\File` namespace contains several filters applicable to processing file uploads (those filters will be covered in the next chapter). Additionally, the `Zend\I18n` component defines several filter classes that are aware of the user's locale.

[2] From figure 8.1, you may also notice that there are several more base filters: `AbstractUnicode` filter is the base class for the `StringToUpper` and `StringToLower` filters, because it provides the string conversion functionality common to both of them. And, the `Decompress` filter inherits from the `Compress` filter, because these filters are in fact very similar. By analogy, the `Decrypt` filter inherits from the `Encrypt` filter, because they are the "mirror reflection" of each other as well.

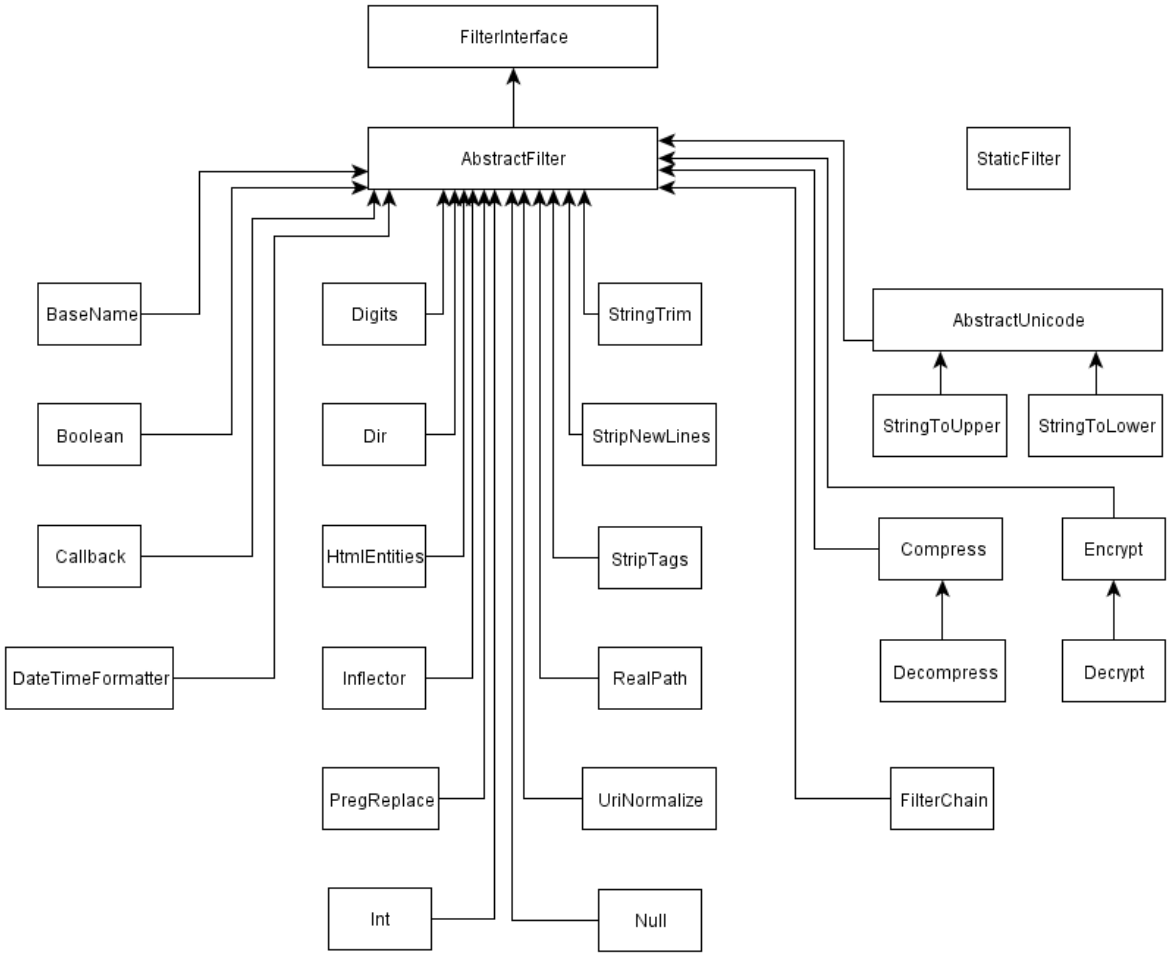


Figure 8.1. Filter class inheritance

Table 8.1. Standard filters

Class name	*Description*
`Boolean`	Returns a boolean representation of `$value`.
`Int`	Casts the input `$value` to `int`.
`Digits`	Returns the string `$value`, removing all but digit characters.
`Null`	Returns `null` if the input value can be treated as null; otherwise returns the `$value` itself.
`DateTimeFormatter`	Takes a date & time string in an arbitrary format and produces a date & time string in a given format.
`BaseName`	Given a string containing the path to a file or directory, this filter will return the trailing name component.
`Dir`	Given a string containing the path of a file or directory, this filter will return the parent directory's path.
`RealPath`	Returns canonicalized absolute pathname.
`Compress`	Compresses the input data with the specified algorithm (GZ by default).

Table 8.1. Standard filters

Class name	*Description*
`Decompress`	Decompresses the input data with the specified algorithm (the effect is inverse to the `Compress` filter).
`Encrypt`	Encrypts the input data with the specified cryptographic algorithm.
`Decrypt`	Decrypts the input data previously encrypted with the specified cryptographic algorithm.
`Inflector`	Performs the modification of a word to express different grammatical categories such as tense, mood, voice, aspect, person, number, gender, and case.
`PregReplace`	Performs a regular expression search and replace.
`StringToLower`	Converts the string to lowercase letters.
`StringToUpper`	Converts the string to uppercase letters.
`StringTrim`	Removes white spaces (space, tabs, etc.) from the beginning and the end of the string.
`StripNewlines`	Removes new line characters from string (ASCII codes #13, #10).
`HtmlEntities`	Returns the string, converting characters to their corresponding HTML entity equivalents where they exist.
`StripTags`	Removes tags (e.g., `<a></a>`) and comments (e.g., `<!-- -->`).
`UriNormalize`	Converts a URL string to the "normalized" form and prepends the schema part (e.g., converts *www.example.com* to *http://www.example.com*).
`Callback`	Allows to use a callback function as a filter.
`FilterChain`	Allows to organize several filters in a chain.
`StaticFilter`	Returns a value filtered through a specified filter class without requiring separate instantiation of the filter object.

The rest of this chapter is skipped in this free sample.

9. Checking Input Data with Validators

In this chapter, we will provide an overview of standard validators that can be used with your forms. A validator is a class designed to take some input data, check it for correctness, and return a boolean result telling whether the data is correct (and error messages if the data has some errors).

In general, you even can use validators *outside* forms to process an arbitrary data. For example, validators may be used in a controller action to ensure that data passed as GET and/or POST variables is secure and conform to certain format.

ZF2 components covered in this chapter:

Component	*Description*
`Zend\Validator`	Implements various validator classes.
`Zend\InputFilter`	Implements a container for filters/validators.

9.1 About Validators

A *validator* is designed to take some input data, check it for correctness, and return a boolean result telling whether the data is correct. If the data is incorrect, the validator generates the list of errors describing why the check didn't pass.

