Learn by doing: less theory, more results

Zend Framework 2.0 by Example

A step-by-step guide to help you build full-scale web applications using Zend Framework 2.0

Beginner's Guide

Krishna Shasankar V

[PACKT] PUBLISHING open source community experience distilled

Zend Framework 2.0 by Example Beginner's Guide

A step-by-step guide to help you build full-scale web applications using Zend Framework 2.0

Krishna Shasankar V

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

Zend Framework 2.0 by Example Beginner's Guide

First published: July 2013

Production Reference: 1180713

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78216-192-9

www.packtpub.com

Cover Image by Abhishek Dhir (abhishekdhirimages@gmail.com)

Credits

Author
Krishna Shasankar V

Reviewers
Wenbert S. Del Rosario
Alex (Shurf) Frenkel
Islam Mohamed Abdel-aziz

Acquisition Editor
Antony Lowe

Lead Technical Editor
Neeshma Ramakrishnan

Technical Editor
Veena Pagare

Project Coordinator
Anugya Khurana

Proofreader
Maria Gould

Indexer
Priya Subramani

Graphics
Abhinash Sahu

Production Coordinator
Arvindkumar Gupta

Cover Work
Arvindkumar Gupta

About the Author

Krishna Shasankar V is a web developer with 7 years of extensive development experience in PHP. He leads a team of engineers at Lister Technologies developing enterprise class retail and e-commerce solutions.

He is a Zend Certified Engineer in PHP 5 and Zend Framework. He also has a Bachelor's degree in Information Technology from Anna University, Chennai, and a Master's degree in Software Systems from Birla Institute of Technology and Science, Pilani.

In his spare time, he enjoys music, photography, and travel (especially when combined). You can contact Krishna and leave some comments on his blog (`www.clickoffline.com`).

I would like to thank my parents, my brother, and all my friends who encouraged and supported me throughout my life.

Thanks to Mukund Deverajan for his full and enthusiastic support, without which, this book would not have existed. Thanks to Apoorv Bhargava, Jayabharathi and Souvik Sengupta for motivating me and helping me rework a majority of the book's content. Special thanks to my amazing team at Lister Technologies for their wonderful support and all the fun. You guys are awesome!

Thanks to the reviewers Wenbert S. Del Rosario, Alex (Shurf) Frenkel, and Islam Mohamed Abdel-aziz for providing me with valuable feedback during the review stages.

Finally, the awesome team, Antony Lowe, Neeshma Ramakrishnan, Veena Pagare, and everyone else at Packt Publishing who had contributed to this book, ensuring quality at each level. I am indebted to Anugya Khurana at Packt Publishing, without her patience and persistence, this book would have stalled many times. Special thanks to Veena Manjrekar for giving me this opportunity, for which I am grateful.

About the Reviewers

Wenbert Del Rosario is a web developer with a couple of years of experience working with open source technologies (Linux, CakePHP, Code Igniter, MySQL, jQuery, Knockout JS, and WordPress). In his free time, he loves to work on personal projects. He also does some freelance and consulting work.

Wenbert has also reviewed a couple of books for Packt Publishing:

- *Zend Framework 1.8 Web Application Development Keith Pope*
- *CouchDB and PHP Web Development Beginner's Guide*, *Tim Juravich*

He shares his ideas, solutions, and day-to-day encounters at work through his blog at `http://blog.ekini.net`. You can also follow him on Twitter `@wenbert`.

For Noeme and our baby Lucas.

Alex Frenkel has been working in the field of web application development since 1998 (the beginning of PHP 3.X) and has extensive experience in system analysis and project management. Alex is a PHP 5.3 Zend Certified Engineer and is considered to be one of the most prominent LAMP developers in Israel.

In the past, Alex was the CTO of ReutNet, one of the leading Israeli web technology based companies, and also worked as the CEO/CTO of OpenIview LTD., a company built around an innovative idea of breaching IBM Mainframe business with PHP applications. He also provided expert consulting services to different companies in various aspects of web-related technology.

Alex is a CTO of a startup called GBooking and the owner of a small consulting company, Frenkel-Online.

GBooking allows consumers to search, compare, and book a wide range of services on the Web, while optimizing prices according to the demand, creating discounts during the weak hours of businesses and propagating them to partners' sites.

Frenkel-Online is a project-based company, working with a number of professional freelance consultants in Israel and abroad. Currently their permanent staff comprises of several consultants in Israel and abroad for the company's PHP projects, and an altering number of specialists in other programming languages for the rest of the projects.

Islam Abdel-Aziz is a senior open source software engineer, and Zend Framework contributor. He has been a Zend Certified Engineer since 2009.

Islam spent 9 years teaching and consulting on the latest web and enterprise technologies.

He is involved in development techniques, including the NO-SQL databases, the scalability of the web, parallel/distributed processing using map/reduce model.

He has contributed to many open source projects in the last 7 years, and he has experience in most open source technologies including PHP5, Python, and Java.

Islam joined Oracle in 2008 as a senior software engineer. He was one of the team for developing the most stable cloud-computing platform in Python.

Islam currently holds the title of Arabic Team Lead in the ADVFN, the most popular financial software company in UK. He is the one who is responsible for the engineering of ME versions of the ADVFN products.

I would like to thank my wife for standing by me while I reviewed this book.

---

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

PACKTLiB™

http://PacktLib.PacktPub.com

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

To my parents Paramajothi and Anuragalatha whose unconditional love and sacrifice resulted in where I stand today.

Table of Contents

Preface 1

Chapter 1: Getting Started with Zend Framework 2.0 7

Zend Framework 2.0 7

Introduction to Zend Server Community Edition (CE) 8

Zend Server CE – system requirements 8

Time for action – installing Zend Server CE 8

Configuring Zend Server CE 11

Zend Server CE – Administration Interface 11

Time for action – configuring Zend Server CE 12

MySQL 14

Time for action – installing MySQL 15

phpMyAdmin 16

Time for action – creating a database 16

Summary 20

Chapter 2: Building Your First Zend Framework Application 21

Prerequisites 21

ZendSkeletonApplication 22

Time for action – creating a Zend Framework project 22

Zend Framework 2.0 – modules 27

Zend Framework 2.0 – project folder structure 28

Time for action – creating a module 29

MVC layer 30

Zend Framework module – folder structure 31

Time for action – creating controllers and views 31

Zend Framework module – configuration 33

Time for action – modifying module configuration 34

Summary 38

Chapter 3: Creating a Communication Application **39**
Zend\Form **39**
Time for action – creating a registration form **40**
Form validation **46**
Zend\InputFilter 46
Time for action – adding validation to the registration form **47**
Models and database access **50**
TableGateway 50
Time for action – creating models and saving the form **51**
Zend\Authentication 55
Time for action – user authentication **56**
Summary **58**
Chapter 4: Data Management and Document Sharing **59**
Zend Framework 2 ServiceManager **59**
Time for action – migrating existing code to ServiceManager **61**
Database operations **63**
More on TableGateway 64
Time for action – implementing an admin UI to manage users **65**
Document management **71**
Time for action – creating a file upload form **71**
Managing file sharing **76**
Time for action – implementing a file sharing system **76**
Summary **82**
Chapter 5: Chat and E-mail **83**
Layouts and views **83**
View helpers 84
The URL helper 84
The BasePath helper 85
The JSON helper 85
Concrete placeholder implementations 85
The HeadLink helper 85
The HeadMeta helper 86
The HeadScript helper 86
The HeadStyle helper 87
The HeadTitle helper 87
Time for action – using jQuery UI in a simple page **88**
Building a simple group chat **90**
Time for action – creating a simple group chat application **90**
Sending mails **95**
Zend\Mail\Transport 96
Zend\Mail\Message 96
Zend\Mime\Message and Zend\Mime\Part 96

Time for action – creating a simple e-mail form **97**
Zend\EventManager **99**
Time for action – setting module layout using ZF events **100**
Summary **103**

Chapter 6: Media Sharing **105**
External modules **105**
Resizing images **106**
Time for action – resizing images using modules **106**
The Photo gallery application **108**
Time for action – implementing a simple photo gallery **109**
Google Data APIs **113**
The Google Photos API 114
Time for action – fetching photos from Google Photos **115**
YouTube Data API **119**
Time for action – listing YouTube videos for a keyword **119**
Summary **122**

Chapter 7: Search Using Lucene **123**
Introduction to Lucene **123**
Time for action – installing ZendSearch\Lucene **124**
Indexing **125**
Time for action – generating a Lucene index **127**
Searching **129**
Time for action – displaying search results **130**
Indexing Microsoft Office documents **133**
Time for action – indexing document files **134**
Summary **137**

Chapter 8: Creating a Simple Store **139**
Shopping cart **140**
Time for action – creating a store front **140**
The store administration **143**
Time for action – creating the Store Admin interface **144**
Payments with PayPal **146**
PayPal and Zend Framework 2.0 146
Time for action – setting up PayPal **147**
PayPal Express Checkout **149**
Time for action – accepting payments using PayPal **150**
Summary **157**

Chapter 9: HTML5 Support 159
HTML5 input elements 160
Time for action – HTML5 input elements 165
HTML5 view helpers 167
Time for action – HTML5 view helpers 168
HTML5 attributes 171
Multiple file uploads 172
Time for action – HTML5 multiple file uploads 172
Summary 176
Chapter 10: Building Mobile Applications 177
Cloud-connected mobile applications 177
Zend Studio 10 178
phpCloud 178
Time for action – configuring your phpCloud account 178
PhoneGap and Zend Studio 182
Time for action – building your first cloud-connected mobile application 182
Native applications versus mobile web applications 186
Time for action – testing as a native application 187
Zend Server Gateway 190
Time for action – creating a mobile search interface 190
Summary 193
Appendix: Pop Quiz Answers 195
Chapter 1, Getting Started with Zend Framework 2.0 195
Chapter 2, Building Your First Zend Framework Application 195
Chapter 3, Creating a Communication Application 195
Chapter 4, Data Management and Document Sharing 196
Chapter 5, Chat and E-mail 196
Chapter 6, Media Sharing 196
Chapter 7, Search Using Lucene 196
Chapter 8, Creating a Simple Store 197
Chapter 9, HTML5 Support 197
Chapter 10, Building Mobile Applications 197
Index 199

Preface

Zend Framework 2 is the latest update to the well-known Zend Framework. This version has considerably eased the process of building complex web applications with minimal development effort using plug and play components. Zend Framework 2 also provides a highly robust and scalable framework for developing web applications.

This book will guide you through the process of developing powerful web applications using ZF2. It covers all aspects of Zend Framework application development right from installation and configuration; the tasks are designed in a way that readers can easily understand and use them to build their own applications with ease.

This book begins with basic installation and configuration of the Zend Framework. As you progress through the exercises, you will become thoroughly acquainted with ZF2. With this book, you will learn about the basic concepts of building solid MVC web applications using Zend Framework 2. The detailed step-by-step instructions will enable you to build functionality such as a group chat, a file and media sharing service, search, and a simple store, to name a few. You will also use a wide range of external modules to implement features that are not natively available.

By the end of the book, you will be well versed in building complex and functionality-rich web applications using Zend Framework 2.

What this book covers

Chapter 1, *Getting Started with Zend Framework 2.0*, introduces you to the configuration of the development environment. In this chapter, we will set up a PHP application server, install MySQL, and create a development database which will be used in subsequent chapters for our Zend Framework learning exercises.

Chapter 2, *Building Your First Zend Framework Application*, explains the creation of the Zend Framework 2 project; we will be reviewing some of the key aspects of building a ZF2 MVC application by creating modules, controllers, and views. We will be creating our own custom module in Zend Framework which will be enhanced further in subsequent chapters of this book.

Chapter 3, *Creating a Communication Application*, introduces you to Zend\Form. In this chapter we will create our first registration form, and set up login and authentication for registered users using Zend Framework components.

Chapter 4, *Data Management and Document Sharing*, covers some of Zend Framework's data and file management concepts. In this chapter, we will learn various aspects of Zend Framework including ServiceManager, the TableGateway pattern, handling uploads, and file sharing.

Chapter 5, *Chat and E-mail*, covers the use of JavaScript in your application. This chapter uses a simple group chat implementation as an example for explaining the usage of JavaScript in your applications; you will also be introduced to sending e-mails using Zend\Mail and the ZF2 event manager.

Chapter 6, *Media Sharing*, explains the management and sharing of images and videos using Zend Framework. In this chapter, we will use of various external Zend Framework 2 modules to work with images and videos.

Chapter 7, *Search using Lucene*, introduces you to the Lucene search implementation using Zend Framework. This chapter begins by explaining the users about the installation of ZendSearch\Lucene module, we then cover the details of implementing search for database records and also document files.

Chapter 8, *Creating a Simple Store*, introduces you to e-commerce. In this chapter, we will be building a simple online store to demonstrate the process involved in development of a shopping cart. We will be using PayPal Express Checkout as our payment processer in this chapter.

Chapter 9, *HTML5 Support*, introduces you to HTML5 support in Zend Framework 2. When compared to the previous version, ZF2 offers exhaustive support for various HTML5 features; this chapter covers two major aspects of ZF2's HTML5 support—new input types and multiple file uploads.

Chapter 10, *Building Mobile Applications*, introduces you to the development of native mobile applications with the help of Zend Framework 2 and Zend Studio 10. In this chapter, we will learn the fundamentals of building cloud-connected mobile applications using Zend Framework; we will also learn about the setup of Zend PHP developer cloud environment.

What you need for this book

You will need a system that is capable of running Zend Server CE along with MySQL. The prerequisite software that is required for working with tasks to be performance in the book is covered in *Chapter 1*, *Getting Started with Zend Framework 2.0*.

Who this book is for

If you are a PHP developer who is new to Zend Framework, but you want to get hands-on with the product quickly, this book is for you. Basic knowledge of object-oriented programming with PHP is expected.

Conventions

In this book, you will find several headings appearing frequently.

To give clear instructions of how to complete a procedure or task, we use:

Time for action – heading

1. Action 1
2. Action 2
3. Action 3

Instructions often need some extra explanation so that they make sense, so they are followed with:

What just happened?

This heading explains the working of tasks or instructions that you have just completed.

You will also find some other learning aids in the book, including:

Pop quiz – heading

These are short multiple-choice questions intended to help you test your own understanding.

Have a go hero – heading

These practical challenges give you ideas for experimenting with what you have learned.

You will also find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "The `TableGateway` class extends `AbstractTableGateway` which implements `TableGatewayInterface`."

A block of code is set as follows:

```
    // Add Document to index
    $indexDoc = new Lucene\Document();
    $indexDoc->addField($label);
    $indexDoc->addField($owner);
    $indexDoc->addField($fileUploadId);
    $index->addDocument($indexDoc);
  }
  // Commit Index
  $index->commit();
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
    // Add Document to index
    $indexDoc = new Lucene\Document();
    $indexDoc->addField($label);
    $indexDoc->addField($owner);
    $indexDoc->addField($fileUploadId);
    $index->addDocument($indexDoc);
  }
  // Commit Index
  $index->commit();
```

Any command-line input or output is written as follows:

```
$ sudo apt-get install php5-cli
$ sudo apt-get install git
$ curl -s https://getcomposer.org/installer | php
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "On the **Select Destination Location** screen, click on **Next** to accept the default destination."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this |book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title through the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website, or added to any list of existing errata, under the Errata section of that title.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Getting Started with Zend Framework 2.0

In this chapter we will get our development environment set up and configured in order to start development with Zend Framework 2.0. We will set up a PHP Application Server, install MySQL, and create a development database that will be used in subsequent chapters for our Zend Framework learning exercises. So, let's get started.

Zend Framework 2.0

The last major release of Zend Framework, which happened in 2007, was version 1.0; during the last five years, Zend Framework has undergone a lot of changes to be a successful PHP-based framework. But by merely updating the framework, Zend Framework has retained some of the issues that were inherently present in Zend Framework 1.0.

Zend Framework 2.0 is an attempt to make Zend Framework better by rearchitecting the framework right from the core. Some of the key features of Zend Framework 2.0 over its previous version are listed as follows:

- PHP 5.3 features such as namespaces and closures
- A modular application architecture
- Event manager
- Dependency Injection (DI)

We will get to know about implementing the new features of Zend Framework 2.0 in the coming chapters.

In this chapter we will cover the installation and configuration of some of the prerequisites of Zend Framework 2.0. ZF2 can be installed on most PHP-enabled web servers that support PHP 5.3.3 or later.

We have used Zend Server Community Edition as our default web server; however, any other PHP stack that supports PHP 5.3.3 can be used. Alternatively, you can also download Apache and PHP separately and install PHP over Apache.

To simplify the installation process, I am using Linux as the primary development environment in this book. All the tools used in this book are available for Windows and can be used to perform the same activity.

Introduction to Zend Server Community Edition (CE)

Zend Server Community Edition is the free version of the popular Zend Server stack. The Zend Server stack provides a pre-integrated PHP application stack that could be used across development, testing, and production. This enables application development teams to have a consistent environment across all stages of development.

Zend Server CE also provides features such as Zend Optimizer+ for PHP bytecode caching and Zend Guard for encoding files.

Zend Server CE – system requirements

Zend Server offers installers for Windows, Mac OS X, and a universal installation package compatible with most Linux distributions.

More details on the installation requirements can be found at `http://www.zend.com/en/products/server/system-requirements`.

Time for action – installing Zend Server CE

Our next step will be to download and install Zend Server CE; I am running Ubuntu 12.04 Precise Pangolin. The installation procedure for other operating systems could be different; you can always refer to the Zend Server website for installation instructions. The following are the steps to install the Zend Server CE:

1. Visit the Zend Server Community Edition website (`http://www.zend.com/en/community/zend-server-ce`) and download the latest version of Zend Server that is applicable to your operating system. In this case, we will be downloading the Linux installer.

2. Once the installer is downloaded, extract the contents of the installer to a temporary location:

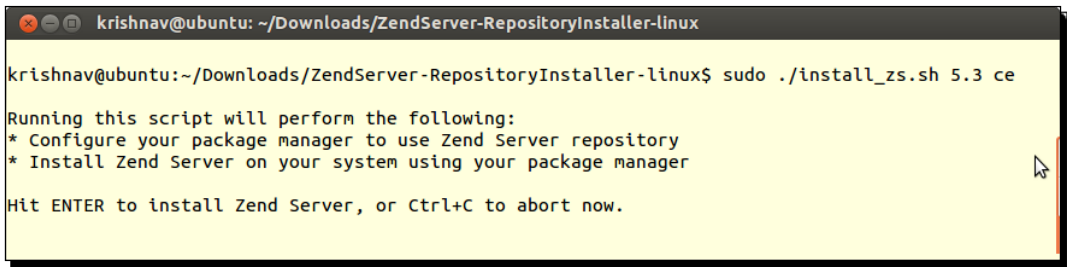
```
$ tar -zxvf ZendServer-5.6.0-RepositoryInstaller-linux.tar.gz
```

3. After extracting, the installer needs to be started with administrator privileges:

```
$ cd ZendServer-RepositoryInstaller-linux/
$ sudo ./install_zs.sh 5.3 ce
```

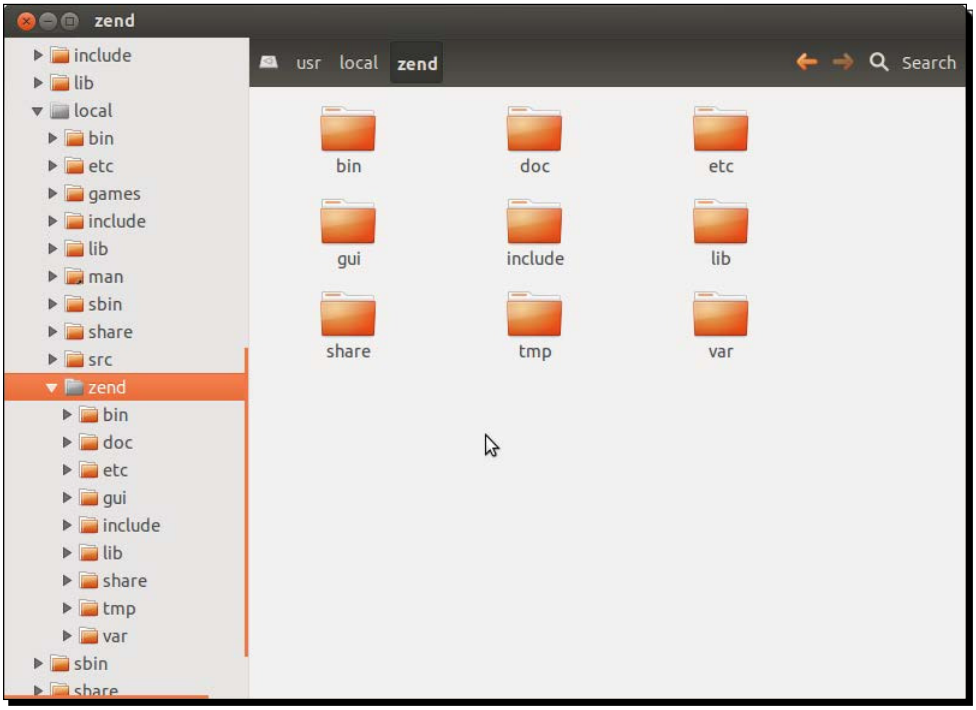
> We are passing two parameters to the installer. The first one is the version of PHP that needs to be installed; in this case it is `5.3`. The second parameter identifies the edition of Zend Server that needs to be installed; in this case it is `ce` for Community Edition.

4. During the installation, the installer will request you to download various packages:

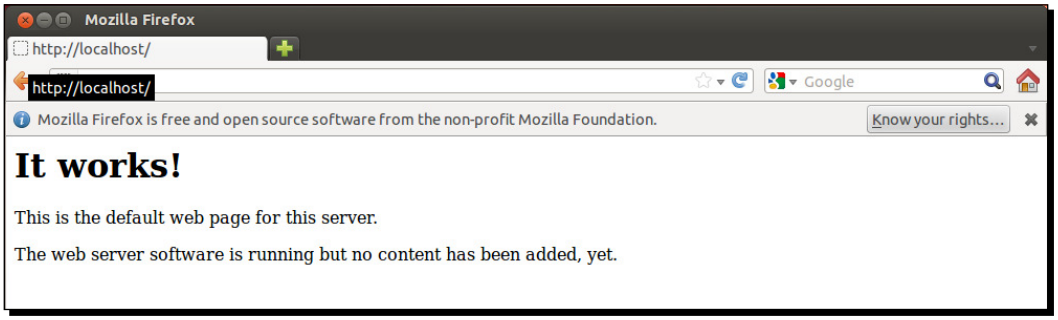


5. Zend Server will be installed into `/usr/local/zend` by default; the default document root will point to `/var/www`. You can use the following files to make configuration changes to the Zend Server instance:
   - Apache master configuration is available in `/etc/apache2/apache2.conf`
   - PHP configuration is controlled by `/var/local/zend/etc/php.ini`

The following screenshot shows the installed location of Zend Server:



6. Once the installation is completed, you should be able to open `http://localhost` on your web browser. This should take you to a test page like the one shown in the following screenshot:



To restart Zend Server, use the `$ sudo service zend-server restart` command.

What just happened?

Zend Server CE is installed and ready to be used. Now we have a web server and a compatible version of PHP running—this satisfies the core requirements for running Zend Framework 2.0.

Have a go hero

We will be using Git to check out Zend Framework from Github; one of the major changes that happened to Zend Framework 2.0 is that the source control has changed from SVN to Git.

Your next task will be to install Git. We will be making use of Git when we are setting up our Zend Framework project.

Git binaries can either be downloaded from `http://www.git-scm.com/` or installed from your operating system's repositories.

Installation instructions for Git can be found at the following link:

`http://git-scm.com/book/en/Getting-Started-Installing-Git`

Configuring Zend Server CE

Our next step will be to set up Zend Server CE and make some configuration changes that will enable us to run other PHP applications.

Zend Server CE – Administration Interface

Zend Server CE's Administration Interface is a web-based user interface that provides the following features:

- Managing PHP extensions
- Configuring PHP directives
- Managing Zend Server components
- Monitoring PHP status, extension status, and application/server logs

In our next task, we will be making a configuration change to Zend Server by using its Administration Interface.

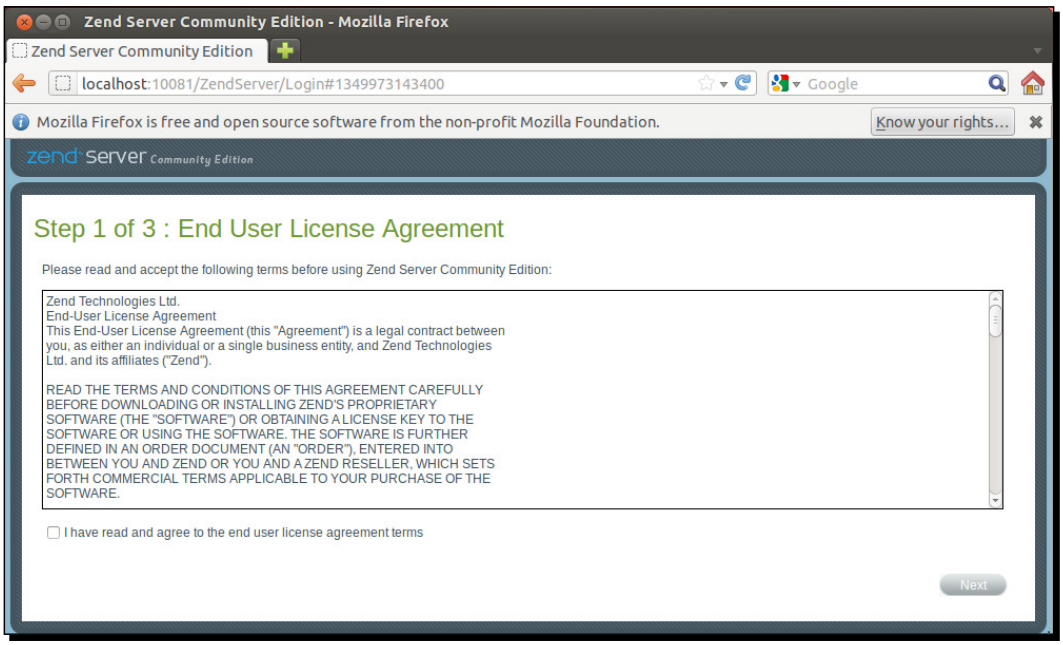
Time for action – configuring Zend Server CE

The Zend Server needs to be configured after the installation is completed. The following are the steps for configuring Zend Server CE:

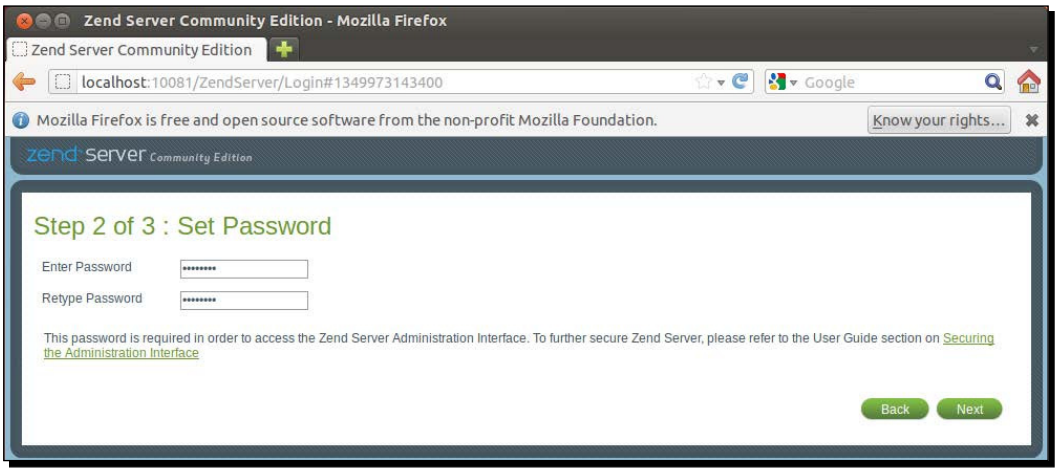
1. Open the admin console of Zend Server in your default browser (`http://localhost:10081/`).

> The Zend Server UI console runs on port 10081 while the web server runs on port 80. This is why we need to implicitly specify the port number in the URL for accessing the UI console.

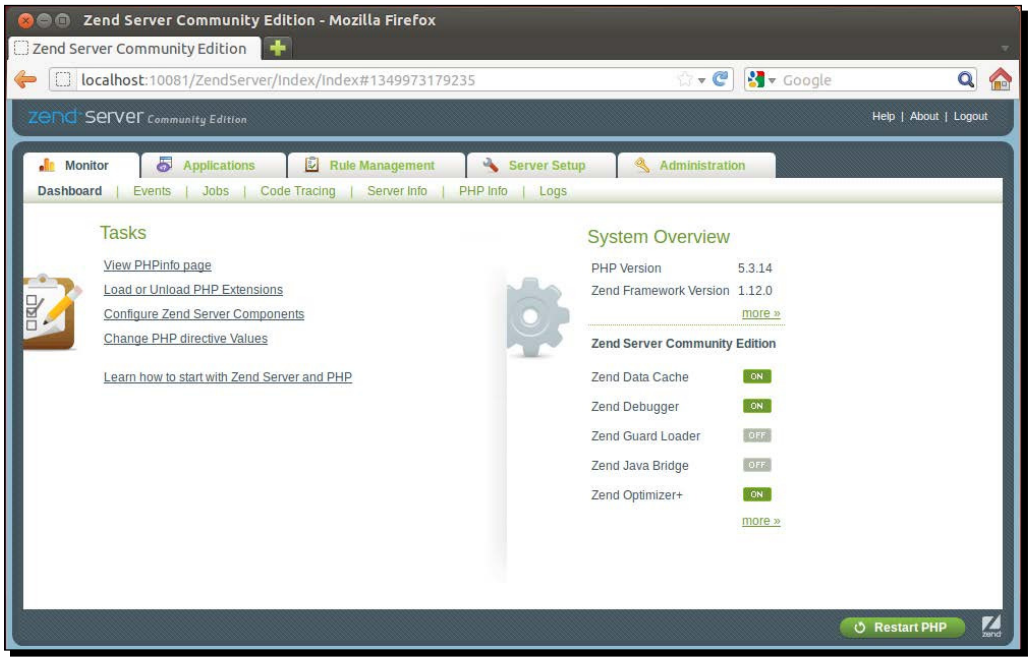
2. When opening the Zend Server Administration Interface for the first time, you will be presented with a configuration wizard. Review and accept the terms and conditions of Zend's **End User License Agreement** page:



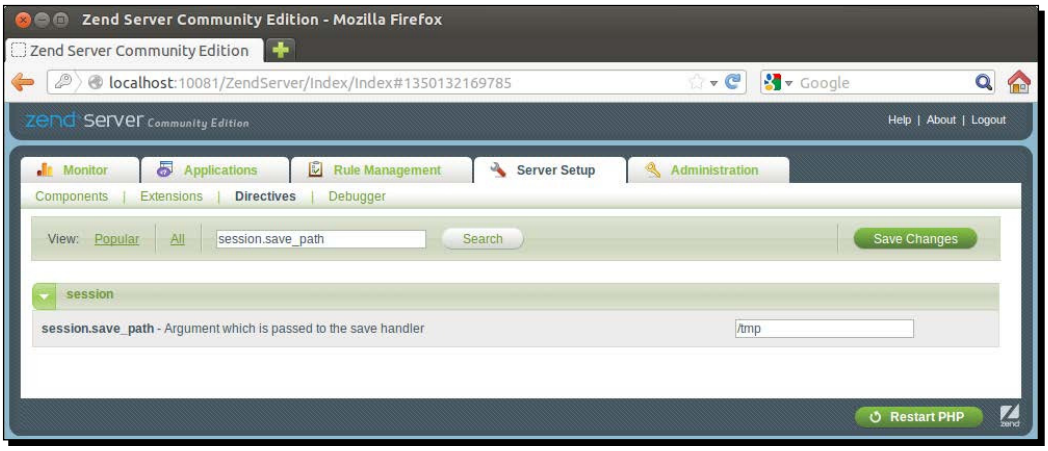
3. As shown in the following screenshot, you will be asked to set the password for the Zend Server installation:



4. After the initial configuration wizard is completed, you will be redirected to the Zend Server Administration Interface's home page.



5. We need to set the session save path. In order to do this, perform the following steps:
    1. Navigate to **Directives** in **Server Setup**.
    2. Search for `session.save_path`.
    3. Set the value to `/tmp`.
    4. Click on **Save Changes** and then **Restart PHP**.



What just happened?

We have successfully modified a server configuration using Zend Server's Administration Interface and we have restarted the PHP instance running on Zend Server.

MySQL

MySQL doesn't need an introduction—it is the world's most widely used open source database application. It's free and is available on the Internet to individuals and businesses that wish to develop their websites and applications using the MySQL database.

Zend Framework 2.0 has driver support for MySQL along with SQLite, PostgreSQL, and Microsoft SQL Server.

Our next exercise will be to install MySQL on our development machine. MySQL is available for download from all Linux repositories. Windows and Mac users will have to download the installer from the MySQL website (`http://dev.mysql.com/downloads/`).

Windows and Mac users can skip this section if they have chosen to install MySQL Server as a part of their Zend Server CE installation. The Zend Server installer allows Windows and Mac users to download and install MySQL Server as a part of the installation.

Time for action – installing MySQL

MySQL Server and Client need to be installed using the following steps; we will be using MySQL as our primary database in this book:

1. In a standard Ubuntu installation, MySQL can be installed by executing the following command in the shell prompt:

   ```
   $ sudo apt-get install mysql-server mysql-client
   ```

2. After the installation is complete, MySQL Server will start automatically. To check if MySQL Server is running, run the following command:

   ```
   $ sudo netstat -tap | grep mysql
   ```

3. The command should give an output that is similar to the following; this means that the MySQL daemon is running:

   ```
   tcp      0      0 localhost:mysql     *:*     LISTEN      923/mysqld
   ```

4. If, for some reason, MySQL Server is not running, you can start the server by running the `restart` command:

   ```
   $ sudo service mysql restart
   ```

What just happened?

We have just installed MySQL; we have the LAMP stack ready too. Our next step will be to create a database in MySQL Server.

Since we are using Zend Server, we don't need to install the `php5-mysql` package. If you are using a stack that doesn't have MySQL support enabled by default, you will have to install the necessary packages manually.

Have a go hero

Having gone through this section, feel free to attempt the task in the following section.

phpMyAdmin

phpMyAdmin is a free, open source web-based database administration tool written in PHP. phpMyAdmin provides a web-based UI to manage MySQL Database Server; add / remove / manage databases, users, privileges; and so on. In this book, we will be using phpMyAdmin as the database Administration Interface for managing our database(s).

Now that we have Apache, PHP, and MySQL installed, our next step will be to create a blank database in MySQL Server.

For doing this, we need to install and configure phpMyAdmin in the Zend Server.

phpMyAdmin can either be downloaded from `http://www.phpmyadmin.net/` or installed from your operating system's repositories.

Installation instructions for phpMyAdmin can be found at the following link:

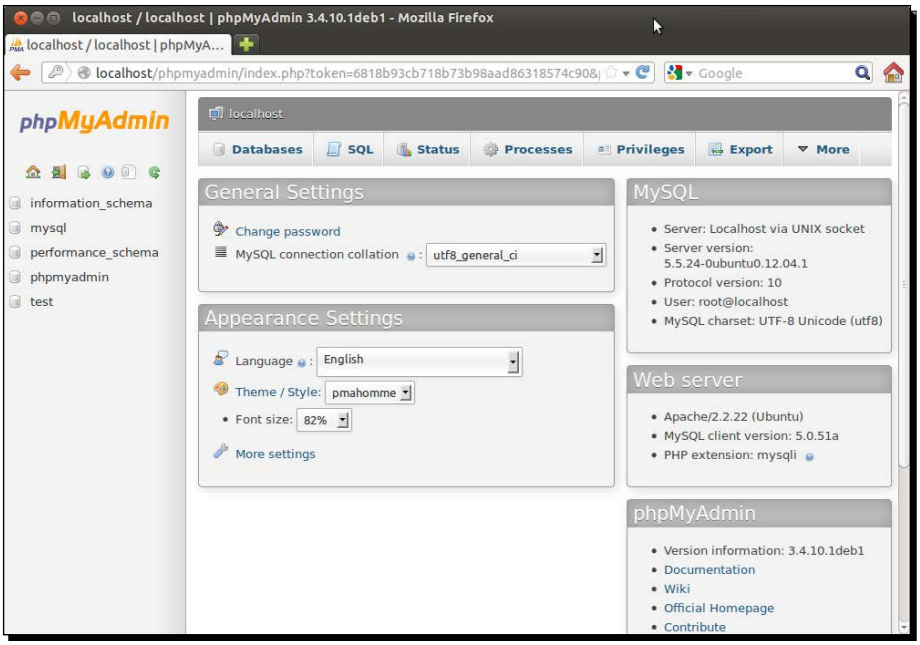
`http://docs.phpmyadmin.net/en/latest/setup.html`

In our next task we will be creating a MySQL database, creating users in the MySQL server and also grant them access permissions to connect to the database and perform database operations.

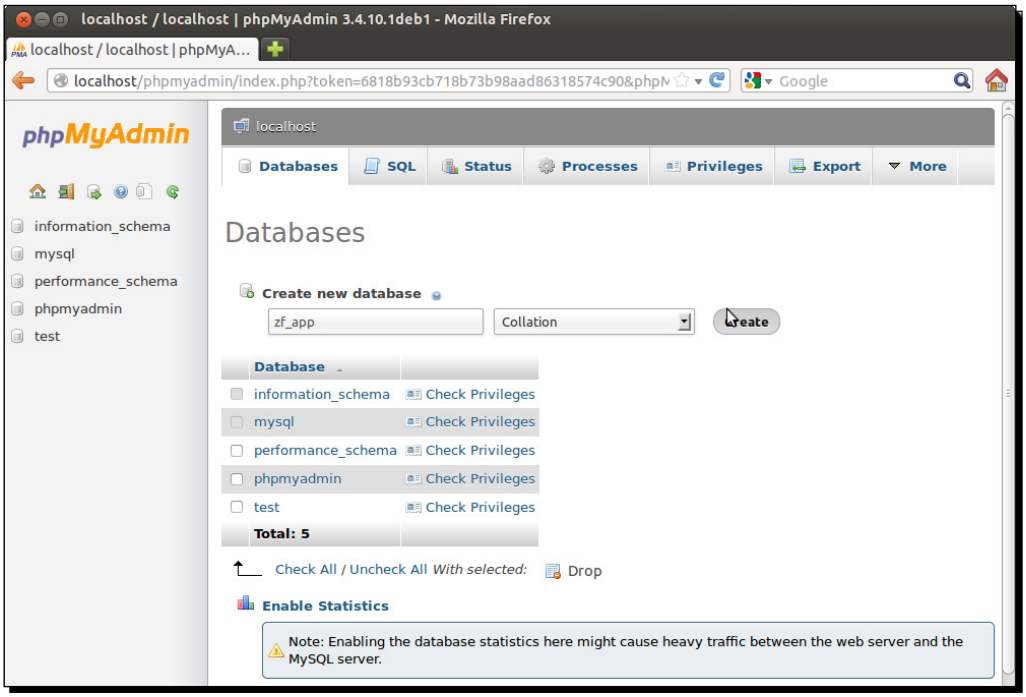
Time for action – creating a database

To create a new database, open an instance of phpMyAdmin in your web browser and follow the steps described here:

1. Open phpMyAdmin in your web browser by visiting `http://localhost/phpmyadmin`:



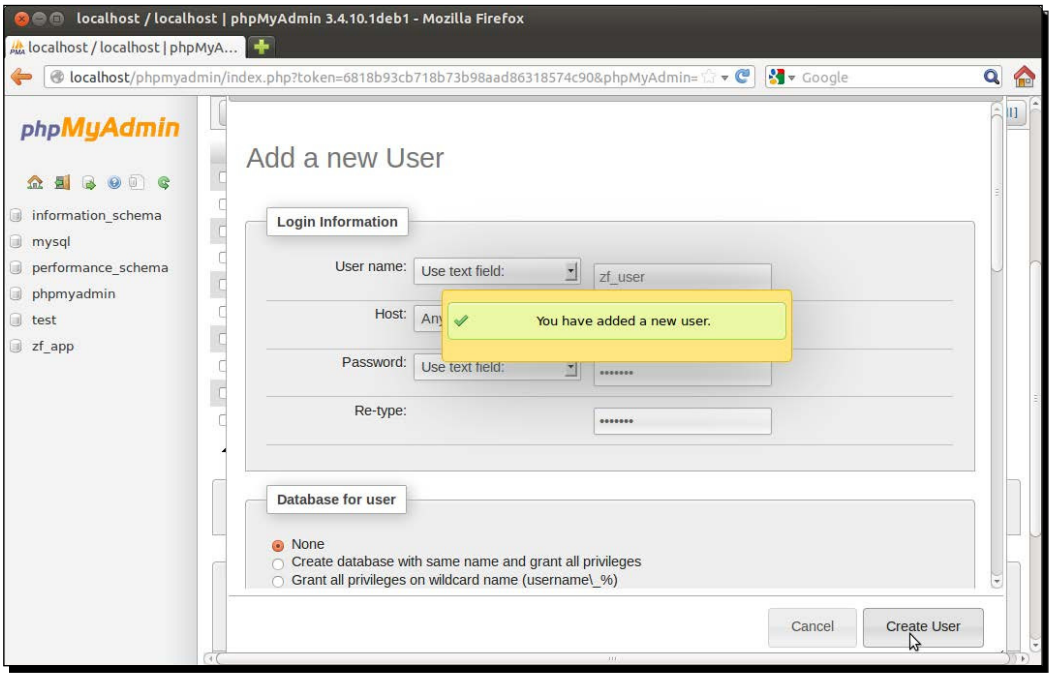
2. Choose **Databases**, enter the name of the new database as `zf_app` in **Create new database**, and click on **Create**:



3. After creating the database, create a database user for this database; this can be done by selecting **Add a new user** from **Privileges**. Provide the following details:

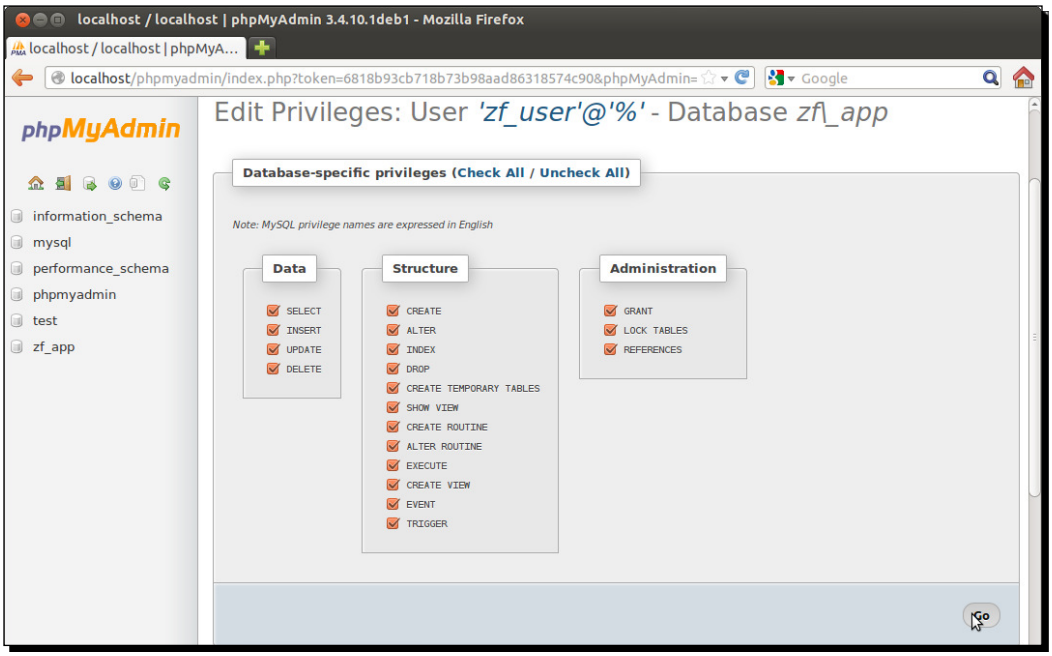
User field	Value
User name	`zf_user`
Host	`localhost`
Password	`zf_pass`

After doing this you will get the following screen:



4. After the user is created, go to the **Privileges** section and choose **Edit Privileges** for the `zf_user`.
5. In the **Database-specific privileges** section, select the `zf_app` database.

6. You will be redirected to the privileges section of the `zf_app` database for the `zf_user` user. Choose **Check All** and click on **Go**.



You can now test the database by logging out of phpMyAdmin and logging in again with the user credentials of `zf_user`. You should now be able to see only the `zf_app` database.

What just happened?

We just created our first database in MySQL. We have also created a user in the database and mapped the user to the database with administrative rights; we can now use these credentials in the application that we will be building in our next chapters.

Have a go hero

Now that you have the PHP web server up and running and also have a MySQL database, create a simple table called `Students` and add a few records to the table using phpMyAdmin.

Your task will be to create a simple PHP web page that will display all the records in the `Students` table in the page.

Pop quiz – Zend Framework 2.0

Q1. What is the minimum version of PHP needed to run Zend Framework 2.0?

1. PHP 4.3 and above
2. PHP 5.2.0 and above
3. PHP 5.3.3 and above
4. PHP 5.4.7 and above

Q2. What is the default location of `php.ini` in the new Zend Server installation?

1. `/home/<user>/etc/php/php.inc`
2. `/etc/php/php.ini`
3. `/var/www/php.ini`
4. `/usr/local/zend/etc/php.ini`

Summary

In this chapter we have learned the setup and configuration of Zend Server's PHP application stack. We went on to install MySQL Server and created our first database. In your exercises, you have learned about the installation of Git and phpMyAdmin.

In the next chapter, we will learn about the structure of a Zend Framework project and core MVC components such as views and controllers.

2

Building Your First Zend Framework Application

In this chapter, we are going to create our first Zend Framework 2.0 project; we will be reviewing some of the key aspects of building a ZF2 MVC Application by creating modules, controllers, and views. We will be creating our own custom module in Zend Framework which will be enhanced further in subsequent chapters of this book.

Prerequisites

Before you get started with setting up your first ZF2 Project, make sure that you have the following software installed and configured in your development environment:

- PHP Command Line Interface
- **Git**: Git is needed to check out source code from various `github.com` repositories
- **Composer**: Composer is the dependency management tool used for managing PHP dependencies

The following commands will be useful for installing the necessary tools to setup a ZF2 Project:

- To install PHP Command Line Interface:
  ```
  $ sudo apt-get install php5-cli
  ```
- To install Git:
  ```
  $ sudo apt-get install git
  ```
- To install Composer:
  ```
  $ curl -s https://getcomposer.org/installer | php
  ```

ZendSkeletonApplication

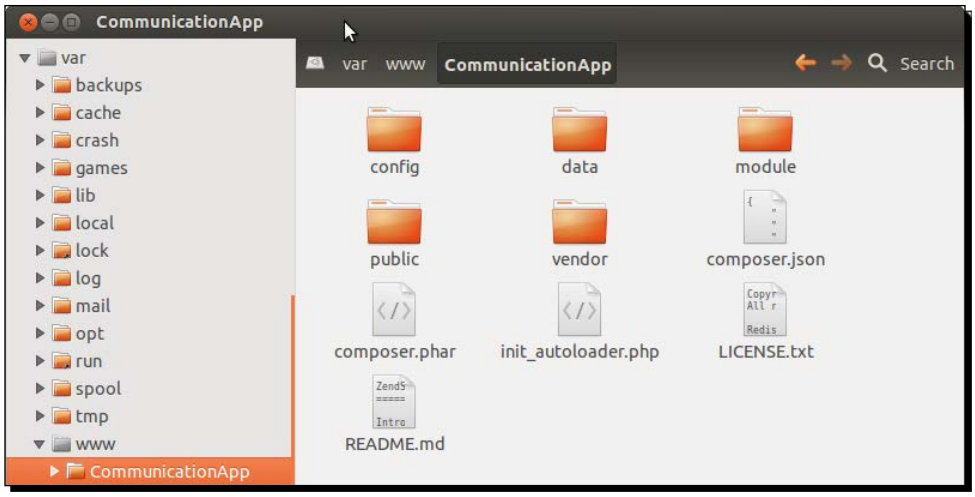
`ZendSkeletonApplication` provides a sample skeleton application that can be used by developers as a starting point to get started with Zend Framework 2.0. The skeleton application makes use of ZF2 MVC, including a new module system.

`ZendSkeletonApplication` can be downloaded from GitHub (`https://github.com/zendframework/ZendSkeletonApplication`).

Time for action – creating a Zend Framework project

To set up a new Zend Framework project, we will need to download the latest version of `ZendSkeletonApplication` and set up a virtual host to point to the newly created Zend Framework project. The steps are given as follows:

1. Navigate to a folder location where you want to set up the new Zend Framework project:
   ```
   $ cd /var/www/
   ```
2. Clone the `ZendSkeletonApplication` app from GitHub:
   ```
   $ git clone git://github.com/zendframework/
   ZendSkeletonApplication.git CommunicationApp
   ```



In some Linux configurations, necessary permissions may not be available to the current user for writing to `/var/www`. In such cases, you can use any folder that is writable and make necessary changes to the virtual host configuration.

3. Install dependencies using Composer:

```
$ cd CommunicationApp/
$ php composer.phar self-update
$ php composer.phar install
```

The following screenshot shows how Composer downloads and installs the necessary dependencies:

```
krishnav@ubuntu:/var/www/CommunicationApp$ php composer.phar self-update
Updating to version 172414a.
    Downloading: 100%
krishnav@ubuntu:/var/www/CommunicationApp$ php composer.phar install
Loading composer repositories with package information
Installing dependencies
  - Installing zendframework/zendframework (2.0.3)
    Downloading: 100%

zendframework/zendframework suggests installing doctrine/common (Doctrine\Common >=2
zendframework/zendframework suggests installing ext-intl (ext/intl for i18n features
zendframework/zendframework suggests installing pecl-weakref (Implementation of weak
zendframework/zendframework suggests installing zendframework/zendpdf (ZendPdf for c
zendframework/zendframework suggests installing zendframework/zendservice-recaptcha (
as in Zend\Captcha and/or Zend\Form)
Writing lock file
Generating autoload files
krishnav@ubuntu:/var/www/CommunicationApp$
```

4. Before adding a virtual host entry we need to set up a hostname entry in our hosts file so that the system points to the local machine whenever the new hostname is used. In Linux this can be done by adding an entry to the `/etc/hosts` file:

```
$ sudo vim /etc/hosts
```

> In Windows, this file can be accessed at `%SystemRoot%\system32\drivers\etc\hosts`.

5. Add the following line to the `hosts` file:

```
127.0.0.1     comm-app.local
```

The final `hosts` file should look like the following:

```
127.0.0.1       localhost
127.0.1.1       ubuntu
127.0.0.1       comm-app.local

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

6. Our next step would be to add a virtual host entry on our web server; this can be done by creating a new virtual host's configuration file:

```
$ sudo vim /usr/local/zend/etc/sites.d/vhost_comm-app-80.conf
```

> This new virtual host filename could be different for you depending upon the web server that you use; please check out your web server documentation for setting up new virtual hosts.
>
> For example, if you have Apache2 running on Linux, you will need to create the new virtual host file in `/etc/apache2/sites-available` and enable the site using the command `a2ensite comm-app.local`.