9.1.1 ValidatorInterface

In ZF2, a validator is a usual PHP class which implements the `ValidatorInterface` interface (it belongs to `Zend\Validator` namespace). The interface definition is presented below:

```
1  <?php
2  namespace Zend\Validator;
3
4  interface ValidatorInterface
5  {
6    // Returns true if and only if $value meets the validation requirements.
7    public function isValid($value);
8
9    // Returns an array of messages that explain why
```

```
10    // the most recent isValid() call returned false.
11    public function getMessages();
12  }
```

As you can see, the `ValidatorInterface` has two methods: the `isValid()` method (line 7) and `getMessages()` method (line 11).

The first one, `isValid()` method, is intended to perform the check of the input value (the `$value` parameter). If the validation of the `$value` passes, the `isValid()` method returns boolean `true`. If the `$value` fails validation, then this method returns `false`.

A concrete validator class implementing the `ValidatorInterface` interface may have additional methods. For example, many validator classes have methods allowing to configure the validator (set validation options).

9.2 Standard Validators Overview

Standard ZF2 validators are provided by the `Zend\Validator` component [1]. Standard validator classes inheritance is shown in figure 9.1. As you can see from the figure, most of them are derived from `AbstractValidator` base class.

Standard validators together with their brief description are listed in table 9.1. As you may notice from the table, they can be roughly divided into several groups:

- validators for checking value conformance to certain format (IP address, host name, E-mail address, credit card number, etc.);
- validators for checking if a numerical value lies in a given range (less than, greater than, between, etc.);
- validators working as "proxies" to other validators (`ValidatorChain`, `StaticValidator` and `Callback`).

[1] Here, we only consider the standard validator classes belonging to the `Zend\Validator` namespace. But, actually there are more validators that can be considered as standard. We will cover them in further chapters.

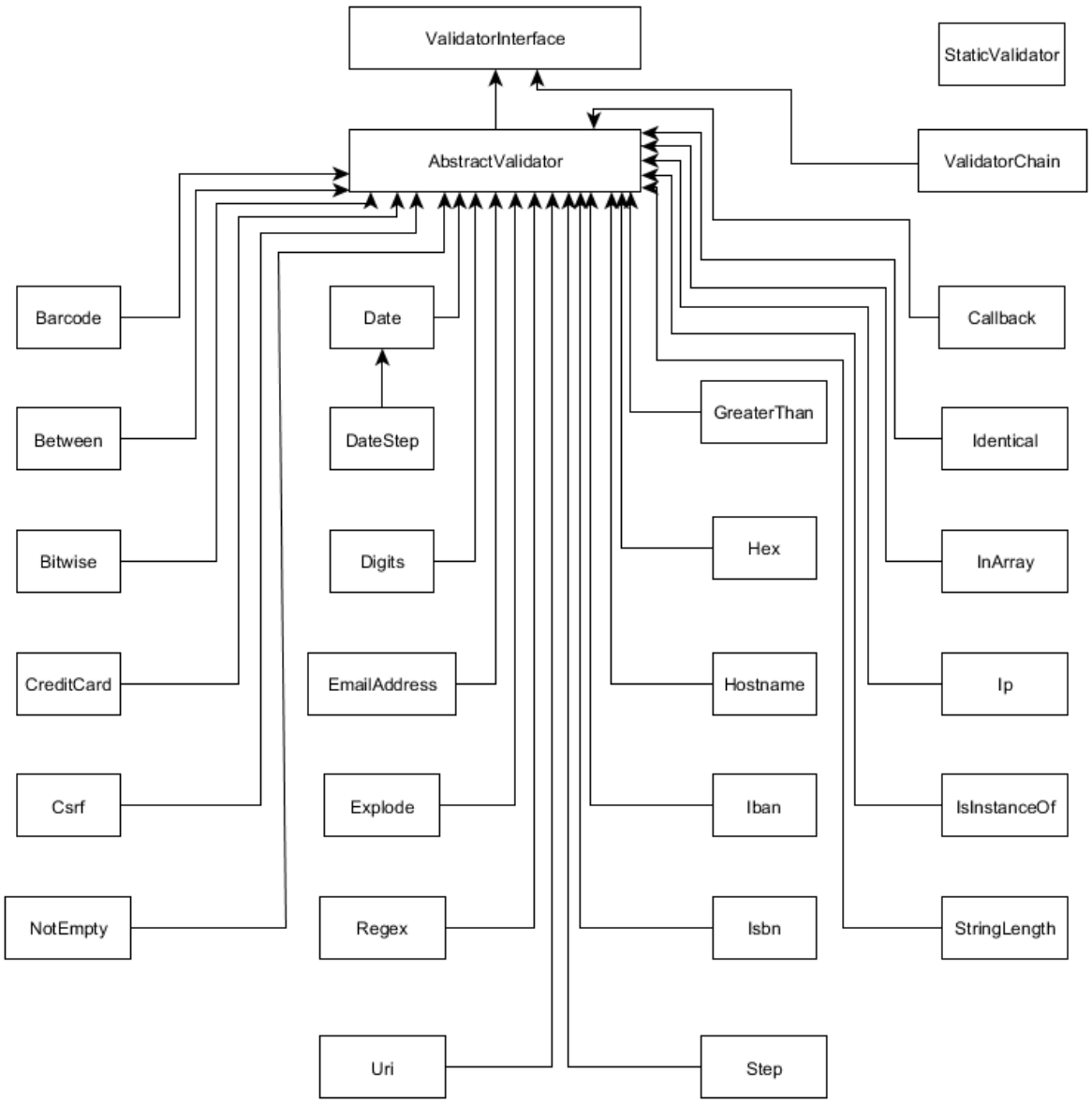


Figure 9.1. Validator class inheritance

Table 9.1. Standard validators

Class name	*Description*
`EmailAddress`	Returns boolean `true` if the value is a valid E-mail address; otherwise returns `false`.
`Hostname`	Checks whether the value is a valid host name.
`Barcode`	Returns boolean `true` if and only if the value contains a valid barcode.
`CreditCard`	Returns `true` if and only if the value follows the common format of credit card number (Luhn algorithm, mod-10 checksum).
`Iban`	Returns `true` if the value is a valid International Bank Account Number (IBAN); otherwise returns `false`.

Table 9.1. Standard validators

Class name	*Description*
`Isbn`	Returns boolean `true` if and only if value is a valid International Standard Book Number (ISBN).
`Ip`	Returns `true` if value is a valid IP address; otherwise returns `false`.
`Uri`	Returns `true` if and only if the value is an Uniform Resource Identifier (URI).
`Between`	Returns `true` if the value lies in certain range; otherwise returns `false`.
`LessThan`	Returns boolean `true` if the value is less than certain number; otherwise returns `false`.
`GreaterThan`	Returns `true` if and only if value is greater than certain number.
`Identical`	Returns boolean `true` if a the value matches a given token.
`Step`	Checks whether the value is a scalar and a valid step value.
`Csrf`	This validator checks if the provided token matches the one previously generated and stored in a PHP session.
`Date`	Returns `true` if value is a valid date of the certain format.
`DateStep`	Returns boolean `true` if a date is within a valid step.
`InArray`	Returns `true` if value is contained in the given array; otherwise returns false.
`Digits`	Returns boolean `true` if and only if $value only contains digit characters.
`Hex`	Returns `true` if and only if value contains only hexadecimal digit characters.
`IsInstanceOf`	Returns `true` if value is instance of certain class; otherwise returns `false`.
`NotEmpty`	Returns `true` if value is not an empty value.
`Regex`	Returns `true` if value matches against given pattern; otherwise returns `false`.
`StringLength`	Returns `true` if the string length lies within given range.
`Explode`	Splits the given value in parts and returns `true` if all parts pass the given check.
`StaticValidator`	This validator allows to execute another validator without explicitly instantiating it.
`Callback`	This validator allows to execute a custom validation algorithm through the user-provided callback function.
`ValidatorChain`	Wrapper validator allowing to organize several validators in a chain. Attached validators are run in the order in which they were added to the chain (FIFO).