7. Add the following configuration to the virtual host file:

```
<VirtualHost *:80>
  ServerName comm-app.local
  DocumentRoot /var/www/CommunicationApp/public
  SetEnv APPLICATION_ENV "development"
  <Directory /var/www/CommunicationApp/public>
```

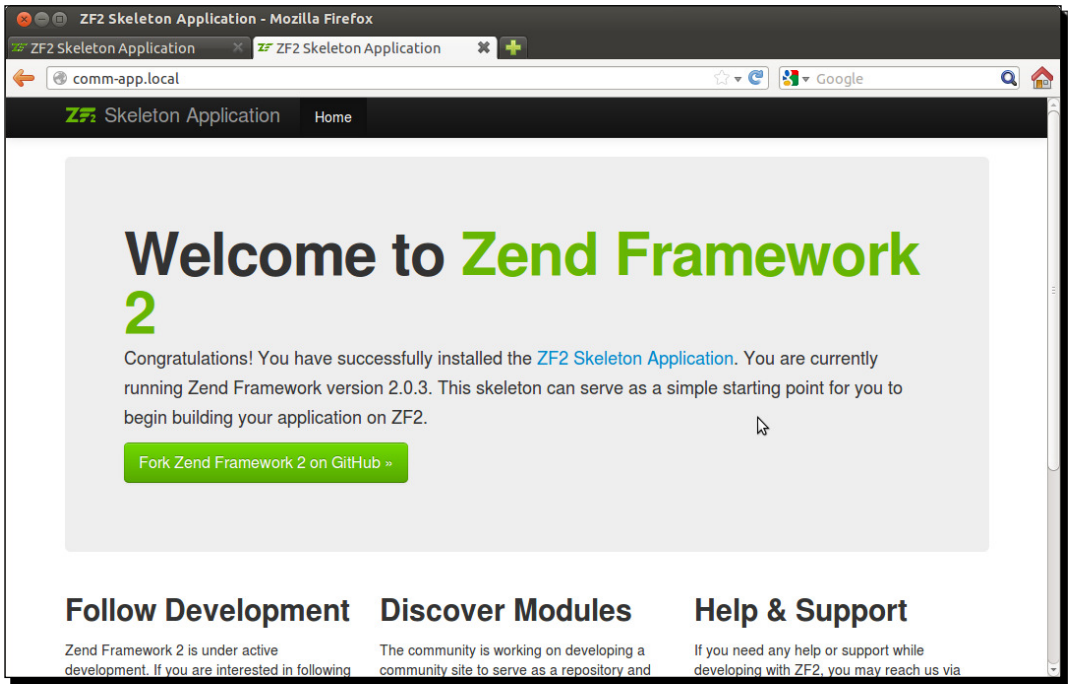
```
    DirectoryIndex index.php
    AllowOverride All
    Order allow,deny
    Allow from all
  </Directory>
</VirtualHost>
```

If you are using a different path for checking out the `ZendSkeletonApplication` project make sure that you include that path for both `DocumentRoot` and `Directory` directives.

8. After configuring the virtual host file, the web server needs to be restarted:

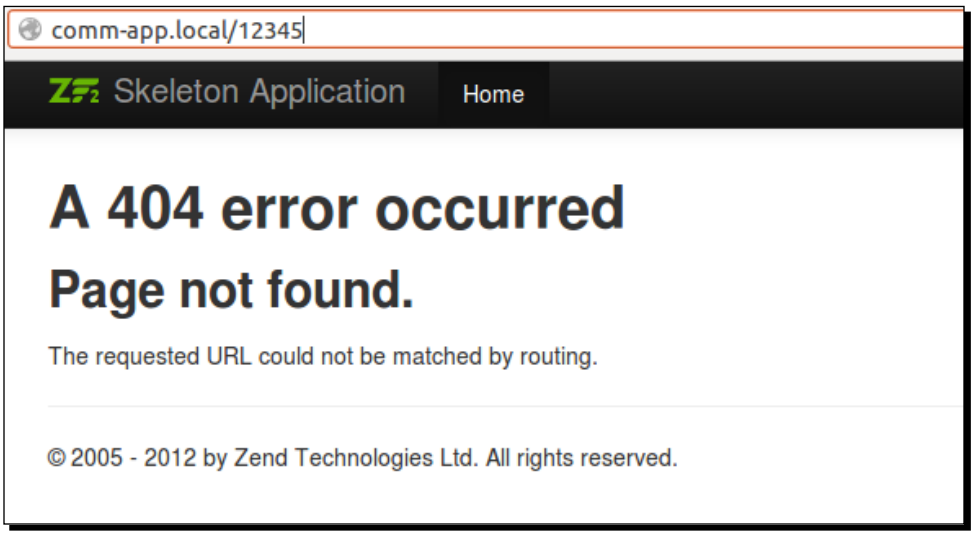
```
$ sudo service zend-server restart
```

9. Once the installation is completed, you should be able to open `http://comm-app.local` on your web browser. This should take you to the following test page :



Test rewrite rules

In some cases, `mod_rewrite` may not have been enabled in your web server by default; to check if the URL redirects are working properly, try to navigate to an invalid URL such as `http://comm-app.local/12345`; if you get an Apache 404 page, then the `.htaccess` rewrite rules are not working; they will need to be fixed, otherwise if you get a page like the following one, you can be sure of the URL working as expected.



What just happened?

We have successfully created a new ZF2 project by checking out `ZendSkeletonApplication` from GitHub and have used Composer to download the necessary dependencies including Zend Framework 2.0. We have also created a virtual host configuration that points to the project's `public` folder and tested the project in a web browser.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Alternate installation options

We have seen just one of the methods of installing `ZendSkeletonApplication`; there are other ways of doing this.

You can use Composer to directly download the skeleton application and create the project using the following command:

```
$ php composer.phar create-project --repository-
url="http://packages.zendframework.com" zendframework/
skeleton-application path/to/install
```

You can also use a recursive Git clone to create the same project:

```
$ git clone git://github.com/zendframework/
ZendSkeletonApplication.git --recursive
```

Refer to:

`http://framework.zend.com/downloads/skeleton-app`

Zend Framework 2.0 – modules

In Zend Framework, a module can be defined as a unit of software that is portable and reusable and can be interconnected to other modules to construct a larger, complex application.

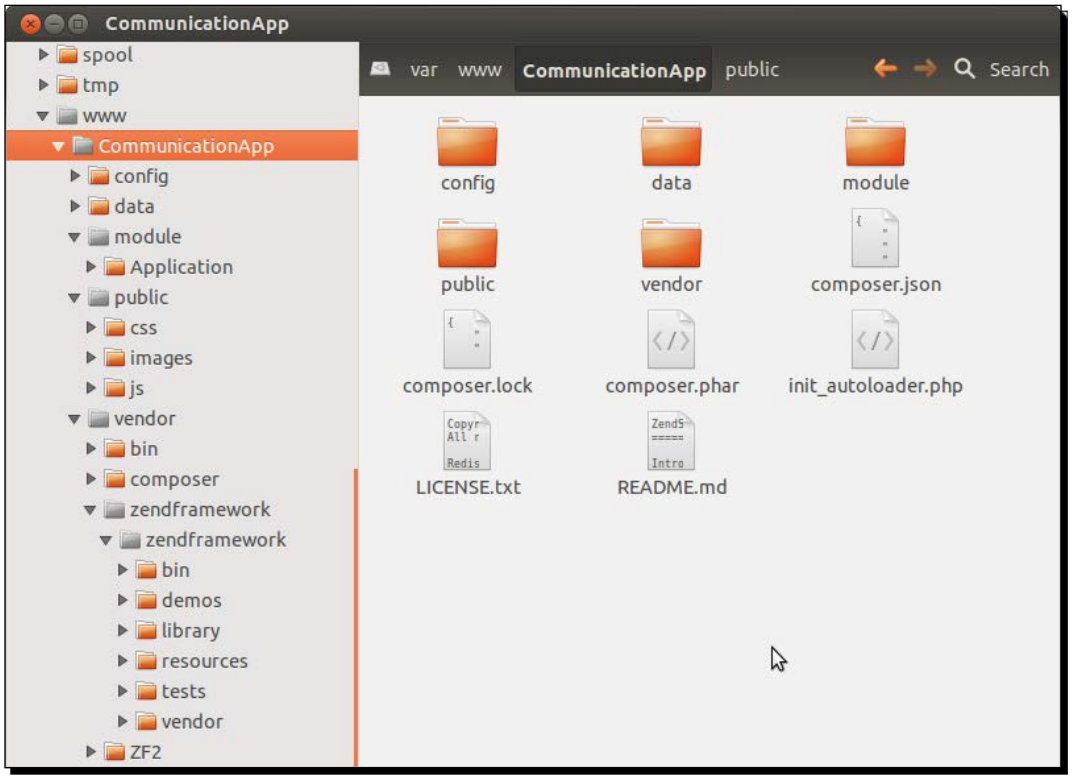
Modules are not new in Zend Framework, but with ZF2, there is a complete overhaul in the way modules are used in Zend Framework. With ZF2, modules can be shared across various systems, and they can be repackaged and distributed with relative ease. One of the other major changes coming into ZF2 is that even the main application is now converted into a module; that is, the application module.

Some of the key advantages of Zend Framework 2.0 modules are listed as follows:

- Self-contained, portable, reusable
- Dependency management
- Lightweight and fast
- Support for Phar packaging and Pyrus distribution

Zend Framework 2.0 – project folder structure

The folder layout of a ZF2 project is shown as follows:



Folder name	Description
`config`	Used for managing application configuration.
`data`	Used as a temporary storage location for storing application data including cache files, session files, logs, and indexes.
`module`	Used to manage all application code.
`module/Application`	This is the default application module that is provided with `ZendSkeletonApplication`.
`public`	Serves as an entry point to the application; the website's document root points here. All web resources including CSS files, images, and JavaScripts are stored here.
`vendor`	Used to manage common libraries that are used by the application. Zend Framework is also installed in this folder.
`vendor/zendframework`	Zend Framework 2.0 is installed here.

Time for action – creating a module

Our next activity will be about creating a new `Users` module in Zend Framework 2.0. The `Users` module will be used for managing users including user registration, authentication, and so on. We will be making use of `ZendSkeletonModule` provided by Zend, shown as follows:

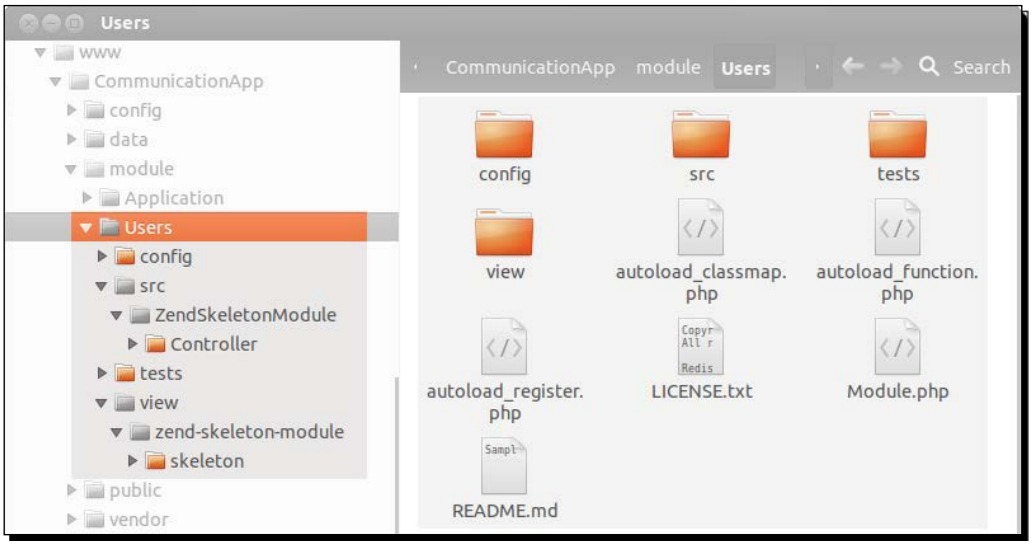
1. Navigate to the application's `module` folder:

   ```
   $ cd /var/www/CommunicationApp/
   $ cd module/
   ```

2. Clone `ZendSkeletonModule` into a desired module name, in this case it is `Users`:

   ```
   $ git clone git://github.com/zendframework/ZendSkeletonModule.git
   Users
   ```

3. After the checkout is complete, the folder structure should look like the following screenshot:



4. Edit `Module.php`; this file will be located in the `Users` folder under modules (`CommunicationApp/module/Users/module.php`) and change the namespace to `Users`. Replace `namespace ZendSkeletonModule;` with `namespace Users;`.

5. The following folders can be removed because we will not be using them in our project:
   * `Users/src/ZendSkeletonModule`
   * `Users/view/zend-skeleton-module`

What just happened?

We have installed a skeleton module for Zend Framework; this is just an empty module, and we will need to extend this by creating custom controllers and views. In our next activity, we will focus on creating new controllers and views for this module.

Creating a module using ZFTool

`ZFTool` is a utility for managing Zend Framework applications/projects, and it can also be used for creating new modules; in order to do that, you will need to install `ZFTool` and use the `create module` command to create the module using `ZFTool`:

```
$ php composer.phar require zendframework/zftool:dev-master
$ cd vendor/zendframework/zftool/
$ php zf.php create module Users2 /var/www/CommunicationApp
```

Read more about ZFTool at the following link:

`http://framework.zend.com/manual/2.0/en/modules/zendtool.introduction.html`

MVC layer

The fundamental goal of any MVC Framework is to enable easier segregation of three layers of the MVC, namely, model, view, and controller. Before we get to the details of creating modules, let's quickly try to understand how these three layers work in an MVC Framework:

- **Model**: The model is a representation of data; the model also holds the business logic for various application transactions.
- **View**: The view contains the display logic that is used to display the various user interface elements in the web browser.
- **Controller**: The controller controls the application logic in any MVC application; all actions and events are handled at the controller layer. The controller layer serves as a communication interface between the model and the view by controlling the model state and also by representing the changes to the view. The controller also provides an entry point for accessing the application.

- In the new ZF2 MVC structure, all the models, views, and controllers are grouped by modules. Each module will have its own set of models, views, and controllers, and will share some components with other modules.

Zend Framework module – folder structure

The folder structure of Zend Framework 2.0 module has three vital components—the configurations, the module logic, and the views. The following table describes how contents in a module are organized:

Folder name	Description
`config`	Used for managing module configuration
`src`	Contains all module source code, including all controllers and models
`view`	Used to store all the views used in the module

Time for action – creating controllers and views

Now that we have created the module, our next step would be having our own controllers and views defined. In this section, we will create two simple views and will write a controller to switch between them:

1. Navigate to the module location:

   ```
   $ cd /var/www/CommunicationApp/module/Users
   ```

2. Create the folder for controllers:

   ```
   $ mkdir -p src/Users/Controller/
   ```

3. Create a new `IndexController` file, `< ModuleName >/src/<ModuleName>/Controller/`:

   ```
   $ cd src/Users/Controller/
   $ vim IndexController.php
   ```

4. Add the following code to the `IndexController` file:

   ```
   <?php
   namespace Users\Controller;
   use Zend\Mvc\Controller\AbstractActionController;
   use Zend\View\Model\ViewModel;
   class IndexController extends AbstractActionController
   {
       public function indexAction()
       {
   ```

```
            $view = new ViewModel();
            return $view;
        }
        public function registerAction()
        {
            $view = new ViewModel();
            $view->setTemplate('users/index/new-user');
            return $view;
        }
        public function loginAction()
        {
            $view = new ViewModel();
            $view->setTemplate('users/index/login');
            return $view;
        }
    }
```

5. The preceding code will do the following actions; if the user visits the home page, the user is shown the default view; if the user arrives with an action `register`, the user is shown the `new-user` template; and if the user arrives with an action set to `login`, then the `login` template is rendered.
6. Now that we have created the controller, we will have to create necessary views to render for each of the controller actions.
7. Create the folder for views:

   ```
   $ cd /var/www/CommunicationApp/module/Users
   $ mkdir -p view/users/index/
   ```

8. Navigate to the views folder, `<Module>/view/<module-name>/index`:

   ```
   $ cd view/users/index/
   ```

9. Create the following view files:
   - `index`
   - `login`
   - `new-user`

   1. For creating the `view/users/index/index.phtml` file, use the following code:

      ```
      <h1>Welcome to Users Module</h1>
      <a href="/users/index/login">Login</a> | <a href="/users/
      index/register">New User Registration</a>
      ```

2. For creating the `view/users/index/login.phtml` file, use the following code:

```
<h2> Login </h2>
<p> This page will hold the content for the login form </p>
<a href="/users"><< Back to Home</a>
```

3. For creating the `view/users/index/new-user.phtml` file, use the following code:

```
<h2> New User Registration </h2>
<p> This page will hold the content for the registration
form </p>
<a href="/users"><< Back to Home</a>
```

What just happened?

We have now created a new controller and views for our new Zend Framework module; the module is still not in a shape to be tested. To make the module fully functional we will need to make changes to the module's configuration, and also enable the module in the application's configuration.

Zend Framework module – configuration

Zend Framework 2.0 module configuration is spread across a series of files which can be found in the skeleton module. Some of the configuration files are described as follows:

- `Module.php`: The Zend Framework 2 module manager looks for the `Module.php` file in the module's root folder. The module manager uses the `Module.php` file to configure the module and invokes the `getAutoloaderConfig()` and `getConfig()` methods.
- `autoload_classmap.php`: The `getAutoloaderConfig()` method in the skeleton module loads `autoload_classmap.php` to include any custom overrides other than the classes loaded using the standard autoloader format. Entries can be added or removed to the `autoload_classmap.php` file to manage these custom overrides.
- `config/module.config.php`: The `getConfig()` method loads `config/module.config.php`; this file is used for configuring various module configuration options including routes, controllers, layouts, and various other configurations.

Time for action – modifying module configuration

In this section will make configuration changes to the `Users` module to enable it to work with the newly created controller and views using the following steps:

1. **Autoloader configuration** – The default autoloader configuration provided by the `ZendSkeletonModule` needs to be disabled; this can be done by editing `autoload_classmap.php` and replacing it with the following content:

```
<?php
return array();
```

2. **Module configuration** – The module configuration file can be found in `config/module.config.php`; this file needs to be updated to reflect the new controllers and views that have been created, as follows:

   - **Controllers** – The default controller mapping points to the `ZendSkeletonModule`; this needs to be replaced with the mapping shown in the following snippet:

```
'controllers' => array(
    'invokables' => array(
        'Users\Controller\Index' =>
        'Users\Controller\IndexController',
    ),
),
```

   - **Views** – The views for the module have to be mapped to the appropriate view location. Make sure that the view uses lowercase names separated by a hyphen (for example, `ZendSkeleton` will be referred to as zend-skeleton):

```
'view_manager' => array(
    'template_path_stack' => array(
        'users' => __DIR__ . '/../view',
    ),
),
```

   - **Routes** – The last module configuration is to define a route for accessing this module from the browser; in this case we are defining the route as `/users`, which will point to the `index` action in the `Index` controller of the `Users` module:

```
'router' => array(
    'routes' => array(
        'users' => array(
            'type'    => 'Literal',
            'options' => array(
                'route'    => '/users',
```

```
                        'defaults' => array(
                            '__NAMESPACE__' =>
                            'Users\Controller',
                            'controller'    => 'Index',
                            'action'        => 'index',
                        ),
                    ),
```

3. After making all the configuration changes as detailed in the previous sections, the final configuration file, `config/module.config.php`, should look like the following:

```
<?php
return array(
    'controllers' => array(
        'invokables' => array(
            'Users\Controller\Index' =>
            'Users\Controller\IndexController',
        ),
    ),
    'router' => array(
        'routes' => array(
            'users' => array(
                'type'    => 'Literal',
                'options' => array(
                    // Change this to something specific to
                    your module
                    'route'    => '/users',
                    'defaults' => array(
                        // Change this value to reflect the
                        namespace in which
                        // the controllers for your module are
                        found
                        '__NAMESPACE__' => 'Users\Controller',
                        'controller'    => 'Index',
                        'action'        => 'index',
                    ),
                ),
                'may_terminate' => true,
                'child_routes' => array(
                    // This route is a sane default when
                    developing a module;
                    // as you solidify the routes for your module,
                    however,
                    // you may want to remove it and replace it
                    with more
```

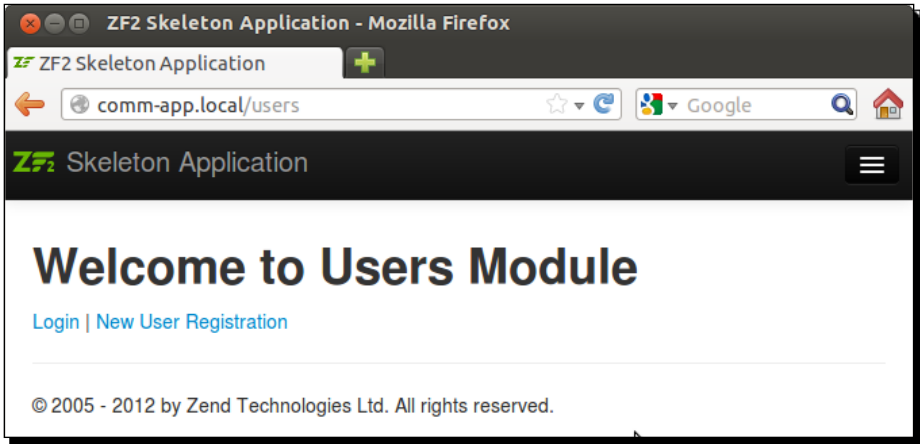
```
                        // specific routes.
                        'default' => array(
                            'type'    => 'Segment',
                            'options' => array(
                                'route'    =>
                                '/[:controller[/:action]]',
                                'constraints' => array(
                                    'controller' =>
                                    '[a-zA-Z][a-zA-Z0-9_-]*',
                                    'action'     =>
                                    '[a-zA-Z][a-zA-Z0-9_-]*',
                                ),
                                'defaults' => array(
                                ),
                            ),
                        ),
                    ),
                ),
            ),
        ),
        'view_manager' => array(
            'template_path_stack' => array(
                'users' => __DIR__ . '/../view',
            ),
        ),
    );
```

4. **Application configuration** – Enable the module in the application's configuration—this can be done by modifying the application's `config/application.config.php` file, and adding `Users` to the list of enabled modules:

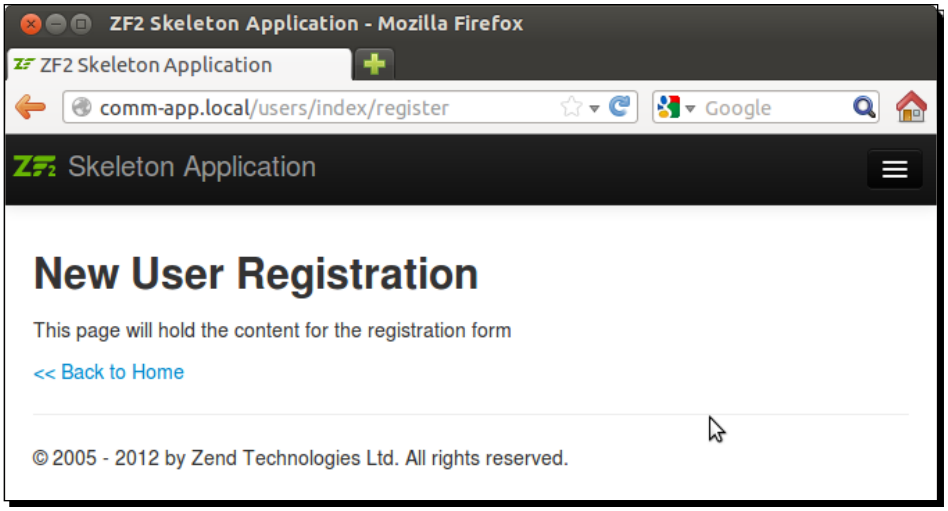
```
    'modules' => array(
        'Application',
        'Users',
    ),
```

5. To test the module in a web browser, open `http://comm-app.local/users/` in your web browser; you should be able to navigate within the module.

The module home page is shown as follows:



The registration page is shown as follows:



What just happened?

We have modified the configuration of `ZendSkeletonModule` to work with the new controller and views created for the `Users` module. Now we have a fully-functional module up and running using the new ZF module system.

Have a go hero

Now that we have the knowledge to create and configure own modules, your next task would be to set up a new `CurrentTime` module. The requirement for this module is to render the current time and date in the following format:

Time: 14:00:00 GMT Date: 12-Oct-2012

Pop quiz – Zend Framework 2.0

Q1. What is the tool used by `ZendSkeletonApplication` for managing dependencies in PHP?

1. Git
2. Composer
3. PHP Command Line Interface
4. Pyrus

Q2. What is the filename of a module's configuration file?

1. `<App>/module/<Module>/config.inc`
2. `<App>/<Module>/config/config.php`
3. `<App>/module/<Module>/module.config.php`
4. `<App>/module/<Module>/config/module.config.php`

Summary

We have now learned about setting up a new Zend Framework project using Zend's skeleton application and module. In our next chapters, we will be focusing on further development on this module and extending it into a fully-fledged application.

3

Creating a Communication Application

In the previous chapter, we covered creating controllers and views in a new Zend Framework module. In this chapter we will create our first registration form, and set up login and authentication for registered users using Zend Framework components.

Some of the key components that we will focus on in this chapter are listed as follows:

- `Zend\Form`
- `Zend\InputFilter`
- `Zend\Validator`
- Models and `Zend\Db`

Zend\Form

Forms are usually built by creating the HTML page for the form, writing separate validation and filtering for various form events, and finally writing the controllers and actions for the form actions. With Zend Framework, the `Zend\Form` component provides all the previously stated features in a single component.

`Zend\Form` allows developers to programmatically create and handle forms in your applications. `Zend\Form` supports form rendering, form handling, input filtering and validation, and form configurations. In our next task we will set up our first form in ZF2.

Time for action – creating a registration form

To create our first registration form, we will create a new controller to display a registration form; we will also create new forms and views. We need to make the following changes to the `Users` module:

1. **Form** – We will also need to create a registration form under `src/Users/Form/RegisterForm.php`:

    1. The `RegisterForm` class extends `Zend\Form\Form`; the form's configuration is added to the constructor:

    ```
    <?php
    // filename : module/Users/src/Users/Form/RegisterForm.php
    namespace Users\Form;
    use Zend\Form\Form;
    class RegisterForm extends Form
    {
        public function __construct($name = null)
        {
            parent::__construct('Register');
            $this->setAttribute('method', 'post');
            $this->setAttribute('enctype','multipart/form-
            data');
    ```

    2. All fields are added to the form using the `$this->add()` method on the form's constructor:

    ```
            $this->add(array(
                'name' => 'name',
                'attributes' => array(
                    'type'  => 'text',
                ),
                'options' => array(
                    'label' => 'Full Name',
                ),
            ));
    ```

    3. Additional validators/filters can be added to the fields while declaring the fields in the form. In this case we are adding special validation for the `EmailAddress` field:

    ```
            $this->add(array(
                'name' => 'email',
                'attributes' => array(
                    'type'  => 'email',
                ),
    ```

```
                'options' => array(
                    'label' => 'Email',
                ),
                'attributes' => array(
                    'required' => 'required'
                ),
                'filters' => array(
                    array('name' => 'StringTrim'),
                ),
                'validators' => array(
                    array(
                        'name' => 'EmailAddress',
                        'options' => array(
                            'messages' => array(
                                \Zend\Validator\
EmailAddress::INVALID_FORMAT => 'Email address format is
invalid'
                            )
                        )
                    )
                )
        ));
```

4. Use the same method to add `password`, `confirm_password`, and `submit` fields; `password` and `confirm_password` will be of type `password`, whereas `submit` will be of type `button`.

2. **Views** – The following views will have to be created to support the registration process:

1. **Registration page**: The view for registration page is created in `src/view/users/register/index.phtml`.
2. The view consists of three main sections—the section to display error messages, the view logic which is used to generate the form tag, and the view helpers used to generate the actual form elements. The following logic is used to display error messages:

```
<section class="register">
<h2>Register</h2>
<?php if ($this->error): ?>
<p class="error">
    There were one or more issues with your submission.
Please correct them as
    indicated below.
</p>
<?php endif ?>
```

3. The following block is used to generate the `<form>` HTML tag using the `form` object assigned to the view in the controller:

```
<?php
$form = $this->form;
$form->prepare();
$form->setAttribute('action', $this->url(NULL,
array('controller'=>'Register', 'action' => 'process')));
$form->setAttribute('method', 'post');
echo $this->form()->openTag($form);
?>
```

4. The following section is used to generate individual form elements for the **Name**, **Email**, **Password**, **Confirm Password**, and **Submit** fields:

```
<dl class="zend_form">
<dt><?php echo $this->formLabel($form->get('name')); ?></dt>
<dd><?php
    echo $this->formElement($form->get('name'));
    echo $this->formElementErrors($form->get('name'));
?></dd>
<dt><?php echo $this->formLabel($form->get('email')); ?></
dt>
<dd><?php
    echo $this->formElement($form->get('email'));
    echo $this->formElementErrors($form->get('email'));
?></dd>
<dt><?php echo $this->formLabel($form->get('password'));
?></dt>
<dd><?php
    echo $this->formElement($form->get('password'));
    echo $this->formElementErrors($form->get('password'));
?></dd>
<dt><?php echo $this->formLabel($form->get('confirm_
password')); ?></dt>
<dd><?php
    echo $this->formElement($form->get('confirm_password'));
    echo $this->formElementErrors($form->get('confirm_
password'));
?></dd>
<dd><?php
    echo $this->formElement($form->get('submit'));
    echo $this->formElementErrors($form->get('submit'));
?></dd>
</dl>
```

5. Finally the `form` HTML tag needs to be closed:

```
<?php echo $this->form()->closeTag() ?>
</section>
```

6. **Confirmation page**: The view for the confirmation page is pretty straightforward, the view is created in `src/view/users/register/confirm.phtml`.

```
<section class="register-confirm">
<h2>Register Sucessfull</h2>
<p> Thank you for your registration. </p>
</section>
```

3. **Controller** – Now that we have the form and views ready, our next step will be to have a controller in place, which will help us to access this form. We will create a new `RegisterController` class and load the newly created form in its index action. The new controller will be created in the `src/Users/Controller/RegisterController.php` file:

```
<?php
namespace Users\Controller;
use Zend\Mvc\Controller\AbstractActionController;
use Zend\View\Model\ViewModel;
use Users\Form\RegisterForm;
class RegisterController extends AbstractActionController
{

    public function indexAction()
    {
                $form = new RegisterForm();
                $viewModel  = new ViewModel(array('form' =>
                $form));
                return $viewModel;
    }
    public function confirmAction()
    {
                $viewModel  = new ViewModel();
                return $viewModel;
    }
}
```

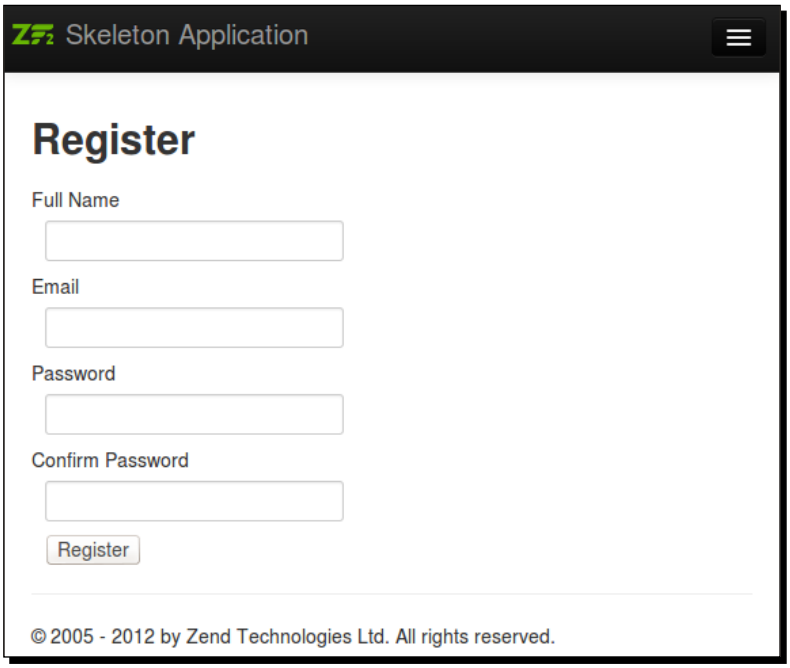
4. **Configuration** – Now we have created all the necessary components to display our form, we need to add our controller to the `invokables` list in the module config (`config/module.config.php`):

```
'controllers' => array(
  'invokables' => array(
    'Users\Controller\Index' =>
    'Users\Controller\IndexController',
    'Users\Controller\Register' =>
    'Users\Controller\RegisterController',
  ),
```

5. To test the registration form's display, open any web browser and try accessing the following URL:

   `http://comm-app.local/users/register`

   The registration form should look like the following:



What just happened?

Until now we have created a form that can be used to display all the necessary fields that can be used during the registration process. Let us try to understand how the form is being rendered. When we invoke the `http://comm-app.local/users/register` page, the controller creates a new instance of the `RegisterForm` class and displays it on the web browser. We have added the following fields to the `RegisterForm` class using its constructor:

- **Name**
- **Email**
- **Password**
- **Confirm Password**
- The **Submit** button

These fields are added to the newly created `Form` object. The `ViewModel` pattern renders the form, and the `form` object gets passed over to the view for rendering, and each field is rendered as per the logic in the view using the `FormElement` view helper.

`FormElement` works as a magic helper to render any form field based on the type of the `Zend\Form\Element` tag that is passed on to it. There are individual helpers for rendering specific form fields. The complete list of form view helpers can be obtained from the ZF documentation on *Form View Helpers* found at `http://framework.zend.com/manual/2.0/en/modules/zend.form.view.helpers.html`.

Have a go hero

Before we move on to the next section, please create a login form in the same way that we used to create the registration form. The form will contain the following fields:

- **Email**
- **Password**
- The **Submit** button

We will be using this login form to perform authentication towards the end of this chapter.

Form validation

If you had taken a closer look at the form code, you would have noticed that we have added some validation for the **Email Address** field as shown in the following snippet:

```
'attributes' => array(
    'required' => 'required'
),
'filters' => array(
    array('name' => 'StringTrim'),
),
'validators' => array(
    array(
        'name' => 'EmailAddress',
        'options' => array(
            'messages' => array(
                \Zend\Validator\EmailAddress::INVALID_
                 FORMAT => 'Email address format is
                 invalid'
            )
```

So, we added the following:

- An attribute to make the field a `required` field
- A filter to trim the string that is passed
- A validator to verify if the e-mail address is in the valid format

With the introduction on Zend Framework's InputFilter, we can validate entire forms instead of attaching validation to each and every form field. This allows much cleaner code and better scalability of Zend Forms. So effectively we can have the same form being used in multiple sections of the website, each having its own set of validation rules that are not dependant on the form's validation. In our next section we will set up a new validator for the registration form.

Zend\InputFilter

Validation for forms and various other inputs can be performed by making use of `Zend\InputFilter`. This component allows filtering and validation of generic sets of input data. For specific form elements you can apply validation and filtering on the specific elements, but if we have to filter an input set like a `$_GET` request or a `$_POST` request, this can be implemented using the `InputFilter` class.

In our next task, we will be adding the `InputFilter` class to our registration form.

Time for action – adding validation to the registration form

To add an `InputFilter` class to an existing form, we need to create a new `InputFilter` class and use it during form submission for validation, as shown in the following steps:

1. Create a new `InputFilter` class in `src/Users/Form/RegisterFilter.php`. The `RegisterFilter` class will extend the `Zend\InputFilter\InputFilter` class and will add all the necessary validators in its constructor:

```
<?php
namespace Users\Form;
use Zend\InputFilter\InputFilter;

class RegisterFilter extends InputFilter
{
    public function __construct()
    {
```

2. Using the `$this->add()` method, we can add various filter options to the registration form:

    1. For the **Email Address** field, we will add a validator to check if the value entered is a valid e-mail address:

```
$this->add(array(
    'name'       => 'email',
    'required'   => true,
    'validators' => array(
        array(
            'name'    => 'EmailAddress',
            'options' => array(
                'domain' => true,
            ),
        ),
    ),
));
```

    2. For the **Name** field, we will add a validator to limit the size between `2` to `140` characters and will also add a filter to strip the HTML tags:

```
$this->add(array(
    'name'       => 'name',
    'required'   => true,
    'filters'    => array(
        array(
            'name'    => 'StripTags',
        ),
```

```
                ),
                'validators' => array(
                    array(
                        'name'    => 'StringLength',
                        'options' => array(
                            'encoding' => 'UTF-8',
                            'min'      => 2,
                            'max'      => 140,
                        ),
                    ),
                ),
            ));
```

3. For the **Password** and **Confirm Password** fields, we will not add any validators but will make them mandatory:

```
'password'        ));
        $this->add(array(
            'name'       => 'confirm_password',
            'required'   => true,
        ));
```

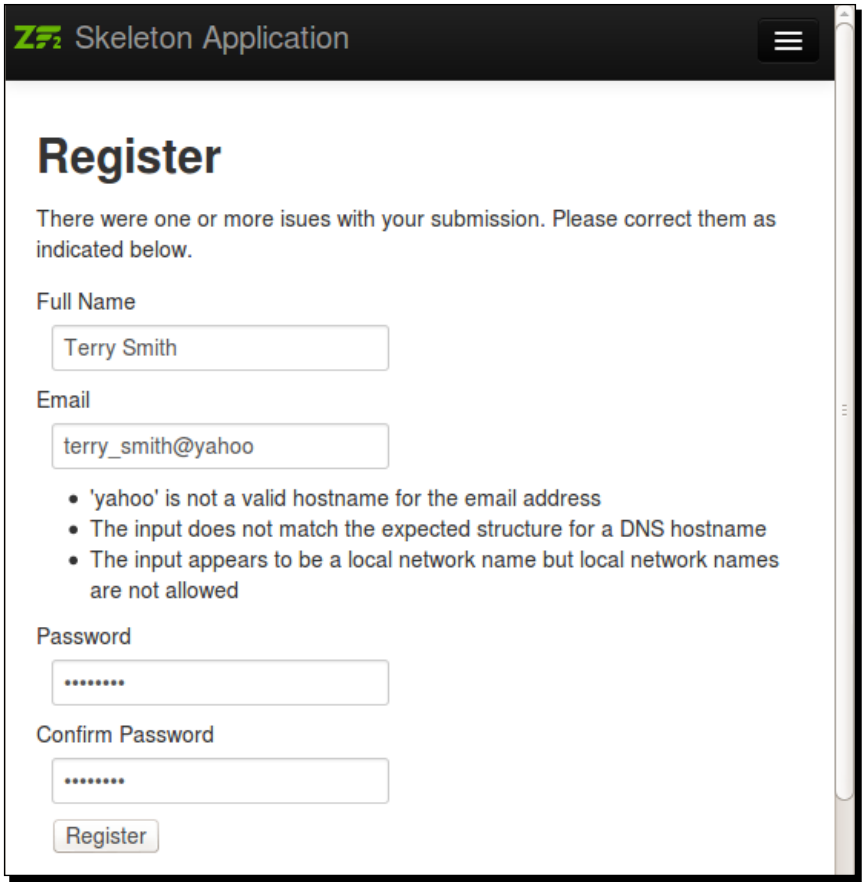
3. This `InputFilter` class is not mapped to the `RegisterForm` class yet; we will be performing the validation during form submission. We need to modify the `RegisterController` class to enable the `processAction` method and validate the form upon submission.

4. Modify the `RegisterController` class to enable the `processAction` method:

```
public function processAction()
{
  if (!$this->request->isPost()) {
    return $this->redirect()->toRoute(NULL ,
      array( 'controller' => 'register',
            'action' =>  'index'
      ));
  }
  $post = $this->request->getPost();
  $form = new RegisterForm();
  $inputFilter = new RegisterFilter();
  $form->setInputFilter($inputFilter);
  $form->setData($post);
  if (!$form->isValid()) {
    $model = new ViewModel(array(
      'error' => true,
      'form'  => $form,
    ));
```

```
        $model->setTemplate('users/register/index');
        return $model;
      }
      return $this->redirect()->toRoute(NULL , array(
        'controller' => 'register',
        'action' =>  'confirm'
      ));
    }
```

5. Now open the registration page in your web browser and test the validation:



What just happened?

We have now enabled validation on the registration form. In the `processAction()` function of the `RegisterController` class, you will see that a new instance of the `RegisterFrom` class is created and `RegisterFilter` is applied to the form using the `$form->setInputFilter()` method. The data entered as input to the form is added again and validation is performed by using the `isValid()` method. Error messages are rendered in the form using the `FormElementErrors` view helper.

We need to ensure that the names in the `InputFilter` class properly map to the names in the form while adding validation to `InputFilter`.

Have a go hero

You've just learned about adding a custom `InputFilter` class to a Zend form using the previous task; before you move on to the next section, set up a validation `InputFilter` for the `Login` form that you have built in your previous exercise.

Models and database access

Models provide a representation of data in the MVC application. There is no `Zend\Model` component that is provided by Zend Framework, so developers have to decide on the implementation part of models. Models by themselves cannot talk to databases and fetch or process data, so they are usually connected to mapper objects or use ORM to connect to databases. For this example, we will be using a `TableGateway` pattern for storing data in the database.

`TableGateway` is a built-in Zend Framework 2 DB pattern which acts as a gateway to a database table, having access to all table rows for performing various SQL operations including `select`, `insert`, `update`, and `delete`.

TableGateway

The `TableGateway` pattern is used for creating an object that represents a table in the database; in this example, we will need a `TableGateway` object for the `User` table.

The `exchangeArray()` method needs to be declared in the model if the model uses `TableGateway` for database storage.

Time for action – creating models and saving the form

In this task, we will be creating a new user model, creating a table in MySQL database to save the registration data using `TableGateway` to store registration data to the table. We will, finally, connect our registration form to `UserTable` so that new registrations are stored in the database. Perform the following steps to do so:

1. A new table needs to be created to store the registration information in the MySQL database:

```
CREATE TABLE user (
  id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  name TEXT NOT NULL,
  email VARCHAR(255) NOT NULL,
  password TEXT NOT NULL,
  PRIMARY KEY (id),
  UNIQUE INDEX idx_email(email)
);
```

2. The application's global configuration needs to be modified to add references to the database connection as shown in the following snippet. This is available under `<Application_Home>/config/autoload/global.php`.

```
return array(
    'db' => array(
        'driver'         => 'Pdo',
        'dsn'            => 'mysql:dbname=test;host=localhost',
        'username'         => 'db_user',
        'password'         => '',
        'driver_options' => array(
            PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES \'UTF8\''
        ),
    ),
    'service_manager' => array(
        'factories' => array(
            'Zend\Db\Adapter\Adapter'
                    => 'Zend\Db\Adapter\AdapterServiceFactory',
        ),
    ),
);
```

3. Create a new model for the `User` class. This needs to be created under `src/Users/Model/User.php`.

```
<?php
namespace Users\Model;
class User
{
  public $id;
  public $name;
  public $email;
  public $password;
}
```

4. The `User` model will define the `setPassword()` and the `exchangeArray()` methods:

1. Implement a `setPassword()` method which will assign a MD5 version password to the `UserTable` entity for storage:

```
public function setPassword($clear_password)
{
  $this->password = md5($clear_password);
}
```

2. Implement the `exchangeArray()` method; this method is used while mapping the `User` entity to the `UserTable` entity:

```
function exchangeArray($data)
{
  $this->name = (isset($data['name'])) ?
  $data['name'] : null;
  $this->email = (isset($data['email'])) ?
  $data['email'] : null;
  if (isset($data["password"]))
  {
    $this->setPassword($data["password"]);
  }
}
```

5. Create a new table reference for `User`. This needs to be created under `src/Users/Model/UserTable.php`:

```
<?php
namespace Users\Model;
use Zend\Db\Adapter\Adapter;
use Zend\Db\ResultSet\ResultSet;
use Zend\Db\TableGateway\TableGateway;
class UserTable
```

```
{
  protected $tableGateway;
  public function __construct(TableGateway $tableGateway)
  {
    $this->tableGateway = $tableGateway;
  }
  public function saveUser(User $user)
  {
    $data = array(
      'email' => $user->email,
      'name'  => $user->name,
      'password'  => $user->password,
    );
    $id = (int)$user->id;
    if ($id == 0) {
      $this->tableGateway->insert($data);
    } else {
      if ($this->getUser($id)) {
        $this->tableGateway->update($data, array('id' => $id));
      } else {
        throw new \Exception('User ID does not exist');
      }
    }
  }
  public function getUser($id)
  {
    $id  = (int) $id;
    $rowset = $this->tableGateway->select(array('id' => $id));
    $row = $rowset->current();
    if (!$row) {
      throw new \Exception("Could not find row $id");
    }
    return $row;
  }
}
```

6. Now we can use `UserTable` to save new registrations to the database. To save registrations, we need to make changes to the `RegisterController` class. First, we will create a new function for saving user registration:

```
protected function createUser(array $data)
{
  $sm = $this->getServiceLocator();
  $dbAdapter = $sm->get('Zend\Db\Adapter\Adapter');
  $resultSetPrototype = new \Zend\Db\ResultSet\ResultSet();
```

```
    $resultSetPrototype->setArrayObjectPrototype(new
    \Users\Model\User);
    $tableGateway = new \Zend\Db\TableGateway\TableGateway('user',
    $dbAdapter, null, $resultSetPrototype);

    $user = new User();
    $user->exchangeArray($data);
    $userTable = new UserTable($tableGateway);
    $userTable->saveUser($user);
    return true;
}
```

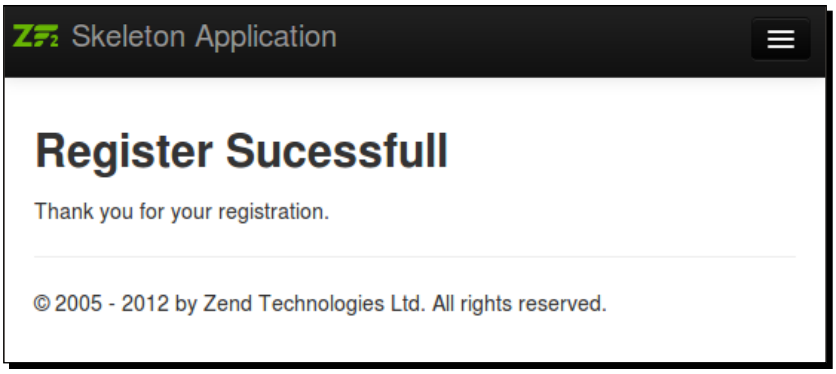
The `TableGateway` constructor takes the following parameters and generates a `TableGateway` object in response:

- **`$table`**: Used to provide the table name for the `TableGateway` object.
- **`Adapter $adapter`**: Used to provide the database adapter name.
- **`$features`** (optional): TableGateway Feature API allows the extension of the `TableGateway` functionality without having to extend the base class. The features can be specified here.
- **`ResultSet $resultSetPrototype`** (optional): Used to provide the `ResultSet` type.
- **`Sql $sql`** (optional): Used to provide any additional SQL criteria; make sure that the SQL object is bound to the same table as in `$table`.
- For more information refer to: `http://framework.zend.com/manual/2.0/en/modules/zend.db.table-gateway.html#zend-db-tablegateway`

7. Next, we need to make sure that the `processAction()` method calls this function before redirecting to the confirmation page:

```
// Create user
$this->createUser($form->getData());
```

8. Open the registration page in your favourite browser and use the MySQL database to check if the registration information is properly stored in the database. The registration confirmation page should look like the following screenshot:



You can check the MySQL database to see if the records have been inserted properly:

```
mysql> SELECT id, name, email, password FROM user;
+----+-------------+----------------------+----------------------------------+
| id | name        | email                | password                         |
+----+-------------+----------------------+----------------------------------+
|  1 | Terry Smith | terry.smith@email.com | 5f4dcc3b5aa765d61d8327deb882cf99 |
|  2 | John Smith  | john.smith@email.com  | 5f4dcc3b5aa765d61d8327deb882cf99 |
|  3 | Mani Raj    | mani.raj@email.com    | 5f4dcc3b5aa765d61d8327deb882cf99 |
|  4 | Test User   | test.user@email.com   | 5f4dcc3b5aa765d61d8327deb882cf99 |
+----+-------------+----------------------+----------------------------------+
4 rows in set (0.00 sec)

mysql>
```

What just happened?

We have now modified the form to save new user registrations to the database; our next step will be to set up authentication based on the information stored in the database.

Zend\Authentication

`Zend\Authentication` is an authentication component provided by Zend Framework which can be used for authentication against a wide number of authentication mechanisms including database table, HTTP authentication, and LDAP authentication. The component also lets you store the session information to a wide range of storages.

In this example, we will be using the `Zend\Authentication` component to validate the user credentials submitted in the login form.

Time for action – user authentication

In this task we will be authenticating the login form using the `Zend\Authentication` component using the following steps:

1. Add a function to return the authentication service in the login controller `src/Users/Controller/LoginController.php`:

```
// References
use Zend\Authentication\AuthenticationService;
use Zend\Authentication\Adapter\DbTable as DbTableAuthAdapter;
// Class definition
public function getAuthService()
{
  if (! $this->authservice) {
    $dbAdapter = $this->getServiceLocator()->get('Zend\Db\Adapter\
Adapter');
    $dbTableAuthAdapter = new DbTableAuthAdapter($dbAdapter,
'user','email','password', 'MD5(?)');
    $authService = new AuthenticationService();
    $authService->setAdapter($dbTableAuthAdapter);
    $this->authservice = $authService;
  }
  return $this->authservice;
}
```

2. In the `processAction()` method for `LoginController`, check if the form submission is valid, and use the `AuthService` method to validate the credentials using the `authenticate` method:

```
public function processAction()
//
$this->getAuthService()->getAdapter()
                              ->setIdentity($this->request-
>getPost('email'))
                              ->setCredential($this->request-
>getPost('password'));
$result = $this->getAuthService()->authenticate();
if ($result->isValid()) {
  $this->getAuthService()->getStorage()->write($this->request-
>getPost('email'));
  return $this->redirect()->toRoute(NULL , array(
                                     'controller' => 'login',
                                     'action' =>  'confirm'
                                     ));
}
```

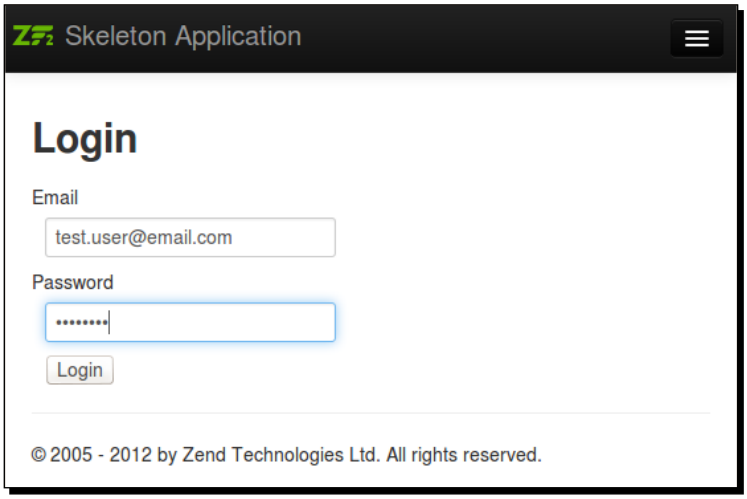
3. The `ConfirmAction` function will render the logged in user's welcome screen:

```
    public function confirmAction()
    {
  $user_email = $this->getAuthService()->getStorage()->read();
  $viewModel  = new ViewModel(array(
            'user_email' => $user_email
        ));
  return $viewModel;
    }
```

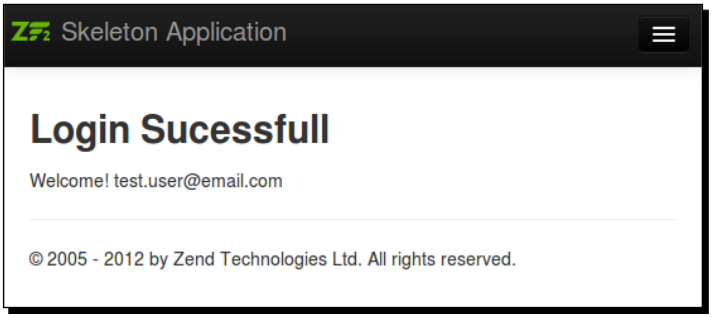
4. The view for the user's home page created under `/view/users/login/confirm.phtml` will be as follows:

```
<section class="login-confirm">
<h2>Login Successful</h2>
<p> Welcome!  <?php echo $this->user_email; ?> </p>
</section>
```

5. Open the login page in your browser and try to log in with the credentials that you used during registration. The login form should look like the following:



Upon successful login, you will be redirected to the login success page as shown below.



What just happened?

We created a new database table authentication adapter for the `user` table to validate the `email` and `password` fields. Using the authentication adapter we have been able to perform authentication for registered users.

Pop quiz – Zend Framework 2.0

Q1. Which file should be modified to store the database credentials application-wide?

1. `<App>/module/<Module>/config.inc`
2. `<App>/config/autoload/global.php`
3. `<App>/module/<Module>/module.config.php`
4. `<App>/module/<Module>/config/module.config.php`

Q2. What is the correct method to assign an input filter to a form?

1. `$form->setInputFilter($inputFilter)`
2. `$form->useInputFilter($inputFilter)`
3. `$form->assignInputFilter($inputFilter)`
4. `$form->mapInputFilter($inputFilter)`

Summary

In this chapter we have learned creating forms, doing basic validations, storing form data to the database, using models, and authenticating with the database. In the next chapter we will be learning about advanced database operations, which will be based on the `TableGateway` pattern that we have covered in this chapter.

4

Data Management and Document Sharing

After getting ready to write your own basic models in the previous chapters, you can now learn how to make the most out of your Zend Framework's data and file management concepts in this chapter.

In this chapter we will cover the following key topics:

- Zend Framework 2 ServiceManager
- The `TableGateway` pattern
- File uploads and file sharing using Zend Framework

Zend Framework 2 ServiceManager

The ZF2 ServiceManager implements the service locator design pattern. The service locator is a service/object locator used for retrieving other objects.

The ServiceManager configurations are classified into six main categories; your application/module configuration will fall under one or more of the categories listed in the following table:

Configuration type	Description
abstract_factories	Used to define an array of abstract classes.
aliases	Used to define an associative array of alias name / target name pairs.

Configuration type	Description
factories	Used to define an array of service name / factory class name pairs. The factory classes defined here should either implement Zend/ServiceManager/FactoryInterface or invokable classes.
invokables	Used to define an array of service name / class name pairs. The classes listed here may be directly instantiated without any constructor arguments.
services	Used to define an array of service name / object pairs. The service is basically an instance of a class. Services can be used to register classes which are already initialized.
shared	Used to define an array of service name / Boolean pairs, indicating whether or not a service should be shared. All services are shared by default; this ServiceManager option can be used to disable sharing on specific services.

The ServiceManager configuration can be stored either in the application configuration or in the module configuration; this can be chosen according to the needs, application, or module. Usually, the configuration, which is static across the application, is stored in the application-level configuration; all other information is stored at a module level.

The configuration for ServiceManager is merged in the following order:

1. Module configuration provided by the `Module` lass using the `getServiceConfig()` method. This will be processed in the same order in which the modules are processed:

```
public function getServiceConfig()
{
    return array(
        'abstract_factories' => array(),
        'aliases' => array(),
        'factories' => array(),
        'invokables' => array(),
        'services' => array(),
        'shared' => array(),
    );
}
```

2. Module configuration is present in the `service_manager` key; again, this is processed in the same order in which the modules are processed.
3. Application configuration is present in various configuration files in the `config/autoload/` directory in the order in which they are processed:

```
<?php
return array(
    'service_manager' => array(
```

```
            'abstract_factories' => array(),
            'aliases' => array(),
            'factories' => array(),
            'invokables' => array(),
            'services' => array(),
            'shared' => array(),
        ),
    );
```

Time for action – migrating existing code to ServiceManager

Our next step will be to migrate existing code blocks to make use of ServiceManager. Some of the key factories that can be moved into ServiceManager are as follows:

- Database connections
- Models and table gateways
- Forms and filters
- Authentication service

If you review the existing code, you will be able to figure out that all the database connections are already using the Zend Framework 2 ServiceManager model for storing credentials. We will take one step forward and move the rest of the factories into ServiceManager using the following steps:

1. Modify `Module.php` and add a new function to load the ServiceManager configuration:

```
public function getServiceConfig()
{
  return array(
    'abstract_factories' => array(),
    'aliases' => array(),
    'factories' => array(

      // DB
      'UserTable' =>  function($sm) {
        $tableGateway = $sm->get('UserTableGateway');
        $table = new UserTable($tableGateway);
        return $table;
      },
      'UserTableGateway' => function ($sm) {
        $dbAdapter = $sm->get('Zend\Db\Adapter\Adapter');
        $resultSetPrototype = new ResultSet();
        $resultSetPrototype->setArrayObjectPrototype(new User());
```

```
            return new TableGateway('user', $dbAdapter, null,
            $resultSetPrototype);
          },

          // FORMS
          'LoginForm' => function ($sm) {
            $form = new \Users\Form\LoginForm();
            $form->setInputFilter($sm->get('LoginFilter'));
            return $form;
          },
          'RegisterForm' => function ($sm) {
            $form = new \Users\Form\RegisterForm();
            $form->setInputFilter($sm->get('RegisterFilter'));
            return $form;
          },

          // FILTERS
          'LoginFilter' => function ($sm) {
            return new \Users\Form\LoginFilter();
          },
          'RegisterFilter' => function ($sm) {
            return new \Users\Form\RegisterFilter();
          },
        ),
        'invokables' => array(),
        'services' => array(),
        'shared' => array(),
      );
    }
```

2. Make sure that the `Module.php` file includes all the necessary namespaces:

```
use Users\Model\User;
use Users\Model\UserTable;

use Zend\Db\ResultSet\ResultSet;
use Zend\Db\TableGateway\TableGateway;
```

Using namespaces

Namespaces can be utilized by making use of PHP 5.3's `namespace` and `use` keywords. All ZF2 classes have a namespace which directly matches with the folder structure of the folder holding that class; all classes stored within that folder are directly determined by their namespace.

By default, the `use` keyword creates an alias for the last segment of the namespace, and this can be changed by using the `as` option on the keyword. For example, see the following code:

```
use Zend\Form\Element as Element;
use Zend\Form\Element; // same as previous line
```

3. Make necessary changes to the controllers to fetch the instances from ServiceManager:

```
// to get Login Form
$form = $this->getServiceLocator()->get('LoginForm');

// to get User Table
$userTable = $this->getServiceLocator()->get('UserTable');
```

4. To check if the changes are working as expected, try to register and log in with new credentials.

What just happened?

We have migrated our code to make use of Zend's ServiceManager framework. ServiceManager provides enormous benefits in terms of a cleaner code, highly effective refactoring ability, and a centralized register for core application components.

Have a go hero

Now that you have understood Zend ServiceManager functionality, here is a simple task for you. The login controller (`CommunicationApp/module/Users/src/Users/Controller/LoginController.php`) makes use of `getAuthService()` for the authentication service. Modify the function, so that the authentication service is obtained from ServiceManger.

Database operations

In the previous chapter we learned how to implement a basic database operation, namely, `table insert`. In this section, you will learn all the basic database operations necessary for building a simple **CRUD** (**Create, Read, Update and Delete**) interface.

More on TableGateway

The `TableGateway` class extends `AbstractTableGateway`, which implements `TableGatewayInterface`. The interface definition of `TableGatewayInterface` is provided in the following code snippet; all the basic table operations are defined in the interface:

```
interface Zend\Db\TableGateway\TableGatewayInterface
{
    public function getTable();
    public function select($where = null);
    public function insert($set);
    public function update($set, $where = null);
    public function delete($where);
}
```

The `TableGateway` class offers a wide range of methods to perform basic database operations; some of the most frequently used methods are explained in the following section:

- `getTable()`: Returns a string which contains the table name mapped with the `TableGateway` object. For example, see the following code:

  ```
  $myTableName = $myTableGateway->getTable();
  ```

- `select($where = null)`: Used to select a set of rows with the criteria specified in `$where`; it can either be a `where` condition based on `Zend\Db\Sql\Where` or an array of criteria. For example, see the following code:

  ```
  $rowset = $myTableGateway->select( array('id' => 2));
  ```

- `insert($set)`: Used to insert the data defined in `$set` into the table as a new record. For example, see the following code:

  ```
  $myTableGateway->insert( array('id' => 2, 'name'=>'Ravi'));
  ```

- `update($set, $where = null)`: Used to update a set of rows with the criteria specified in `$where`; it can either be a `where` condition based on `Zend\Db\Sql\Where` or an array of criteria. `$set` holds the data that will be updated for all the records matched with `$where`. For example, see the following code:

  ```
  $rowset = $myTableGateway->update(array('name' => 'Jerry') ,
  array('id' => 2));
  ```

- `delete($where)`: Used to delete a set of rows with the criteria specified in `$where`; it can either be a `where` condition based on `Zend\Db\Sql\Where` or an array of criteria. For example, see the following code:

  ```
  $myTableGateway->delete( array('id' => 2));
  ```

- `getLastInsertValue()`: Returns the last `insert` value for the table's primary key. the return type is an integer. For example, see the following code:

```
$myTableGateway->insert( array('name'=>'Ravi'));
$insertId = $myTableGateway-> getLastInsertValue ();
```

Time for action – implementing an admin UI to manage users

In this task we will be creating an administration user interface for managing users in our application. The following operations will include listing all users, editing existing users, deleting users, and adding users:

1. Modify `CommunicationApp/module/Users/src/Users/Model/UserTable.php` using the following code. Add the following functions:
   - `fetchAll()`
   - `getUser($id)`
   - `getUserByEmail($userEmail)`
   - `deleteUser($id)`

```
public function fetchAll()
{
  $resultSet = $this->tableGateway->select();
  return $resultSet;
}

public function getUser($id)
{
  $id  = (int) $id;
  $rowset = $this->tableGateway->select(array('id' => $id));
  $row = $rowset->current();
  if (!$row) {
    throw new \Exception("Could not find row $id");
  }
  return $row;
}

public function getUserByEmail($userEmail)
{
  $rowset = $this->tableGateway->select(array('email' =>
  $userEmail));
  $row = $rowset->current();
  if (!$row) {
    throw new \Exception("Could not find row $ userEmail");
  }
```

```
    return $row;
  }

  public function deleteUser($id)
  {
    $this->tableGateway->delete(array('id' => $id));
  }
```

2. Create a new controller for user management under `CommunicationApp/module/Users/src/Users/Controller/UserManagerController.php`.
3. The `UserManagerController` controller will have the following actions:
   - `indexAction()`: This is used to render all available users in the system, and we will also render links to add/edit and delete links as shown in the following code:

     ```
     $userTable = $this->getServiceLocator()
                                  ->get('UserTable');
     $viewModel  = new ViewModel(array(
                     'users' => $userTable->fetchAll()));
     return $viewModel;
     ```

   - `editAction()`: This action is used to render the `edit` form to modify the information related to the user:

     ```
     $userTable = $this->getServiceLocator()
                                  ->get('UserTable');
     $user = $userTable->getUser(
                          $this->params()->fromRoute('id'));
     $form = $this->getServiceLocator()
                            ->get('UserEditForm');
     $form->bind($user);
     $viewModel  = new ViewModel(array(
               'form' => $form,
               'user_id' => $this->params()->fromRoute('id')
              ));
     return $viewModel;
     ```

The bind method

The `bind` method used in the `Form` function allows the mapping of the model to a form. The function works in two directions—it updates the form in the view with the data from the model and it updates the model with the form submission data if the form is validated, that is, `$form->isValid()`.

Read more here:

`http://framework.zend.com/manual/2.2/en/modules/zend.form.quick-start.html#binding-an-object`

- `processAction()`: The `processAction` action is used when the user `edit` form is submitted; `processAction` saves the updated record and returns to `indexAction`:

```
// Get User ID from POST
$post = $this->request->getPost();
$userTable = $this->getServiceLocator()
                              ->get('UserTable');
// Load User entity
$user = $userTable->getUser($post->id);

// Bind User entity to Form
$form = $this->getServiceLocator()
                        ->get('UserEditForm');
$form->bind($user);
$form->setData($post);

// Save user
$this->getServiceLocator()
          ->get('UserTable')->saveUser($user);
```

- `deleteAction()`: This action is used to delete the user record:

```
$this->getServiceLocator()->get('UserTable')
        ->deleteUser($this->params()
        ->fromRoute('id'));
```

4. Create the necessary views and modify the module's `config/module.config.php` file to specify a unique child route to access this controller:

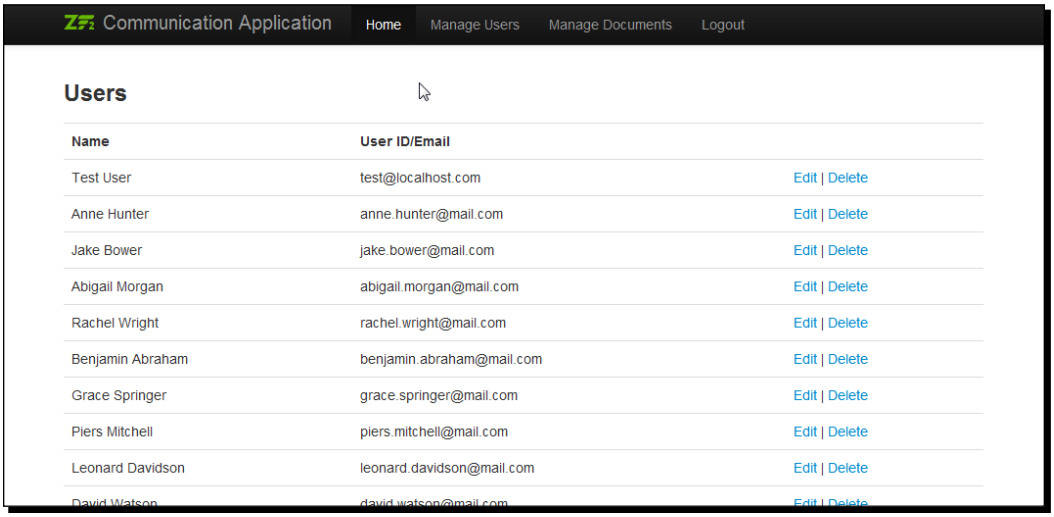
```
'user-manager' => array(
  'type'    => 'Segment',
  'options' => array(
    'route'    => '/user-manager[/:action[/:id]]',
    'constraints' => array(
      'action'     => '[a-zA-Z][a-zA-Z0-9_-]*',
      'id'     => '[a-zA-Z0-9_-]*',
```

```
        ),
        'defaults' => array(
          'controller' => 'Users\Controller\UserManager',
          'action'     => 'index',
        ),
      ),
    ),
```

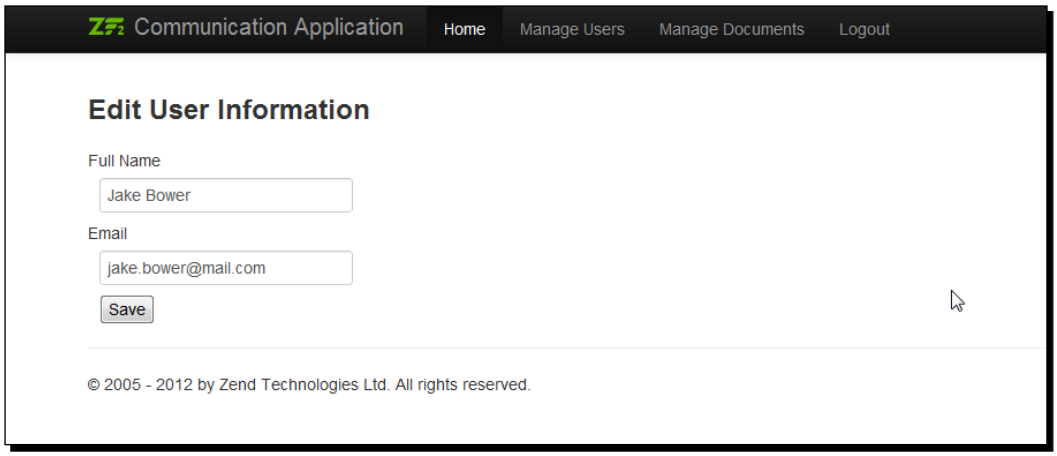
5. Finally add the new controller to the `invokables` array:

```
'Users\Controller\UserManager' => 'Users\Controller\
UserManagerController',
```

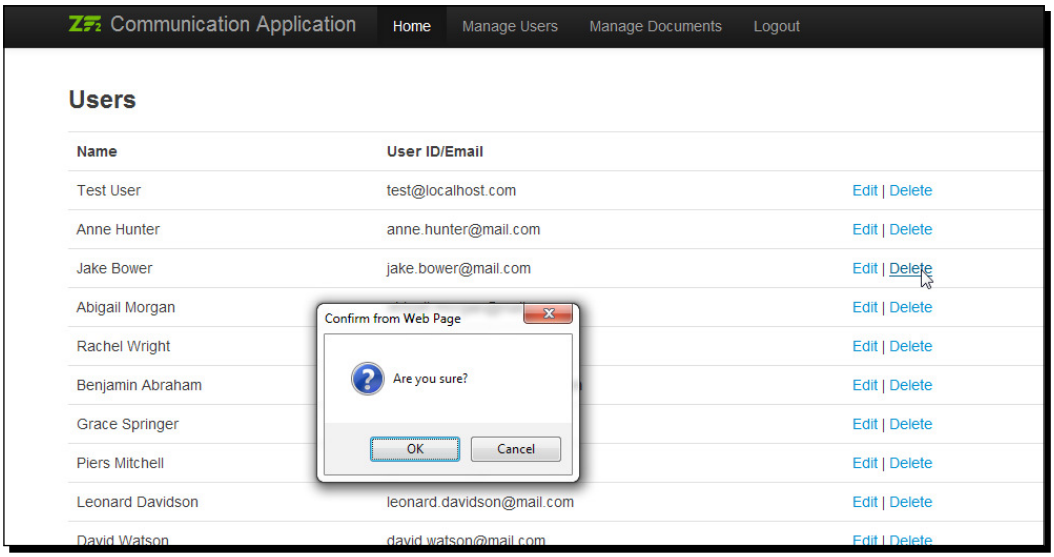
6. Now open your web browser and access the controller, log in to your application, and open `http://comm.-app.local/users/user-manager`. You should be able to see a page similar to the one given in the following screenshot:



The **Edit user** link should redirect you to an user edit form like the one in the following screenshot:



The **Delete user** link can be used to remove the user from the user list:



What just happened?

We have now created an administration user interface for adding, modifying, and removing users from our communication application. We have utilized all the core functionalities of the `TableGateway` model and created functions for performing CRUD operations on the table access objects.

Going forward, we will be making use of some of the more advanced applications of `TableGateway`.

Have a go hero

Before we move on to the next section, here is a small task for you to practice. Your task for this section will be to create a new `Add User` form. Refer to the following screenshot:

This form will be similar to the `Register Form` that we created in the previous chapter. Once the form is submitted, the user will be taken back to the user listing page. A link to this form will have to be added in the user listing page.

Document management

In this section we will create a new document management interface. The document management interface will allow users to upload documents, manage uploads, and share uploaded documents with other users. The user interface will also allow users to manage sharing, and add/remove shares.

In this section, we will focus on providing users with options to create file uploads and manage those uploads. We will be using the filesystem to store the uploaded file and the relative path of the uploaded file will be stored in the database mapped to the user who uploaded the file.

Some of the important Zend Framework components used in file uploads are:

- File upload form element (`Zend\Form\Element\File`): The `File` upload element is used in the upload form to display a file input box. This element is an equivalent of the `<input type='file'../>` style element in HTML used for allowing users to upload files. The file input element can be rendered by setting `'type' => 'file'` in the form definition.
- File transfer adapter (`Zend\File\Transfer\Adapter\Http`): The file transfer adapter handle file uploads upon form submission. The `setDestination()` method in the file transfer adapter allows the user to set a destination and receive the file in that destination. The `receive()` method is used to initiate the transfer.

Time for action – creating a file upload form

In this task, we will be creating a new document upload form; file uploads will be stored in the filesystem, and the information regarding the file upload will be stored in the database in a table named `uploads`. The file uploads are stored in a folder location defined in the module configuration. Perform the following steps to do so:

1. Our first step will be to define a location where files can be uploaded in the module's configuration (`config/module.config.php`):

```
<?php
return array(
  // Other configurations
  // ..
  // ..
  // MODULE CONFIGURATIONS
  'module_config' => array(
    'upload_location' => __DIR__ . '/../data/uploads',
  ),
);
```

2. Next, we need to create a table which will store the upload information:

```
CREATE TABLE IF NOT EXISTS uploads (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
  filename VARCHAR( 255 ) NOT NULL ,
  label VARCHAR( 255 ) NOT NULL ,
  user_id INT NOT NULL,
  UNIQUE KEY (filename)
);
```

3. Create the `Upload` and `UploadTable` classes for interacting with the `uploads` table. Add default methods such as `saveUpload()`, `fetchAll()`, `getUpload()`, and `deleteUpload()`. Also, add a method to get uploads made by a specific user `getUploadsByUserId($userId)`:

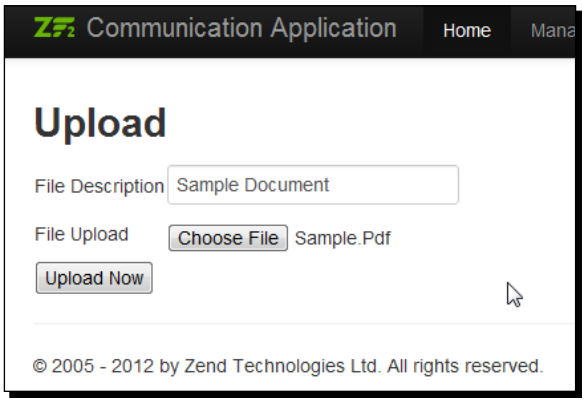
```
public function getUploadsByUserId($userId)
{
  $userId  = (int) $userId;
  $rowset = $this->tableGateway->select(
        array('user_id' => $userId));
  return $rowset;
}
```

4. Create an `UploadManagerController` controller for managing file uploads. Add `indexAction()` to display the list of uploads done by the user:

```
$uploadTable = $this->getServiceLocator()
                            ->get('UploadTable');
$userTable = $this->getServiceLocator()
                            ->get('UserTable');
// Get User Info from Session
$userEmail = $this->getAuthService()
                            ->getStorage()->read();
$user = $userTable->getUserByEmail($userEmail);

$viewModel  = new ViewModel( array(
'myUploads' => $uploadTable->getUploadsByUserId($user->id),
));
return $viewModel;
```

5. Create an upload form with a file input as described in the following code snippet:

```
$this->add(array(
  'name' => 'fileupload',
  'attributes' => array(
    'type'  => 'file',
  ),
  'options' => array(
    'label' => 'File Upload',
  ),
));
```



Upload form

6. Create views for the file upload form, and the `index` action. Now we have all the necessary elements to handle a file upload. We need to read the configuration for the file upload path and use the Zend HTTP file transfer adapter to receive the file in the configuration location. The `get('config')` method on the service locator is used to retrieve the configuration. The following code is used to read the file upload location from the configuration:

```
public function getFileUploadLocation()
{
  // Fetch Configuration from Module Config
  $config  = $this->getServiceLocator()->get('config');
  return $config['module_config']['upload_location'];
}
```

7. The last step is to handle the file upload process. There are two actions that need to happen once the form is successfully submitted:

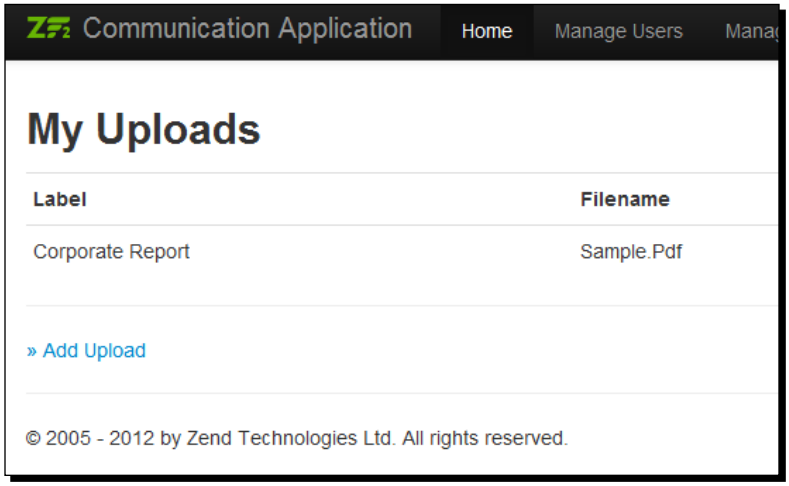
1. The uploaded file has to be moved to the file upload locations.
2. An entry needs to be added describing the upload in the `'uploads'` table using the following code:

```
$uploadFile = $this->params()->fromFiles('fileupload');
$form->setData($request->getPost());

if ($form->isValid()) {
  // Fetch Configuration from Module Config
  $uploadPath    = $this->getFileUploadLocation();

  // Save Uploaded file
  $adapter = new \Zend\File\Transfer\Adapter\Http();
  $adapter->setDestination($uploadPath);
  if ($adapter->receive($uploadFile['name'])) {
    // File upload sucessfull
    $exchange_data = array();
    $exchange_data['label'] = $request->getPost()
                                    ->get('label');
    $exchange_data['filename'] = $uploadFile['name'];
    $exchange_data['user_id'] = $user->id;

    $upload->exchangeArray($exchange_data);
    $uploadTable = $this->getServiceLocator()
                            ->get('UploadTable');
    $uploadTable->saveUpload($upload);

    return $this->redirect()
            ->toRoute('users/upload-manager' ,
                  array('action' =>  'index'
              ));
  }
}
```

8. Add a child route (upload manger) for the `UploadManager` controller and the controller to the `invokables` list.

9. Open the web browser and test the upload form.

The final form will look like the following screenshot:

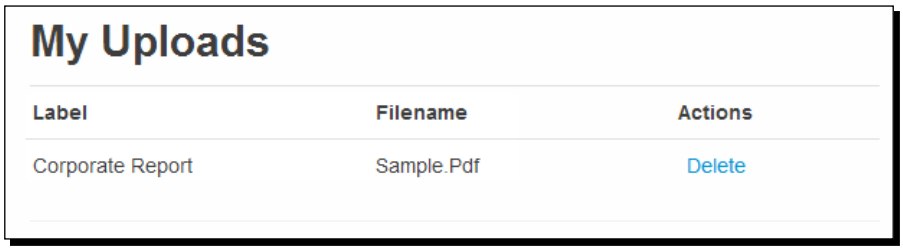


What just happened?

We have now created a file upload process, which allows users to upload files into the application and view the files that are uploaded. We have used Zend Framework's file upload handling components to handle a file upload. In our next section, we will set up a file sharing mechanism such that the documents can be shared with different users. Before we move on to implement file sharing, please complete the following task.

Have a go hero

Your next task will be to add a **Delete** option that allows users to delete uploaded files as shown in the following screenshot. Also, ensure that the file is removed from the filesystem when the delete action is triggered.



Managing file sharing

Now that we have a fully functional document management section, our next task is to extend this document management system to support file sharing with other users. The most important part of implementing a file sharing mechanism is to store the information about upload sharing; we do this by linking documents with user IDs in a table called `upload_sharing`.