9.3 Validator Behaviour in Case of Invalid or Unacceptable Data

If you pass a validator some data that doesn't pass the check, the validator internally creates the list of error messages that can be retrieved with the `getMessages()` method. For example, look

below for possible validation errors that the `EmailValidator` returns if you pass it the "abc@ewr" value (the back-slash (‘’) character indicates line breaks where code doesn't fit book page):

```
array(3) {
  ["emailAddressInvalidHostname"] =>
    string(51) "'ewr' is not a valid hostname for \
the email address"
  ["hostnameInvalidHostname"] =>
    string(66) "The input does not match the expec\
ted structure for a DNS hostname"
  ["hostnameLocalNameNotAllowed"] =>
    string(84) "The input appears to be a local ne\
twork name but local network names are not allowed"
}
```

Validator's `getMessages()` method will return an array of messages that explain why the validation failed. The array keys are validation failure message identifiers, and the array values are the corresponding human-readable message strings.

If `isValid()` method was never called or if the most recent `isValid()` call returned `true`, then the `getMessages()` method returns an empty array. Also, when you call `isValid()` several times, the previous validation messages are cleared, so you see only validation errors from the last call.

Some validators may work with input data in certain format only (for example, a validator may require that the input data be a string, but not an array). If you pass it data in unacceptable format, the validator may throw an `Zend\Validator\Exception\RuntimeException` exception or raise a PHP warning.

It is recommended to check certain validator's documentation to be aware of its actual behaviour in case of unacceptable data.

9.4 Instantiating a Validator

In Zend Framework 2, there are several methods of creating a validator:

- instantiating it manually (with the `new` operator);
- creating it with a factory class (by passing an array configuration); this way is used the most frequently when adding validation rules in a form;
- instantiating it implicitly with the `StaticValidator` wrapper class.

Next, we will cover these three methods in more details.

9.4.1 Method 1. Manual Instantiation of a Validator

A validator in general can be used not only with forms, but also for validation of an arbitrary data. In order to do that, you simply create an instance of the validator class, configure the validator by using the methods it provides, and call the `isValid()` method on the validator.

For example, let's consider the usage of the `EmailAddress` validator which checks an E-mail address for conformance to RFC-2822[2] standard. An E-mail address typically consists of the local part (user name) followed by the "at" character (@), which is in turn followed by the host name. For example, in the "name@example.com" E-mail address, "name" is the local part, and "example.com" is the host name.

The `EmailAddress` validator is useful for checking an user-entered E-mail addresses on your forms for correctness. The validator will check for the correctness of the local part and the host name, for presence of the "at" character (@) and, optionally, will connect to the recipient's host and query the DNS service for existence of the MX (Mail Exchanger) record [3].

The methods provided by the `EmailAddress` validator are listed in table 9.2:

Table 9.2. Public methods of the EmailAddress validator

Method name	*Description*
`__construct($options)`	Constructs the validator. Accepts the list of options allowing to configure it.
`isValid($value)`	Returns `true` if the value is a valid E-mail address according to RFC-2822; otherwise returns `false`.
`getMessages()`	If validation failed, this method will return an array of error messages.
`useDomainCheck($domain)`	Tells the validator to check the host name part for correctness.
`getDomainCheck()`	Returns `true` if host name part check is enabled.
`setHostnameValidator($hostnameValidator)`	Attaches the validator to use for checking host name part of the E-mail address.
`getHostnameValidator()`	Returns the validator used for checking host name part of the E-mail address.
`setAllow($allow)`	Sets the allowed types of host names to be used in an E-mail address.
`getAllow()`	Returns the allowed types of host names.
`useMxCheck($mx)`	Sets whether to perfrom the check for a valid MX record via DNS service.
`getMxCheck($mx)`	Returns `true` if MX check mode is enabled.
`useDeepMxCheck($deep)`	Sets whether to use deep validation for MX records.
`getDeepMxCheck()`	Returns `true` if the deep MX check mode is enabled; otherwise returns `false`.
`isMxSupported()`	Returns `true` if MX checking via `getmxrr()` PHP function is supported in the system; otherwise returns `false`.

[2] https://tools.ietf.org/html/rfc2822

[3] An MX record is a type of record used in the Domain Name System (DNS). MX records define one or several mail server addresses assigned to recipient's domain.

Table 9.2. Public methods of the EmailAddress validator

Method name	*Description*
`getMXRecord()`	After validation, returns the found MX record information.

As you can see from table, the `EmailAddress` validator, additionally to the `isValid()` and `getMessages()` methods, provides the constructor method to which you can (optionally) pass the complete list of options for initializing the validator.

All standard validators have the constructor method (optionally) accepting an array of options for configuring the validator when instantiating it manually.

The `EmailAddress` class also provides a number of methods that can be used for setting specific validator options.

The `useDomainCheck()` method tells whether to check the host name for correctness, or not. By default, this check is enabled. The `setAllow()` method provides an ability to specify which types of host names are allowed. You can pass an OR combination of the `ALLOW_`-prefixed constants [4] to the `setAllow()` method:

- `ALLOW_DNS` Allow a domain name (this is the default),
- `IP_ADDRESS` Allow an IP address,
- `ALLOW_LOCAL` Allow local network name,
- `ALLOW_ALL` Allow all of the above.

Internally, the `EmailAddress` validator uses the `Hostname` validator for checking the host name part of an E-mail address. Optionally, you can attach a custom host name validator by using the `setHostnameValidator()` method, however it is unlikely you will need to do such.

The `useMxCheck()` method tells whether the validator should connect to the recipient's host and query the DNS server for the MX record(s). If the server has no MX records, than the validation fails. You can additionally use the `useDeepMxCheck()` method to tell the validator to compare the mail server addresses extracted from the MX records against the black list of reserved domain names, and perform additional checks per each detected address.

It is not recommended to perform MX check (and deep MX check), because that may take a lot of time and increase the web page load time. By default, these checks are disabled.

Below, we provide code examples showing two equivalent methods of manual creating of an instance of the `EmailAddress` validator, setting its options and checking an input value:

Example 1. Passing options to the constructor method.

[4] The `ALLOW_`-prefixed constants are provided by the `Hostname` validator.

```
1  <?php
2  // Optionally, define a short alias for the validator class name.
3  use Zend\Validator\EmailAddress;
4  use Zend\Validator\Hostname;
5
6  // Create an instance of the validator, passing options to the constructor.
7  $validator = new EmailAddress(array(
8                  'allow' => Hostname::ALLOW_DNS|Hostname::ALLOW_IP|Hostname::ALLOW_LOCAL,
9                  'mxCheck' => true,
10                 'deepMxCheck' => true
11         ));
12
13 // Validate an E-mail address.
14 $isValid = $validator->isValid('name@example.com'); // Returns true.
15 $isValid2 = $validator->isValid('abc'); // Returns false.
16
17 if(!$isValid2) {
18   // Get error messages in case of validation failure.
19   $errors = $validator->getMessages();
20 }
```

In the code above, we create the `EmailAddres` validator object with the help of the `new` operator (line 7). We pass the array of options to the constructor. We use the `allow` key to allow an E-mail address to be a domain name, an IP address or local network address. Also, we use the `mxCheck` and `deepMxCheck` to enable MX record check and deep MX record check, respectively.