Time for action – implementing a file sharing system

For implementing file sharing, we will need to create a new table called `upload_sharing` and store all sharing-related information in that table. The following steps will explain how this is implemented in our application:

1. Create a new table called `upload_sharing`; this table will hold the relationship about uploads shared with users:

```
CREATE TABLE IF NOT EXISTS uploads_sharing (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
  upload_id INT NOT NULL ,
  user_id INT NOT NULL,
  UNIQUE KEY (upload_id, user_id)
);
```

2. In the module definition `Module.php`, add a simple `TableGateway` object for the `uploads_sharing` table:

```
'UploadSharingTableGateway' => function ($sm) {
  $dbAdapter = $sm->get('Zend\Db\Adapter\Adapter');
  return new TableGateway('uploads_sharing', $dbAdapter);
},
```

3. Modify the constructor of the `UploadTable` class to take in an additional parameter of the upload sharing `TableGateway` object:

```
public function __construct(TableGateway $tableGateway,
                TableGateway $uploadSharingTableGateway)
{
    $this->tableGateway = $tableGateway;
    $this->uploadSharingTableGateway = $uploadSharingTableGateway;
}
```

4. Modify the module configuration (`Module.php`) for the `UploadTable` factory to support `UploadSharingTableGateway`:

```
'UploadTable' =>  function($sm) {
  $tableGateway = $sm->get('UploadTableGateway');
```

```
  $uploadSharingTableGateway = $sm->get('UploadSharingTableGatew
ay');
  $table = new UploadTable($tableGateway,
$uploadSharingTableGateway);
  return $table;
},
```

5. Modify the `UploadTable` class to support the following file sharing functions:

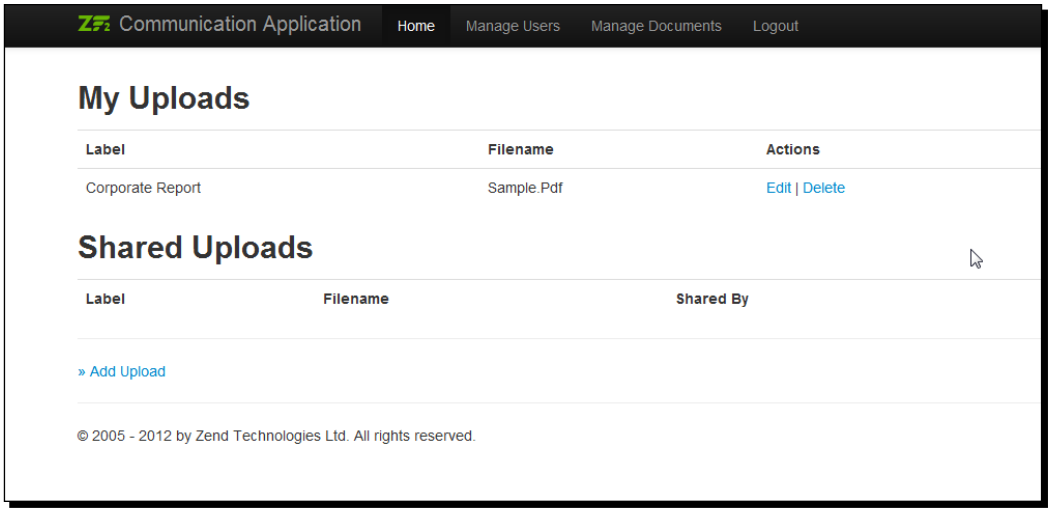
- `addSharing()`: Adds a new sharing permission for the given upload with the user
- `removeSharing()`: Removes the sharing permission for the specific upload/user combination
- `getSharedUsers()`: Gets the list of users for which the upload is shared
- `getSharedUploadsForUserId()`: Gets the list of uploads that are shared for that user

This can be done using the following code:

```
public function addSharing($uploadId, $userId)
{
  $data = array(
    'upload_id' => (int)$uploadId,
    'user_id'  => (int)$userId,
  );
  $this->uploadSharingTableGateway->insert($data);
}

public function removeSharing($uploadId, $userId)
{
  $data = array(
    'upload_id' => (int)$uploadId,
    'user_id'  => (int)$userId,
  );
  $this->uploadSharingTableGateway->delete($data);
}

public function getSharedUsers($uploadId)
{
  $uploadId  = (int) $uploadId;
  $rowset = $this->uploadSharingTableGateway->select(
                      array('upload_id' => $uploadId));
  return $rowset;
```

```
}

public function getSharedUploadsForUserId($userId)
{
  $userId  = (int) $userId;

  $rowset = $this->uploadSharingTableGateway->select(
  function (Select $select) use ($userId){
    $select->columns(array())
      ->where(array('uploads_sharing.user_id'=>$userId))
      ->join('uploads', 'uploads_sharing.upload_id = uploads.id');
  });

  return $rowset;
}
```

The **Manage Documents** section lists all uploads for a specific user and also lists uploads shared by others with the user:

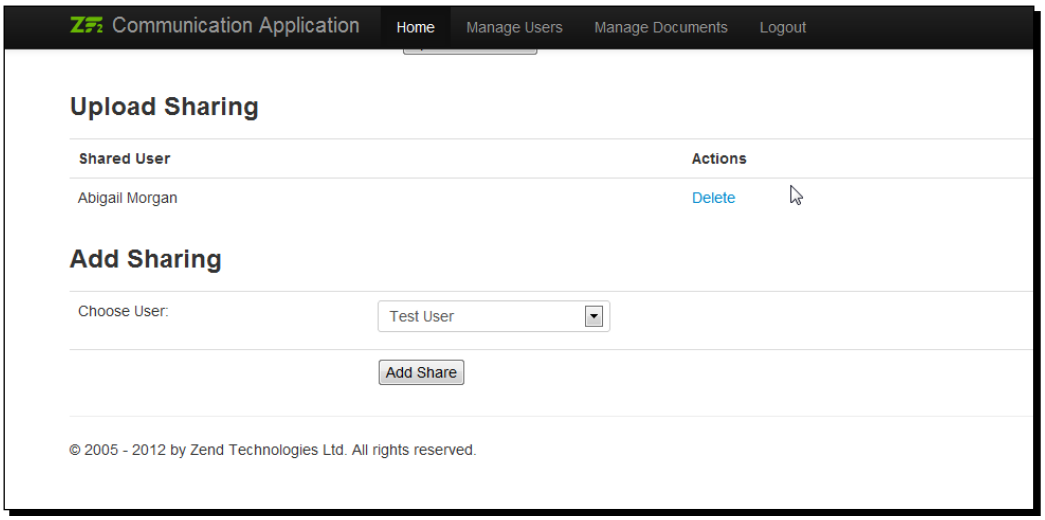


6. Modify the edit upload form to display the list of users the upload is shared with; this can be achieved by passing the upload ID to the `getSharedUsers()` method of the `UploadTable` object.

7. Add a new section in the edit upload form which allows the addition of new shares; this is achieved by displaying the list of all users in the system in a drop-down list. When the user clicks on **Add Share**, a new record is added to the `upload_sharing` table:

```
$userTable = $this->getServiceLocator()
              ->get('UserTable');
$uploadTable = $this->getServiceLocator()
              ->get('UploadTable');
$form = $this->getServiceLocator()->get('UploadForm');
$request = $this->getRequest();
if ($request->isPost()) {
  $userId = $request->getPost()->get('user_id');
  $uploadId = $request->getPost()->get('upload_id');
  $uploadTable->addSharing($uploadId, $userId);
}
```

The following screenshot shows the **Upload Sharing** page with a drop-down list to add shares:



8. The last section of the file sharing implementation is to allow an option for users to download shared files. This is provided by the `fileDownloadAction()` function defined in our file sharing application:

```
public function fileDownloadAction()
{
  $uploadId = $this->params()->fromRoute('id');
  $uploadTable = $this->getServiceLocator()
                            ->get('UploadTable');
```

```
    $upload = $uploadTable->getUpload($uploadId);

    // Fetch Configuration from Module Config
    $uploadPath    = $this->getFileUploadLocation();
    $file = file_get_contents($uploadPath ."/" . $upload->filename);

    // Directly return the Response
    $response = $this->getEvent()->getResponse();
    $response->getHeaders()->addHeaders(array(
      'Content-Type' => 'application/octet-stream',
      'Content-Disposition' => 'attachment;filename="'
      .$upload->filename . '"',

    ));
    $response->setContent($file);

    return $response;
  }
```

File download

For implementing a file download, we need to disable the layout. This can be achieved by directly providing the HTTP response object as output for that particular action as shown in the previous code. This can also be achieved by `setTerminal()`, as shown in the following code:

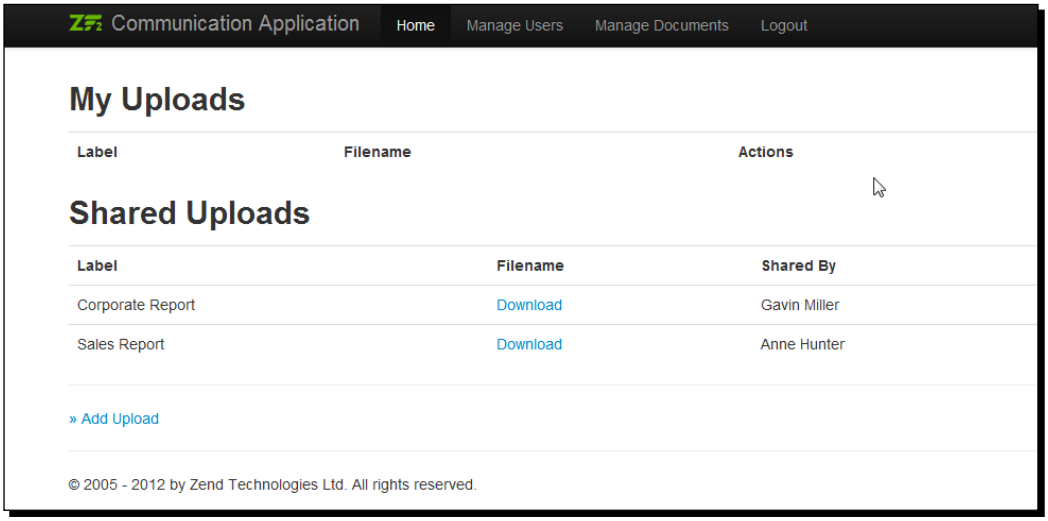
```
$result = new ViewModel();
$result->setTerminal(true);
return $result;
```

Large file downloads

The `file_get_contents()` method is capable of handling small file uploads and consume a lot of memory when processing large files. For better performance, you can create a stream HTTP response object `Zend\Http\Response\Stream()` and stream the file download.

9. Now we have a fully functional file sharing system in place. Test the file sharing system; start by sharing the file with different users, and log in and out as different users.

The final form should look like the following screenshot:



What just happened?

You created a table that can store user and upload relationships; you modified the `UploadTable` class to support additional sharing functions. You created controllers and views to enable file sharing, and finally you provided the ability for the user to download the shared file using a file download script. With this, you have successfully implemented the file sharing system, where users can now upload, edit, and share documents within the system.

Pop quiz – data management and document sharing

Q1. In `TableGateway`, which function is used to determine the last inserted record ID?

1. `getLastId()`
2. `getLastInsertId()`
3. `get('last_insert_id')`
4. `getLastInsertValue()`

Q2. Which method can be used to disable layouts in a view model?

1. `$viewModel->setNoLayouts(true)`
2. `$ viewModel->Layouts(false)`
3. `$viewModel->setTerminal(true)`
4. `$viewModel->setLayouts(false)`

Summary

In this chapter, we have discussed several topics in the context of data and file management. First, we elaborated on the usage of the `TableGateway` database pattern. We then implemented a simple file upload service by making use of Zend Framework's file transfer components. Finally, we implemented a simple file sharing service by utilizing both Zend Framework's file transfer components and the `TableGateway` pattern. In the next chapter, we will be working closely on the frontend, especially with JavaScript and AJAX calls.

5
Chat and E-mail

In any web application development, there will be very high dependency on client-side scripts primarily including JavaScript and CSS. The MVC model of Zend Framework provides basic support of controlling the output that is sent across to the browser. The view helper classes in Zend Framework 2 offer maximum control over the content that gets rendered in the client browser.

In this chapter we will focus on building a simple group chat and e-mail component which will make use of various frontend capabilities of Zend Framework 2.0. Some of the important topics covered in this chapter include:

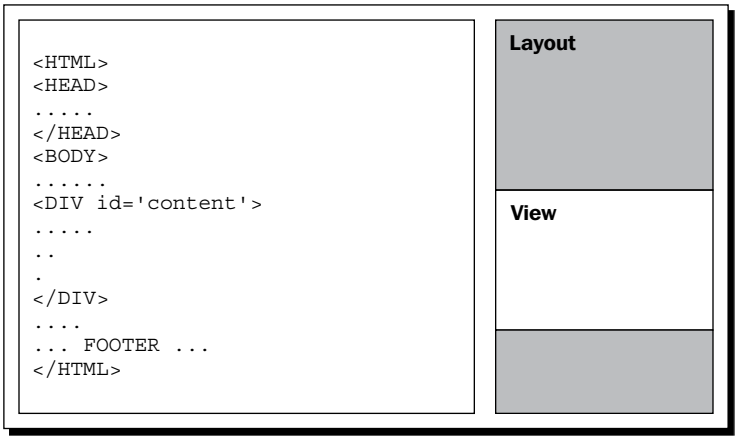
- Using external JavaScript libraries in the Zend Framework 2 application
- Implementing a simple group chat application using Zend Framework 2 and JavaScript
- Using Zend\Mail to send e-mails
- Introduction to the Zend Framework event manager

Layouts and views

Zend Framework MVC uses layouts and views to render pages in the web browser; the overall page content is controlled by the layout specification, and the view level information is contained in the views. The concept is to minimize the amount of redundant HTML code that needs to be generated for each of these views.

By using layouts, the application can have a consistent user interface, which is also easy to customize; the views offer the flexibility to modify the targeted content and allow customization to the maximum possible extent. This is also known as *two-step* view.

When a new view is generated, the appropriate layout is identified from the layout definitions in the `view_manager` configuration and the view is rendered with that layout.



The preceding schematic explains how the layout and view are combined to form an HTML page, so for each and every view, the view part changes and the layout part remains static.

View helpers

Zend Framework 2 offers a wide range of view helpers that help us perform complex operations on views; if the included helpers are not sufficient, you can define your own custom helper by implementing the interface `Zend\View\HelperInterface`.

In this section, we will quickly review some of the included helpers in Zend Framework 2.

The URL helper

The syntax for this helper is `url($name, $urlParams, $routeOptions = array(), $reuseMatchedParams = array())`.

The URL helper is used to generate the URL for a specific route. The route's segment match parameters can be passed over the URL helper to form a URL based on the route option; for example, see the following:

```
<a href="<?php $this->url('users/upload-manager',
array('action'=>'edit', 'id' =>  10));">Edit</a>
```

This code will generate `<a href="/users/upload-manager/edit/10">Edit</a>` if the route definition is as follows:

```
'route' => '/user-manager[/:action[/:id]]'
```

The BasePath helper

The syntax for this helper is `basePath()`.

The `BasePath` helper returns the base URL of the view, this can be used by developers to prepend to their custom URLs and form links for various resources.

The JSON helper

The syntax for this helper is `json($jsonData = array())`.

The JSON helper is used to render PHP arrays as JSON-encoded data. Most AJAX libraries classify JSON content by its content header, and this helper also sets the content type header to `application/json`.

Concrete placeholder implementations

Zend Framework makes use of placeholder helpers to perform some standard operations on the HTML `head` sections including adding/removing references to new JavaScript libraries, linking with new styles, adding and cross referencing scripts, and adding/removing HTML `head` section's `meta` content.

This is achieved by the following list of helpers called as **concrete placeholder helpers**. The reason why they are called placeholder helpers is because the helpers themselves don't make any changes to the way in which the content is rendered. For example, if you add `<?php echo $this->headLink(); ?>` to the HTML code, this won't do anything, until you add something to the `headLink` helper by using `appendStylesheet` or some other function.

The HeadLink helper

The `HeadLink` helper is used to modify the `<link>` tag in the HTML `head` section; this helper is used to attach or manage external CSSs.

Some of the most-used functions in this helper are listed as follows:

- `appendStylesheet($href, $media, $conditionalStylesheet, $extras)`
- `offsetSetStylesheet($index, $href, $media, $conditionalStylesheet, $extras)`
- `prependStylesheet($href, $media, $conditionalStylesheet, $extras)`
- `setStylesheet($href, $media, $conditionalStylesheet, $extras)`

To render the `Link` tags in an HTML layout/view, use the following script:

```
<?php echo $this->headLink(); ?>
```

The HeadMeta helper

The `HeadMeta` helper is used to modify the `<meta>` tag in the HTML `head` section; this helper is used to manipulate the HTML `meta` information.

Some of the most-used functions in this helper are listed as follows:

- `appendName($keyValue, $content, $conditionalName)`
- `offsetSetName($index, $keyValue, $content, $conditionalName)`
- `prependName($keyValue, $content, $conditionalName)`
- `setName($keyValue, $content, $modifiers)`
- `appendHttpEquiv($keyValue, $content, $conditionalHttpEquiv)`
- `offsetSetHttpEquiv($index, $keyValue, $content, $conditionalHttpEquiv)`
- `prependHttpEquiv($keyValue, $content, $conditionalHttpEquiv)`
- `setHttpEquiv($keyValue, $content, $modifiers)`
- `setCharset($charset)`

To render the `meta` tags in an HTML layout/view, use the following script:

```
<?php echo $this->headMeta(); ?>
```

The HeadScript helper

The `HeadScript` helper is used to modify the `<script>` tag in the HTML `head` section; this helper is used to attach external JavaScript and also add the `<script>` tags to the HTML `head` section.

Some of the most-used functions in this helper are listed as follows:

- `appendFile($src, $type = 'text/javascript', $attrs = array())`
- `offsetSetFile($index, $src, $type = 'text/javascript', $attrs = array())`
- `prependFile($src, $type = 'text/javascript', $attrs = array())`
- `setFile($src, $type = 'text/javascript', $attrs = array())`
- `appendScript($script, $type = 'text/javascript', $attrs = array())`

- `offsetSetScript($index, $script, $type = 'text/javascript', $attrs = array())`
- `prependScript($script, $type = 'text/javascript', $attrs = array())`
- `setScript($script, $type = 'text/javascript', $attrs = array())`

To render the `Script` tags in an HTML layout/view use the following script:

```
<?php echo $this->headScript(); ?>
```

The HeadStyle helper

The `HeadStyle` helper is used to modify the `<style>` tag in HTML `head` section; this helper is used to add internal styles by adding the `<style>` tags to the HTML `head` section.

Some of the most-used functions in this helper are listed as follows:

- `appendStyle($content, $attributes = array())`
- `offsetSetStyle($index, $content, $attributes = array())`
- `prependStyle($content, $attributes = array())`
- `setStyle($content, $attributes = array())`

To render the `Style` tags in an HTML layout/view use the following script:

```
<?php echo $this->headStyle(); ?>
```

The HeadTitle helper

The `HeadTitle` helper is used to render title in the `<title>` tags on the HTML `head` section; multiple calls to a `headTitle()` helper create a list of titles which are rendered when tag is outputted in the layout/view. The optional parameter `$setType` can be set to override the pre-existing array of titles, the default is `APPEND`, it can be overridden to `PREPEND` or `SET`(overwrite).

The syntax for this helper is `headTitle($title, $setType = null);`.

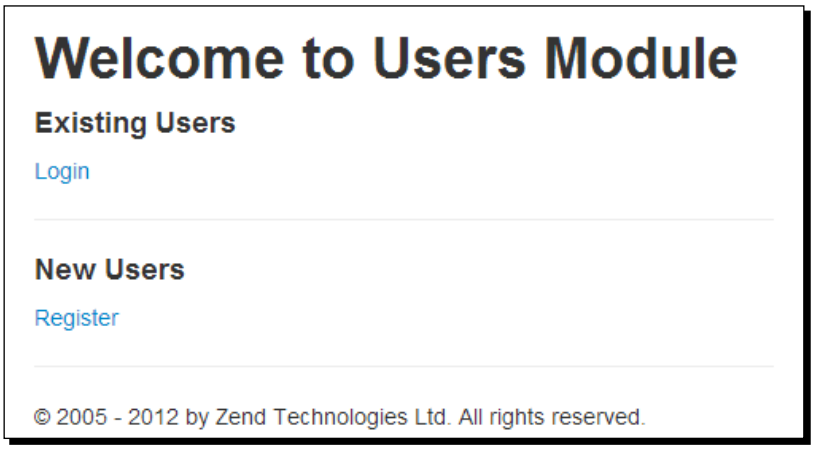
To render the `Title` tags in an HTML layout/view, use the following script:

```
<?php echo $this->headTitle(); ?>
```

Time for action – using jQuery UI in a simple page

In this task we will be converting some of our existing pages to make use of the jQuery UI library and render buttons in that page using jQuery UI:

1. View the existing application home page as shown in the following screenshot; our next task is to convert the **Login** and **Register** links to render as jQuery UI buttons:



Existing application home page

2. Replace the **Login** and **Register** links in the `index` view (`module/Users/view/users/index/index.html`), and add the `ui-button` class to the links as shown in the following code snippet:

```
<a href="/users/login" class='ui-button'>Login</a>
<a href="/users/register" class='ui-button'>Register</a>
```

3. Add external references to jQuery UI towards the beginning of the view:

```
// Attached jQuery UI Scripts
$this->headScript()
->appendFile('http://code.jquery.com/jquery-1.8.3.js','text/
javascript');

$this->headScript()
->appendFile('http://code.jquery.com/ui/1.10.0/jquery-ui.
js','text/javascript');

// Attach jQuery UI Styles
$this->headLink()->appendStylesheet('http://code.jquery.com/
ui/1.10.0/themes/base/jquery-ui.css');
```

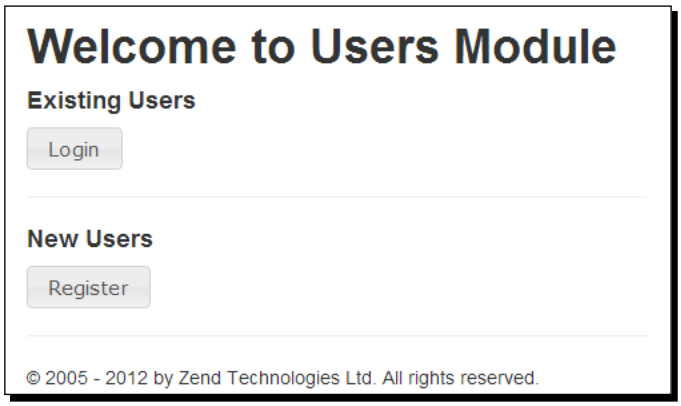
Referencing custom JavaScript libraries

Instead of directly referencing the external scripts, you can also optionally download the scripts to the `/public` folder in your application and pass relative links as parameters to the `appendFile` and `appendStylesheet` functions. You can also make use of the `basePath()` helper to prepend the base URL.

4. Add a UI initialization script to apply the button look and feel to both the links:

```
// UI Initializer for buttons
$this->headScript()->appendScript(
'$(function() {
    $("a.ui-button").button();
  });', 'text/javascript');
```

5. Preview the home page in the browser now, and you will be able to see that both the **Login** and **Register** buttons are styled using jQuery UI as shown in the following screenshot:



A **View Source** link on the index page will reveal the application of `headScript()` as shown in the following code:

```
<!DOCTYPE html>
<html lang="en">
...
...
<script type="text/javascript" src="http://code.jquery.com/jquery-
1.8.3.js"></script>
<script type="text/javascript" src="http://code.jquery.com/ui/1.10.0/
jquery-ui.js"></script>
<script type="text/javascript">
    //<!--
```

```
      $(function() {
     $("a.ui-button").button();
    });
      //-->
</script>
...
...
</html>
```

What just happened?

We have made use of Zend Framework's view helpers to connect to the external JavaScript library; we then added custom JavaScript to the HTML `head` section using the `headScript()` view helper.

Now we have integrated our application with an external JavaScript; in the next exercise we will learn a little bit more on how scripts can be added to the HTML `head` section.

Have a go hero

Before we move on to building the **Group Chat** interface, here is a simple task for you to complete. Now that you have understood how to link external JavaScript libraries, you can download jQuery UI from its website, extract it to the `public/` folder, and modify the previously listed page to use the downloaded version of jQuery UI.

jQuery UI can be downloaded from `http://jqueryui.com/`.

Building a simple group chat

Our next task is to build a simple group chat application that allows multiple users to log in to our system and chat with each other. The backend for this tool is pretty straightforward. We need to create a table that will store all user messages and render them in a separate view; we will create a simple form that will allow users to send messages.

Time for action – creating a simple group chat application

1. Create a new `chat_messages` table to store all user messages:

```
CREATE TABLE IF NOT EXISTS chat_messages (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
  user_id INT NOT NULL,
  message VARCHAR( 255 ) NOT NULL ,
  stamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)
```

2. Create a controller for group chat in `CommunicationApp/module/Users/src/Users/Controller/GroupChatController.php`.
3. Make necessary changes to `CommunicationApp/module/Users/config/module.config.php` and add the new controller to invokables and routes:

```
// Invokable
'Users\Controller\GroupChat' => 'Users\Controller\
GroupChatController',

// Route
'group-chat' => array(
  'type'    => 'Segment',
  'options' => array(
    'route'    => '/group-chat[/:action[/:id]]',
    'constraints' => array(
      'action'     => '[a-zA-Z][a-zA-Z0-9_-]*',
      'id'     => '[a-zA-Z0-9_-]*',
    ),
    'defaults' => array(
      'controller' => 'Users\Controller\GroupChat',
      'action'     => 'index',
    ),
  ),
),
```

4. Create a new view in `CommunicationApp/module/Users/view/users/group-chat/index.phtml`:

```
<?php
$this->headScript()->appendScript(
'$(function() {
    $( "#btnRefresh" )
      .click(function( event ) {
  document.getElementById("messageListFrame").contentWindow.
location.reload(true);
   })
  });', 'text/javascript');

$this->headStyle()->appendStyle('
  #userName { width:100px; margin-top:10px; display: inline}
  #messageText { width:700px; margin-top:10px;}
');
?>
<h3>Group Chat</h3>
<iframe src="<?php echo $this->url('users/group-chat', array(
```

```
                        'action' =>  'messageList'
                    )) ?>" width="80%" height="400px"
id="messageListFrame"></iframe>

<?php
// Render the opening tag
echo $this->form()->openTag($form);

// ...loop through and render the form elements...
echo '<label id="userName">'. $userName .': </label>';
foreach ($form as $element) {
    echo $this->formElement($element);       // <-- Magic!
    echo $this->formElementErrors($element);
}

// Render the closing tag
echo $this->form()->closeTag();
?>
```

5. Add the `messageList` action to `GroupChatController - CommunicationApp/module/Users/src/Users/Controller/GroupChatController.php`; this action will query the `chat_messages` table and get all the records from that table and pass that on to the view:

```
public function messageListAction()
{
  $userTable = $this->getServiceLocator()->get('UserTable');
  $chatMessageTG = $this->getServiceLocator()->get('ChatMessagesTa
bleGateway');
  $chatMessages = $chatMessageTG->select();

  $messageList = array();
  foreach($chatMessages as $chatMessage) {
    $fromUser = $userTable->getUser($chatMessage->user_id);
    $messageData = array();
    $messageData['user'] = $fromUser->name;
    $messageData['time'] = $chatMessage->stamp;
    $messageData['data'] = $chatMessage->message;
    $messageList[] = $messageData;
  }

  $viewModel  = new ViewModel(array('messageList' =>
$messageList));
  $viewModel->setTemplate('users/group-chat/message-list');
  $viewModel->setTerminal(true);
  return $viewModel;
}
```

6. Create a simple message listing view, `CommunicationApp/module/Users/view/users/group-chat/message-list.phtml`, which will list messages from the `$messageList` array:

```
<!DOCTYPE html>
<html lang="en">
<body>
<section id="messages" >
  <?php foreach ($messageList as $mesg) : ?>
  <div class="message" style="clear:both;">
    <span class='msg-time'>
  [<?php echo $this->escapeHtml($mesg['time']);?>]
    </span>
    <span class='msg-user'>
  <?php echo $this->escapeHtml($mesg['user']);?>:
    </span>
    <span class='msg-data'>
  <?php echo $this->escapeHtml($mesg['data']);?>
    </span>
  </div>
  <?php endforeach; ?>
</section>
</body>
</html>
```

7. Create a method called `sendMessage()`, which is called when a user sends a message to store the message in the database, as shown in the following code. This needs to be placed in the group chat controller `CommunicationApp/module/Users/src/Users/Controller/GroupChatController.php`.

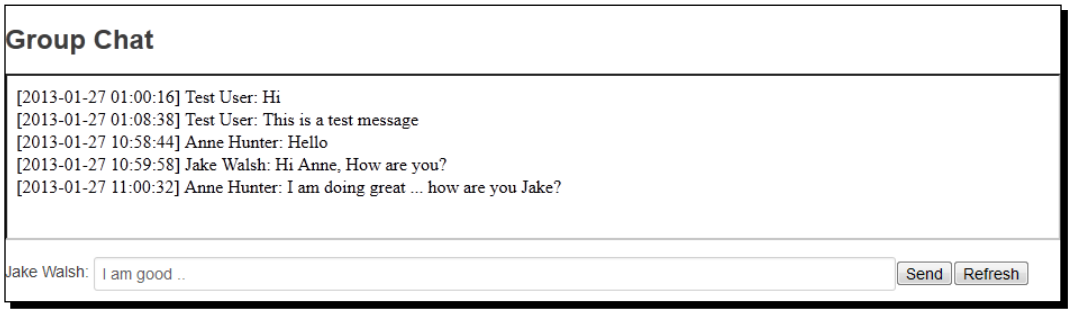
```
protected function sendMessage($messageTest $fromUserId)
{
  $chatMessageTG = $this->getServiceLocator()
                   ->get('ChatMessagesTableGateway');
  $data = array(
    'user_id' => $fromUserId,
    'message'  => $messageTest,
    'stamp' => NULL
  );
  $chatMessageTG->insert($data);
  return true;
}
```

8. Modify the `indexAction` function to display a `Send Message` form and to call `sendMessage()` on form submission. This needs to be placed in the group chat controller `CommunicationApp/module/Users/src/Users/Controller/GroupChatController.php`.

```
public function indexAction(
{
  $user = $this->getLoggedInUser();
  $request = $this->getRequest();
  if ($request->isPost()) {
    $messageTest = $request->getPost()->get('message');
    $fromUserId = $user->id;
    $this->sendMessage($messageTest, $fromUserId);
    // to prevent duplicate entries on refresh
    return $this->redirect()->toRoute('users/group-chat');
  }

  //Prepare Send Message Form
  $form    = new \Zend\Form\Form();

  $form->add(array(
    'name' => 'message',
    'attributes' => array(
      'type'  => 'text',
      'id' => 'messageText',
      'required' => 'required'
    ),
    'options' => array(
      'label' => 'Message',
    ),
  ));

  $form->add(array(
          'name' => 'submit',
          'attributes' => array(
              'type'  => 'submit',
              'value' => 'Send'
          ),
  ));

  $form->add(array(
    'name' => 'refresh',
    'attributes' => array(
      'type'  => 'button',
      'id' => 'btnRefresh',
```

```
            'value' => 'Refresh'
        ),
    ));

    $viewModel  = new ViewModel(array('form' => $form,
                                      'userName' => $user->name));
    return $viewModel;
}
```

9. To test the changes, log in to the browser from two different computers or two different browsers using different credentials, and test the **Group Chat** interface.



What just happened?

We have now successfully implemented a **Group Chat** interface using Zend Framework; the interface is effective for multiple people chatting with each other in a group. Our next task will need to build a mechanism to send e-mails to other users in the system; for that we will be exhaustively using the Zend Framework's mailing capabilities.

Have a go hero

Here is a simple exercise for you to try before you move on to the next section. In the **Group Chat** interface, we have a **Refresh** button that reloads the `iframe` tag. Write some JavaScript and attach it to the view, which will reload the IFrame every five seconds.

Sending mails

Zend Framework offers the `Zend\Mail` library to send and receive e-mails. In this section, we will cover the basics of Zend Framework's mailing capabilities, and will also implement a simple mailing script.

`Zend\Mail` supports both plain text and MIME complaint multipart e-mail messages. The framework by default supports Sendmail, SMTP, and File transports; new transports can be implemented using `Zend\Mail\Transport\TransportInterface`.

Zend\Mail\Transport

The `Mail` transport is used to send the e-mail message to recipients; `Zend\Mail` supports the following transports:

- Sendmail using `Zend\Mail\Transport\Sendmail`
- SMTP using `Zend\Mail\Transport\Smtp`
- File Transport using `Zend\Mail\Transport\File`

The `Mail` transport implements the `send()` method; this method accepts an object of type `Zend\Mail\Message` as the parameter; this object (`Zend\Mail\Message`) contains all the necessary information for an e-mail message; the message is sent using the transport.

Zend\Mail\Message

`Zend\Mail\Message` is used to compose the mail message in Zend Framework; this object takes various parameters including the from address, to address, subject, and body. If the message is a MIME complaint multipart message, then the body of the message can be set to a `Zend\Mime\Message` mail message object using the `setBody()` method, and the message can be sent. Some of the most frequently-used methods in `Zend\Mail\Message` are listed as follows:

- `setFrom()`
- `setHeaders`
- `setTo()`
- `addCc()` and `addBcc()`
- `setSubject()`
- `setBody()`

Zend\Mime\Message and Zend\Mime\Part

For sending HTML or multi-part content, each message part is defined as a `Zend\Mime\Part` object along with its type and associated to the `Zend\Mime\Message` object using the `setParts()` method. The `Zend\Mime\Message` object is assigned to the `Zend\Mail\Message` object using the `setBody()` method.

Time for action – creating a simple e-mail form

In this activity, we will be creating an e-mail form making use of Zend's mailing capabilities:

1. Create a simple e-mail form with input fields for subject, message content, and addressee.
2. Set up a new controller to display the form and write the necessary views.
3. Modify the controller so that it references the `Zend\Mail` namespace.

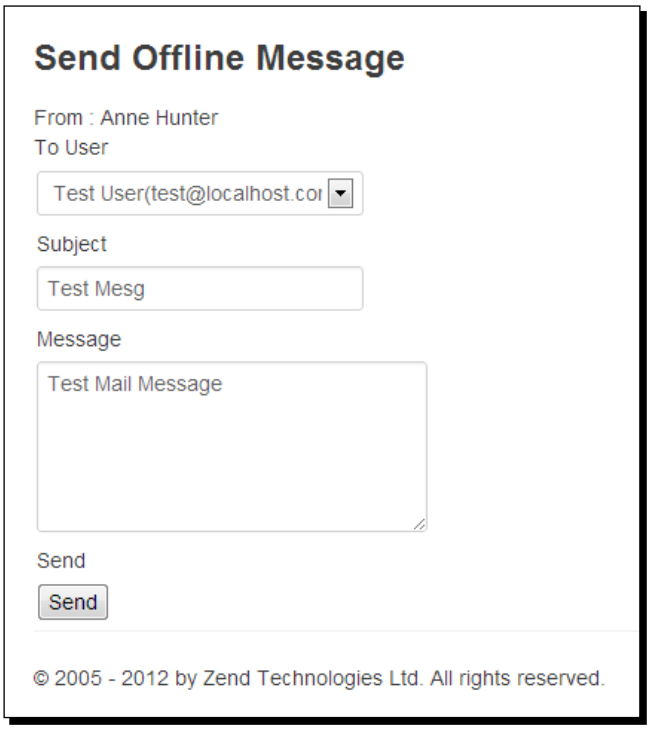
   ```
   use Zend\Mail;
   ```

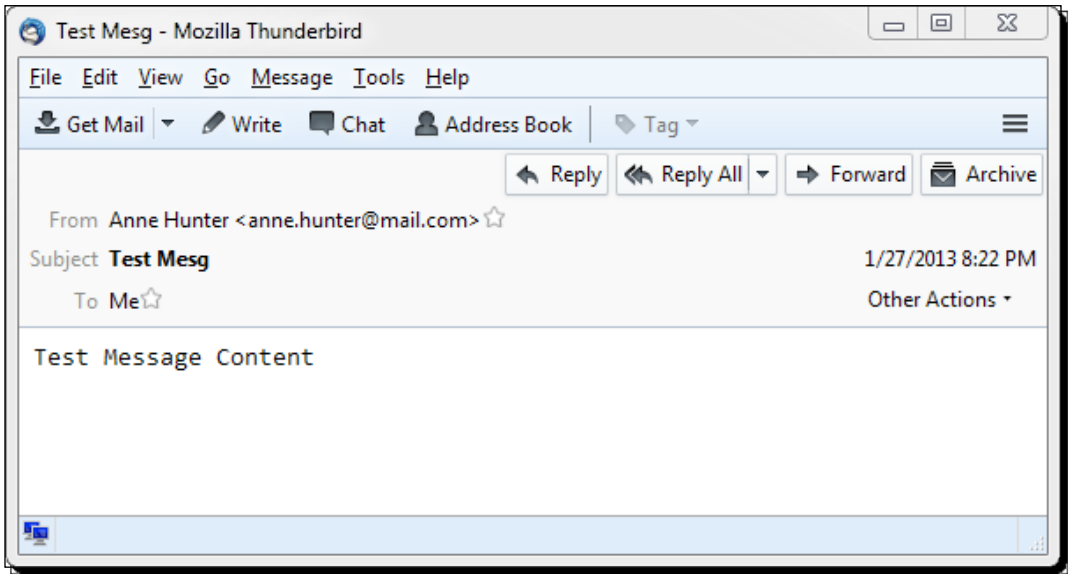
4. Create a new controller method that does the actual e-mailing; this can be placed within our group chat controller (`CommunicationApp/module/Users/src/Users/Controller/GroupChatController.php`) using the following code:

   ```
   protected function sendOfflineMessage($msgSubj,
                           $msgText, $fromUserId, $toUserId)
   {
     $userTable = $this->getServiceLocator()
                                ->get('UserTable');

     $fromUser = $userTable->getUser($fromUserId);
     $toUser = $userTable->getUser($toUserId);

     $mail = new Mail\Message();
     $mail->setFrom($fromUser->email, $fromUser->name);
     $mail->addTo($toUser->email, $toUser->name);
     $mail->setSubject($msgSubj);
     $mail->setBody($msgText);

     $transport = new Mail\Transport\Sendmail();
     $transport->send($mail);

     return true;
   }
   ```

> The `Sendmail` transport (`Zend\Mail\Transport\Sendmail`) is available in Linux by default and can be used for sending e-mail messages. Windows users can make use of SMTP transport (`Zend\Mail\Transport\Smtp`) to connect an SMTP server to send e-mail messages. The following reference link provides a quick example on using SMTP transport:
>
> `https://packages.zendframework.com/docs/latest/manual/en/modules/zend.mail.transport.html#zend-mail-transport-quick-start-smtp-usage`

5. Preview the form in a web browser and test if the e-mail is being received; a message similar to the following one would be received by the recipient:





What just happened?

We have used the `Zend\Mail` object to send e-mails within the system using the `Sendmail` mail transport; we have also learned about how to send HTML or multi-part mail messages.

Have a go hero

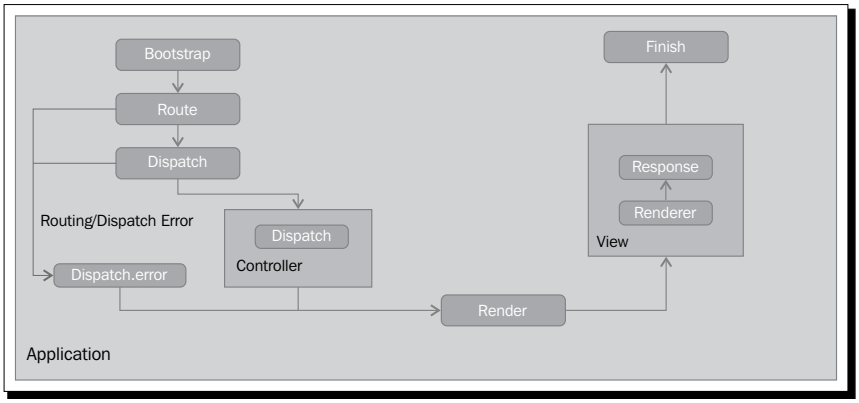
Before moving on to the next section, try to implement the e-mailing form for sending out HTML e-mails.

Zend\EventManager

Zend Framework 2 is an event-driven framework; the event manager allows you to attach events to almost any functionality. There are three main terms used in Zend Framework's event management, which are as follows:

- **Event manager**: The `EventManager` object is the object that holds a collection of listeners and their relative events
- **Listener**: The listener is the callback that reacts to the triggered event
- **Event**: The event is the action that is triggered by the event manager

The event manager provides `attach()` and `trigger()` to create and trigger events respectively. Mostly we will be depending on MVC events for various operation, and the sequence of execution of MVC application events is described in the following diagram:



The article at the following link explains the sequence of events in a ZF2 application:

`http://akrabat.com/zend-framework-2/a-list-of-zf2-events/`

Flow of events for successful execution is as follows:

1. `Zend\Mvc\Application`: **Bootstrap**
2. `Zend\Mvc\Application`: **Route**
3. `Zend\Mvc\Application`: **Dispatch**
4. `Zend\Mvc\Controller\ActionController`: **Dispatch** (if controller extends this class)
5. `Zend\Mvc\Application`: **Render**
6. `Zend\View\View`: **Renderer**
7. `Zend\View\View`: **Response**
8. `Zend\Mvc\Application`: **Finish**

In case of errors during dispatch (or) route, the flow of events will be as follows:

1. `Zend\Mvc\Application`: **Dispatch.error**
2. `Zend\Mvc\Application`: **Render**
3. `Zend\View\View`: **Renderer**
4. `Zend\View\View`: **Response**
5. `Zend\Mvc\Application`: **Finish**

In our next activity, we will try to set a new layout for multiple controllers using the shared event manager in Zend Framework.

Time for action – setting module layout using ZF events

Perform the following steps for setting the module layout using ZF events:

1. Create a new layout for the **My Account** page and save it under `CommunicationApp/module/Users/view/layout/myaccount-layout.phtml`.
2. Add the layout to the `CommunicationApp/module/Users/config/module.config.php` file under `view_manager -> template_map`:

   ```
   'layout/myaccount' => __DIR__ . '/../view/layout/myaccount-layout.phtml',
   ```

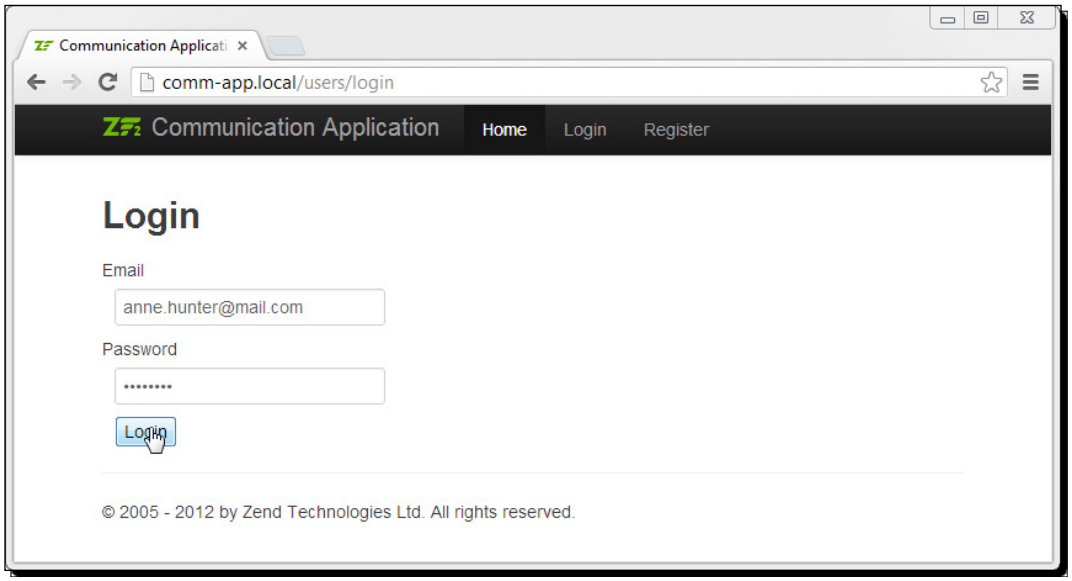
3. Open the `CommunicationApp/module/Users/module.php` file and add references to `MvcEvent`:

   ```
   use Zend\Mvc\MvcEvent;
   ```

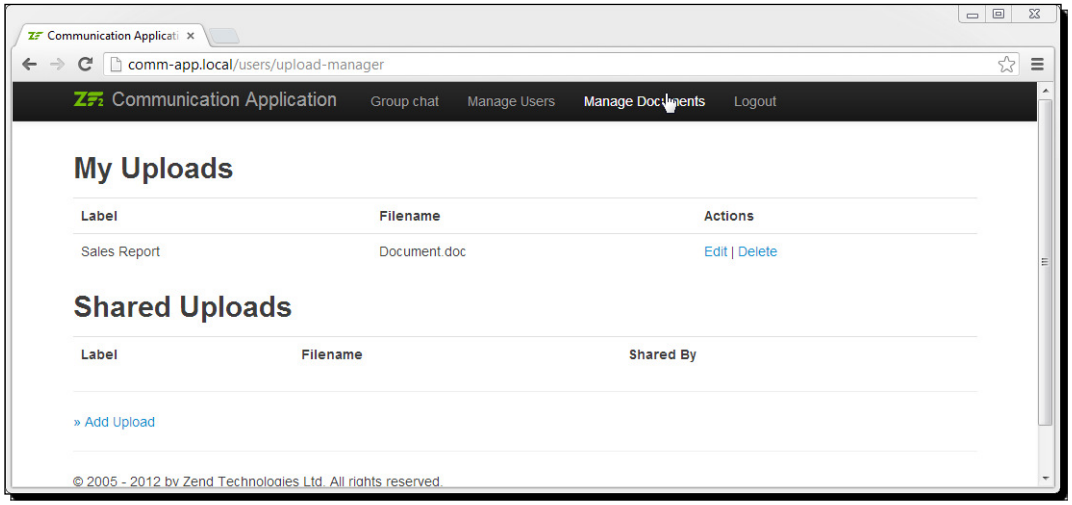
4. Overwrite the `onBootStrap()` method with the following code:

```
public function onBootstrap($e)
{
  $eventManager  = $e->getApplication()->getEventManager();
  $moduleRouteListener = new ModuleRouteListener();
  $moduleRouteListener->attach($eventManager);

  $sharedEventManager = $eventManager->getSharedManager(); // The
shared event manager
  $sharedEventManager->attach(__NAMESPACE__, MvcEvent::EVENT_
DISPATCH, function($e) {
    $controller = $e->getTarget(); // The controller which is
dispatched
    $controllerName = $controller->getEvent()
           ->getRouteMatch()->getParam('controller');
    if (!in_array($controllerName,
              array('Users\Controller\Index', 'Users\Controller\
Register', 'Users\Controller\Login'))) {
      $controller->layout('layout/myaccount');
    }
  });
}
```

5. Open the **Communication Application** page in any web browser; make a note of the layout:



6. Log in to the application and see the new layout being applied:



What just happened?

We have used the Zend Framework event manager to attach a listener to the `Dispatch` event of the module. So every time the controller is dispatched, this event is triggered. The callback checks if the controller is valid and if the controller is not among the list of controllers that have the `default` layout, then the `myaccount` layout is applied to these controllers.

Pop quiz – chat and e-mail

Q1. Which of the following helpers can be used to define/attach CSS styles inside the HTML `head` section?

1. `HeadLink`
2. `HeadScript`
3. `HeadCss`
4. `HeadStyle`

Q2. Which of the following are valid mail transports supported by Zend Framework 2?

1. `Zend\Mail\Transport\Pop`
2. `Zend\Mail\Transport\Smtp`
3. `Zend\Mail\Transport\Imap`
4. `Zend\Mail\Transport\File`

Summary

We have covered a wide range of topics in this chapter; first we learned about making use of external JavaScripts. Next we created a simple group chat application and then we learned about `Zend\Mail` and implemented a simple mailing form. Towards the end, we learned about events and how to make use of these events in Zend Framework. In the next chapter we will be working on media sharing using Zend Framework by working with various media-sharing APIs.

6
Media Sharing

Uploading and managing images/videos on the Internet has become very common with the advent of social media. More and more applications now allow you to share and retrieve media with external media hosts/services such as Google, Flickr, and YouTube. In Zend Framework 1.0, the `Zend_Service` *package offered a large number of third-party integrations. This has changed with ZF2 and the new module framework.*

In this chapter, we will use various external Zend Framework 2.0 modules to manage images and videos. Let's quickly look at the topics that we will be learning in this chapter:

- Installing external modules in the Zend Framework application
- Setting up a simple photo gallery
- Resizing and manipulating images using `WebinoImageThumb`
- Introduction to the Zend GData API
- Using the GData API to fetch albums from Google Photos and YouTube

External modules

One of the most important features of Zend Framework 2.0 is the ability to integrate external modules in your PHP application, and this integration is completely managed using a dependency management tool (in our case, `Composer`).

This feature allows development of PHP applications without having to worry about maintaining external libraries inside your application. Libraries and applications can be decoupled and maintained separately.

In this chapter, we will be using an external module for resizing images; we will also make use of external libraries for connecting to Google services.

Composer

Composer is the one of the dependency management solutions used in Zend Framework. Composer allows developers to declare the dependencies needed for their application and will handle the installation of those libraries. The dependency configuration is stored in a file named `composer.json`.

Resizing images

Zend Framework 1.0 had a resize filter that allowed images to be resized on upload; with Zend Framework 2.0, this option no longer exists. Our next task will be to find a simple image-resizing module and install it in our application. So let's get started.

Time for action – resizing images using modules

Carry out the following steps:

1. Go to the Zend Framework 2 module's site:

   `http://modules.zendframework.com/`

2. Run a search for `WebinoImageThumb`.
3. To install this module, you will need to update `composer.json` in the application root and include this module as a required module.
4. To do this, edit `CommunicationApp/composer.json` and modify the required section:

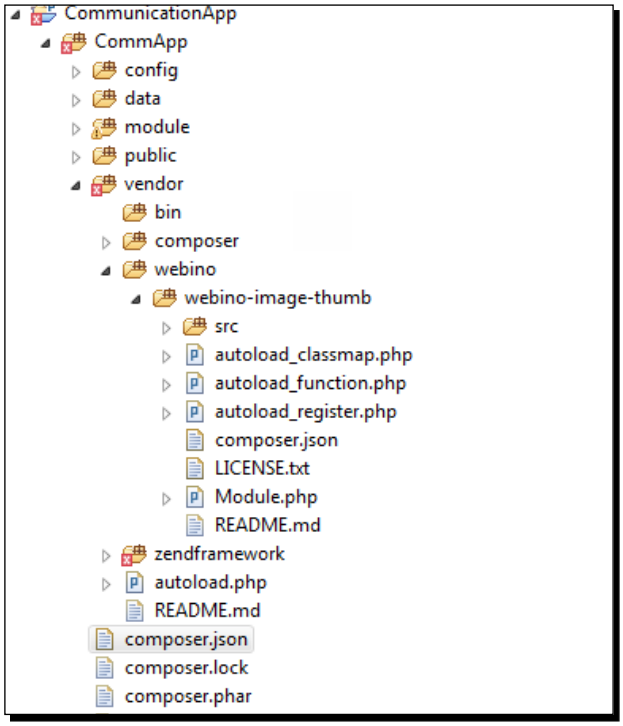
```
"require": {
    "php": ">=5.3.3",
    "zendframework/zendframework": "2.0.*",
    "webino/webino-image-thumb": "1.*",
}
```

5. Now run `composer.phar update` to install the newly added dependency.

```
$ php composer.phar update
Loading composer repositories with package information
Updating dependencies
  - Installing webino/webino-image-thumb (1.0.0)
```

```
    Downloading: 100%

Writing lock file
Generating autoload files
```

6. You will be able to see the newly installed modules in the `vendor` folder as follows:



7. Now that the module is downloaded, we will need to activate the module in `CommunicationApp/config/application.config.php` by adding `'WebinoImageThumb'` to the `modules` array.

```
return array(
    'modules' => array(
        'Application',
      'WebinoImageThumb',
    'Users',
    ),
```

What just happened?

We have installed an external module into our application using the dependency management tool, Composer. We have also activated the module in our application so that the module is accessible across the application.

Have a go hero

Now that you know how to install new modules in the Zend Framework 2 application, here is a simple task for you. Install the Zend GData package on this application. Instructions for installing this package are available at `https://packages.zendframework.com/`. We will be using this module in the subsequent sections of this chapter.

The Photo gallery application

Let us get started with implementing our custom photo gallery using Zend Framework 2. Since we have already implemented a file management interface, we will use a similar interface to implement a photo gallery.

The schema for a photo gallery will be similar to the `Upload` entity; additionally, we will have a field to store the `thumbnail` filename, which is generated during upload. Both the images and the generated thumbnails will be stored in the `<Module>\data\images` folder. We will use a custom action to display the images in the browser.

Before we get started, let's quickly review some of the important methods that are supported by `WebinoImageThumb`:

- `resize ($maxWidth = 0, $maxHeight = 0)`: This function resizes the image to the specified height and width; if either of the values is set to `0`, that dimension will not be considered as a limiter
- `adaptiveResize ($width, $height)`: This function attempts to get the image as close to the provided dimensions as possible, and then crops the remaining overflow (from the center) to get the image to be the size specified
- `crop ($startX, $startY, $cropWidth, $cropHeight)`: This function crops the images from the given coordinates to the specified width and height
- `rotateImage ($direction = 'CW')`: Rotates the image by 90 degrees clockwise or counterclockwise
- `rotateImageNDegrees ($degrees)`: Rotates the image by the specified degrees
- `save ($fileName, $format = null)`: Saves the image by the specified filename

Time for action – implementing a simple photo gallery

Carry out the following steps:

1. Create a new entity called `ImageUpload` with the following table structure:

```
CREATE TABLE IF NOT EXISTS image_uploads (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
  filename VARCHAR( 255 ) NOT NULL ,
  thumbnail VARCHAR( 255 ) NOT NULL ,
  label VARCHAR( 255 ) NOT NULL ,
  user_id INT NOT NULL,
  UNIQUE KEY (filename)
);
```

2. Create the relevant `ImageUpload` entity in the `src/Users/Model/ImageUpload.php` file, the `TableGateway` object in the `src/Users/Model/ImageUploadTable.php` file, and the `Controller` (`MediaManagerController`) inside the module (`CommunicationApp/module/Users`) in the `src/Users/Controller/MediaManagerController.php` file.

3. In the `Upload` form's `Submit` process, generate the thumbnail by using a new method called `generateThumbnail()`; this method will take the filename of the existing image as the parameter. The `resize` method resizes the image to 75x75 px and saves it to the image upload directory with a `tn_` prefix.

   This method needs to be placed in the `MediaManagerController` file, `src/Users/Controller/MediaManagerController.php`.

```
public function generateThumbnail($imageFileName)
{
  $path = $this->getFileUploadLocation();
  $sourceImageFileName = $path . '/' . $imageFileName;
  $thumbnailFileName = 'tn_' . $imageFileName;

  $imageThumb = $this->getServiceLocator()
        ->get('WebinoImageThumb');
  $thumb = $imageThumb->create(
                  $sourceImageFileName,
                  $options = array());
  $thumb->resize(75, 75);
  $thumb->save($path . '/' . $thumbnailFileName);

  return $thumbnailFileName;
}
```

4. Our next step is to write an action to render the image in the `Full` and `Thumbnail` modes; for this we will need to create a custom route that will take the `action`, `id`, and `subaction` parameters. This is achieved by the following route definition in the module configuration file, `CommunicationApp/module/Users/config/module.config.php`:

```
'media' => array(
  'type'    => 'Segment',
  'options' => array(
    'route'    => '/media[/:action[/:id[/:subaction]]]',
    'constraints' => array(
      'action'     => '[a-zA-Z][a-zA-Z0-9_-]*',
      'id'     => '[a-zA-Z0-9_-]*',
      'subaction'     => '[a-zA-Z][a-zA-Z0-9_-]*',

    ),
    'defaults' => array(
      'controller' => 'Users\Controller\MediaManager',
      'action'     => 'index',
    ),
  ),
),
```

5. Our next step is to write an action that will respond to the various image requests. This action needs to be placed in the `MediaManagerController` file, `src/Users/Controller/MediaManagerController.php`.

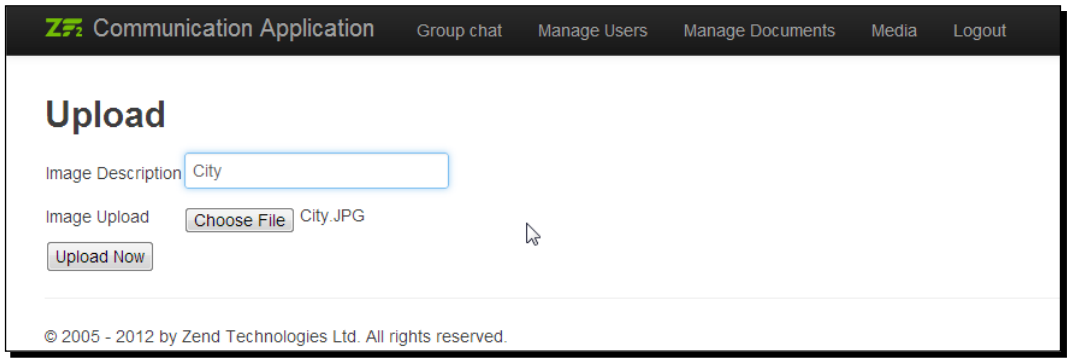
```
public function showImageAction()
{
  $uploadId = $this->params()->fromRoute('id');
  $uploadTable = $this->getServiceLocator()
      ->get('ImageUploadTable');
  $upload = $uploadTable->getUpload($uploadId);

  // Fetch Configuration from Module Config
  $uploadPath    = $this->getFileUploadLocation();
  if ($this->params()->fromRoute('subaction') == 'thumb')
  {
    $filename = $uploadPath ."/" . $upload->thumbnail;
  } else {
    $filename = $uploadPath ."/" . $upload->filename;
  }
  $file = file_get_contents($filename);

  // Directly return the Response
  $response = $this->getEvent()->getResponse();
```

```
        $response->getHeaders()->addHeaders(array(
          'Content-Type' => 'application/octet-stream',
          'Content-Disposition' => 'attachment;filename="'
          .$upload->filename . '"',

        ));
        $response->setContent($file);
        return $response;
      }
```

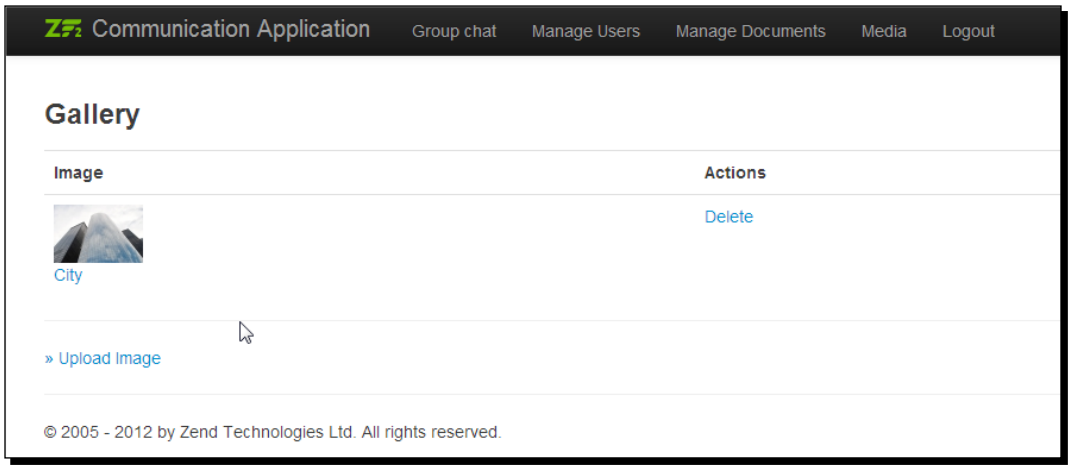
6. Make sure the process works completely, from uploading the picture to the gallery to displaying it in the photo page. See the following code for the usage of `showImageAction()` in the `upload` view in the media manager, `CommunicationApp/module/Users/view/users/media-manager/view.phtml`:

```
<section class="upload">
<h2><?php echo $this->escapeHtml($upload->label);?></h2>
<h4><?php echo $this->escapeHtml($upload->filename);?></h4>
<img src="<?php echo $this->url('users/media',
            array('action'=>'showImage',
                  'id' => $upload->id,
                  'subaction' => 'full'));?>" />
</section>
<a href="<?php echo $this->url('users/media');?>">
      &raquo; Show Gallery</a>
```

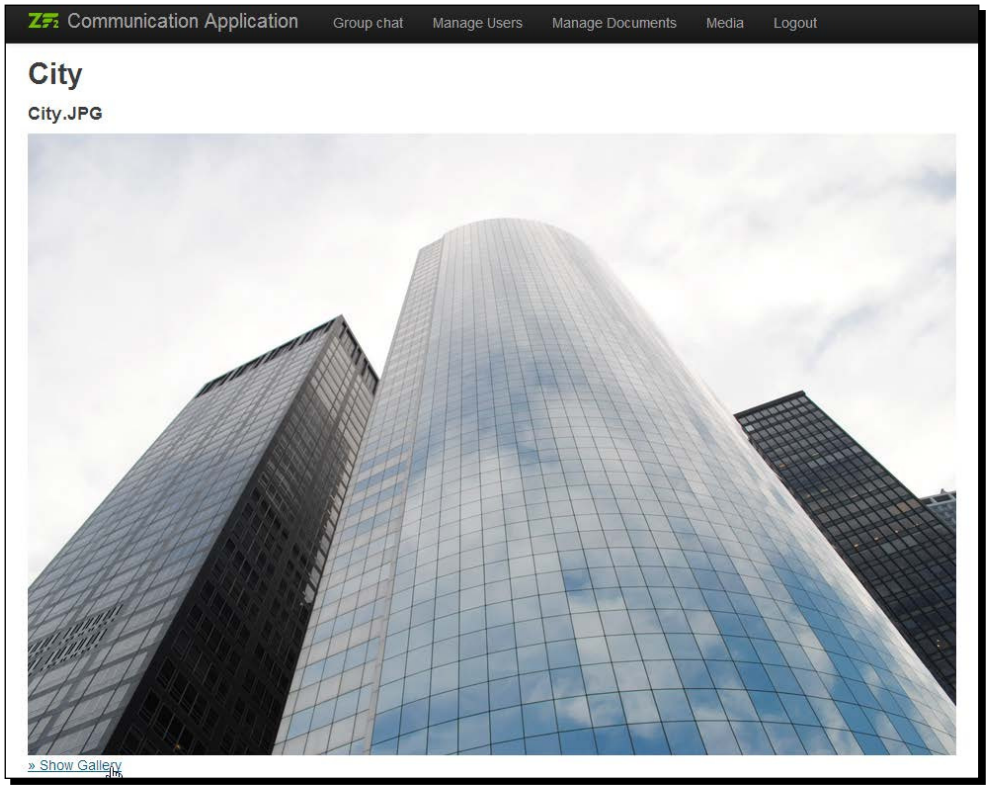
7. Now test the application on a browser of your choice. The image upload page should look like the following screenshot:



Once the image upload form is successfully submitted, the image will be resized and shown in the gallery as shown in the following screenshot:



The **View Image** link on top of the resized image takes you to a page with the full-sized image:



What just happened?

We have implemented a simple photo gallery by making use of an external image manipulation library. We utilized the `resize` function to create thumbnails and we created a custom action to handle image rendering in the web browser.

Have a go hero

Now that you understand how to work with the `WebinoImageThumb` module, your next task will be to extend the photo gallery to support the `rotate` function. Add a `rotate` function to the **View Image** page and allow the user to rotate the image both clockwise and anticlockwise.

Google Data APIs

Google Data APIs provide a simple interface for applications to read and write data into various Google services. The Data APIs use a protocol similar to the Atom Publishing Protocol for data transfer. All the services are implemented in the package called `ZendGdata`.

Some of the most frequently used Google services that are supported by the `ZendGdata` API are listed as follows:

- Picasa Web Albums
- YouTube
- Google Calendar
- Google Spreadsheets
- Google Documents
- Google Provisioning
- Google Analytics
- Google Blogger
- Google CodeSearch
- Google Notebook

Since `ZendGdata` is not provided with the default Zend Framework installation, this needs to be installed manually. This can be performed using Composer and by fetching `"zendframework/zendgdata": "2.*"`.

The Google Photos API

The Google Photos API allows you to fetch, edit, and manage your photos and albums in your Picasa or Google+ accounts. The Data API provides all kinds of services; some of the key functions are listed as follows:

- `getUserFeed()`: Gets all the associated albums for that user
- `insertAlbumEntry()`: Creates a new album
- `getAlbumFeed()`: Fetches the specified album
- `insertPhotoEntry()`: Creates a new photo
- `getPhotoFeed()`: Fetches the specified photo
- `insertCommentEntry()`: Creates a new comment
- `getCommentEntry()`: Fetches the specified comment
- `insertTagEntry()`: Creates a new tag
- `getTagEntry()`: Fetches the specified tag
- `deleteAlbumEntry()`: Deletes the album
- `deletePhotoEntry()`: Deletes the photo
- `deleteCommentEntry()`: Deletes the comment
- `deleteTagEntry()`: Deletes the tag

In this example we will fetch the user's existing albums and the photos stored inside those albums.

Before moving on, ensure that the `ZendGdata` library is installed in your application using Composer. Refer to the following installation instructions:

- Add the following line to the `requires` section of `CommunicationApp/composer.json`:

  ```
  "zendframework/zendgdata": "2.*"
  ```

- Update the application dependencies using Composer:

  ```
  $ php composer.phar update
  ```

Before getting started, make sure you have uploaded some photos on your Google Photos account.

Time for action – fetching photos from Google Photos

Follow these steps to fetch photos from your Google Photos account:

1. Create a method, `getGooglePhotos()`, in your controller that will connect to Google Photos and fetch all albums from Google Photos. This method needs to be placed in the `MediaManagerController` file, `src/Users/Controller/MediaManagerController.php`.
2. Set up the API client to make use of the `Curl` request with the option to disable `sslverifypeer`.

```
$adapter = new \Zend\Http\Client\Adapter\Curl();
$adapter->setOptions(array(
  'curloptions' => array(
    CURLOPT_SSL_VERIFYPEER => false,
  )
));

$httpClient = new \ZendGData\HttpClient();
$httpClient->setAdapter($adapter);

$client = \ZendGData\ClientLogin::getHttpClient(
        self::GOOGLE_USER_ID,
        self::GOOGLE_PASSWORD,
        \ZendGData\Photos::AUTH_SERVICE_NAME,
        $httpClient);
```

3. Now create a new Google Photos client using the API client.

```
$gp = new \ZendGData\Photos($client);
```

4. Now fetch the list of albums using `getUserFeed()` and get the list of images inside the album using `getAlbumFeed()`.

```
$userFeed = $gp->getUserFeed( self::GOOGLE_USER_ID );
foreach ($userFeed as $userEntry) {

  $albumId = $userEntry->getGphotoId()->getText();
  $gAlbums[$albumId]['label'] = $userEntry->getTitle()-
  >getText();

  $query = $gp->newAlbumQuery();
  $query->setUser( self::GOOGLE_USER_ID );
  $query->setAlbumId( $albumId  );
```

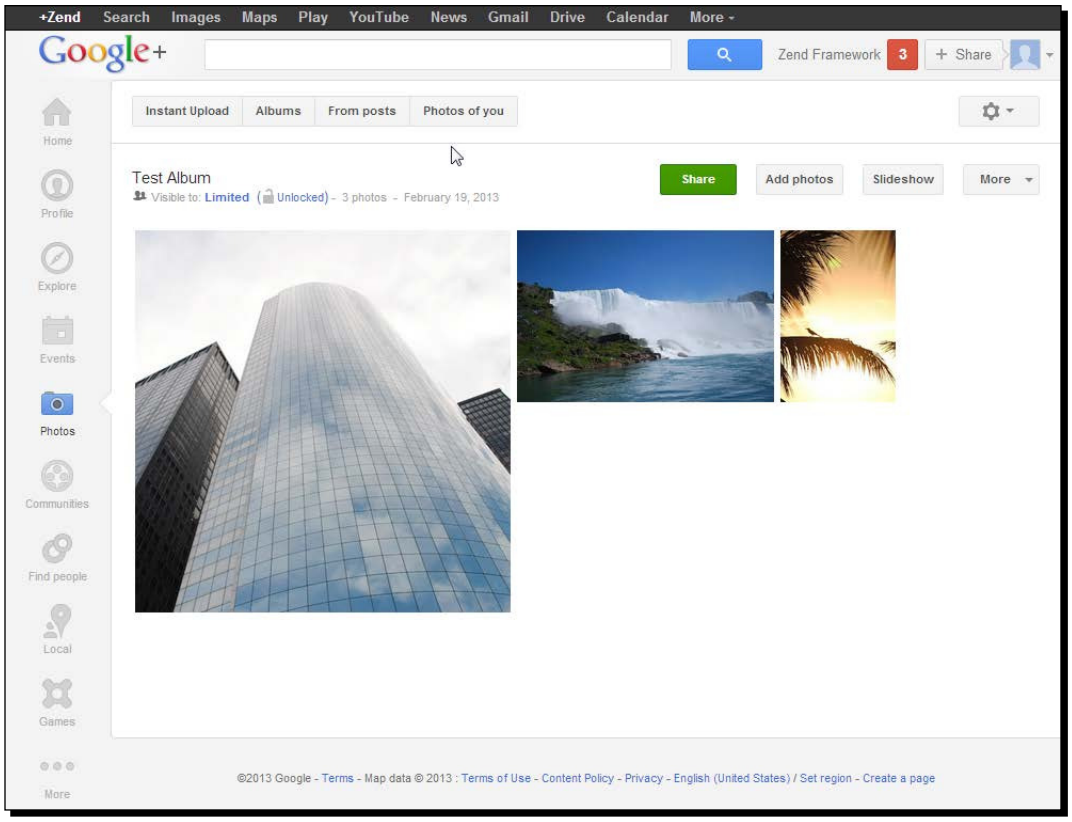
```
        $albumFeed = $gp->getAlbumFeed($query);

        foreach ($albumFeed as $photoEntry) {
          $photoId = $photoEntry->getGphotoId()->getText();
          if ($photoEntry->getMediaGroup()->getContent() != null) {
            $mediaContentArray = $photoEntry->getMediaGroup()-
            >getContent();
            $photoUrl = $mediaContentArray[0]->getUrl();
          }

          if ($photoEntry->getMediaGroup()->getThumbnail() != null)
{
            $mediaThumbnailArray = $photoEntry->getMediaGroup()-
            >getThumbnail();
            $thumbUrl = $mediaThumbnailArray[0]->getUrl();
          }

          $albumPhoto = array();
          $albumPhoto['id'] = $photoId;
          $albumPhoto['photoUrl'] = $photoUrl;
          $albumPhoto['thumbUrl'] = $thumbUrl;

          $gAlbums[$albumId]['photos'][] =$albumPhoto;
        }
      }
// Return the consolidated array back to the view for rendering
return $gAlbums;
```

5. The following code block in the `album` view is used to render the albums; this can be placed in the media manager's `index` view, `CommunicationApp/module/Users/view/users/media-manager/index.phtml`:

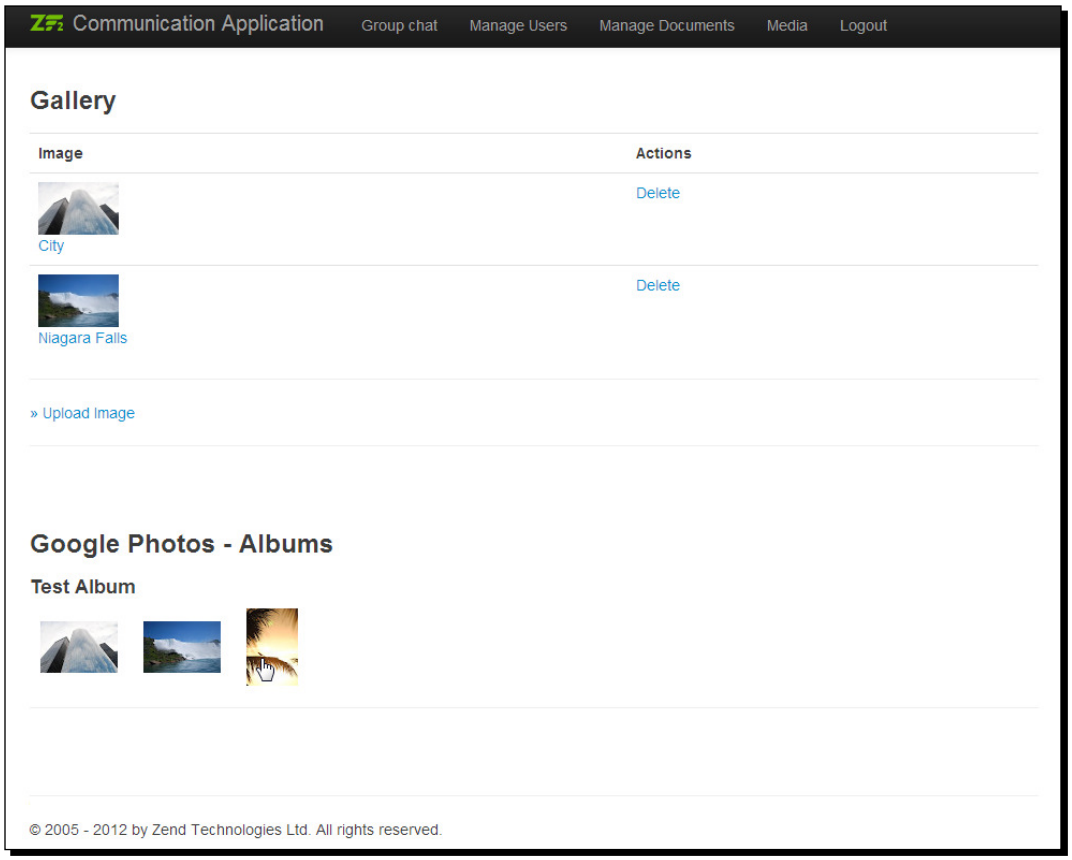
```
<?php foreach ($googleAlbums as $googleAlbum) : ?>
<h4> <?php echo $this->escapeHtml($googleAlbum['label']);?>
</h4>
<?php foreach ($googleAlbum['photos'] as $googleAlbumPhoto) : ?>
  <div class = "googleAlbumPhoto"
  style="padding:10px; display:inline">
    <a href="<?php echo $this->escapeHtml($googleAlbumPhoto['p
    hotoUrl']);?>">
    <img src="<?php echo $this->escapeHtml($googleAlbumPhoto['
    thumbUrl']);?>" />
```

```
        </a>
      </div>
      <?php endforeach; ?>
    <?php endforeach; ?>
    <hr />
```

6. Upload pictures to your Google Photos album:



7. Open the page in a browser window; you should be able to see all available albums and photos inside the album:



What just happened?

We have successfully used Google Data APIs to fetch Picasa upload information from Google and used that information to render galleries in our application.

Have a go hero

Your next task will be to implement the photo upload option using Google Data APIs when viewing a photo in the photo gallery; you will have a button that will allow you to upload the photo to Google Photos.

YouTube Data API

The YouTube Data API allows access to YouTube content; you can use this API to fetch videos, playlists, channels, post comments, and upload and manage videos. Users are allowed to perform unauthenticated requests for the retrieval of feeds on popular videos, post comments, and so on.

Some of the most frequently used YouTube API methods are listed as follows:

- `getVideoFeed()`: Retrieve videos from a video query
- `getTopRatedVideoFeed()`: Retrieve top-rated videos for the specific video query
- `getUserUploads()`: Retrieve the user's uploaded videos
- `getUserFavorites()`: Retrieve the user's favorite videos
- `getVideoResponseFeed()`: Get video responses for a specific video
- `getVideoCommentFeed()`: Get comments for a specific video
- `getPlaylistListFeed()`: Get a user's playlists
- `getSubscriptionFeed()`: Get a user's subscriptions
- `insertEntry()`: Upload a video to YouTube

In this example, we will be retrieving videos for a specific keyword and then render them in the web page.

Time for action – listing YouTube videos for a keyword

Perform the following steps for listing YouTube videos for a keyword:

1. Create a function that will get the YouTube videos for the `Zend Framework` keyword.
2. Establish the connection in a similar way to the previous connection made for Google Photos. This needs to be placed in a new method, `getYoutubeVideos()`, in the `MediaManagerController` file, `src/Users/Controller/MediaManagerController.php`:

```
$adapter = new \Zend\Http\Client\Adapter\Curl();
$adapter->setOptions(array(
  'curloptions' => array(
    CURLOPT_SSL_VERIFYPEER => false,
  )
));

$httpClient = new \ZendGData\HttpClient();
```

```
$httpClient->setAdapter($adapter);

$client = \ZendGData\ClientLogin::getHttpClient(
  self::GOOGLE_USER_ID,
  self::GOOGLE_PASSWORD,
  \ZendGData\YouTube::AUTH_SERVICE_NAME,
  $httpClient);
```

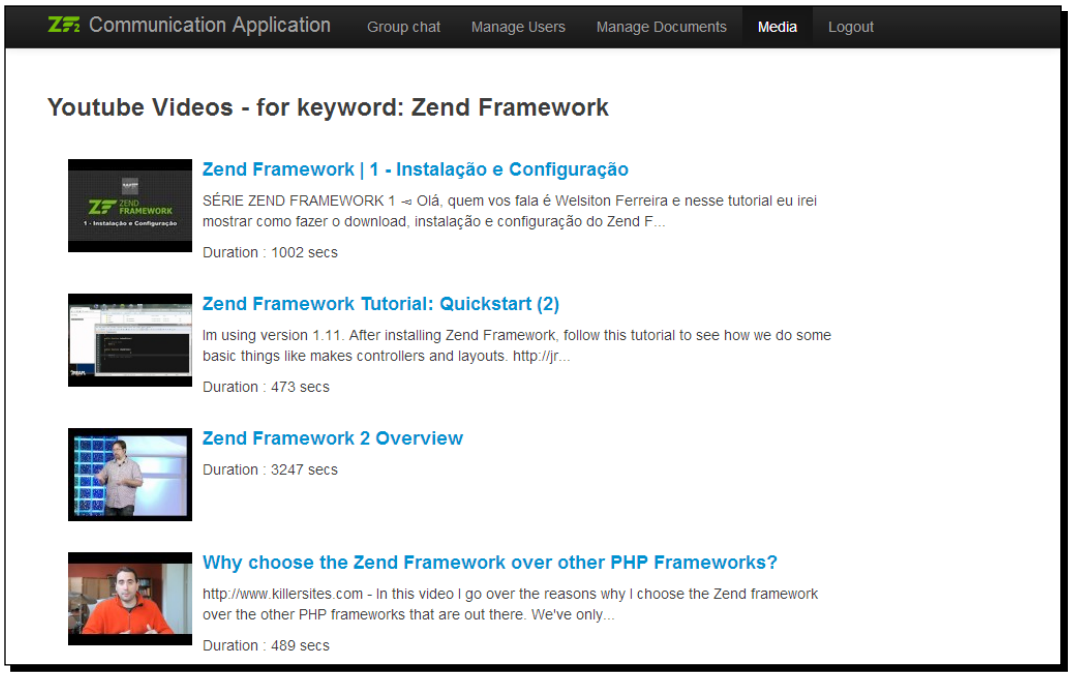
3. Initialize the YouTube client and execute a video query for the keyword `Zend Framework`:

```
$yt = new \ZendGData\YouTube($client);
$yt->setMajorProtocolVersion(2);
$query = $yt->newVideoQuery();
$query->setOrderBy('relevance');
$query->setSafeSearch('none');
$query->setVideoQuery('Zend Framework');
```

4. Parse the query results and store it in an array:

```
$videoFeed = $yt->getVideoFeed($query->getQueryUrl(2));

$yVideos = array();
foreach ($videoFeed as $videoEntry) {
   $yVideo = array();
   $yVideo['videoTitle'] = $videoEntry->getVideoTitle();
   $yVideo['videoDescription'] =
   $videoEntry->getVideoDescription();
   $yVideo['watchPage'] = $videoEntry->getVideoWatchPageUrl();
   $yVideo['duration'] = $videoEntry->getVideoDuration();
   $videoThumbnails = $videoEntry->getVideoThumbnails();

   $yVideo['thumbnailUrl'] = $videoThumbnails[0]['url'];
   $yVideos[] = $yVideo;
}
return $yVideos;
```

5. The resulting content is rendered in the view and a video listing as shown in the following screenshot:



What just happened?

We have utilized the `ZendGData` API's YouTube APIs to retrieve a simple list of videos from YouTube for a specific keyword.

Pop quiz – media sharing

Q1. Which command is used in Composer to install a newly configured dependency?

1. `php composer.phar setup`
2. `php composer.phar self-update`
3. `php composer.phar show`
4. `php composer.phar update`

Q2. Which of the following is a valid method to upload a new photo to Google Photos?

1. `uploadPhoto()`
2. `insertPhoto()`
3. `uploadNewPhoto()`
4. `insertPhotoEntry()`

Summary

In this chapter, we have learned various techniques to manage media; initially we started with implementing our own photo gallery and later on we moved on to using Google GData APIs to retrieve and store media on the Web.

In our next chapter, we will be working on implementing a simple search interface.

7

Search Using Lucene

More often than not, we will come across web applications that need support for built-in search capabilities. Sometimes the search could involve searching a simple field in a MySQL table, or at times you may want to search a document or a plain text file; there are multiple ways to address the search requirements using various search libraries. Lucene is one such library that offers excellent search capabilities for implementing full text search.

In this chapter we will be using Zend Framework's Lucene search implementation. Zend Framework 1.0 had a built-in `Zend_Search_Lucene` library which supported indexing and searching with Lucene; in ZF 2.0, this library is available as `ZendSearch\Lucene`, which can be downloaded and installed on your web application. In this chapter, we will be learning the fundamentals of implementing a full-text search using the Lucene search library in the following topics:

- Installing the `ZendSearch` library in your application
- Creating data index for simple MySQL data
- Querying the Lucene index
- Adding new documents files to the index

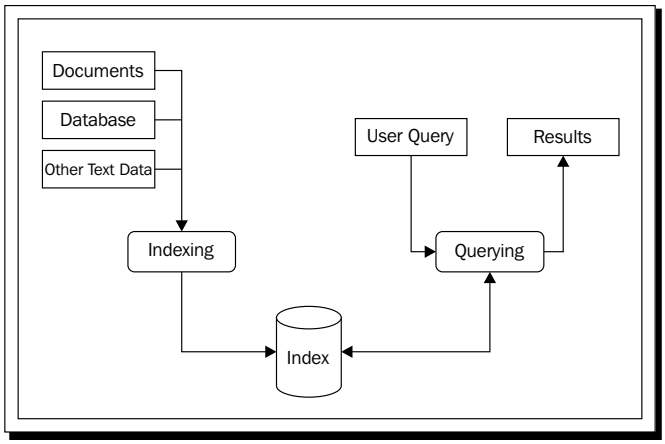
Introduction to Lucene

Lucene is a high-performance, scalable information retrieval (search) library developed by Apache Foundation, which can be used for implementing free-text search in web applications. Lucene provides a simple-to-use API, which will provide powerful indexing and searching capability to your web application. To read more about Lucene visit `http://lucene.apache.org/`.

The most important components of the Lucene search library are explained as follows:

- **Index**: Lucene index is the data store that holds all the indexed documents; queries are executed against the index to fetch the documents.
- **Document**: A document is the default building block for a Lucene index; documents can be compared to records in a table. Each document holds a number of fields upon which queries can be executed.
- **Field**: Each Lucene document comprises of one or more fields; it is not necessary that all the fields are indexed, fields can also be stored without indexing.

The Lucene search works based on the index, so it is necessary to have the index updated with the latest content to get the best search results. The following diagram explains the overview of the Lucene search:



Time for action – installing ZendSearch\Lucene

Perform the following steps for installing `ZendSearch\Lucene`:

1. `ZendSearch\Lucene` was not available as a composer package at the time of writing this book. So, we will check out the source from the GitHub repository. The repository is available at `https://github.com/zendframework/ZendSearch`.
2. Next we need to navigate to the `vendor` folder:

   ```
   $ cd /var/www/CommunicationApp/vendor/
   ```

3. Clone the Zend search repository into the `vendor` folder:

   ```
   $ git clone https://github.com/zendframework/ZendSearch.git
   ZendSearch
   ```

4. Next we should configure the ZendSearch library using composer:

```
$ cd ZendSearch/
$ curl -s https://getcomposer.org/installer | php
$ php composer.phar install
```

5. Once the library is configured, we will need to define a module-level configuration to store the index location. To do this, we need to modify `CommunicationApp/module/Users/config/module.config.php`, and add a new configuration for `search_index`:

```
// MODULE CONFIGURATIONS
'module_config' => array(
   'upload_location'        => __DIR__ . '/../data/uploads',
   'image_upload_location'  => __DIR__ . '/../data/images',
   'search_index'         => __DIR__ . '/../data/search_index'
),
```

What just happened?

We have now downloaded and configured the `ZendSearch` library for Zend Framework 2.0; the previous tutorial also provides us with a guideline for downloading and installing packages which cannot be downloaded directly from Composer.

Now that we have the `ZendSearch\Lucene` search library installed, our next task will be to create a Lucene index for some of the data that is already stored in our communication application.

Indexing

Indexing is a fairly straightforward process using `ZendSearch\Lucene`. All we need is to create documents with fields and values, and keep adding the document to the index. You can also remove documents, update documents, and clear an index. The following classes are used in index generation:

- `Field` – The `ZendSearch\Lucene\Document\Field` class allows users to define a new document field; this field can be classified into one the following types:
  - `Field::keyword($name, $value, $encoding = 'UTF-8')`: the `keyword` field type is used to identify string fields that don't have to be tokenized, yet need to be indexed and stored. For example, date and URL.
  - `Field::unIndexed($name, $value, $encoding = 'UTF-8')`: The `unIndexed` field type is used to store fields in the index without having to index/tokenize them. For example, ID fields.

- `Field::binary($name, $value)`: The `binary` field type is used for storing binary values in the index.
- `Field::text($name, $value, $encoding = 'UTF-8')`: The `text` field type is the most common field type used for describing short strings which are tokenized and stored in the index.
- `Field::unStored($name, $value, $encoding = 'UTF-8')`: The `unStored` field type is used to identify fields that will be tokenized and indexed, but not stored in the index.

- `Document` – The `ZendSearch\Lucene\Document` class allows definition of a new index document. Some of the most commonly-used methods in this class are described as follows:
  - `addField(Document\Field $field)`: Adds a new field to the document
  - `getFieldNames()`: Used to retrieve all field names from the document
  - `getField($fieldName)`: Used to retrieve a specific field from the document
  - `getFieldValue($fieldName)`: Used to retrieve a specific field value from the document
- `Index` – `Index` can be retrieved using the `create()` and `open()` methods in the `ZendSearch\Lucene` class. Both the methods take the index path as the parameter and return an index of type `ZendSearch\Lucene\SearchIndexInterface`. The `SearchIndexInterface` provides the following methods for manipulating the documents inside the index:
  - `addDocument(Document $document)`: Adds a new document to the index
  - `delete($id)`: Deletes the indexed document based on the internal document ID
  - `optimize()`: Helps in optimizing the index, by merging all segments into a single segment, thereby increasing the performance
  - `commit()`: Used to commit transactions to the search index

Now that we have learned about the methods that are used for index generation, let's get started and generate the index for the `uploads` table that is available in our communication application.

Time for action – generating a Lucene index

Perform the following steps for generating a Lucene index:

1. Create a new search controller, `CommunicationApp/module/Users/src/Users/Controller/SearchController.php`, which will be used for searching and generating indexes.
2. Add references to `ZendSearch\Lucene`:

```
use ZendSearch\Lucene;
use ZendSearch\Lucene\Document;
use ZendSearch\Lucene\Index;
```

3. Add a method to fetch the index location from the module configuration:

```
public function getIndexLocation()
{
  // Fetch Configuration from Module Config
  $config  = $this->getServiceLocator()->get('config');
  if ($config instanceof Traversable) {
    $config = ArrayUtils::iteratorToArray($config);
  }
  if (!empty($config['module_config']['search_index'])) {
    return $config['module_config']['search_index'];
  } else {
    return FALSE;
  }
}
```

4. The index document needs to be generated in the following format:

Index field	Description
`upload_id`	This is non-indexed field which will be used for retrieving the uploaded file that gets returned in the search results
`label`	This field is used to index the `label` field of the `uploads` table
`owner`	This field is used to index the `name` field of the user who uploaded the document

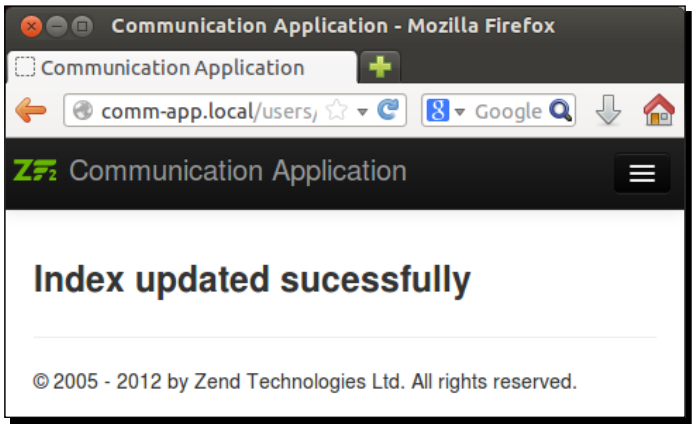
5. Create a new action to generate the index:

```
public function generateIndexAction()
{
  $searchIndexLocation = $this->getIndexLocation();
  $index = Lucene\Lucene::create($searchIndexLocation);

  $userTable = $this->getServiceLocator()->get('UserTable');
```

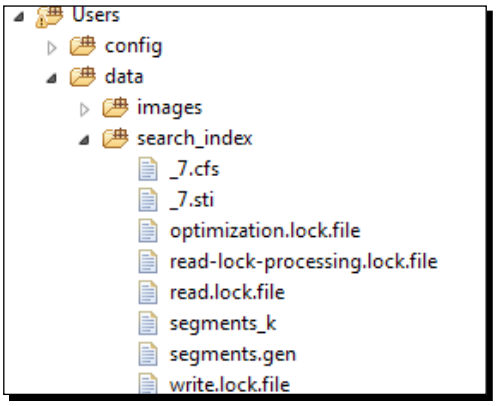
```
    $uploadTable = $this->getServiceLocator()->get('UploadTable');
    $allUploads = $uploadTable->fetchAll();
    foreach($allUploads as $fileUpload) {
      //
      $uploadOwner = $userTable->getUser($fileUpload->user_id);

      // create lucene fields
      $fileUploadId = Document\Field::unIndexed(
                            'upload_id', $fileUpload->id);
      $label = Document\Field::Text(
                            'label', $fileUpload->label);
      $owner = Document\Field::Text(
                            'owner', $uploadOwner->name);

      // create a new document and add all fields
      $indexDoc = new Lucene\Document();
      $indexDoc->addField($label);
      $indexDoc->addField($owner);
      $indexDoc->addField($fileUploadId);
      $index->addDocument($indexDoc);
    }
    $index->commit();
  }
```

6. Now open the action URL (`http://comm-app.local/users/search/generateIndex`) in your web browser, and if everything works as expected, you will see that the index files that created in the `search_index` folder.

The following screenshot shows the browser response upon a successful index update:



You can see in the following screenshot that the index files are generated and stored in the `search_index` folder:



What just happened?

Now we have created a method to index the data stored in the `MySQL` table to the Lucene data store; our next step will be to have some queries executed against the Lucene index and to fetch and show the results.

Searching

Searching the index is relatively simple using `ZendSearch\Lucene`. The index needs to be opened for querying and the query string needs to be passed to the `find()` method in `ZendSearch\Lucene\Index`. The `find` methods return an array matching the hits for the specific query, and this in turn can be used to render the search results.

There are two options for querying the index—you can pass the plain text query string to the find function or you can build your own `Query` object using `ZendSearch\Lucene\Search\Query`.