In line 14, we call the `isValid()` method and pass it the string value "name@example.com" to be checked. The expected output of this call is the boolean `true`.

In line 15, we pass the "abc" string value to the validator. The validation procedure is expected to fail (`false` is returned). Then, the error messages are retrieved with the `getMessages()` method (line 19).

Example 2. Without passing options to the constructor.

```
1  <?php
2  // Optionally, define a short alias for the validator class name.
3  use Zend\Validator\EmailAddress;
4  use Zend\Validator\Hostname;
5
6  // Create an instance of the validator.
7  $validator = new EmailAddress();
8
9  // Optionally, configure the validator
10 $validator->setAllow(
11        Hostname::ALLOW_DNS|Hostname::ALLOW_IP|Hostname::ALLOW_LOCAL);
12 $validator->useMxCheck(true);
```

```
13 $validator->useDeepMxCheck(true);
14
15 // Validate an E-mail address.
16 $isValid = $validator->isValid('name@example.com'); // Returns true.
17 $isValid2 = $validator->isValid('abc'); // Returns false.
18
19 if(!$isValid2) {
20   // Get error messages in case of validation failure.
21   $errors = $validator->getMessages();
22 }
```

In the code above, we create the `EmailAddres` validator object with the help of the `new` operator (line 7).

In lines 10-13, we configure the validator. We call the `setAllow()` method to allow an E-mail address to be a domain name, an IP address or local network address. Also, we use the `useMxCheck()` and `useDeepMxCheck()` to enable MX record check and deep MX record check, respectively.

In line 16, we call the `isValid()` method and pass it the string value "name@example.com" to be checked. The expected output of this call is the boolean `true`.

In line 17, we pass the "abc" string value to the validator. The validation procedure is expected to fail. Then, the error messages are retrieved with the `getMessages()` method (line 21).

The rest of this chapter is skipped in this free sample.

10. Uploading Files with Forms

In this chapter, you will learn about uploading files with forms. First, we will review the basic theory like HTTP file upload capability and binary content transfer encoding, and then provide a complete working Image Gallery example showing how to upload images to a web server.

ZF2 components covered in this chapter:

Component	*Description*
`Zend\Form`	Contains base form model classes.
`Zend\Filter`	Contains various filters classes.
`Zend\Validator`	Implements various validator classes.
`Zend\InputFilter`	Implements a container for filters/validators.

10.1 About HTTP File Uploads

HTML forms have capability for uploading files of arbitrarily large size [1]. The files are typically transmitted through HTTP POST method [2].

By default, HTTP uses the *URL encoding* for transfers of form data, and you could see how that encoding looks like when reading the *GET and POST Methods* section of the previous chapter. However, this is inefficient for uploading large files, since URL-encoding binary data dramatically increases the length of the HTTP request. For the purpose of uploading files, it is instead recommended to use the so called "binary transfer encoding" described in the next section.

10.1.1 HTTP Binary Transfer Encoding

A simple HTML form capable of file uploads is shown in the code example below. The binary encoding type is enabled by setting the `enctype` attribute of the form with the value of "multipart/form-data":

[1] HTTP file uploads are described in RFC-1867. This mechanism allows to upload large files by using binary content transfer encoding. The "multipart/form-data" encoding type is utilized for this purpose.

[2] The HTTP GET method is inefficient for file uploads, because URL length has some upper limit. Also, URL-encoding file data greatly increases the URL length.

```
1  <form action="upload" method="POST" enctype="multipart/form-data">
2    <input type="file" name="myfile">
3    <br/>
4    <input type="submit" name="Submit">
5  </form>
```

In line 1, we explicitly set form encoding (`enctype` attribute) to "multipart/form-data" to utilize effective binary content transfer encoding for the form.

In line 2, we define an input field with type "file" and name "myfile". This input field will allow site visitor to select the file for upload.

If you now save the above mentioned markup to an *.html* file and open it in your web browser, you will see the page like in figure 10.1.

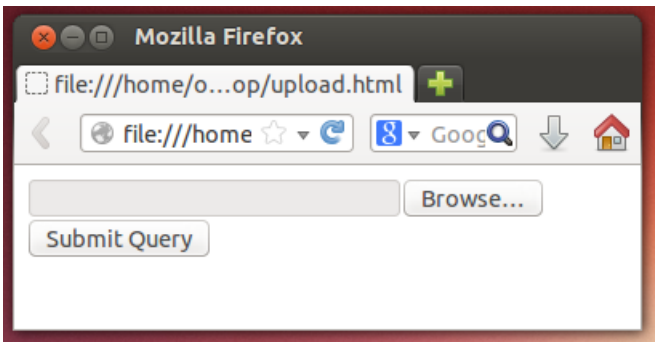


Figure 10.1. A simple HTML form capable of file upload

The file element has the *Browse...* button allowing to pick a file for upload. When the site user picks some file and clicks the *Submit* button on the form, the web browser will send an HTTP request to the web server, and the request will contain the data of the file being uploaded. The example below illustrates how the HTTP request may look like:

```
1  POST http://localhost/upload HTTP/1.1
2  Host: localhost
3  Content-Length: 488
4  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64)
5  Content-Type: multipart/form-data; boundary=----j1bOrwgLvOC3dy7o
6  Accept-Encoding: gzip,deflate,sdch
7
8  ------j1bOrwgLvOC3dy7o
9  Content-Disposition: form-data; name="myfile"; filename="Somefile.txt"
10 Content-Type: text/html
11
12 (file binary data goes here)
13 ------j1bOrwgLvOC3dy7o
14 Content-Disposition: form-data; name="Submit"
15
16 Submit Request
17 ------j1bOrwgLvOC3dy7o--
```

As you can see from the example above, the HTTP request with "multipart/form-data" encoding type looks analogous to a usual HTTP request (has the status line, the headers, and the content area), however it has the following important differences:

- Line 5 sets the "Content-Type" header with "multipart/form-data" value; The form is assembled of the fields marked by the "boundary" – a unique randomly generated sequence of characters delimiting form fields of each other.
- Lines 8-17 represent the content of the HTTP request. The form fields are delimited by the "boundary" sequences (lines 8, 13, 17). The data of the file being uploaded are transmitted in binary format (line 12), and that allows to reduce the content size to its minimum.

By default, PHP engine's settings do not allow to upload large files (larger than 2MB). In order to upload large files, you may need to edit the *php.ini* configuration file and modify the `post_max_size` and `upload_max_filesize` parameters (please refer to Appendix A for information on how to do that). Setting these with `100M` allows to upload files up to 100 Mb in size, and this would typically be sufficient. If you plan to upload very large files up to 1 GB in size, than better set these with 1024M. Do not forget to restart your Apache Web Server after editing the configuration file.

The rest of this chapter is skipped in this free sample.

11. Advanced Usage of Forms

In previous chapters, you've learned about form usage basics: what HTML forms are and how you define form models and form presentation in Zend Framework 2. In this chapter, you will learn some advanced form usage topics such as security form elements (CAPTCHA and CSRF), and so on.

ZF2 components covered in this chapter:

Component	*Description*
`Zend\Captcha`	Implements various CAPTCHA algorithms.
`Zend\Form`	Contains base form model classes.
`Zend\Filter`	Contains various filters classes.
`Zend\Validator`	Implements various validator classes.
`Zend\InputFilter`	Implements a container for filters/validators.