> To read more about various query options in ZendSearch\Lucene, check the following developer documentation:
>
> `https://zf2.readthedocs.org/en/release-2.2.0/modules/zendsearch.lucene.queries.html`

In the following example, we will be using plain text queries, and you can manipulate the search results by using operators such as `:`,`+`,`-`, and field searches. For example, see the following list:

- A search for all documents uploaded by Anne could be retrieved by the following query:

  ```
  owner:Anne
  ```

- A search for all documents having the word `report` and uploaded by the user named `Anne` could be retrieved by the following query:

  ```
  report AND owner:Anne
  ```

- A search for all documents having the word `report` and excluding the ones uploaded by `Anne` could be retrieved by the following query:

  ```
  report -owner:Anne
  ```

Time for action – displaying search results

Perform the following steps for displaying search results:

1. For displaying the search results, we will need to create a new form which will display the search textbox and render the search results right below the search form. The form will be placed in `SearchController` under `CommunicationApp/module/Users/src/Users/Controller/SearchController.php`.
2. Create a new view which will be used for displaying the query window and also rendering search results. This will be placed under `CommunicationApp/module/Users/view/users/search/index.phtml`.

   ```
   <h3>Document Search</h3>
   <?php
   // Search Form
   echo $this->form()->openTag($form);
   foreach ($form as $element) {
       echo $this->formElement($element);
       echo $this->formElementErrors($element);
   }
   echo $this->form()->closeTag();

   // Search Results
   if (count($searchResults)) {
   ?>
   <h5>Results</h5>
   <table  style="width: 600px; border:1px solid #f5f5f5;">
     <tr>
   ```

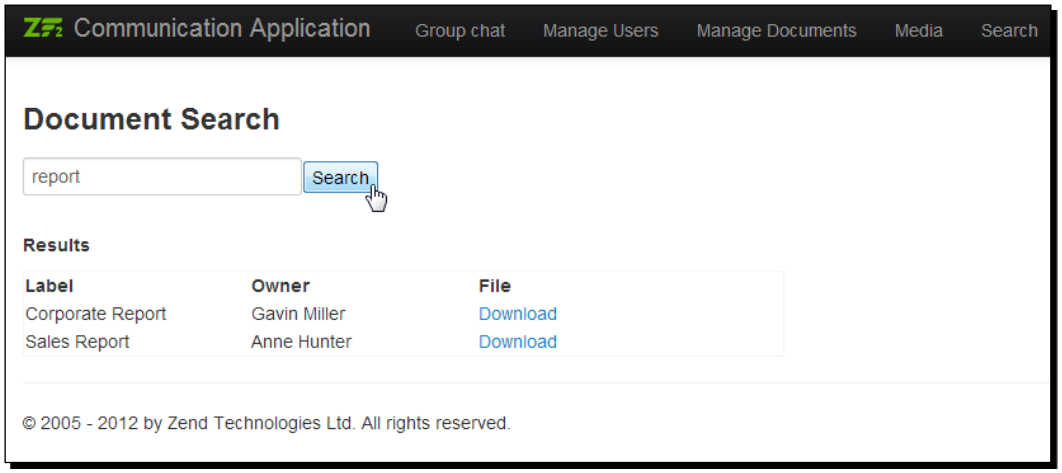
```
    <th width="30%" align="left"> Label</th>
    <th width="30%" align="left"> Owner</th>
    <th align="left"> File</th>
  </tr>
  <?php   foreach ($searchResults as $searchResult) {
  ?>
  <tr>
    <td><?php echo $searchResult->label; ?></td>
    <td><?php echo $searchResult->owner; ?></td>
    <td><a href="<?php echo $this->escapeHtml($this->url('users/
    upload-manager',
            array('action'=>'fileDownload', 'id' =>
            $searchResult->upload_id)));?>">Download</a></td>
  </tr>
  <?php
  }
  ?>
</table>
<?php }?>
```

3. Now create a new action which will display the `Search` form and also query the Lucene index with the input provided in the `Search` form. This will be placed in `SearchController` under `CommunicationApp/module/Users/src/Users/Controller/SearchController.php`.

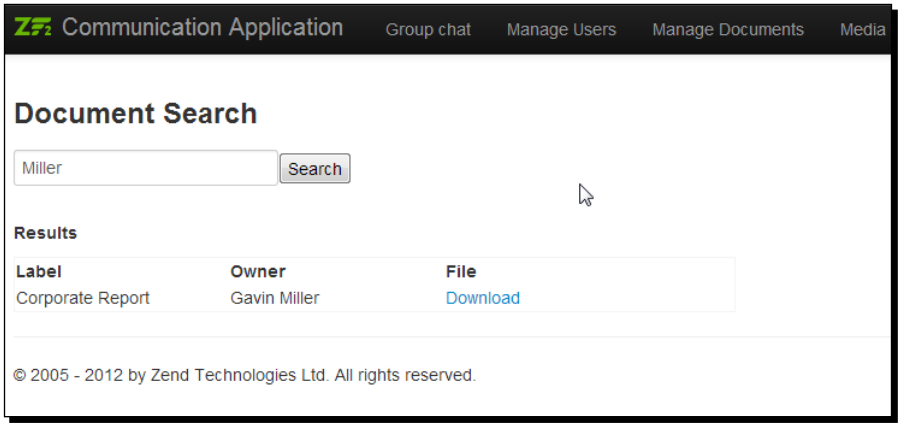
```
public function indexAction()
{
  $request = $this->getRequest();
  if ($request->isPost()) {
    $queryText = $request->getPost()->get('query');
    $searchIndexLocation = $this->getIndexLocation();
    $index = Lucene\Lucene::open($searchIndexLocation);
    $searchResults = $index->find($queryText);
  }

  // prepare search form
  $form  = new \Zend\Form\Form();
  $form->add(array(
    'name' => 'query',
    'attributes' => array(
      'type'  => 'text',
      'id' => 'queryText',
      'required' => 'required'
    ),
    'options' => array(
```

```
            'label' => 'Search String',
        ),
    ));
    $form->add(array(
            'name' => 'submit',
            'attributes' => array(
                'type'  => 'submit',
                'value' => 'Search'
            ),
    ));

    $viewModel  = new ViewModel(array(
        'form' => $form,
        'searchResults' => $searchResults
        )
    );
    return $viewModel;
}
```

4. Test the page in your browser; you should be able to see search results for keywords that are available in the `label` and `owner` fields:



On searching using `Owner Name`, you will get the following search results:



What just happened?

We have now implemented the search results page, which allows us to query for uploaded documents using their labels and owners. The retrieved search results are displayed in a customized view which allows us to download the document from the search result.

Our next step will be to expand the search to search the contents of the uploaded documents; for this we will need to make changes to the way we generate the index.

Indexing Microsoft Office documents

As we have seen in the previous example, it is usually insufficient to index the documents' meta information. Most of the time the query string is only present in the document's content. In order to achieve that, we need to parse the document and index the content; `ZendSearch\Lucene` provides support indexing the contents of the following document types:

- For HTML documents the following are the index document creation methods:

  ```
  ZendSearch\Lucene\Document\Html::loadHTMLFile($filename)
  ZendSearch\Lucene\Document\Html::loadHTML($htmlString)
  ```

- For Word 2007 documents the following is the index document creation method:

  ```
  ZendSearch\Lucene\Document\Docx::loadDocxFile($filename)
  ```

- For Powerpoint 2007 documents the following is the index document creation method:

  `ZendSearch\Lucene\Document\Pptx::loadPptxFile($filename)`

- For Excel 2007 documents the following is the index document creation method:

  `ZendSearch\Lucene\Document\Xlsx::loadXlsxFile($filename)`

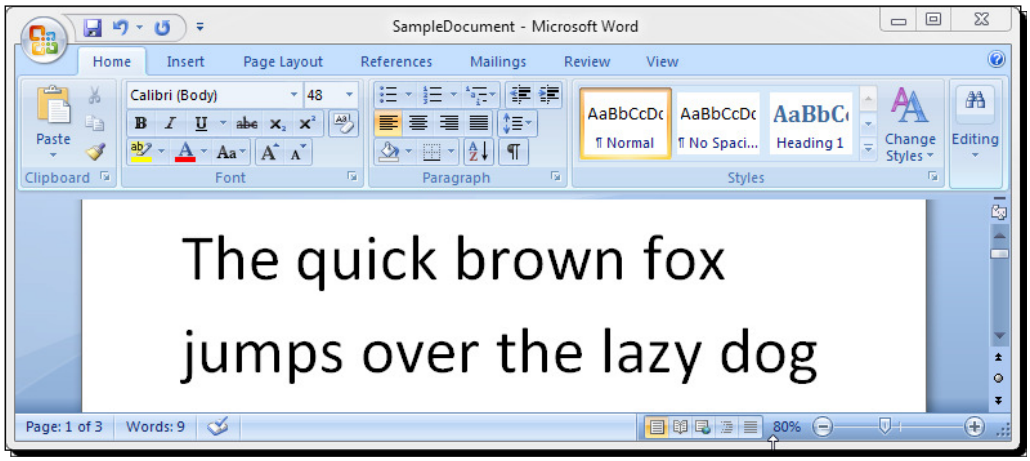
All these methods return a document of type `ZendSearch\Lucene\Document`, which can be improvised further by adding more index fields to it.

So let's get started by indexing the documents that are available in the `uploads` section.

Time for action – indexing document files

Perform the following steps for indexing document files:

1. To index office documents, add a new `uploads` section for sample Word and Excel documents. In this case, we will upload a Word document and an Excel spreadsheet as follows:



Sample Word 2007 document

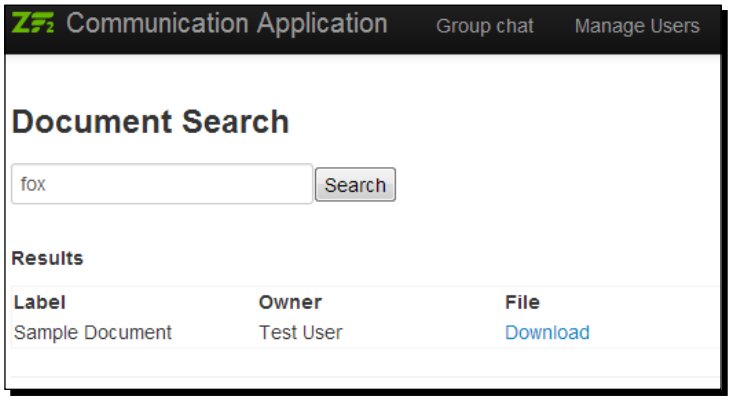
Sample Excel 2007 spreadsheet

2. Add the following lines to the indexing function present in `SearchController`, which is present in `CommunicationApp/module/Users/src/Users/Controller/SearchController.php`, so that the method picks up and indexes Word documents and Excel spreadsheets separately:

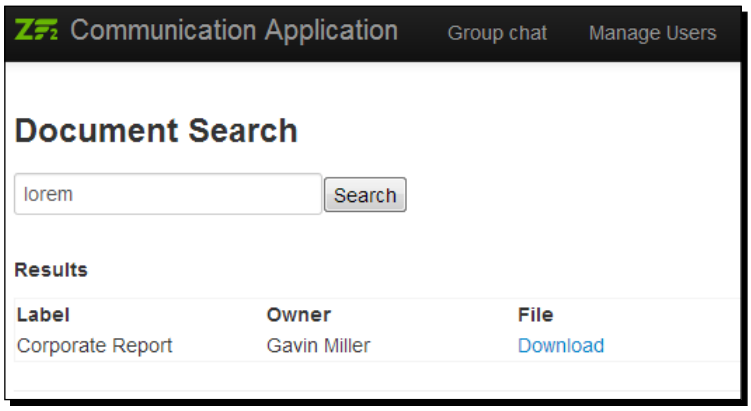
```
if (substr_compare($fileUpload->filename,
        ".xlsx",
        strlen($fileUpload->filename) - strlen(".xlsx"),
        strlen(".xlsx")) === 0) {
  // index excel sheet
  $uploadPath = $this->getFileUploadLocation();
  $indexDoc = Lucene\Document\Xlsx::loadXlsxFile(
                    $uploadPath ."/" . $fileUpload->filename);
} else if (substr_compare($fileUpload->filename,
        ".docx",
        strlen($fileUpload->filename) - strlen(".docx"),
        strlen(".docx")) === 0) {
  // index word doc
  $uploadPath= $this->getFileUploadLocation();
  $indexDoc = Lucene\Document\Docx::loadDocxFile(
                    $uploadPath ."/" . $fileUpload->filename);
} else {
  $indexDoc = new Lucene\Document();
}

$indexDoc->addField($label);
```

```
$indexDoc->addField($owner);
$indexDoc->addField($fileUploadId);
$index->addDocument($indexDoc);
```

3. Now update the index (navigate to `http://comm-app.local/users/search/generateIndex`), come back to the **Document Search** page, and try searching for keywords that are present in the document. You should be able to see the search results as shown in the following screenshot:



Search results for the content inside Office documents will be as shown in the following screenshot:



What just happened?

In the last task we saw the implementation of indexing and searching the content of Microsoft Office documents. As you can see, it is relatively easy to implement these features using `ZendSearch\Lucene`.

Have a go hero

Here is a simple task for you before you move on to the next chapter. Now that we have implemented indexing and searching, your task will be to modify the entities so that the index is updated each time changes are made to uploads. If a new upload is made, a document needs to be added to the index, and if an upload is deleted, it should be removed from the index, and so on.

Pop quiz – search

Q1. Which of the following field types is not tokenized, yet is indexed and stored?

1. `keyword ()`
2. `unStored ()`
3. `text()`
4. `unIndexed()`

Q2. Which of the following file formats is not supported for `ZendSearch\Lucene` as a valid document format for content indexing?

1. `.docx`
2. `.pdf`
3. `.xslx`
4. `.html`

Summary

In this chapter we have learned about implementing a simple search interface using `ZendSearch\Lucene`. This would be very useful when implementing search in any web application that you work with. In the next chapter we will be learning about implementing a simple e-commerce store using Zend Framework 2.0.

8

Creating a Simple Store

Over the last few years e-commerce has evolved from just online advertisements to providing fully functional shopping experiences online. More and more products and services are being made available online everyday through the use of various online payment systems. The role of e-commerce applications and payment gateways has become crucial in this environment.

In this chapter we will be building a simple online store to demonstrate the process involved in setting up a simple shopping cart. We will be using PayPal Express Checkout as our payment processer during this example. Some of the key topics that will be covered in this chapter include:

- Setting up a shopping cart
- Creating a online store administration interface
- Configuring Zend Framework 2.0 for PayPal
- An introduction to PayPal Express Checkout
- The implementation of PayPal Express Checkout

Shopping cart

One of the first things that have to be designed while setting up an online store is the shopping cart. The shopping cart should ideally allow the end user to choose and add multiple products to the cart and be able to check out from the website.

The checkout process is outlined as follows:

1. Customer visits the product listing page.
2. Customer selects a product; he/she is taken to the product detail page.
3. Customer then chooses to purchase the product; customer is expected to add the desired quantity to the cart.
4. Customer is redirected to the shopping cart page; here the customer may make any changes to the order if necessary.
5. Customer chooses the mode of payment and enters the payment information.
6. If successful, the customer is presented with an option to update the shipping details.
7. Customer then confirms the order.
8. The order is received at the retailer; the retailer then goes ahead and processes the order.

So let's get started and create our store front; our next step will be to design a table structure which will support this store. For this we create the following two tables:

- `store_products`: This table will store all product related information
- `store_orders`: This table will store all order-related information

Time for action – creating a store front

For simplicity, we will shorten the `Checkout` process by skipping some steps. We have modified the process so that we can only have one product per order; we will also skip the updating of shipping details and the customer order confirmation steps:

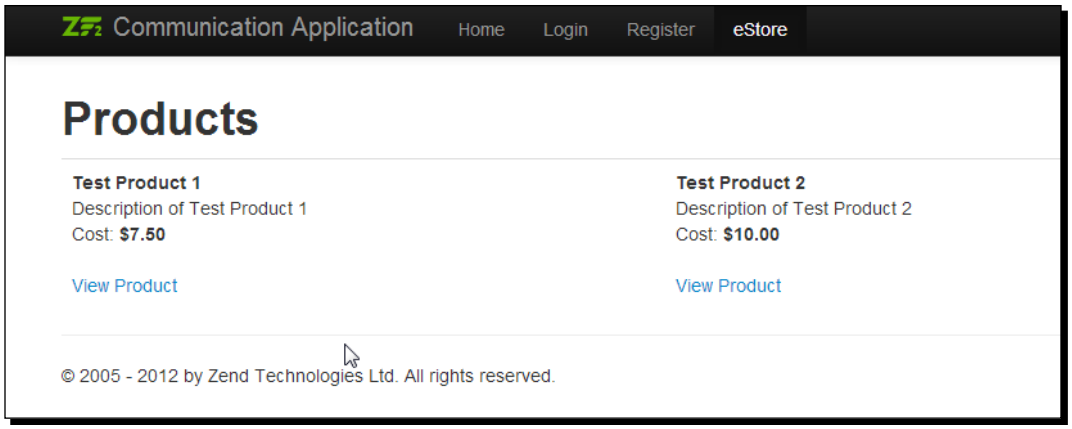
1. Create tables to hold the products and orders data:

```
CREATE TABLE IF NOT EXISTS store_products (
  id int(11) NOT NULL AUTO_INCREMENT,
  name varchar(255) NOT NULL,
  desc varchar(255) NOT NULL,
  cost float(9,2) NOT NULL,
  PRIMARY KEY (id)
```

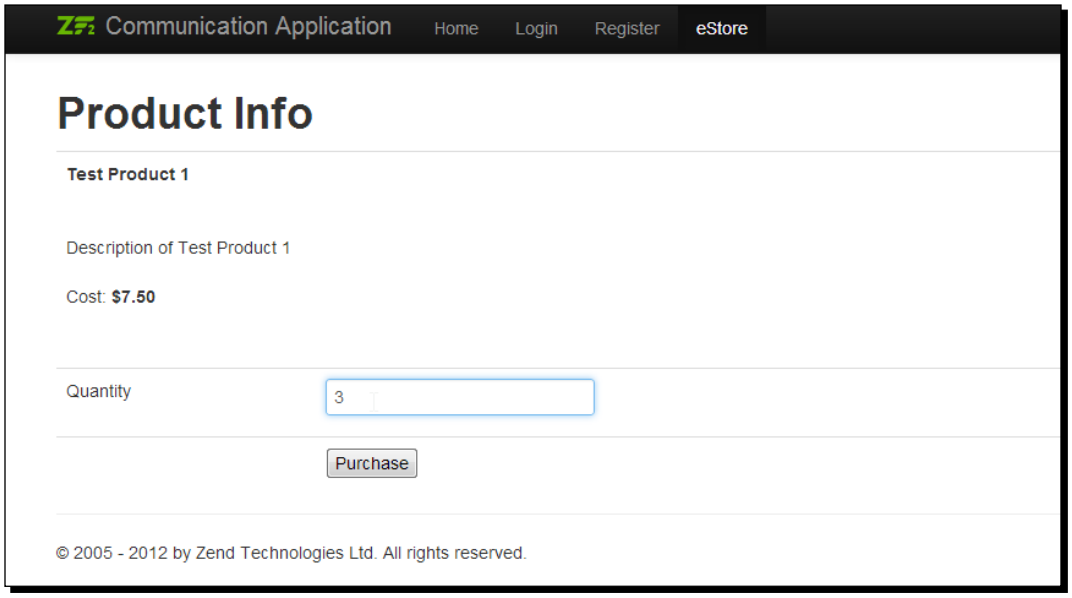
```
);

CREATE TABLE IF NOT EXISTS store_orders (
  id int(11) NOT NULL AUTO_INCREMENT,
  store_product_id int(11) NOT NULL,
  qty int(11) NOT NULL,
  total float(9,2) NOT NULL,
  status enum('new', 'completed',
        'shipped', 'cancelled')  DEFAULT NULL,
  stamp timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  first_name varchar(255) DEFAULT NULL,
  last_name varchar(255) DEFAULT NULL,
  email varchar(255) DEFAULT NULL,
  ship_to_street varchar(255) DEFAULT NULL,
  ship_to_city varchar(255) DEFAULT NULL,
  ship_to_state varchar(2) DEFAULT NULL,
  ship_to_zip int(11) DEFAULT NULL,
  PRIMARY KEY (id)
);
```

2. Create entities for `StoreOrder` and `StoreProduct`, and also create necessary table gateway objects for data access.
3. Create a `StoreController` controller, which will be used as our shopping cart.
4. `StoreController` will support the following actions:
   - `indexAction()`: This action will list all products in the website
   - `productDetailAction()`: This will display the details of a specific product; this will also allow the customer to add a product to the cart
   - `shoppingCartAction()`: This action is used to render the shopping cart before leaving for the payment processing page
   - `paypalExpressCheckoutAction()`: This action will redirect the user to the PayPal Express Checkout page
   - `paymentConfirmAction()`: This action will handle the redirection from PayPal Express Checkout back to the shopping cart upon successful payment
   - `paymentCancelAction()`: This action will handle the redirection from PayPal Express Checkout back to the shopping cart upon failed payment
5. Create the necessary views to display the content of the shopping cart.
6. Add the necessary methods to `StoreOrder` to calculate the order total upon adding items to the orders.

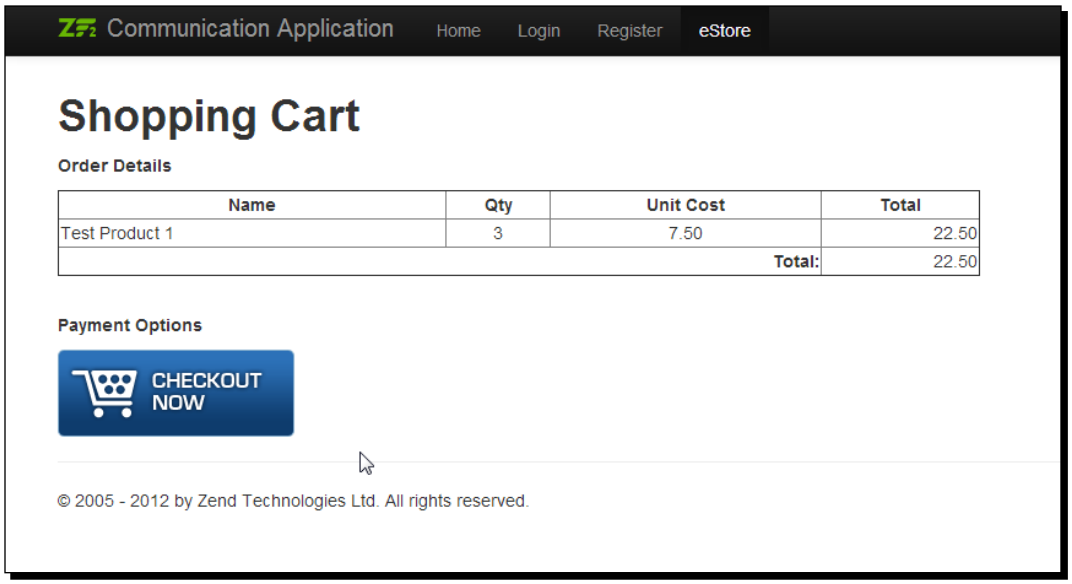
7. The final user interface should look like the following screenshot. The product listing page lists all products in the website/category; in this case, the two test products are listed in the following screenshot:



The product detail page allows users to view details of a product, and also add the specified quantity to the shopping cart:



The **Shopping Cart** page lists all products that are added to the cart along with their unit price, quantity, and subtotal:



What just happened?

We have created a shopping cart interface for our new store; we will be modifying this interface further in order to add support for the payment processor. But before we get to that stage, let's create a simple store administration interface to enable us to manage the store and orders.

The store administration

The store administration user interface is used to check the status of orders once they are created and also to manage the list of products that are available for sale in the store. There are two key aspects for the store administration user interface:

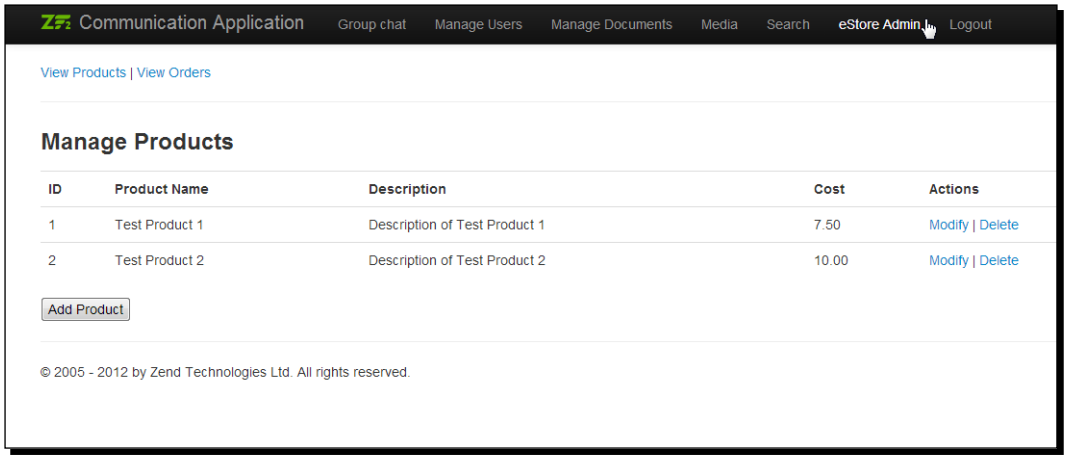
- The administrator should be able to add, remove, and manage products
- The administrator should be able manage order and change statuses using this interface

Time for action – creating the Store Admin interface

Perform the following steps for creating the `Store Admin` interface:

1. Create a new controller for store administration, and name it `StoreAdminController`.
2. This controller will have the following basic actions:
   - `indexAction()`: Used for listing all products
   - `addProductAction()`: Used for adding a new product
   - `deleteProductAction()`: Used for deleting an existing product
   - `listOrdersAction()`: Used for listing all orders
   - `viewOrderAction()`: Used for viewing a specific order
   - `updateOrderStatusAction()`: Used for updating order status
3. Create the necessary views, and map the actions accordingly.
4. Open phpMyadmin and create test records in both the `store_products` and `store_orders` tables to test the functionality for the administration UI.
5. Open your favorite browser, log in to the application, and open the **eStore Admin** interface. The interface should look like the following one.

   The **Manage Products** page lets you add, remove, and edit products from the administration interface:



The orders listing page lists all orders placed in the store and allows you to view orders and modify their statuses:

Communication Application | Group chat | Manage Users | Manage Documents | Media | Search | eStore Admin | Logout

View Products | View Orders

Store Orders

ID	Date	Total	Status	Actions
1	2013-04-01 21:38:30	15.00	cancelled	View Order
2	2013-04-02 08:25:58	30.00	cancelled	View Order
3	2013-04-03 21:20:49	637.50	cancelled	View Order
8	2013-04-03 22:16:15	15.00	cancelled	View Order
9	2013-04-03 22:36:32	37.50	cancelled	View Order
10	2013-04-03 22:40:59	7.50	shipped	View Order
11	2013-04-03 22:44:37	20.00	completed	View Order
20	2013-04-04 00:36:31	22.50	new	View Order

A screenshot of the **Order Information** page listing the order information and providing options to change their status is shown as follows:

Communication Application | Group chat | Manage Users | Manage Documents | Media | Search | eStore Admin | Logout

View Products | View Orders

Order Information

Order Num # 11

Status : completed

Shipping/Billing Information
Test User zendframework2development@gmail.com 1 Main St San Jose, CA 95131

Order Details

Name	Qty	Unit Cost	Total
Test Product 2	2	10.00	20.00

Update Order Status

Set as New | Set as Complete | Set as Shipped | Set as Cancelled

What just happened?

The store administration UI is now ready, and our next step is to set up PayPal Express checkout and to integrate it with our store, which will enable our user to make payments using PayPal. Before we move on to the next section, the following section gives you a simple task to try out.

Have a go hero

Now that you know how to integrate search into a Zend Framework 2.0 application, try to add free text search functionality for the **Manage Products** section of our store application.

Payments with PayPal

PayPal is the most commonly used payment processor across the world; one of the key contributors to PayPal's success is its easy-to-use API and exhaustive documentation that supports this payment gateway. For any new merchant, PayPal offers a wide range of options for setting up their payment processor, the most important being the types of integrations that are offered. PayPal offers various products under Payment Processing; some of them include:

- Express Checkout
- PayPal Payments Standards (Website Payments Standards)
- PayPal Payments Pro (Website Payments Pro)

We will be working with Express Checkout in this chapter, since it is the most basic implementation method of PayPal.

PayPal and Zend Framework 2.0

At the time of writing this book, there were no native packages that were offered by Zend Framework which supported PayPal integration. There are always third-party options that support this integration. In this example, we have made use of one such third party package called `SpeckPaypal`.

Time for action – setting up PayPal

Perform the following steps for setting up PayPal:

1. Open `https://packagist.org/`, search for `speckpaypal`.
2. Get the repository details.
3. Modify the application's Composer configuration file to include the `speckpaypal` repository:

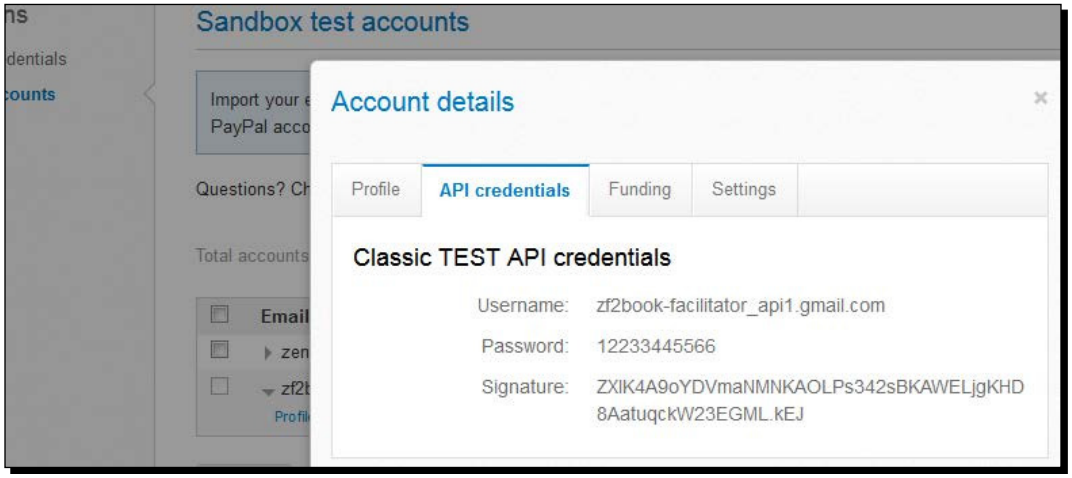
```
"require": {
    "php": ">=5.3.3",
    "zendframework/zendframework": "2.0.*",
    "webino/webino-image-thumb": "1.0.0",
    "zendframework/zendgdata": "2.*",
    "speckcommerce/speck-paypal": "dev-master"
}
```

4. Update the project dependencies using the Composer update:

```
.
Loading composer repositories with package information
Updating dependencies
  - Removing zendframework/zendframework (2.0.7)
  - Installing zendframework/zendframework (2.0.8)
    Downloading: 100%

  - Installing speckcommerce/speck-paypal (dev-master d951518)
    Cloning d951518fd2c98148da5609e23a41697e6cfca06e

Writing lock file
Generating autoload files
```

5. Now we will need API credentials for accessing PayPal Express Checkout. This can be accessed by logging into `https://developer.paypal.com` with your PayPal credentials.
6. Open **Sandbox Accounts** from **Applications**.

7. Choose the appropriate merchant account and select **API Credentials** in **Profile**.



8. Make a note of the API credentials.
9. Now create a new configuration in the `config` file (`CommunicationApp/module/Users/config/module.config.php`) in the module's configuration file and name the array index `speck-paypal-api`:

```
'speck-paypal-api' => array(
  'username'  => '',
  'password'  => '',
  'signature' => '',
  'endpoint'  => 'https://api-3t.sandbox.paypal.com/nvp'
)
```

10. Different PayPal services have different end points. For Express Checkout in Sandbox this is `https://api-3t.sandbox.paypal.com/nvp`; if you are switching live/production environment, this needs to be changed to `https://api-3t.paypal.com/nvp`.

What just happened?

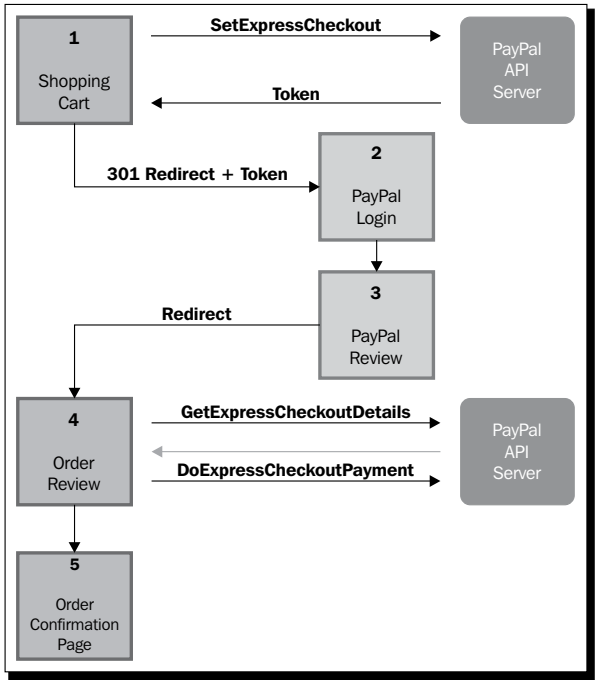
Now we have configured PayPal and SpeckPaypal in our application, our next step is to test receiving payments using PayPal Express Checkout.

PayPal Express Checkout

PayPal Express Checkout allows sellers to receive credit card / PayPal payments on their websites by redirecting them to PayPal Express Checkout for secure web payment and returning them back to the merchant's website once the transaction is completed.

The workflow is explained as follows:

1. Customer on the **Shopping Cart** page chooses to pay by **PayPal Express Checkout**; the merchant calls the `SetExpressCheckout` API call and gets the payment token.
2. Using the Payment token, the customer is redirected to the PayPal Express Checkout login page; here the customer can enter his/her PayPal login information or get a new PayPal account.
3. On the next page, the customer is presented with a **Review** option to review the payment information before proceeding to continue the checkout with the merchant.
4. Now the customer is redirected back to the merchant page; the merchant then calls the `GetExpressCheckoutDetails` API call and gets the customer information. The customer reviews the order and confirms the order. The merchant then completes the payment request using the `DoExpressCheckoutPayment` API call.
5. The customer is shown the transaction results along with the order summary.



PayPal Express Checkout—overview

More about PayPal Express Checkout

You can read more about PayPal Express Checkout at the PayPal website `https://www.paypal.com/webapps/mpp/express-checkout`.

Developer documentation on PayPal Express Checkout is available at:

`https://developer.paypal.com/webapps/developer/docs/classic/express-checkout/integration-guide/ECGettingStarted/`.

Time for action – accepting payments using PayPal

Preform the following steps for accepting payments using PayPal:

1. Now add a button on the **Shopping Cart** page (optionally with Checkout by PayPal Image). This button should link to the `paypalExpressCheckoutAction()` function.
2. Add a method in the store controller which will be used to generate the PayPal request:

```
protected function getPaypalRequest()
{
  $config  = $this->getServiceLocator()->get('config');
  $paypalConfig = new \SpeckPaypal\Element\Config(
              $config['speck-paypal-api']);

  $adapter = new \Zend\Http\Client\Adapter\Curl();
  $adapter->setOptions(array(
    'curloptions' => array(
      CURLOPT_SSL_VERIFYPEER => false,
    )
  ));

  $client = new \Zend\Http\Client;
  $client->setMethod('POST');
  $client->setAdapter($adapter);

  $paypalRequest = new \SpeckPaypal\Service\Request;
  $paypalRequest->setClient($client);
  $paypalRequest->setConfig($paypalConfig);

  return $paypalRequest;
}
```

3. Modify the `paypalExpressCheckoutAction()` function to send the order information to PayPal and redirect the user to PayPal Express Checkout:

```
public function paypalExpressCheckoutAction()
{
  $request = $this->getRequest();
  $orderId = $request->getPost()->get('orderId');

  $orderTable = $this->getServiceLocator()-
  >get('StoreOrdersTable');
  $order = $orderTable->getOrder($orderId);

  $paypalRequest = $this->getPaypalRequest();

  $paymentDetails = new \SpeckPaypal\Element\PaymentDetails
  (array('amt' => $order->total
  ));
  $express = new \SpeckPaypal\Request\SetExpressCheckout(
      array('paymentDetails' => $paymentDetails)
  );

  $express->setReturnUrl(
    'http://comm-app.local/users/store/paymentConfirm');
  $express->setCancelUrl(
    'http://comm-app.local/users/store/paymentCancel');

  // Send Order information to PayPal
  $response = $paypalRequest->send($express);
  $token = $response->getToken();

  $paypalSession = new \Zend\Session\Container('paypal');
  $paypalSession->tokenId = $token;
  $paypalSession->orderId = $orderId;


  // Redirect user to PayPal Express Checkout
  $this->redirect()->toUrl('https://www.sandbox.paypal.com/
  webscr?cmd=_express-checkout&token=' . $token);
}
```

4. Add a method to handle successful payment from Express Checkout—`paymentConfirmAction()`; this method will capture the payment information from PayPal, confirm the payment, and then update the order status in our system using the code as shown in the following list:

   - Capture payment information from PayPal:

     ```
     // To capture Payer Information from PayPal
     $paypalSession = new \Zend\Session\Container('paypal');
     $paypalRequest = $this->getPaypalRequest();

     $expressCheckoutInfo =
             new \SpeckPaypal\Request\
     GetExpressCheckoutDetails();
     $expressCheckoutInfo->setToken($paypalSession->tokenId);
     $response = $paypalRequest->send($expressCheckoutInfo);
     ```

   - Confirm order with PayPal:

     ```
     //To capture express payment
     $orderTable = $this->getServiceLocator()-
     >get('StoreOrdersTable');
     $order = $orderTable->getOrder($paypalSession->orderId);
     $paymentDetails = new \SpeckPaypal\Element\
     PaymentDetails(array(
       'amt' => $order->total
     ));

     $token = $response->getToken();
     $payerId = $response->getPayerId();

     $captureExpress = new \SpeckPaypal\Request\
     DoExpressCheckoutPayment(
       array(
           'token'             => $token,
           'payerId'           => $payerId,
           'paymentDetails'    => $paymentDetails
       ));
     $confirmPaymentResponse = $paypalRequest-
     >send($captureExpress);
     ```

   - Save order with updated shipping/billing information:

     ```
     //To Save Order Information
     $order->first_name = $response->getFirstName();
     $order->last_name = $response->getLastName();
     $order->ship_to_street = $response->getShipToStreet();
     ```

```
$order->ship_to_city = $response->getShipToCity();
$order->ship_to_state = $response->getShipToState();
$order->ship_to_zip = $response->getShipToZip();
$order->email = $response->getEmail();
$order->store_order_id = $paypalSession->orderId;
$order->status = 'completed';
$orderTable->saveOrder($order);
```

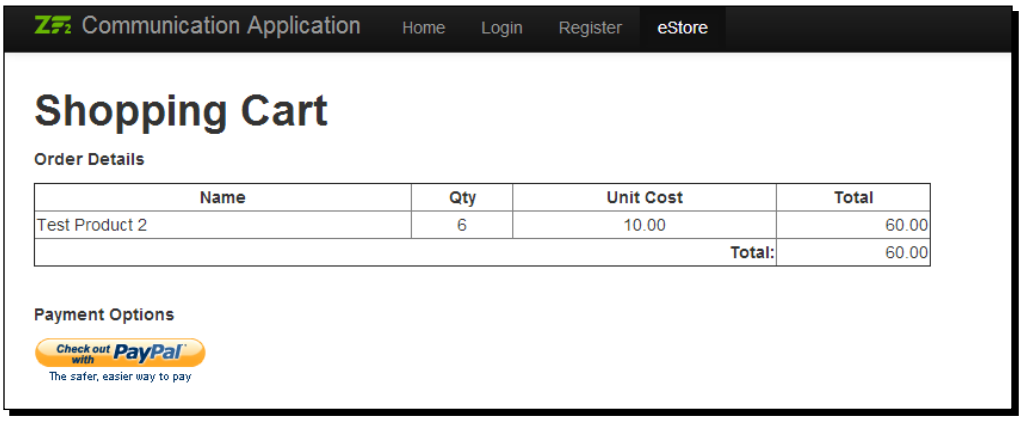
5. Finally add a method to handle failed payment from Express Checkout—`paymentCancelAction()`:

```
public function paymentCancelAction()
{
  $paypalSession = new \Zend\Session\Container('paypal');

  $storeOrdersTG = $this->getServiceLocator()
                     ->get('StoreOrdersTableGateway');
  $storeOrdersTG->update(
            array('status' => 'cancelled'),
            array('id' => $paypalSession->orderId));
  $paypalSession->orderId = NULL;
  $paypalSession->tokenId = NULL;
  $view = new ViewModel();
  return $view;
}
```

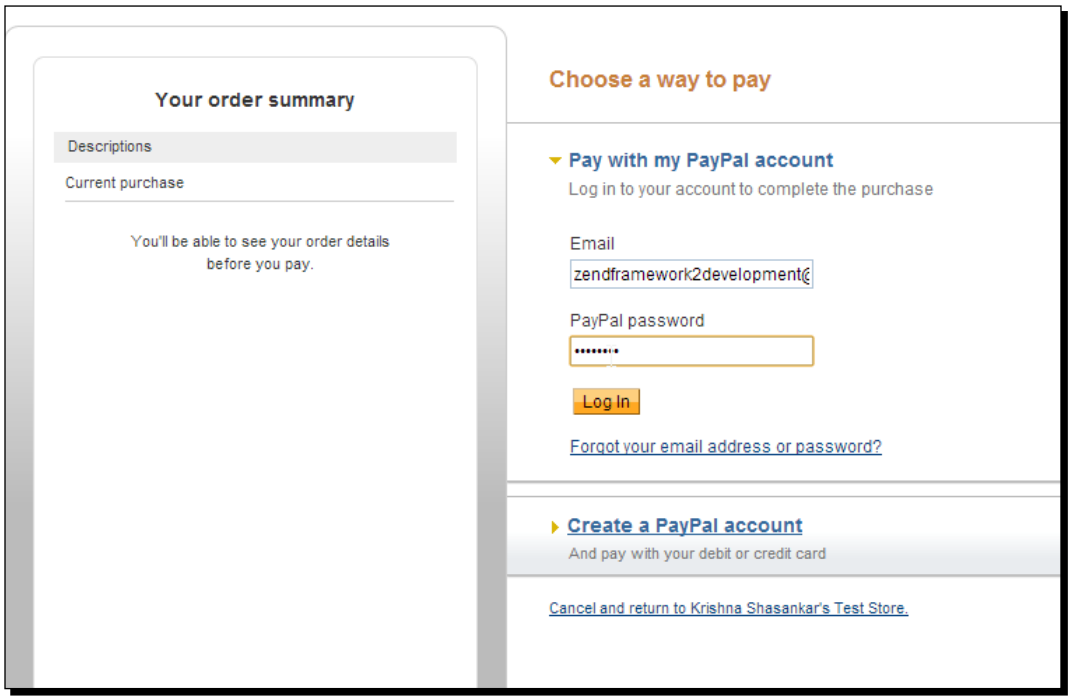
6. Now log in to `https://developer.paypal.com` again.

7. Generate a new sandbox account of type `PERSONAL`.

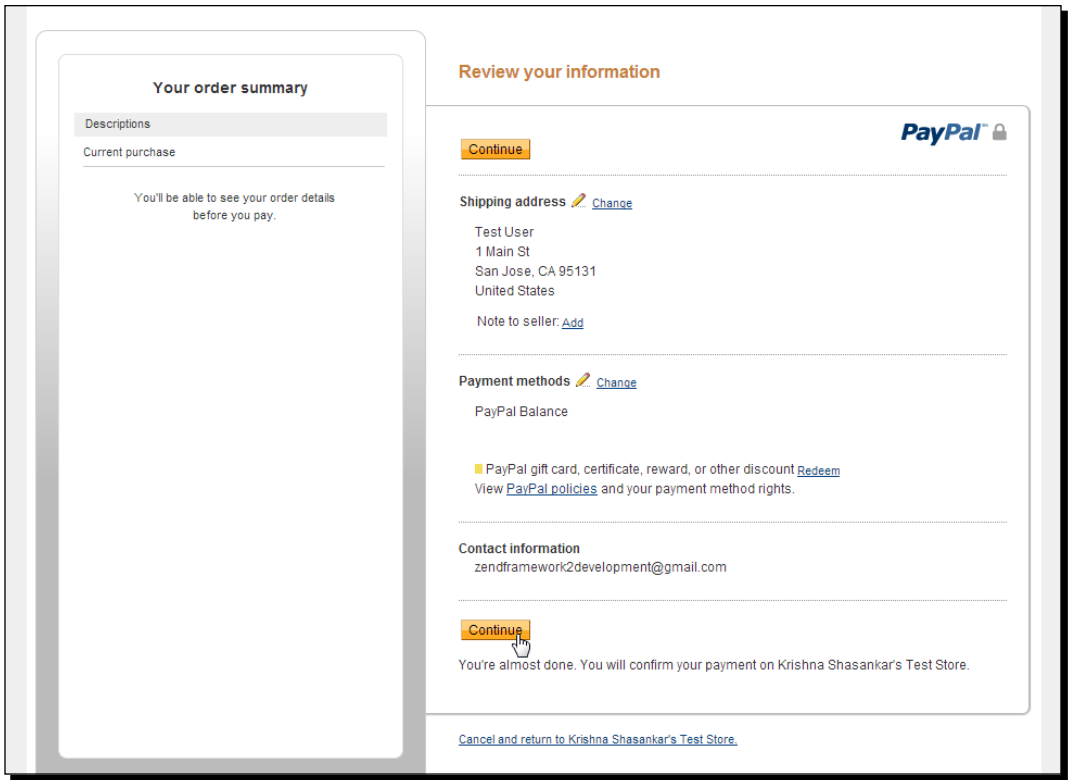
8. Now access the store and try to purchase using the newly created Sandbox account. The final store should look like the following screenshot:



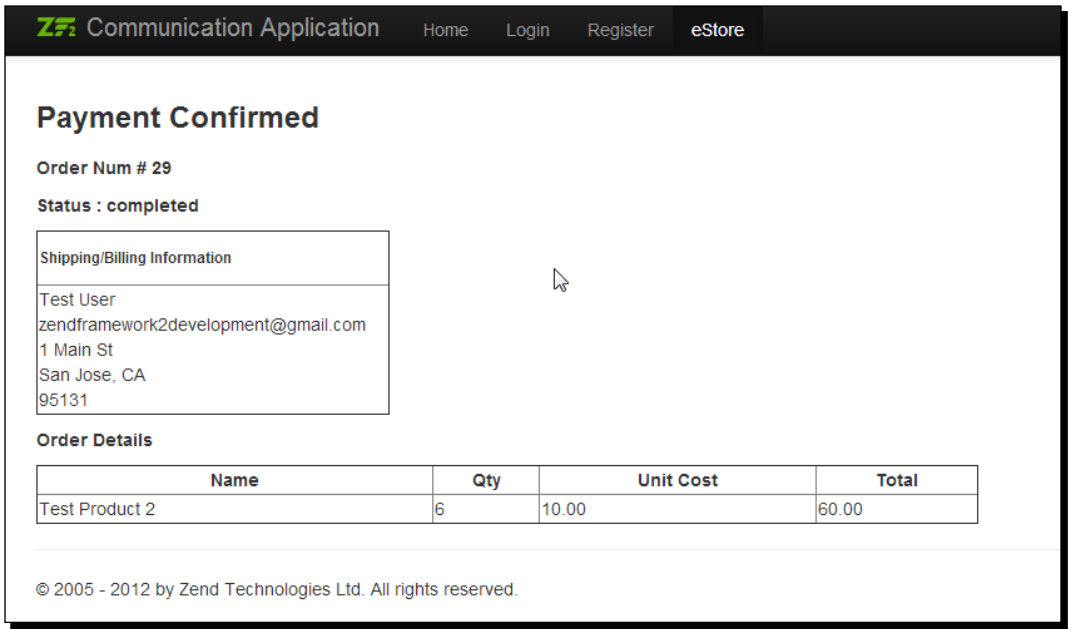
After choosing the checkout from the **Shopping Cart** page, you will be redirected to the **Pay with my PayPal account** login page as shown in the following screenshot:



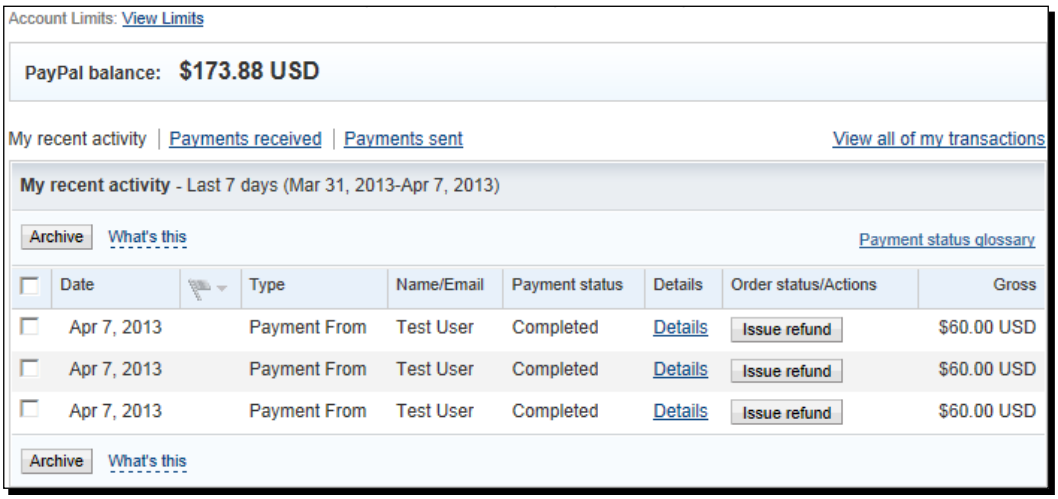
A screenshot of the PayPal Express Checkout's order reviewing page is shown in the following screenshot; this page is used to review the payment that is being made to the merchant from the customer's PayPal account:



Once the order is successfully placed, the user is redirected to the order confirmation page as shown in the following screenshot:



9. Now log in to the Sandbox site for the merchant account to see if the payments are credited:



What just happened?

We just used PayPal Express Checkout to receive payments in our web application and complete the simple store application. As you can see, the PayPal API makes it relatively easy to set up the payment gateway.

Have a go hero

In your next task, make use of the `DoDirectPayment` API call to directly make a payment on the website without having to redirect the user to the PayPal website and back again.

Pop quiz – creating a simple store

Q1. Which of the following methods is used to send the initial payment information for PayPal redirection?

1. `RedirectExpressCheckout`
2. `SetExpressCheckout`
3. `GetExpressCheckoutDetails`
4. `DoExpressCheckoutPayment`

Q2. Which of the following fields is needed for requesting payment information from PayPal?

1. `token`
2. `payerId`
3. `paymentDetails`
4. `orderID`

Summary

In this chapter we have learned the basics of setting up a simple store online and trying to receive payments using PayPal. As you can see from the previous example, Zend Framework's use of modules simplifies application development by giving developers the ability to download and install external third-party modules based on their integration needs. In the next chapter, we will be working on HTML5 development with Zend Framework 2.0.

9
HTML5 Support

HTML5 is the latest version of HTML specification; the final draft is not likely to be completed anywhere soon, but most browsers support a majority of features that are specified in the latest working draft.

Some of the most important offerings of HTML5 are listed as follows:

- Audio and video tags
- CSS3 support
- Support for drawing graphics using SVG and CSS3 2D and 3D
- Local storage, Web/JS workers, and geo location
- HTML5 form elements

For the scope of this book, we will be more focused on new form elements. HTML5 introduces a lot of new form elements. In previous versions of HTML, web developers were limited to use just the standard input types provided in the earlier HTML specifications. Now with the HTML5 specification, we have different elements for various different user inputs.

The list of newly available input elements is listed as follows:

- `datetime`
- `datetime-local`
- `time`
- `date`
- `week`
- `month`
- `email`
- `url`
- `number`
- `range`
- `color`
- `tel`
- `search`

HTML5 specification

For further reading, please refer to the HTML5.0 specification available on the W3C website: `http://www.w3.org/TR/html5/`.

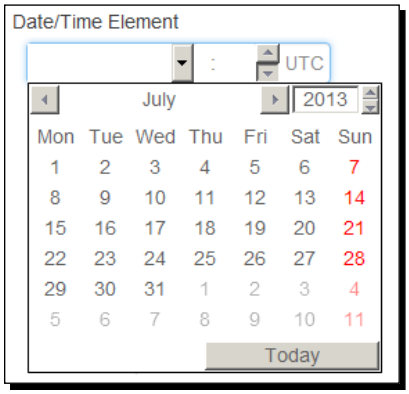
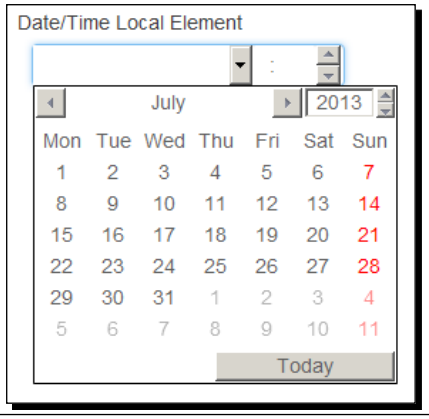
The following link points to specification for the `<input>` element:

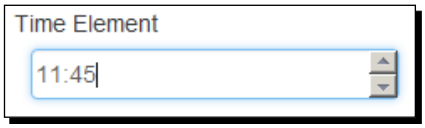
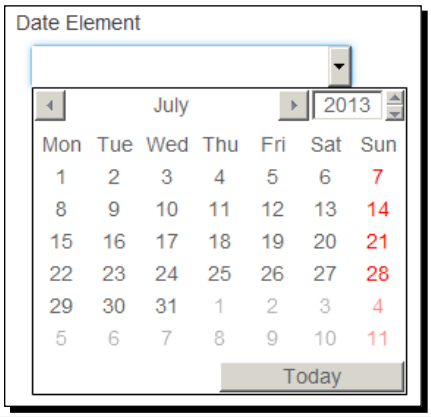
`http://www.w3.org/TR/html5/forms.html#the-input-element`

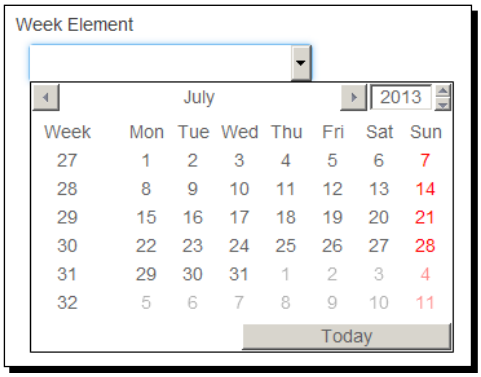
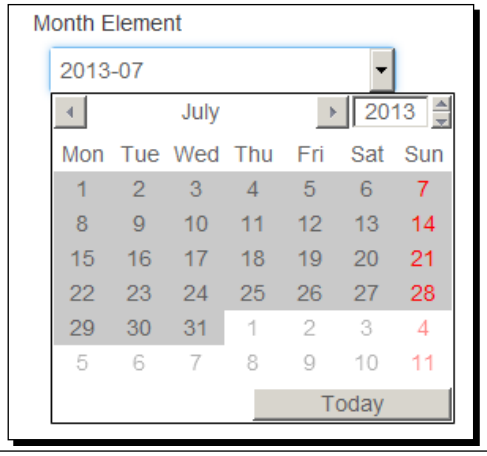
In this chapter we will understand the usage of these input elements.

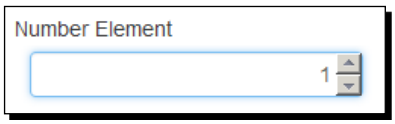
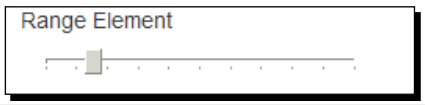
HTML5 input elements

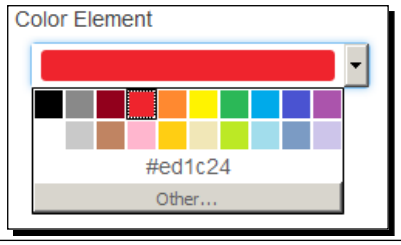
Zend Framework 2.0 now supports all of the newly specified HTML5 input types; these inputs are available under `Zend\Form\Element` like any other input types. The following table describes each of these elements along with their class names:

Input type	Description
`datetime`	◆ Element: `Zend\Form\Element\DateTime` ◆ Used to render the **Date/Time Element** input field with the time zone set to `UTC` ◆ HTML tag: `<input type="datetime" name="element-date-time">` ◆ The `datetime` element rendered in Opera 12.0 is shown in the following screenshot: 
`datetime-local`	◆ Element: `Zend\Form\Element\DateTimeLocal` ◆ Used to render the **Date/Time Local Element** input field for the client browser's time zone ◆ HTML tag: `<input type="datetime-local" name="element-date-time-local">` ◆ The `datetime-local` element rendered in Opera 12.0 is shown in the following screenshot: 

Input type	Description
`time`	◆ Element: `Zend\Form\Element\Time` ◆ Used to render the **Time Element** field ◆ HTML tag: `<input type="time" name="element-time">` ◆ The time element rendered in Opera 12.0 is shown in the following screenshot: 
`date`	◆ Element: `Zend\Form\Element\Date` ◆ Used to render the **Date Element** field ◆ HTML tag: `<input type="date" name="element-date">` ◆ The `date` element rendered in Opera 12.0 is shown in the following screenshot: 

Input type	Description
`week`	◆ Element: `Zend\Form\Element\Week` ◆ Used to render the **Week Element** field ◆ HTML tag: `<input type="week" name="element-week">` ◆ The `week` element rendered in Opera 12.0 is shown in the following screenshot: 
`month`	◆ Element: `Zend\Form\Element\Month` ◆ Used to render the **Month Element** field ◆ HTML tag: `<input type="month" name="element-month">` ◆ The `month` element rendered in Opera 12.0 is shown in the following screenshot: 

Input type	Description
`email`	◆ Element: `Zend\Form\Element\Email` ◆ Used to render the **Email** input field ◆ HTML tag: `<input type="email" name="element-email">`
`url`	◆ Element: `Zend\Form\Element\Url` ◆ Used to render the **URL** input field ◆ HTML tag: `<input type="url" name="element-url">`
`number`	◆ Element: `Zend\Form\Element\Number` ◆ Used to render the **Number Element** input field ◆ HTML tag: `<input type="number" name="element-number">` ◆ The `number` element rendered in Opera 12.0 is shown in the following screenshot: 
`range`	◆ Element: `Zend\Form\Element\Range` ◆ Used to render the **Range Element** input field using slider control ◆ HTML tag: `<input type="range" name="element-range">` ◆ The `range` element rendered in Opera 12.0 is shown in the following screenshot: 

Input type	Description
`color`	◆ Element: `Zend\Form\Element\Color` ◆ Used to render the **Color Element** input field with a color picker ◆ HTML tag: `<input type="color" name="element-color">` ◆ The `color` element rendered in Opera 12.0 is shown in the following screenshot: 

Time for action – HTML5 input elements

In this example we will be creating a test HTML5 form for rendering various types of HTML5 input elements:

1. Create a test action for rendering the form element `formAction()`; it can be created under the new controller `Html5TestController - module/Users/src/Users/Controller/Html5TestController.php`.
2. Add references to `Zend\Form\Form` and `Zend\Form\Element`:

```
use Zend\Form\Element;
use Zend\Form\Form;
```

3. Add various HTML5 form elements to the form:

```
$form  = new Form();

// Date/Time Element
$dateTime = new Element\DateTime('element-date-time');
$dateTime
->setLabel('Date/Time Element')
->setAttributes(array(
  'min'  => '2000-01-01T00:00:00Z',
  'max'  => '2020-01-01T00:00:00Z',
  'step' => '1',
));
```

```
$form->add($dateTime);

// Date/Time Local Element
$dateTime = new Element\DateTimeLocal('element-date-time-local');
$dateTime
->setLabel('Date/Time Local Element')
->setAttributes(array(
  'min'  => '2000-01-01T00:00:00Z',
  'max'  => '2020-01-01T00:00:00Z',
  'step' => '1',
));
$form->add($dateTime);

// Time Element
$time = new Element\Time('element-time');
$time->setLabel('Time Element');
$form->add($time);

// Date Element
$date = new Element\Date('element-date');
$date
->setLabel('Date Element')
->setAttributes(array(
  'min'  => '2000-01-01',
  'max'  => '2020-01-01',
  'step' => '1',
));
$form->add($date);

// Week Element
$week = new Element\Week('element-week');
$week->setLabel('Week Element');
$form->add($week);

// Month Element
$month = new Element\Month('element-month');
$month->setLabel('Month Element');
$form->add($month);

// Email Element
$email = new Element\Email('element-email');
$email->setLabel('Email Element');
```

```
$form->add($email);

// URL Element
$url = new Element\Url('element-url');
$url->setLabel('URL Element');
$form->add($url);

// Number Element
$number = new Element\Number('element-number');
$number->setLabel('Number Element');
$form->add($number);

// Range Element
$range = new Element\Range('element-range');
$range->setLabel('Range Element');
$form->add($range);

// Color Element
$color = new Element\Color('element-color');
$color->setLabel('Color Element');
$form->add($color);
```

What just happened?

We have created a simple form purely using HTML5 elements that are supported by Zend Framework 2.0. The form in its current shape can be rendered by creating the necessary view. Our next task will be to build the view for this form with the use of HTML5 helpers and render all the form elements that were added to the form.

HTML5 view helpers

Zend Framework provides view helpers for rendering all the form elements described in the previous section. The `formElement()` view helper can be used to render any kind of input dynamically based on the input type, however it is not the suggested practice.

The following table gives you the list of standard HTML5 helpers available for the HTML5 input elements:

Input type	Helper	Helper function
datetime	Zend\Form\View\Helper\FormDateTime	formDateTime()
datetime-local	Zend\Form\View\Helper\FormDateTimeLocal	formDateTimeLocal()
time	Zend\Form\View\Helper\FormTime	formTime()
date	Zend\Form\View\Helper\FormDate	formDate()
week	Zend\Form\View\Helper\FormWeek	formWeek()
month	Zend\Form\View\Helper\FormMonth	formMonth()
email	Zend\Form\View\Helper\FormEmail	formEmail()
url	Zend\Form\View\Helper\FormUrl	formUrl()
number	Zend\Form\View\Helper\FormNumber	formNumber()
range	Zend\Form\View\Helper\FormRange	formRange()
color	Zend\Form\View\Helper\FormColor	formColor()

Apart from the standard list of view helpers, Zend Framework also provides helpers for the `tel` and `search` input types; these input types are an extension of the text input, but certain browsers (especially mobile browsers) support stylized input options in both these elements.

The following table gives you the list of additional HTML5 helpers available for the HTML5 input elements:

Input type	Helper	Helper function
tel	Zend\Form\View\Helper\FormTel	formTel()
search	Zend\Form\View\Helper\FormSearch	formSearch()

Time for action – HTML5 view helpers

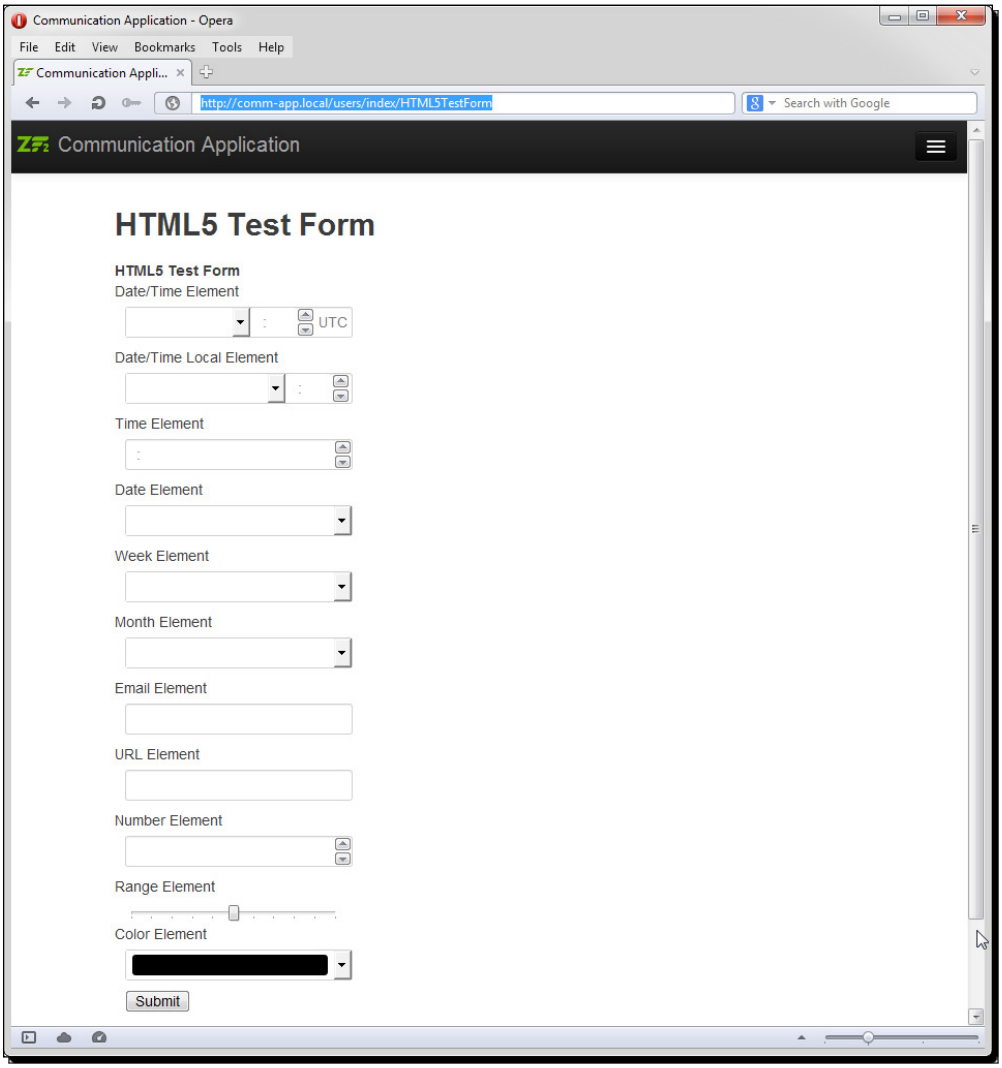
In this task we will render all the form elements that we created in the previous task. We will make use of ZF's HTML5 view helpers to render these elements. Perform the following steps:

1. Create a simple view that can be used to render the form.
2. Make use of view helpers to render various form elements using the following code:

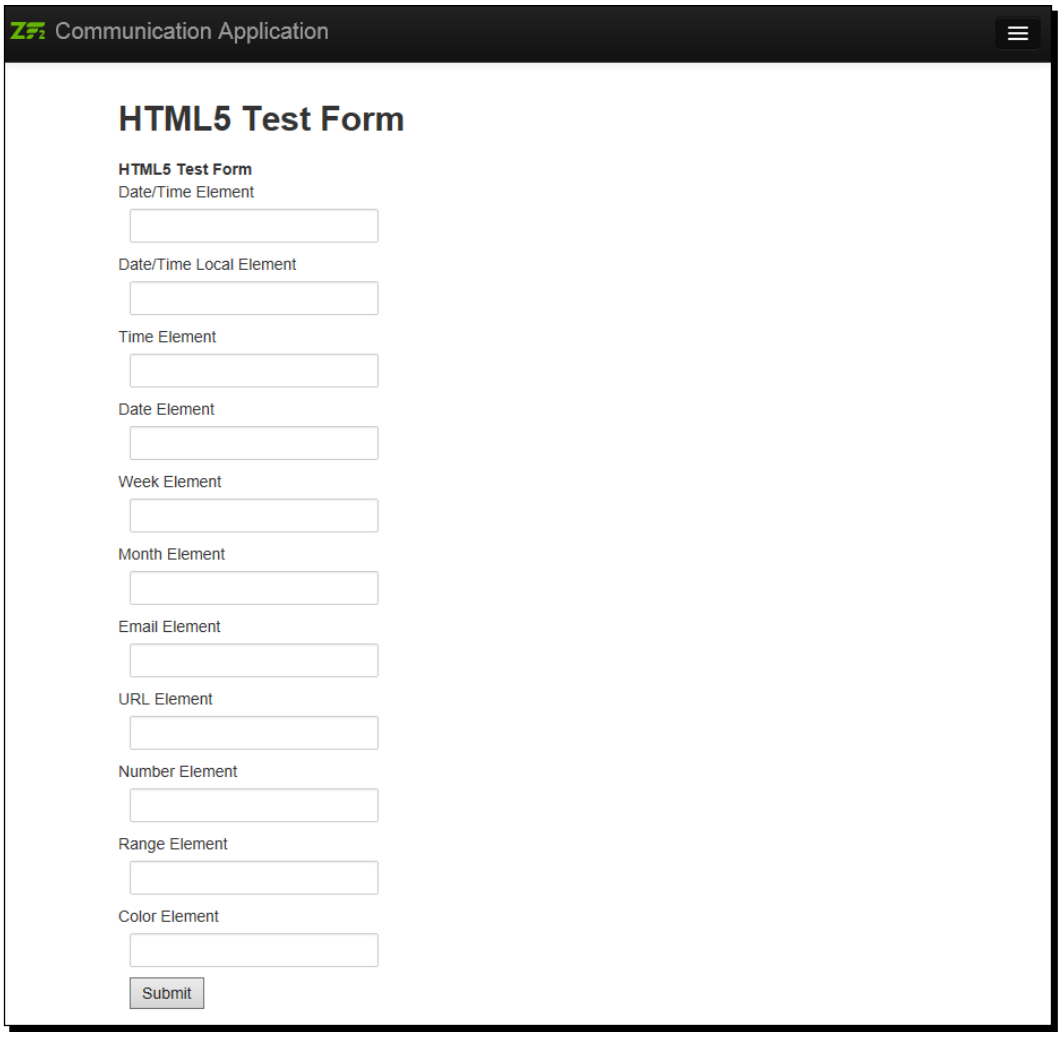
```
$this->formDateTime($form->get('element-date-time'));
$this->formDateTimeLocal($form->get('element-date-time-local'));
$this->formTime($form->get('element-time'));
```

```
$this->formDate($form->get('element-date'));
$this->formWeek($form->get('element-week'));
$this->formMonth($form->get('element-month'));
$this->formEmail($form->get('element-email'));
$this->formUrl($form->get('element-url'));
$this->formNumber($form->get('element-number'));
$this->formRange($form->get('element-range'));
$this->formColor($form->get('element-color'));
```

3. Test the form in an HTML5-compatible browser such as Opera 12. You should be able to see a form like the one shown in the following screenshot:



4. Now, test the same form in an HTML5 non-compatible browser such as IE 9. You should be able to see a form like the one shown in the following screenshot. You can see that the unsupported input elements are replaced with textboxes:



What just happened?

We have created our first HTML5 form using ZF2 form elements. As of now, Opera 12 offers the best support for HTML5; other browsers such as Chrome and Safari are also good in terms of support. So, if you are testing your HTML5 forms, make sure that you are testing them in a browser that is compatible, such as Opera 12.

HTML5 browser compatibility

Support for HTML5 specifications is inconsistent among various browsers; Opera and Chrome seem to offer best support in terms of compliance, but none of them are fully compliant. With each new browser version, there is additional support for these features. There are many resources available on the Internet that allow you to check your browser's compatibility with HTML5.

`http://html5test.com/` is a portal that ranks and compares browsers based on their HTML5 support.

`http://caniuse.com/` is also a great website that lets users check if they can use a specific HTML5 feature on a specific browser.

Have a go hero

Here is a simple task for you before you move on to using advanced HTML5 attributes. Now that you have created a form using all the standard HTML5 elements, try to extend the form by using the view helpers to render the `tel` and `search` type inputs.

HTML5 attributes

You might have noticed in the beginning of the chapter that we were using new attributes such as `min`, `max`, and `step`. These are new attributes that are defined in the HTML5 specification that allow developers to specify additional configuration on the input element. Some important attributes are discussed in the following list:

- `max`: Applicable to the **Number**, **Range**, and **Date** fields; allows specification of maximum value in the input.
- `min`: Applicable to the **Number**, **Range**, and **Date** fields; allows specification of a minimum value in the input.
- `step`: Applicable to the **Number**, **Range**, and **Date** fields; allows specification of an increment value in the input.
- `list`: Applicable to various textbox style inputs. Allows developers to map the field to a data list, thus allowing end users to pick them from the list.
- `placeholder`: Applicable to various textbox style inputs. Allows developers to show placeholder text until the element gains focus.
- `pattern`: Applicable to various textbox style inputs. Allows developers to validate the user input against a regular express-and-throw-a-validation error.

- `required`: Prevents users from submitting the form with empty values in the required fields.
- `multiple`: Applicable to file input; allows multiple file uploads from a single file control.

Multiple file uploads

For implementing multiple file uploads, you will need to set the `multiple` attribute on the file input element to `TRUE`. If the browser supports multiple file uploads, then the user will be allowed to select multiple files, otherwise the control will limit to just one file selection.

Time for action – HTML5 multiple file uploads

Perform the following steps for HTML5 multiple file uploads:

1. Create a new `ImageUpload` form; make sure that the `multiple` attribute for the `File` element is set to `TRUE`:

```
<?php
// filename : module/Users/src/Users/Form/MultiImageUploadForm.php
namespace Users\Form;

use Zend\Form\Form;
use Zend\Form\Element;
use Zend\InputFilter;

class MultiImageUploadForm extends Form
{
    public function __construct($name = null, $options = array())
    {
        parent::__construct($name, $options);
        $this->addElements();
        $this->addInputFilter();
    }

    public function addElements()
    {
      $imageupload = new Element\File('imageupload');
      $imageupload->setLabel('Image Upload')
        ->setAttribute('id', 'imageupload')
        ->setAttribute('multiple', true);
        //Enables multiple file uploads
```

```
        $this->add($imageupload);

        $submit = new Element\Submit('submit');
        $submit->setValue('Upload Now');
        $this->add($submit);
      }

      public function addInputFilter()
      {
        $inputFilter = new InputFilter\InputFilter();
        // File Input
        $fileInput = new InputFilter\FileInput('imageupload');
        $fileInput->setRequired(true);
        $fileInput->getFilterChain()->attachByName(
          'filerenameupload',
          array(
            'target' => './data/images/temp.jpg',
            'randomize' => true
          )
        );
        $inputFilter->add($fileInput);
        $this->setInputFilter($inputFilter);
      }
}
```

Zend\Filter\File\RenameUpload

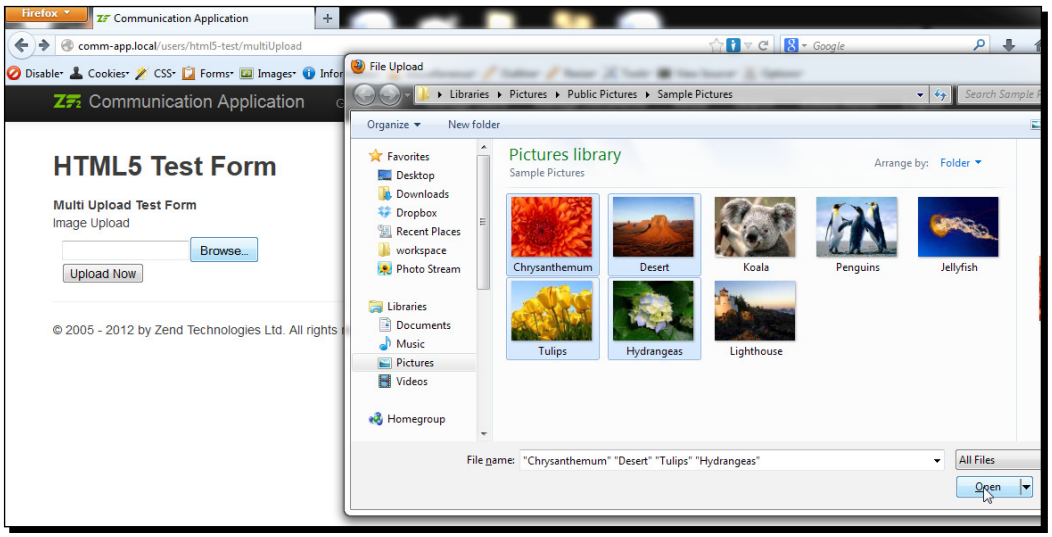
The `RenameUpload` filter is used to rename and move the uploaded file to a new path specified in the `target`. To find out more please refer to the framework documentation at `http://framework.zend.com/manual/2.2/en/modules/zend.filter.file.rename-upload.html`.

2. Set up an action to handle the file uploads, and to redirect the user to an upload confirmation page:

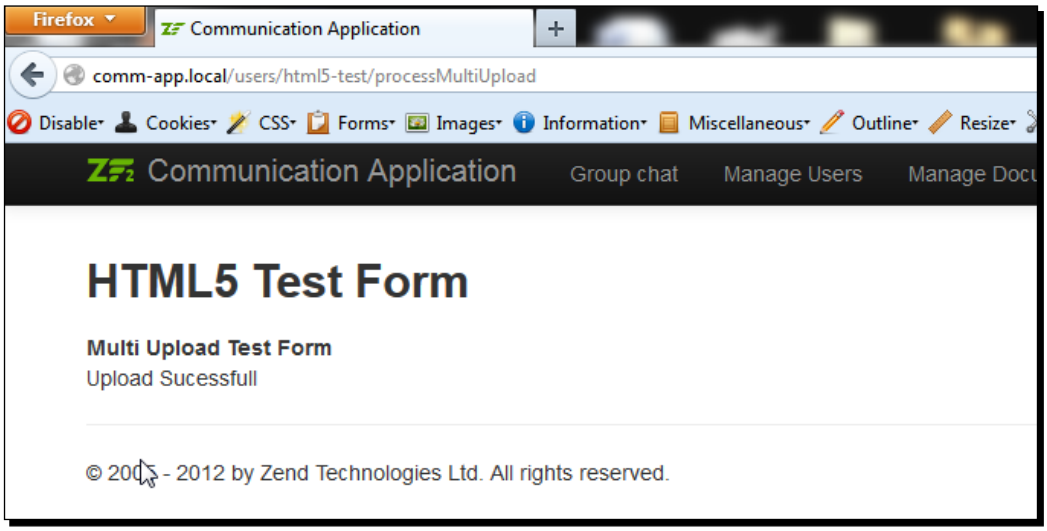
```
public function multiUploadAction()
{
  // prepare form
  $form = $this->getServiceLocator()->get('MultiImageUploadForm');
  $request = $this->getRequest();
  if ($request->isPost()) {
    $post = array_merge_recursive(
      $request->getPost()->toArray(),
      $request->getFiles()->toArray()
```

```
        );
            $form->setData($post);
        if ($form->isValid()) {
          $data = $form->getData();
          // Form is valid, save the form!
          return $this->redirect()->toRoute('users/html5-test',
array('action' => 'processMultiUpload'));
        }
      }
      $viewModel  = new ViewModel(array('form' => $form));
      return $viewModel;
    }
```

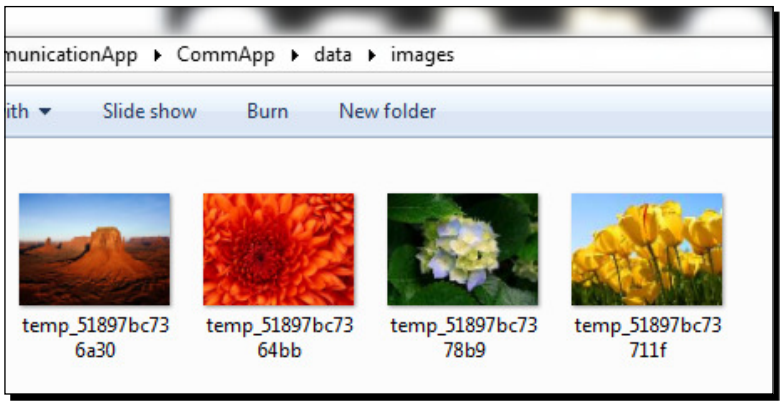
3. Now test the form in your browser that supports multiple file uploads with HTML5, for example, Opera 12. You will see that the file selector interface allows the selection of more than one file as shown in the following screenshot:



4. After you choose **Upload now** and once the upload process is completed, you will see the confirmation page as shown in the following screenshot:



5. You can verify if the files are uploaded successfully and the filters are applied by navigating through the `data/images` directory and looking up for the uploaded files. You can see that all files start with `temp` and have a `_<random_number>` suffix in their filenames:



Filters with multiple file uploads

When applying filters with multiple file uploads, the filter(s) will be applied to all the files that are successfully uploaded with the same filter option settings.

What just happened?

We have now created an HTML5 multiple file upload form using HTML5 attributes and Zend form elements. We have also applied a filter to rename the uploaded files and have also seen how filters work in multiple file uploads.

Pop quiz – HTML5 support

Q1. Which of the following methods is a newly supported HTML5 input type?

1. `text`
2. `radio`
3. `checkbox`
4. `number`

Which of the following input types do not have a `Form` element defined in ZF 2.1?

1. `tel`
2. `date`
3. `color`
4. `search`

Summary

HTML5 is a very robust and powerful specification of HTML which is still partially supported by most browsers. As newer versions of browsers come out in the market, you will get to see much more enhanced support for this specification. In our next chapter, we will be using ZF2 to build mobile web applications.

10
Building Mobile Applications

One of the major hurdles in mobile application development is the diversified number of platforms that have to be targeted while building mobile applications. Platforms such as PhoneGap and Titanium enable developers to build cross-platform mobile applications, but one of the disadvantages with this model is to manage multiple projects on different platforms for mobile and web services. Zend, with the release of Zend Studio 10, has tried to address the same gap by providing a development platform based on PhoneGap, which supports end-to-end mobile apps in a cloud-based environment.

With the release of Zend Studio 10, Zend now offers extremely simplified mobile application development platform using Zend Framework 2, known as **Cloud Connected Mobile Tool**. In this chapter we will be learning about the basics of building cloud-connected mobile applications using Zend Studio. Some of the key learning areas discussed are as follows:

- Building your first **cloud-connected mobile** (**CCM**) application
- Testing as a native application
- Implementing a simple search interface

Cloud-connected mobile applications

Zend Studio now offers a CCM tool enabling developers to build native mobile applications using the cloud platform. CCM supports development of RPC-based or REST-based web services for the cloud using Zend Framework 2 and Zend Server Gateway.

CCM also offers support for developing native mobile applications by integrating with various mobile SDKs (Android SDK/ADT for Android, Xcode for iOS, and Windows Phone SDK for Windows Phone). This enables developers to build and test the applications in native environments/devices.

CCM tool also offers a simple and easy-to-use mobile GUI editor which helps developers to effortlessly build great user interfaces for their mobile applications.

Zend Studio 10

As a first step towards building your mobile application, please ensure that you install Zend Studio 10 on your development machine. Zend Studio 10 offers integrated support for building cloud-connected mobile applications and allows developers to deploy their mobile application on the cloud.

Zend Studio 10 is available for purchase from the Zend Online Store; there is a free 30-day trial as well. For further information visit `http://www.zend.com/en/products/studio/`.

phpCloud

Zend Developer Cloud is a cloud-based PHP development environment, which enables developers to build and deploy applications on the cloud, without undergoing the hassle of setting up a PHP development environment, and configuring and maintaining the environment.

This environment has Zend Framework 2 installed with a large set of PHP extensions; developers can make use of various development tools such as Zend Studio, Eclipse PDT, and CLI to build and deploy their applications on the developer cloud. Zend Developer Cloud also provides capabilities to push your application to other external cloud services such as Amazon and IBM SmartCloud.

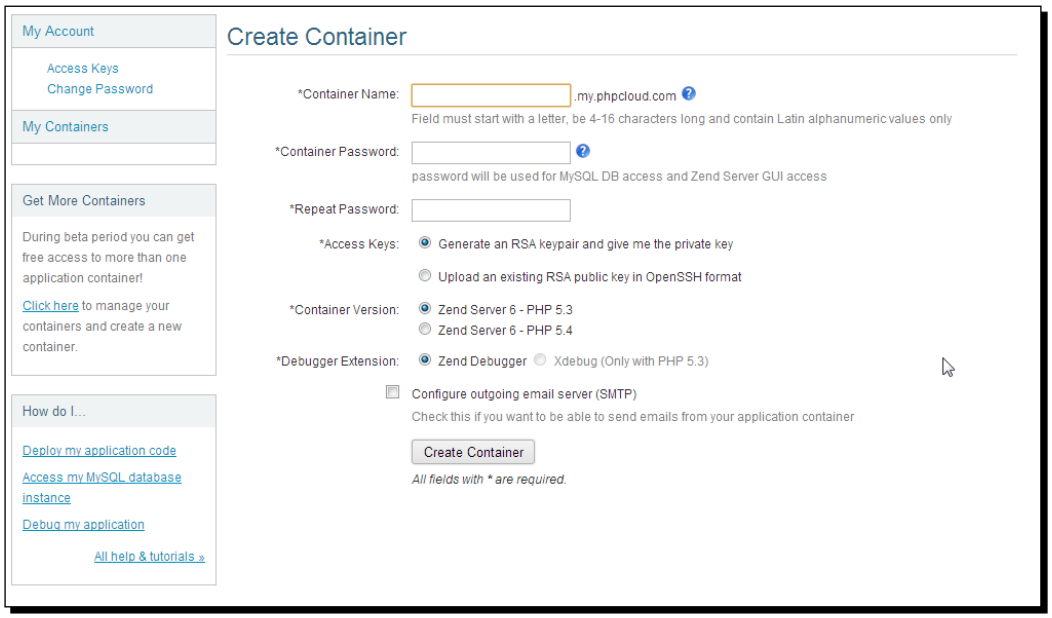
Zend Developer Cloud is currently in free developer beta. For further information about Zend Developer Cloud, please refer to their website: `http://www.phpcloud.com/`.