11.1 Form Security Elements

We will consider the usage of two form security elements provided by Zend Framework 2: `Captcha` and `Csrf` (both classes belong to `Zend\Form\Element` namespace). By adding those elements to your form model (and rendering them in a view template), you will make your form resistant to hacker attacks.

11.1.1 CAPTCHA

A CAPTCHA (stands for "Completely Automated Public Turing test to tell Computers and Humans Apart") is a challenge-response test used in web sites for determining whether the user is a human or a robot.

There are several types of CAPTCHA. The most widely used one requires that the user type the letters of a distorted image that is shown on the web page (see figure 11.1 for some examples).

a. b. c.

Figure 11.1. CAPTCHA examples

A typical CAPTCHA test works using the following algorithm:

1. Some secret sequence of characters (word) is generated server-side.
2. The secret word is saved in a PHP session variable.
3. The distorted image is generated based on the secret word. The image is then displayed on the web page to site user.
4. The site user is asked to type characters shown on the image.
5. If the characters typed by user are the same as the secret word saved in the session, the test is considered passed.

The goal of the CAPTCHA test is to protect your form from filling and submission by an automated process (so called robot). Usually, such robots send spam messages to forums, hack passwords on site login forms, or perform some other malicious actions.

The CAPTCHA test allows to reliably distinguish humans from robots, because humans are easily able to recognise and reproduce characters from the distorted image, while robots are not (at the current stage of evolution of computer vision algorithms).

11.1.1.1 CAPTCHA Types

In Zend Framework 2, there are several CAPTCHA types available (they all belong to the `Zend\Captcha` component):

- *Dumb.* This is a very simple CAPTCHA algorithm which requires that site user enter the word letters in reverse order. We will not consider this type in details here, because it provides too low protection level.
- *Image.* A CAPTCHA algorithm distorting an image with addition of some noise in form of dots and line curves (figure 11.1, a).
- *ReCaptcha.* An adapter providing the access to reCAPTCHA service (figure 11.1, c). The reCAPTCHA[1] is a free service that is provided by Google for generating distorted images and using them for CAPTCHA test.
- *Figlet.* An unusual CAPTCHA type using FIGlet program instead of an image distortion algorithm. The FIGlet is an open-source program which generates the CAPTCHA image of many small ASCII letters (figure 11.1, b).

The `Zend\Captcha` component provides a unified interface for all CAPTCHA types (the `AdapterInterface` interface). The `AbstractAdapter` base class implements that interface, and all other CAPTCHA algorithms are derived from the abstract adapter class [2]. The class inheritance diagram is shown in figure 11.2 below.

As you can see from the figure 11.2, there is another base class for all CAPTCHA types that utilize some secret word of characters: the `AbastractWord` class. This base class provides methods for generating random sequence of characters and for adjusting word generation options.

[1] http://recaptcha.net

[2] The *adapter* is a design pattern that translates one interface for a class into a compatible interface, which helps two (or several) incompatible interfaces to work together. Typically, CAPTCHA algorithms have different public methods, but since they all implement `AbstractAdapter` interface, the caller may use any CAPTCHA algorithm in the same common manner (by calling the methods provided by the base interface).

11.1.1.2 CAPTCHA Form Element & View Helper

ZF2 provides the dedicated form element class and view helper class for letting you use CAPTCHA fields on your forms.

To add a CAPTCHA field to a form model, you use the `Captcha` class that belongs to `Zend\Form` component and lives in `Zend\Form\Element` namespace.

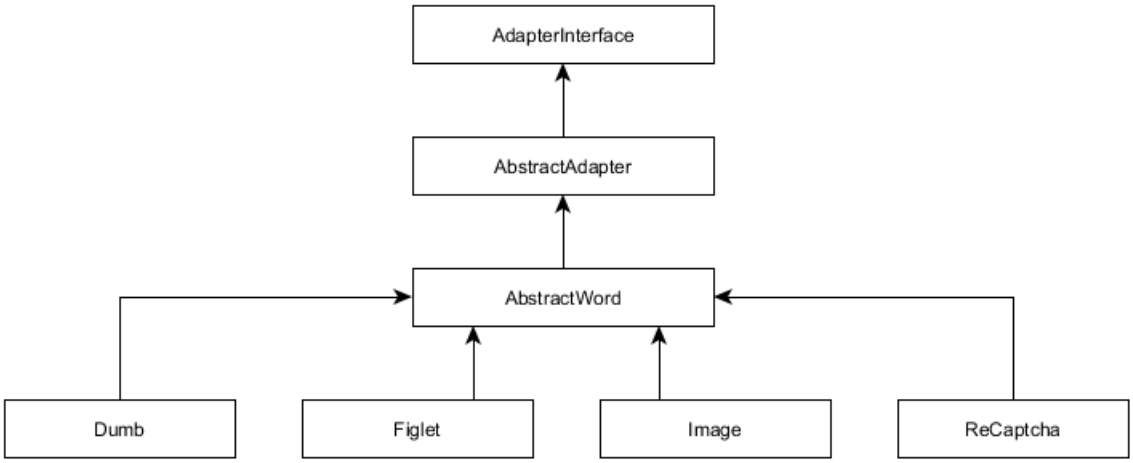


Figure 11.2. CAPTCHA adapter classes

The `Captcha` element class can be used with any CAPTCHA algorithm (listed in the previous section) from `Zend\Captcha` component. For this purpose, the element class has the `setCaptcha()` method which takes either an instance of a class implementing `Zend\Captcha\AdapterInterface` interface, or an array containing CAPTCHA configuration [3]. By the `setCaptcha()` method, you can attach the desired CAPTCHA type to the element.

You add the `Captcha` element to a form model as usual, with the `add()` method provided by the `Zend\Form\Form` base class. As usual, you can pass it either an instance of the `Zend\Form\Element\Captcha` class or provide an array of configuration options specific to certain CAPTCHA algorithm (in that case, the element and its associated CAPTCHA algorithm will automatically be instantiated and configured by the factory class).

The code example below shows how to use the latter method (passing a configuration array). We prefer this method because it requires less code to write. It is assumed that you call this code inside of form model's `addElements()` protected method:

[3] In the latter case (configuration array), the CAPTCHA algorithm will be automatically instantiated and initialized by the factory class `Zend\Captcha\Factory`.

```
1  <?php
2  // Add the CAPTCHA field to the form model
3  $this->add(array(
4    'type'  => 'captcha',
5    'name' => 'captcha',
6    'options' => array(
7      'label' => 'Human check',
8      'captcha' => array(
9        'class' => '<captcha_class_name>', //
10       // Certain-class-specific options follow here ...
11     ),
12   ),
13 ));
```

In the example above, we call the `add()` method provided by the `Form` base class and pass it an array describing the element to insert (line 3):

- The `type` key of the array (line 4), as usual, may either be a full name of the element (`Zend\Form\Element\Captcha`) or its short alias ("captcha").
- The `name` key (line 5) is the value for the "name" attribute of the HTML form field.
- The `options` key contains the options for the attached CAPTCHA algorithm. The `class` key (line 9) may either contain the full CAPTCHA class name (e.g. `Zend\Captcha\Image`) or its short alias (e.g. "Image"). Other, adapter-specific, options may be added to the key as well. We will show how to do that a little bit later.

For generating the HTML markup for the element, you may use the `FormCaptcha` view helper class (belonging to `Zend\Form\View\Helper` namespace). But, as you might learn from the previous chapter, typically you use the generic `FormElement` view helper instead, like shown in the code below:

```
<?php echo $this->formElement($form->get('captcha')); ?>
```

It is assumed that you call the view helper inside of your view template.

Next, we provide three examples illustrating how to use different CAPTCHA types provided by ZF2: the `Image`, `Figlet` and `ReCaptcha`. We will show how to add a CAPTCHA field to the feedback form that we used in examples of the previous chapter.

The rest of this chapter is skipped in this free sample.

12. Database Management with Doctrine ORM

Doctrine is an open-source PHP library providing convenient methods for managing your database in object-oriented way. For working with relational databases, Doctrine provides a component named *Object Relational Mapper* (shortly, ORM). With Doctrine ORM you map your database table to a PHP class (in terms of Domain Driven Design, it is also called an *entity* class) and a row from that table is mapped to an instance of the entity class. If you are new to Doctrine, it is recommended that you also refer to Appendix D for introductory information about the Doctrine library architecture.

Doctrine is a third-party library, it is not part of Zend Framework 2. We cover it in this book because it provides an easy way of adding database support to your ZF2-based web application.

12.1 Get Blog Example from GitHub

For demonstration of Doctrine ORM usage, in this chapter, we will create a real-life *Blog* web site that does the following:

- Stores blog posts in a database and provides user interface for accessing and managing those posts.
- It is assumed that the blog has the single author of its posts, while comments can be added by multiple blog readers.
- The web site has two pages: *Home* page and *Admin* page. The first one displays the list of recently added posts, while the latter one allows to add, edit, view and delete posts.

For example screen shots of the *Blog* web site, please look at the figures 12.1 and 12.2 below:

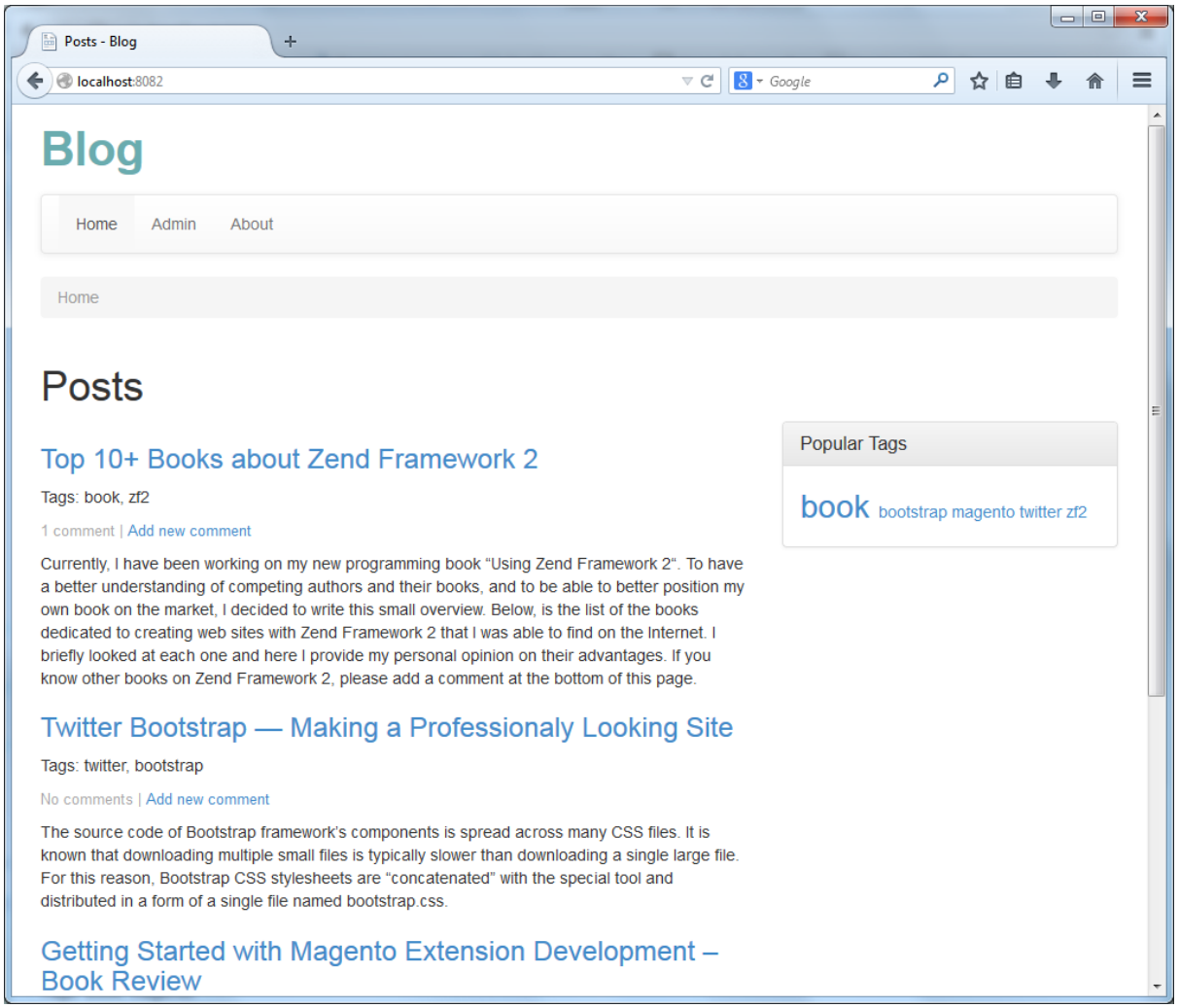


Figure 12.1. Blog home page

To download the *Blog* application, visit this page[1] and click the *Download ZIP* button to download the code as a ZIP archive. When download is complete, unpack the archive to some directory.

Then navigate to the `blog` directory containing the source code of the *Blog* web application:

```
/using-zend-framework-2-book
  /blog
  ...
```

The *Blog* is a sample web site which can be installed on your machine. To install the example, you can either edit your default Apache virtual host file or create a new one. After editing the file, restart the Apache HTTP Server and open the web site in your web browser.

For the *Blog* example to work, you have to create a MySQL database. Instructions on how to do that are provided in the next section.

[1] https://github.com/olegkrivtsov/using-zend-framework-2-book

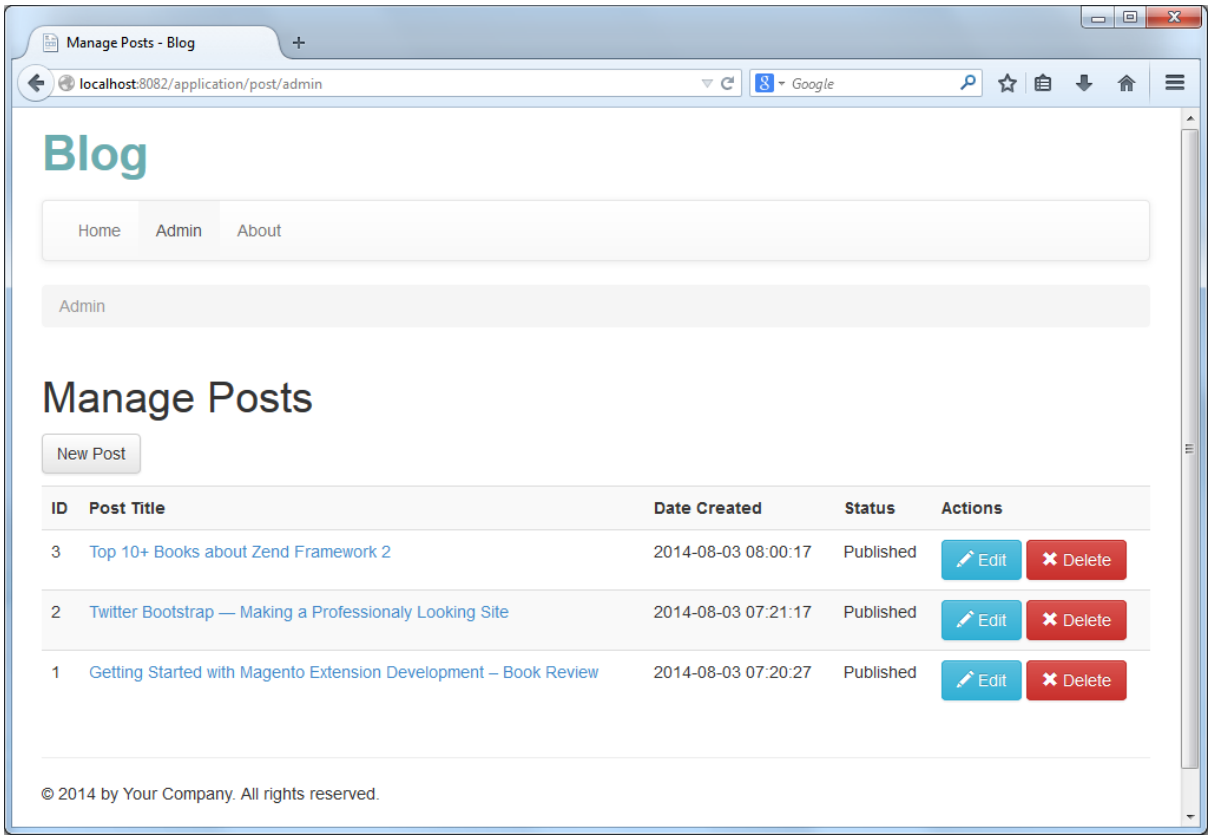


Figure 12.2. Blog admin page

12.2 Creating a Simple MySQL Database

For the *Blog* example to work, we need to have a database. In this book, we use MySQL database management system, which is very simple in installation and administration.

For OS-specific instructions on how to install MySQL server and client, please refer to Appendix A.

Once you install MySQL, type the following command from your command shell to log into MySQL client console:

```
mysql -u root -p
```

When asked for, type the password of the *root* user (the password of the *root* user is the one you've specified during MySQL server installation). On successful login, you should see the following welcome message:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1768
Server version: 5.5.37-0ubuntu0.12.04.1 (Ubuntu)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Now you are able to type MySQL client commands (like `show databases`, `show tables`, etc.) or SQL queries (like `SELECT` or `INSERT`) in the MySQL prompt and see their output.

If you want to quit of the MySQL prompt, type `quit` and press Enter.

12.2.1 Creating New Schema

Let's create a database schema and name it `blog`. To do that, type the following MySQL statement and press Enter:

```
CREATE SCHEMA blog;
```

The expected output of this command is the following:

```
Query OK, 1 row affected (0.01 sec)
```

MySQL commands are case insensitive, so you could type `create schema blog;` with the same result. We recommend using upper case for SQL queries, since this is a common convention.

Next, we create the user named `blog` and grant it all privileges for accessing and modifying the `blog` database and all its tables:

```
GRANT ALL PRIVILEGES ON blog.* TO `blog`@`localhost` IDENTIFIED BY '<passwd>';
```

In the command above, replace the password placeholder with the new password for the `blog` user. This password should be different than the password of the *root* user.

Here, we create the second user `blog`, because it is not recommended to give the web application to log into database under the `root` user. The *root* user has unlimited rights and it would be just insecure to give the application an ability to do any actions it wants. The `blog` user will have permissions to modify the `blog` database only, which is sufficient in our case.

You can check that the database has been created by typing the following command and pressing Enter:

```
show databases;
```

You should be able to see the output like below (note the `blog` line in the list of databases):

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| blog               |
| mysql              |
| performance_schema |
+--------------------+
```

12.2.2 Creating Tables

Next, we will create three tables typical for any simple blog: the `post` table will contain posts, the `comment` table will contain comments to posts, and, finally, the `tag` table will contain tags (a tag is some kind of a key word describing a blog post well).

Additionally, we will create the fourth auxiliary table `post_tag` that will be used to create many-to-many relation between the `post` and the `tag` tables.

Make the `blog` database current by typing the following from MySQL command prompt:

```
use blog;
```

To create the `post` table, type the following MySQL statement:

```
CREATE TABLE `post` (
  `id` int(11) PRIMARY KEY AUTO_INCREMENT,
  `title` text NOT NULL,
  `content` text NOT NULL,
  `status` int(11) NOT NULL,
  `date_created` timestamp NOT NULL
);
```

MySQL client allows to enter multi-line commands easily. Just press Enter when you want to move the caret to the next line. The command is considered to be fully entered when the semicolon (;) character is encountered.

The expected output of this command is the following:

```
Query OK, 0 rows affected (0.22 sec)
```

Next, create the `comment` table by typing the following:

```
CREATE TABLE `comment` (
  `id` int(11) PRIMARY KEY AUTO_INCREMENT,
  `content` text NOT NULL,
  `author` varchar(128) NOT NULL,
  `date_created` timestamp NOT NULL
);
```

Then, create the `tag` table:

```
CREATE TABLE `tag` (
  `id` int(11) PRIMARY KEY AUTO_INCREMENT,
  `name` VARCHAR(128)
);
```

And finally, create the `post_tag` table:

```
CREATE TABLE `post_tag` (
  `id` int(11) PRIMARY KEY AUTO_INCREMENT,
  `post_id` int(11) NOT NULL,
  `tag_id` int(11) NOT NULL
);
```

Let's fill the tables we have created with some sample data:

```
INSERT INTO tag(`name`) VALUES('zf2');
INSERT INTO tag(`name`) VALUES('book');
INSERT INTO tag(`name`) VALUES('magento');

INSERT INTO post(`title`, `content`, `status`, `date_created`) VALUES(
   'Top 10+ Books about Zend Framework 2',
   'Post content', 2, '2014-08-09 18:49');

INSERT INTO post(`title`, `content`, `status`, `date_created`) VALUES(
   'Getting Started with Magento Extension Development  Book Review',
   'Post content 2', 2, '2014-08-09 18:51');

INSERT INTO post_tag(`post_id`, `tag_id`) VALUES(1, 1);
INSERT INTO post_tag(`post_id`, `tag_id`) VALUES(1, 2);
INSERT INTO post_tag(`post_id`, `tag_id`) VALUES(2, 2);
INSERT INTO post_tag(`post_id`, `tag_id`) VALUES(2, 3);
```

```
INSERT INTO comment(`post_id`, `content`, `author`, `date_created`) VALUES(
    1, 'Excellent post!', 'Oleg Krivtsov', '2014-08-09 19:20');
```

If necessary, you can easily remove the schema and all tables and data it contains by typing the following command from MySQL prompt:

```
DROP SCHEMA blog;
```

Figure 12.3 graphically illustrates what entities we have in the schema and what relations between those entities present.

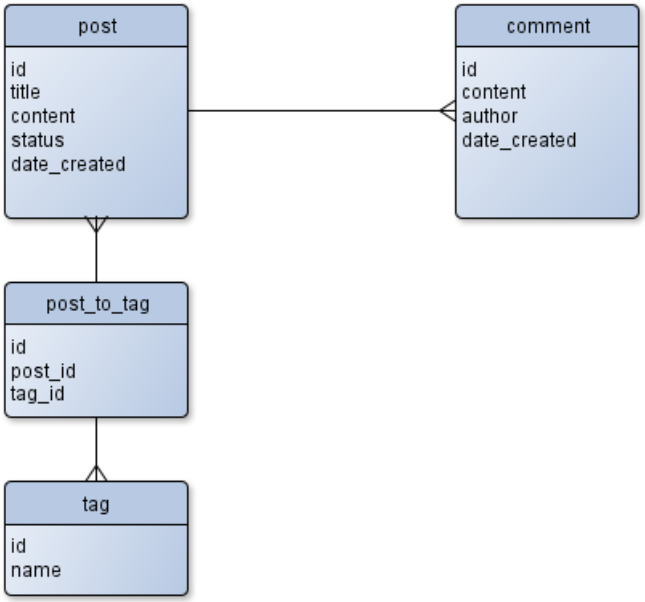


Figure 12.3. Graphical representation of database schema

As you can see from figure 12.3, the `post` table is related to `comment` table as *one-to-many*, because a single post may have many comments. This is also called the "one-to-many" relation.

The `post` table is also related to the `tag` table as *many-to-many*. A single post may have many tags, and a single tag may belong to many posts as well. Many-to-many relation is typically implemented through an auxiliary table (`post_tag` table in our case).

12.2.3 Importing Ready Database Schema

In the previous section, we've shown how to create the complete database schema that is used in the *Blog* sample web application. In the real life, you typically do not type all those SQL statements in MySQL prompt. Instead, you could type the `CREATE TABLE` statements to a file and save it to disk. Then you could just import that file and have ready schema.

For your convenience, the ready schema for *Blog* sample can be found in *APP_DIR/data/schema.mysql.sql* file. The file is a plain text file containing SQL statements. To import the file, go to the

APP_DIR/data/ directory and type the following command from your command shell (but not from MySQL prompt):

```
mysql -uroot -p blog < schema.mysql.sql
```

When prompted for password, enter the password of the root user and type Enter.

Once this is done, log into MySQL client and type the following commands:

```
use blog;

show tables;
```

You should see the list of tables created, something like below:

```
+----------------+
| Tables_in_blog |
+----------------+
| comment        |
| post           |
| post_tag       |
| tag            |
+----------------+
4 rows in set (0.00 sec)
```

12.3 Integrating Doctrine ORM with Zend Framework 2

For easy integration with Zend Framework 2, Doctrine project provides the following two components (that are actually ZF2 modules):

- DoctrineModule[2] is a ZF2 module that provides Doctrine basic functionality required by the ORM component;
- DoctrineORMModule[3] integrates Doctrine 2 Object Relational Mapper with Zend Framework 2.

Each of the above Doctrine components is distributed as a Composer-installable package and is registered in Packagist.org[4] catalogue. This is very similar to the way that Zend Framework 2 uses for installing its components.

Since Composer packages may depend on each other, it is enough to declare dependency only on *DoctrineORMModule*. This package depends on *DoctrineModule* and on some other Doctrine components (*Doctrine\ORM*, *Doctrine\DBAL*, *Doctrine\Common*, *Doctrine\Annotations*, etc.). So, when you install this component, Composer will install other required components automatically.

[2] https://github.com/doctrine/DoctrineORMModule
[3] https://github.com/doctrine/DoctrineORMModule
[4] https://packagist.org/

12.3.1 Installing Doctrine Components with Composer

In order to install required Doctrine components, we first *add a dependency* to the *composer.json* file located in the root directory of the web application (in this book, we typically denote that directory as *APP_DIR*).

To add the dependency, type the following commands from your command shell (replace the *APP_DIR* placeholder with the actual directory name of your application):

```
cd APP_DIR
```

```
php composer.phar require doctrine/doctrine-orm-module *
```

The `cd` command above is used to make the *APP_DIR* directory current working directory.

And the `require` command tells Composer to add the package `doctrine/doctrine-orm-module` as a dependency to your web application, and to download and install that dependency. The asterisk (*) parameter means that any version of the package is acceptable.

Specifying the asterisk as a version, will result in installing the latest available version of Doctrine, which typically is the desired behavior.

Once you run the commands above, Composer will first modify the *composer.json* file and create the following line under its `require` key:

```
{
  ...
  "require": {
    "doctrine/doctrine-orm-module": "*",
    ...
  },
  ...
}
```

Then Composer will try to locate the dependency packages, download them to the local machine and install the files into the *APP_DIR/vendor* directory.

Composer will output lines indicating installation process to the terminal:

```
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
  - Installing doctrine/lexer (v1.0)
    Downloading: 100%

  - Installing doctrine/annotations (v1.1.2)
    Downloading: 100%
```

```
  - Installing doctrine/collections (v1.2)
    Downloading: 100%

  - Installing doctrine/cache (v1.3.0)
    Downloading: 100%

  - Installing doctrine/inflector (v1.0)
    Downloading: 100%

  - Installing doctrine/common (v2.4.2)
    Downloading: 100%

  - Installing doctrine/dbal (v2.4.2)
    Downloading: 100%

  - Installing symfony/console (v2.5.0)
    Downloading: 100%

  - Installing doctrine/orm (v2.4.2)
    Downloading: 100%

  - Installing doctrine/doctrine-module (0.8.0)
    Downloading: 100%

  - Installing doctrine/doctrine-orm-module (0.8.0)
    Downloading: 100%

symfony/console suggests installing symfony/event-dispatcher ()
symfony/console suggests installing psr/log (For using the console logger)
doctrine/orm suggests installing symfony/yaml (If you want to use YAML
Metadata Mapping Driver)
doctrine/doctrine-module suggests installing doctrine/data-fixtures (Data
Fixtures if you want to generate test data or bootstrap data for your
deployments)
doctrine/doctrine-orm-module suggests installing zendframework/zend-developer-
tools (zend-developer-tools if you want to profile operations executed by the
ORM during development)
doctrine/doctrine-orm-module suggests installing doctrine/migrations
(doctrine migrations if you want to keep your schema definitions versioned)
Writing lock file
Generating autoload files
```

As you can see from the output above, when you install `DoctrineORMModule` component, Composer automatically installs the `DoctrineModule` and all necessary Doctrine components (*Doctrine\DBAL*, *Doctrine\ORM*, etc.)

As a bonus, at the end of installation, Composer "suggests" you to install some additional packages that might be useful for you (`doctrine/migrations`, `doctrine/data-fixtures`, etc.) If you strongly wish, you may add those dependencies with the Composer's `require` command as well.

When the installation has been finished, you can find the Doctrine files in your *APP_DIR/vendor* directory (see the figure 12.4 below).

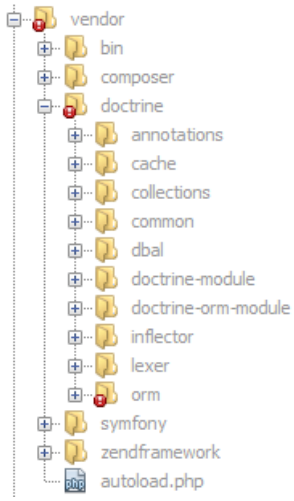


Figure 12.4. Doctrine files are installed to vendor directory

You use the `php composer.phar require` command for the first time you install Doctrine. Once the *composer.json* (and *composer.lock*) files have been modified by Composer, you are able to install (or update) all dependencies as usual by typing the `php composer.phar install` or `php composer.phar update` commands, respectively, from your command shell.

The rest of this chapter is skipped in this free sample.