Time for action – configuring your phpCloud account

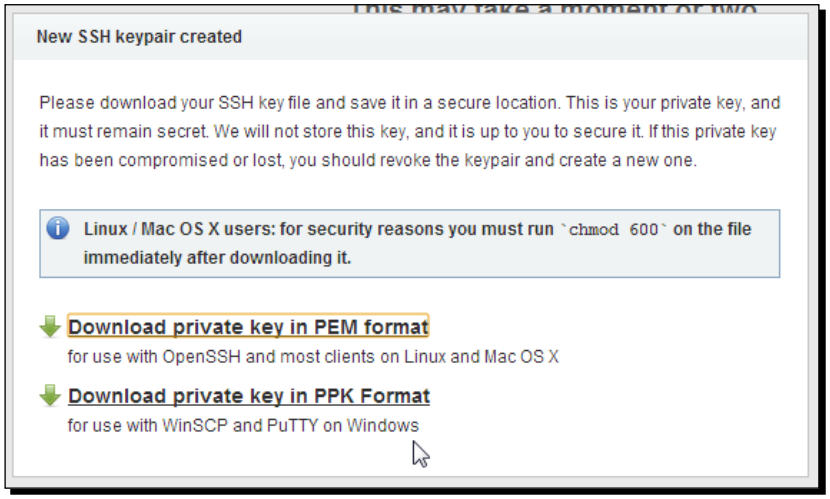
In this task we will set up our phpCloud account and configure the cloud environment in Zend Studio 10 using the following steps:

1. Visit `https://my.phpcloud.com/user/login`, register for a new account, and log in to your phpCloud account.

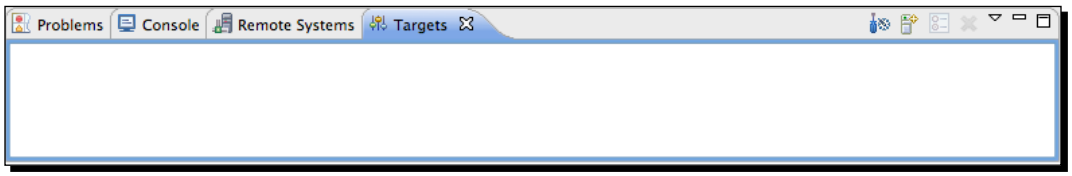
2. After the login, you will be asked to create a container. You can specify a container name which will be a part of the container URL; you can also choose to generate a SSH key pair or use your own SSH keys; in this case, we will generate a new SSH key pair. The following screenshot describes the container creation screen:



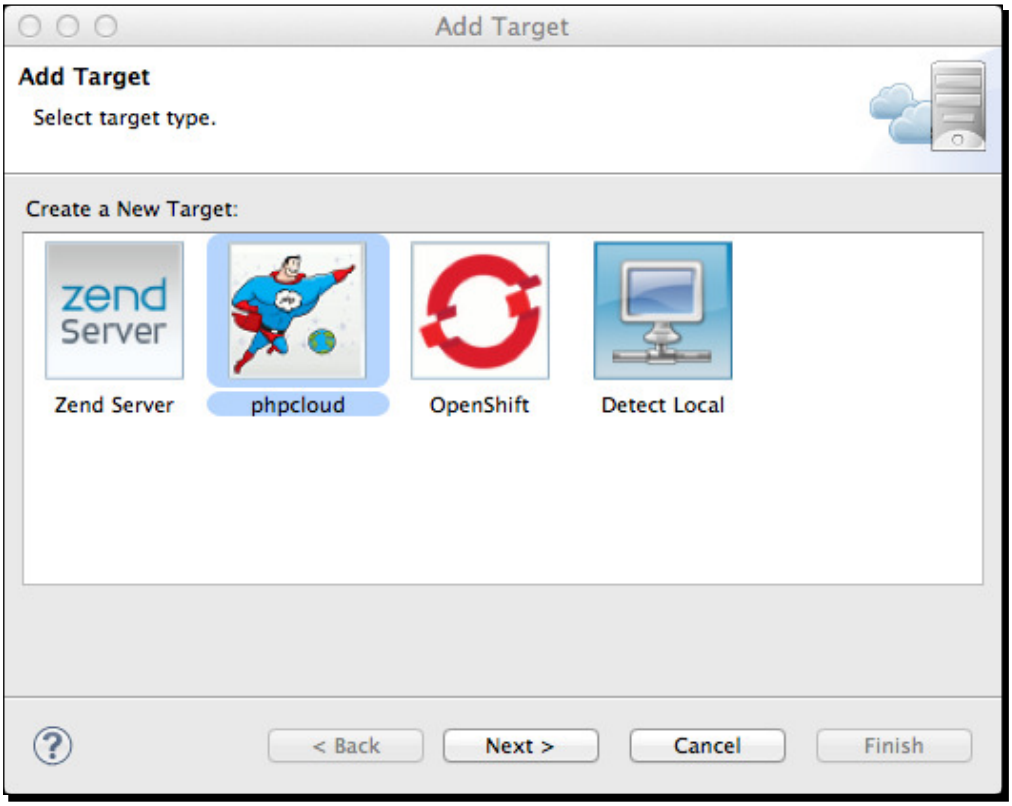
3. Now download the SSH keys; we will be using these keys to set up our deployment target in Zend Studio:



4. In Zend Studio, navigate to **Window** | **Show View** | **Targets**:



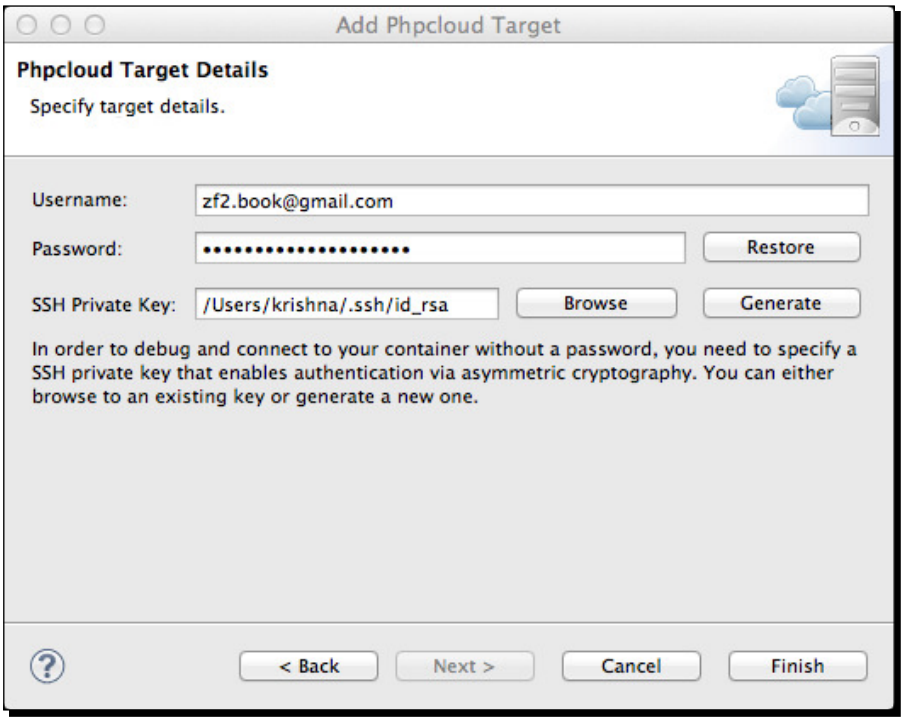
5. Click on the **Add Target** icon and choose **phpcloud** as shown in the following screenshot:



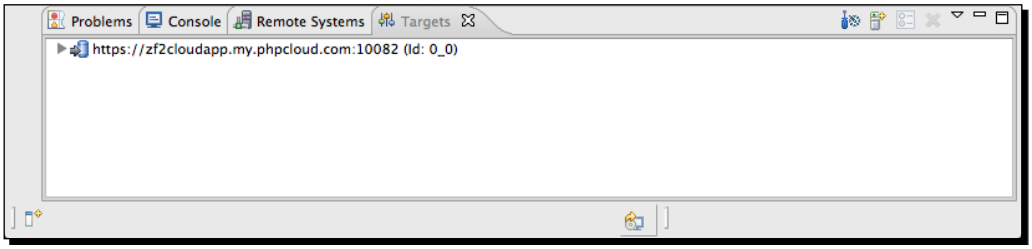
6. On the **phpcloud Target Details** page, you will be asked to provide the following details:

- **Username**: Used to specify your Zend Developer Cloud username
- **Password**: Used to specify your Zend Developer Cloud password

- **SSH Private Key**: Used to point to the SSH key that was just generated in the phpcloud container creation screen



7. After you click on **Finish**, you will see that the new target is added to the list of targets:



What just happened?

We have successfully created our first mobile application using Zend's cloud-connected mobile application projects. In the subsequent sections we will understand how to extend these web services using Zend Framework 2 to build additional functionality into mobile applications.

PhoneGap and Zend Studio

PhoneGap is a mobile application development framework which allows developers to build mobile applications using HTML, CSS, and JavaScript. The PhoneGap framework is used to convert these applications into native mobile applications, without having to rewrite the applications in native languages like Objective-C for iOS.

Zend Studio 10 now integrates PhoneGap into the Zend Studio IDE; this enables developers to easily build and test mobile applications without having to depend on external libraries.

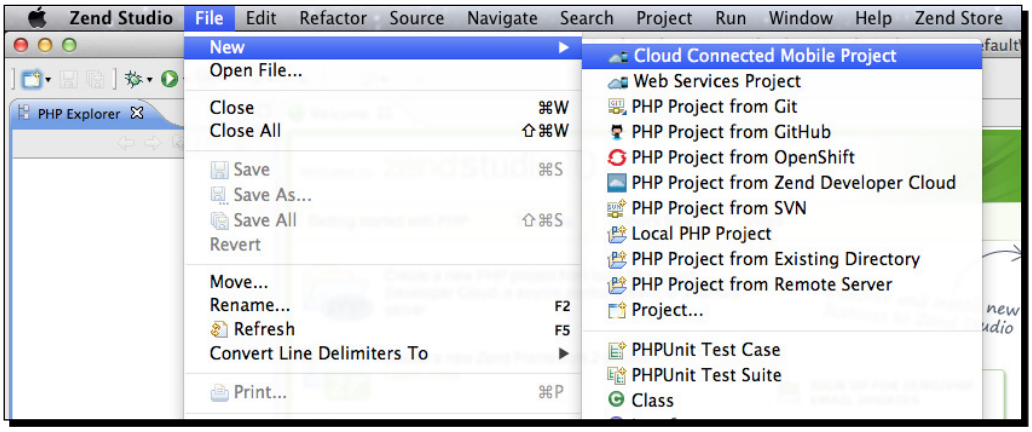
For more information on cloud-connected mobile applications using Zend Studio 10; please refer the following documentation page:

`http://files.zend.com/help/Zend-Studio-10/zend-studio.htm#cloud_connect_mobile.htm`

Time for action – building your first cloud-connected mobile application

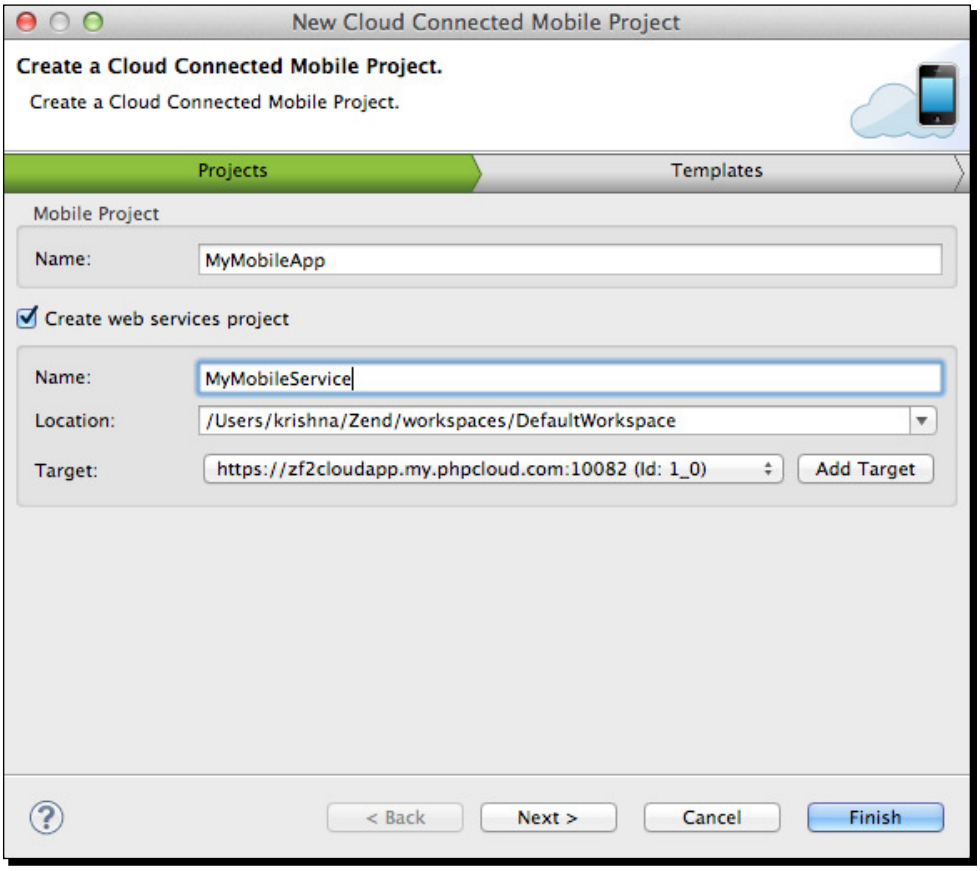
Perform the following steps for building your first cloud-connected mobile application:

1. Choose the **Cloud Connected Mobile Project** option from the **New** menu:

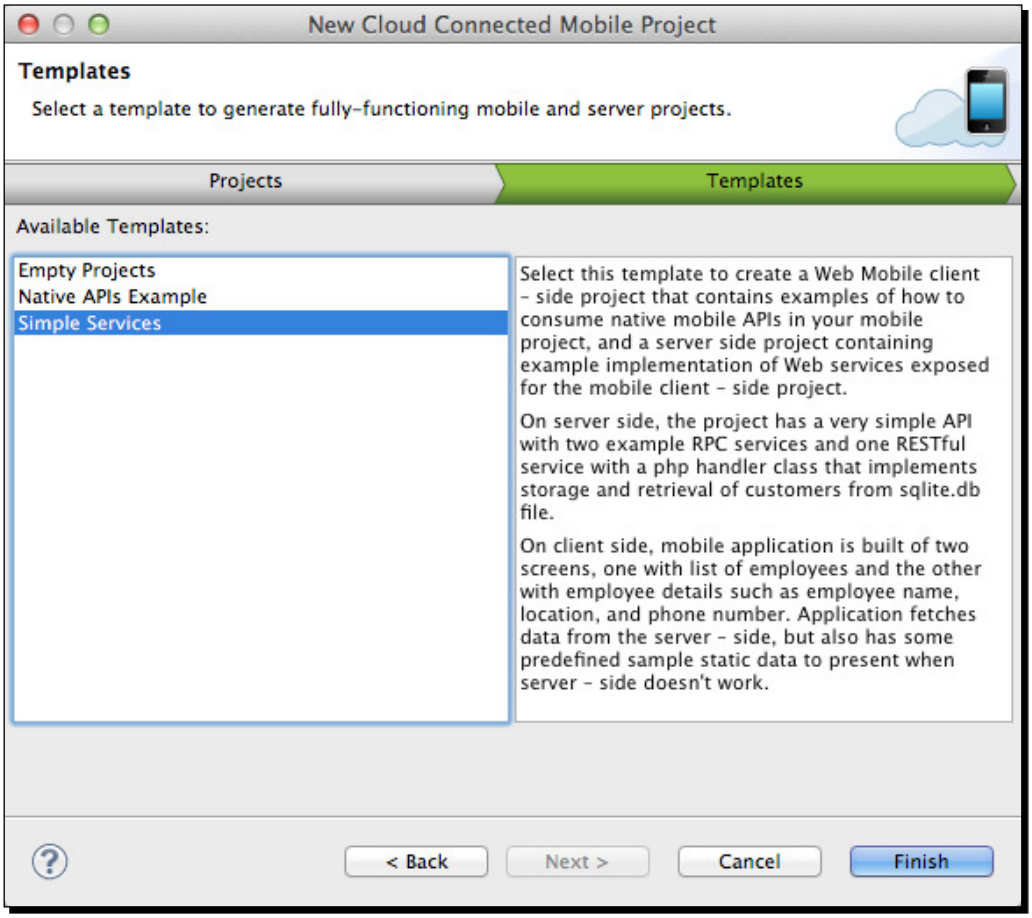


2. In the **Project** wizard, you will be asked to provide the following details:
   - **Mobile Project Name**: Name of the client-side mobile application project
   - **Web Services Project Name**: Name of the web services project for the mobile application
   - **Web Services Project Deployment Target**: Deployment target for the mobile application (you can choose the previously created phpcloud target here)

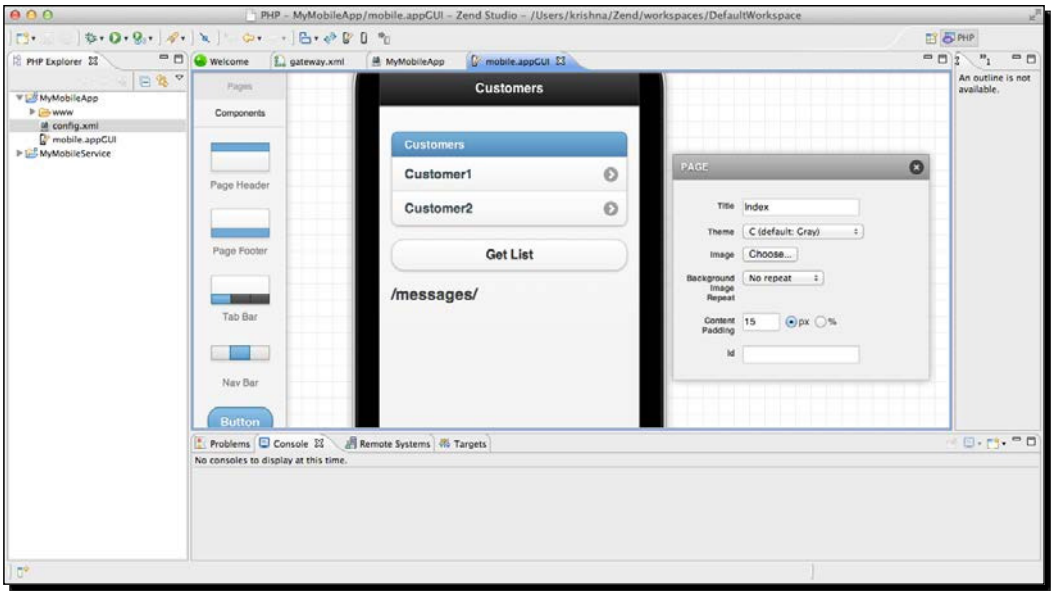
Zend Studio 10 supports various deployment options; it can automatically detect local Zend Server installation or deploy an application to one of the targets— the local Zend Server, remote Zend Server, Zend Developer Cloud (phpCloud), or OpenShift Cloud.



3. In the template selection page, choose **Simple Services** as it will create a simple project with a client/server-side example as shown in the following screenshot:



4. Clicking on **Finish** will create the mobile and web services projects. The user interface designer in the mobile project lets us easily make changes to the mobile interface as shown in the following screenshot:

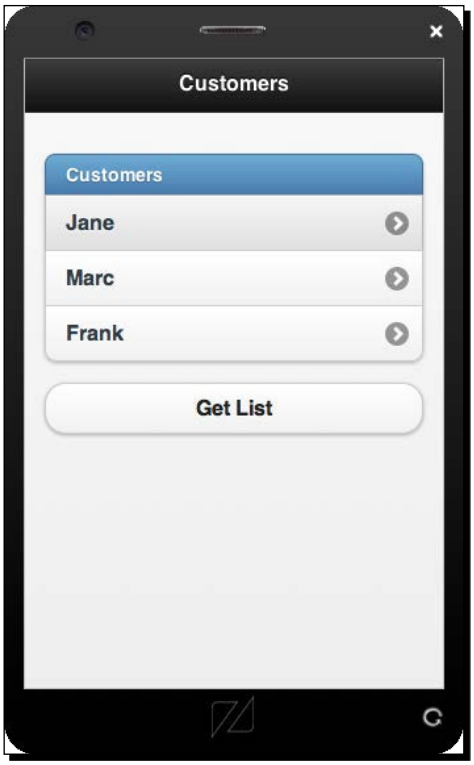


5. Now run the project from the Zend Studio IDE; it should launch a Zend emulator interface as shown in the next screenshot:

The **Get List** button should return the list of customers from the web services project via an RPC call. If the request doesn't return a response and throws an error such as **Ajax error. Error: Access is denied. Trying static data!**, then check the `gatewayURL` variable in `MobileApplication/www/js/my.js`.

Make sure it points to the correct deployment URL as follows:

```
var gatewayURL = 'http://zf2cloudapp.my.phpcloud.
com/MobileService';
```



What just happened?

We have successfully created our first mobile application using Zend's cloud-connected mobile application projects. In the subsequent sections we will understand how to extend these web services using Zend Framework 2 to build additional functionality into mobile applications.

Native applications versus mobile web applications

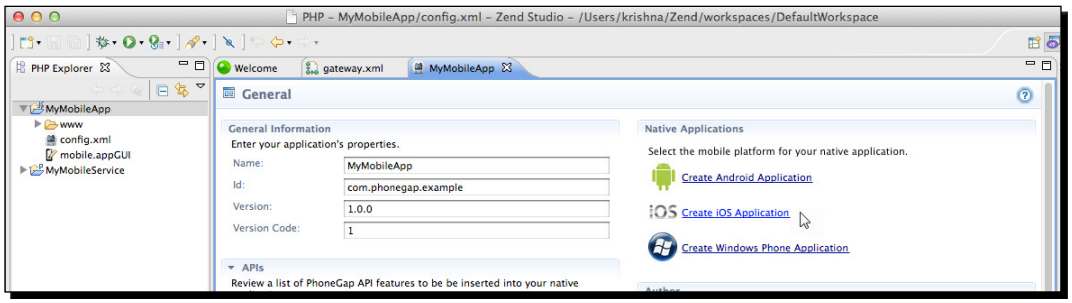
Native mobile applications provide great benefits over mobile web applications. Native web applications are run from the device memory, so there is little need for network interaction; these applications tend to load and run faster. One of the other key advantages of native mobile applications is that they have access to the device's native features such as camera, device information, and accelerometer; this gives native applications an added advantage over mobile web applications.

Time for action – testing as a native application

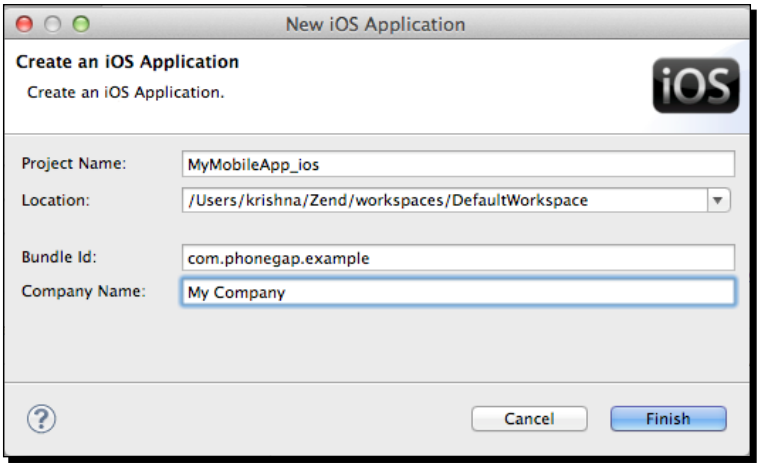
In this task we will create a native iOS application using the **Native Applications** section of Zend Studio. Before you get started, make sure that Xcode IDE in installed on your Mac. Perform the following steps:

> For Android applications, you will need to have **Android Development Tool** (**ADT**) installed; this can be installed directly from Zend Studio.
>
> For a Windows phone application, the Windows Phone SDK needs to be installed.

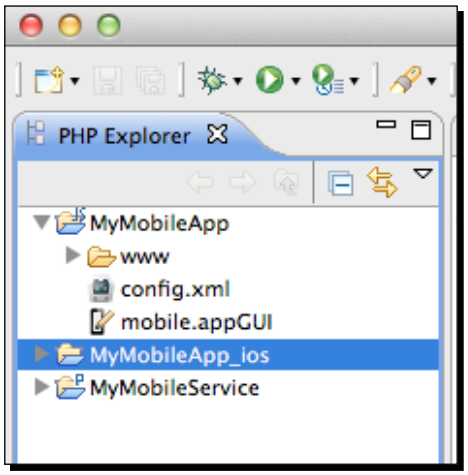
1. Now, from our mobile application project choose **Create iOS Application**:



2. You will be asked to provide the project details; please specify the **Company Name** and **Bundle Id** values. The **Bundle Id** value refers to the unique name that is used to identify the application; this is usually provided in the `com.my-company-name.my-application-name` format. When you register the application with the Apple Store, ensure that the bundle identifier matches with the one provided at Apple.



3. Now the new iOS project is created in the workspace as you can see in the following screenshot:

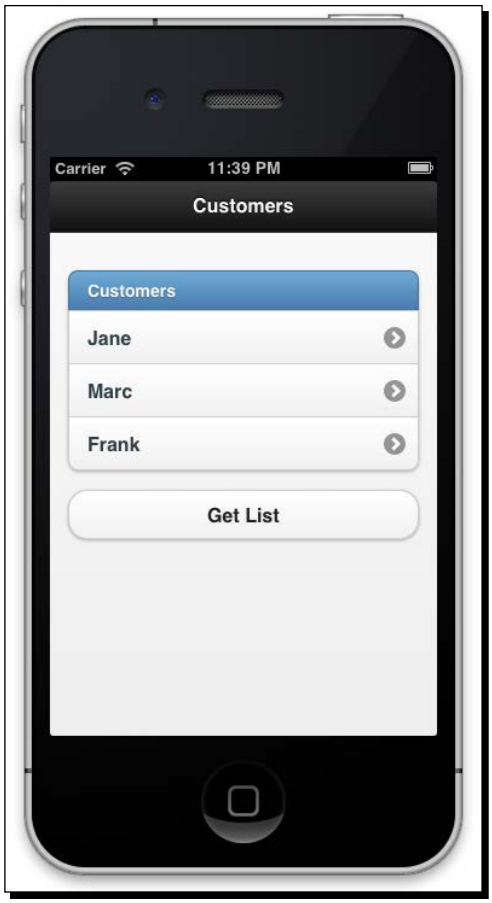


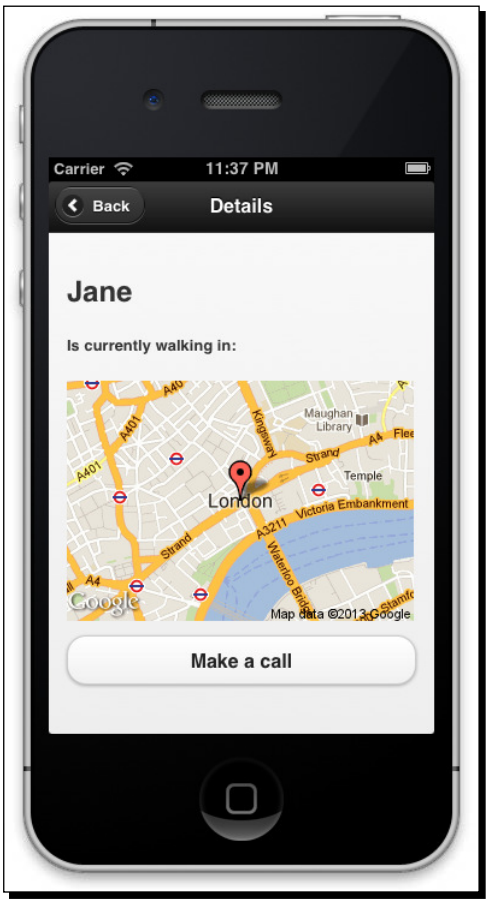
Zend Studio allows for the creation of multiple dependent mobile application projects. If you have to make any changes to the client code, the changes can be made in the parent mobile project and that will automatically update all dependent client projects.

For more information on creating native applications, please refer to the Zend Studio documentation at the following link:

`http://files.zend.com/help/Zend-Studio-10/zend-studio.htm#creating_native_applications.htm`

4. If you run the project, the application will launch the iOS emulator and will launch the mobile application as shown in the following screenshot:





What just happened?

We have created a new native iOS application using Zend Studio support for a native application; in our next section we will be using Zend Framework 2 to provide web services for this application.

Have a go hero

Now that you have created an iOS native application, try creating an Android version of the same application using Zend Studio. For this, you will need to install the Android Development Tool on your Zend Studio installation.

Zend Server Gateway

Zend Server Gateway is a lightweight web services gateway based on Zend Framework 2, which allows for the mapping of web service routes to various controller/actions of the web services. Zend Server Gateway is responsible for authentication, validation, filtering, and routing for RPC and RESTful APIs used in CCM projects.

The routing configurations are mapped into `config/gateway.xml`; the routes and configurations can be managed using the gateway editor interface provided in Zend Studio.

Time for action – creating a mobile search interface

In this task, we will be creating a simple search interface for searching the existing customer records by name using the following steps:

1. We will need to create a search function in the `CustomerRepository` model (`MyMobileService\src\MyCompany\Model\CustomerRepository.php`):

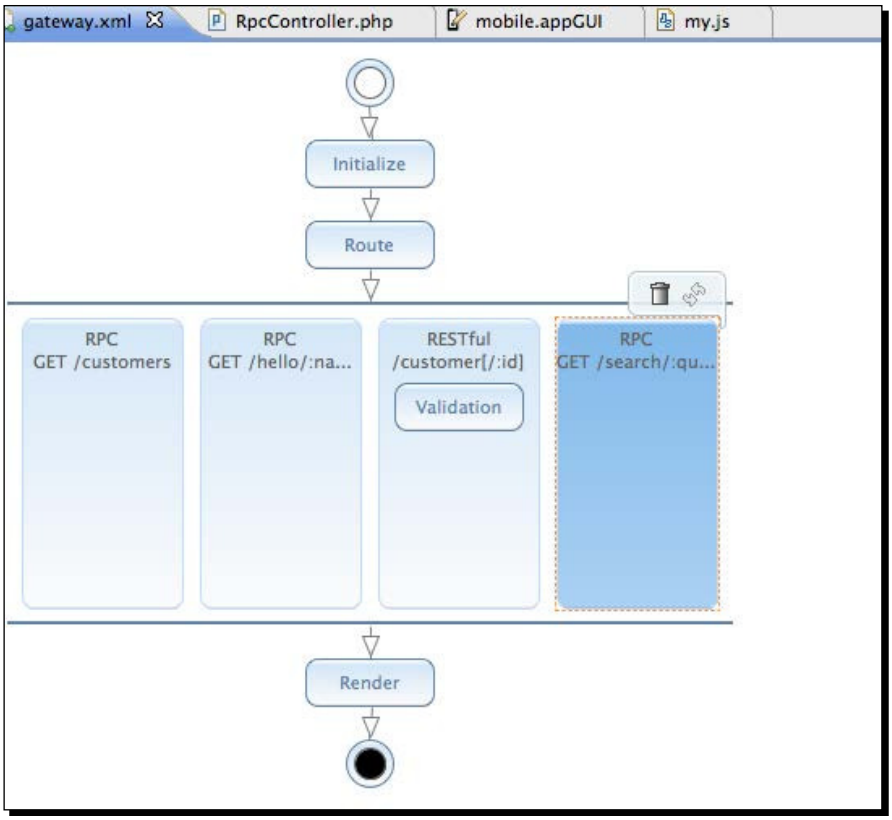
```
public function getSearch($query)
{
  $where = new \Zend\Db\Sql\Where();
  $where->like('name', "%$query%");
  return $this->customerTable->select($where)->toArray();
}
```

2. Add a new action in `RpcController` (`MyMobileService\src\MyCompany\Controller\RpcController.php`); this will handle the web service request:

```
public function getSearchCustomersAction ($query)
{
  $cr = new CustomerRepository();
  return $cr->getSearch($query);
}
```

3. In the gateway editor, create a new RPC service; set the following options:
   - **URL**: `/search`
   - **Method**: `GET`
   - **Request Parameters(Add)**: **Name** – `query`; **Source** – `Route`
   - **Handler Method**: `MyCompany\Controller\RpcController::getSearchCustomersAction`

4. You can test the RPC service by right-clicking on the service and choosing **Test Service**. On the right-hand side you will be presented with an interface to provide test input and validate the service response:



5. In the mobile GUI editor, create a new page `searchCustomers`, and add the following elements:
   - **Text Box**: `custsearchinput`
   - **Button**: `searchbutton`
   - **List View**: `custlistview`

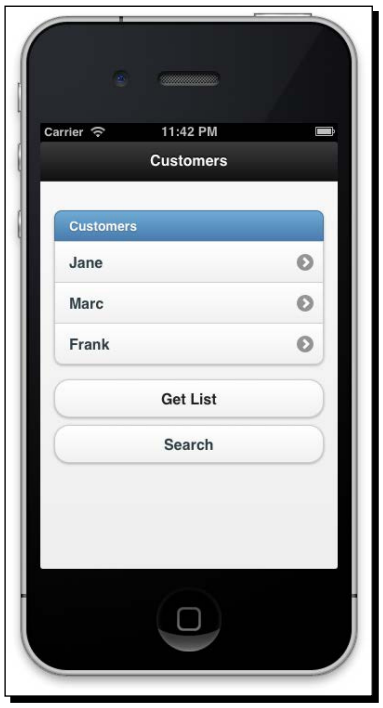
6. In the `binding` section of the `Search` button, bind the button with the `GET /search:query()` web service. Map the `custsearchinput` textbox to the `query` route parameter in the `data` section. This action will bind the search text to the `query` route parameter. Note that the `query` route parameter is already mapped to `getSearchCsutomerAction`.

7. Modify the `onGetSearchquery` JavaScript method in `MyMobileApp/www/js/my.js` to handle the RPC response:

```
function onGetSearchquery(response) {
  // TODO Custom logic to handle server response
  customers = response;

  var newCustomers = '';
  $.each(customers, function(index, item) {
    newCustomers += '<li data-theme="">'
        + '<a href="#page2?empId=' + index
       + '" data-transition="none">' + item.name + '</a>' +
       '</li>';
  });

  $('#custlistview li[role!=heading]').remove();
  $('#custlistview').append(newCustomers).listview('refresh');
}
```

8. Make sure that you link the `Search` page from the `index` page using a button.

9. Now run the project in native mode; you will be able to see the search page, like the one shown in the following screenshot:



What just happened?

We have now created new web services for the existing cloud-connected mobile application and have tested the mobile app in a native emulator. With Zend Studio 10, you can see the simplicity in building mobile apps which are supported by web services running on the cloud.

Pop quiz – building mobile applications

Q1. Which of the following platforms are supported in Zend Studio 10 for native mobile application development?

1. Android
2. Firefox OS
3. MeeGo
4. Brew

Q2. Which of the following web services are not supported by Zend Server Gateway for building cloud-connected mobile applications?

1. RPC
2. SOAP
3. REST

Summary

Cloud-connected mobile applications are a great step by Zend towards enabling PHP developers to build and support mobile apps using the cloud platform. With CCM, Zend is offering an extremely robust, yet simple-to-use platform for building these applications.

Having completed this chapter, you have come to the end of this book. You have covered a lot of ground in various different applications of Zend Framework through this book and have accomplished a number of tasks. This book has shown you the building blocks for developing applications using Zend Framework 2; there is lot more to learn in Zend Framework, most of which is explained in an extremely detailed manner in the Zend Framework documentation (`http://framework.zend.com/manual/2.2/en/index.html`).

Thanks for reading. Feel free to give your feedback on how you felt reading this book.

Pop Quiz Answers

Chapter 1, Getting Started with Zend Framework 2.0

Pop quiz – Zend Framework 2.0

Q1	3
Q2	4

Chapter 2, Building Your First Zend Framework Application

Pop quiz – Zend Framework 2.0

Q1	2
Q2	4

Chapter 3, Creating a Communication Application

Pop quiz – Zend Framework 2.0

Q1	2
Q2	1

Chapter 4, Data Management and Document Sharing

Pop quiz – data management and document sharing

Q1	4
Q2	3

Chapter 5, Chat and E-mail

Pop quiz – chat and e-mail

Q1	1 and 2
Q2	2 and 4

Chapter 6, Media Sharing

Pop quiz – media sharing

Q1	4
Q2	4

Chapter 7, Search Using Lucene

Pop quiz – search

Q1	1
Q2	4

Chapter 8, Creating a Simple Store

Pop quiz – creating a simple store

Q1	2
Q2	1

Chapter 9, HTML5 Support

Pop quiz – HTML5 Support

Q1	4
Q2	1 and 4

Chapter 10, Building Mobile Applications

Pop quiz – building mobile applications

Q1	1
Q2	2

Index

Symbols

$features parameter 54
$form->setInputFilter() method 50
$ sudo service zend-server restart command 10
$table parameter 54
$this->add() method 47
.htaccess 26
<input> element
  URL 160
<link> tag 85
<script> tag 86

A

a2ensite comm-appp.local command 24
abstract_factories, ServiceManager 59
Adapter $adapter parameter 54
adaptiveResize ($width, $height) function 108
addBcc() method 96
addCc() method 96
addDocument(Document $document) method 126
addProductAction() action 144
Add Share 79
addSharing() function 77
Add Target icon 180
Administration Interface, Zend Server CE 11
admin UI
  implementing, for managing users 65-70
album view 116
aliases, ServiceManager 59
Android Development Tool (ADT) 187
appendFile function 89
appendStylesheet function 89
attach() 99
attributes, HTML5
  list attribute 171
  max attribute 171
  min attribute 171
  multiple attribute 172
  pattern attribute 171
  placeholder attribute 171
  required attribute 172
  step attribute 171
autoload_classmap.php file 33
Autoloader configuration 34

B

basePath() 89
BasePath helper 85
Bundle Id value 187

C

CCM application
  about 177, 178
  building 182-186
  phpCloud 178
  Zend Studio 10 178
CCM Tool 177
cloud-connected mobile (CCM) 177, 178
code
  migrating, to ServiceManager 61-63
color element 165

commit() method 126
Communication Application page 101
Company Name value 187
composer
about 21, 106
installing 22
composer.json file 106
concrete placeholder helpers
HeadLink helper 85
HeadMeta helper 86
HeadScript helper 86
HeadStyle helper 87
HeadTitle helper 87
implementing 85
jQuery UI, using in simple page 88-90
config/application.config.php file 36
config component 31
config file 148
config/module.config.php file 33
ConfirmAction function 57
confirm_password field 41
Confirm Password field 42, 48
controller layer 30
controllers
about 34
creating 31-33
Create operation. *See* **CRUD**
create module command 30
create operation 66, 67
crop ($startX, $startY, $cropWidth, $cropHeight) function 108
CRUD 63
CurrentTime module 38
setting up 38
custsearchinput textbox 191

D

Data API
deleteAlbumEntry() function 114
deleteCommentEntry() function 114
deletePhotoEntry() function 114
deleteTagEntry() function 114
functions 114
getAlbumFeed() function 114
getCommentEntry() function 114
getPhotoFeed() function 114
getTagEntry() function 114
getUserFeed() function 114
insertAlbumEntry() function 114
insertCommentEntry() function 114
insertPhotoEntry() function 114
insertTagEntry() function 114
database
creating 16-18
creating, in MySQL Server 15
database operation
CRUD 63
data/images directory 175
data section 191
date element 162
Date field 171
datetime element 161
datetime-local element 161
Delete operation. *See* **CRUD**
delete($id) method 126
delete($where) method 64
deleteAction() action 67
deleteAlbumEntry() function 114
deleteCommentEntry() function 114
Delete option 75
deletePhotoEntry() function 114
deleteProductAction() action 144
deleteTagEntry() function 114
deleteUpload() method 72
deleteUser($id) function 65
Delete user link 69
Dependency Injection (DI) 7
Dispatch event 102
Document class 126
Document component 124
document files
indexing 134-137
document management
file upload form, creating 71-75
DoDirectPayment 157

E

editAction() action 66
Edit user link 69
Email Address field 46, 47
EmailAddress field 40
email element 164

email field 42, 58
e-mail form
creating 97, 99
etExpressCheckout 149
Event 99
Event manager object 99
exchangeArray() method 50, 52
external modules 105

F

factories, ServiceManager 60
fetchAll() function 65
fetchAll() method 72
Field class 125
Field component 124
file download
implementing 80
fileDownloadAction() function 79
File element 172
file_get_contents() method 80
file sharing
implementing 76-81
managing 76
File transfer adapter 71
File upload form element 71
Filters
with multiple file uploads 176
find() method 129
Finish button 181
form
validating 46
formAction() 165
FormElement 45
formElement() view helper 167
form object 42
Form object 45
Form View Helpers
URL 45
framework
documentation, URL 173
Full mode 110
functions, HeadLink helper 85
functions, HeadMeta helper 86
functions, HeadScript helper 86
functions, HeadStyle helper 87

G

gatewayURL variable 185
generateThumbnail() method 109
getAlbumFeed() 115
getAlbumFeed() function 114
getAuthService() function 63
getAutoloaderConfig() method 33
getCommentEntry() function 114
get('config') method 73
getConfig() method 33
GetExpressCheckoutDetails 149
getGooglePhotos() method 115
getLastInsertValue() method 65
Get List button 185
getPhotoFeed() function 114
getPlaylistListFeed() method 119
getSearchCsutomerAction 191
getServiceConfig() method 60
getSharedUploadsForUserId() function 77
getSharedUsers() function 77
getSharedUsers()method 78
getSubscriptionFeed() method 119
getTable() method 64
getTagEntry() function 114
getTopRatedVideoFeed() method 119
getUpload() method 72
getUser($id) function 65
getUserByEmail($userEmail) function 65
getUserFavorites() method 119
getUserFeed() 115
getUserFeed() function 114
getUserUploads() method 119
getVideoCommentFeed() method 119
getVideoFeed() method 119
getVideoResponseFeed() method 119
getYoutubeVideos() method 119
Git
about 11, 21
installation, URL 11
installing 22
URL 11, 21
used, for Zend Framework 2.0 11
Google Data API 113
Google Photos
photos, fetching 115-118

Google Photos API 114
Google services
Google Analytics 113
Google Blogger 113
Google Calendar 113
Google CodeSearch 113
Google Documents 113
Google Notebook 113
Google Provisioning 113
Google Spreadsheets 113
Picasa Web Albums 113
YouTube 113
group chat
building 90
group chat application
creating 90-95

H

HeadLink helper
about 85
functions 85
HeadMeta helper
about 86
functions 86
HeadScript helper
about 86
functions 86
headScript() view helper 89, 90
HeadStyle helper
about 87
functions 87
headTitle() helper 87
HeadTitle helper 87
HTML5
about 159
attributes 171
browser, compatibility 171
browser compatibility, URL 171
elements 160
feature, URL 171
input elements 160-167
multiple file uploads 172-175
offerings 159
URL 160
view helpers 167-171

I

iframe tag 95
images
resizing, modules used 106-108
ImageUpload entity 109
index 124
index action 34
indexAction() action 66, 141, 144
indexAction function 94
Index class
about 126
addDocument(Document $document) method 126
commit() method 126
delete($id) method 126
optimize() method 126
IndexController file 31
indexing process
about 125
Document class 126
Field class 125
Index class 126
Lucene index, generating 127-129
ZendSearch\Lucene, using 125
index, searching
ZendSearch\Lucene, using 129-133
index view 116
input elements, HTML5
color element 165
date element 162
datetime element 161
datetime-local element 161
email element 164
month element 163
number element 164
range element 164
time element 162
url element 164
week element 163
InputFilter class
about 46-48, 50
validation, adding to registration form 47-50
insert($set) method 64
insertAlbumEntry() function 114
insertCommentEntry() function 114

insertEntry() method 119
insertPhotoEntry() function 114
insertTagEntry() function 114
insert value 65
invokables array 68
invokables, ServiceManager 60
isValid() method 50

J

jQuery UI
URL 90
using, in simple page 88-90
JSON helper 85

L

label field 132
label index field 127
layouts 83
list attribute 171
Listener 99
listOrdersAction() action 144
localhost value 18
Login button 89
Lucene
about 123
components 124
Document 124
Field 124
index component 124
overview diagram 124
used, for searching 123
ZendSearch\Lucene, installing 124, 125
Lucene index
generating 127-129
label field 127
owner field 127
upload_id field 127

M

mails
sending 95
mails, sending
Zend\Mail\Message 96
Zend\Mail\Transport 96
Zend\Mime\Message 96
Zend\Mime\Part 96
Mail transport 96
Manage Documents 78
max attribute 171
media
sharing 105
MediaManagerController file 110, 115, 119
MediaManagerController file method 109
messageList action 92
meta tags 86
Microsoft Office documents
document files, indexing 134-137
indexing 133, 135, 136
min attribute 171
mobile search interface
creating 190-193
mobile web application
versus native application 186-189
model layer 30
models
creating 51-55
modify operation 65
mod_rewrite 26
module
creating 29
creating, ZFTool used 30
used, for resizing images 106-108
module configuration
about 34
controllers 34
modifying 34-38
routes 34
views 34
Module.php file 29, 33, 62
month element 163
multiple attribute 172
MVC layer
about 30, 31
controller layer 30
model layer 30
view layer 30
MySQL
about 14
database, creating 16-19
installing 15
phpMyAdmin 16
URL 14

MySQL Server
database, creating 15

N

Name field 42, 47
namespaces
about 63
using 63
native application
testing as 187-189
versus mobile web application 186-189
number element 164
Number field 171

O

onBootStrap() method 101
onGetSearchquery method 192
optimize() method 126
owner field 132
owner index field 127

P

password field 41, 42, 58
pattern attribute 171
paymentCancelAction() action 141
paymentCancelAction() method 153
paymentConfirmAction() action 141
paymentConfirmAction() method 152
payments
accepting, PayPal used 150-157
PayPal
about 146
and Zend Framework 2.0 146
setting up 147, 148
URL 147, 153
used, for accepting payments 150-157
used, for payments 146
PayPal Express Checkout
developer documentation, URL 150
PayPal, used for accepting payments 150-157
URL 150
workflow 149-157
paypalExpressCheckoutAction() function 150, 151
peckpaypal
URL 147
PhoneGap
about 182
CCM application, building 182-186
photo gallery
application 108
implementing 109-113
phpCloud
about 178
account, configuring 179-181
PhoneGap 182
URL 178
Zend Studio 182
phpCloud account
configuring 178
registering, URL 178
PHP Command line
installing 22
phpMyAdmin
about 16
installation, URL 16
URL 16
placeholder attribute 171
processAction() action 67
processAction method 48
processAction() method 54, 56
productDetailAction() action 141

Q

query route parameter 191

R

range element 164
Range field 171
Read operation. *See* **CRUD**
receive() method 71
Refresh button 95
Register button 89
RegisterController class 43, 48, 53
RegisterFilter class 47
RegisterForm class 40, 45, 50
registration form
configuration 44
controller 43

creating 40-45
display, URL 44
form 40, 41
validation, adding 47-50
views 41
views, confirmation page 43
views, registration page 41
removeSharing() function 77
RenameUpload filter 173
required attribute 172
required field 46
resize ($maxWidth = 0, $maxHeight = 0) function 108
resize method 109
ResultSet $resultSetPrototype parameter 54
Review option 149
rotate function 113
rotateImage ($direction = 'CW') function 108
rotateImageNDegrees ($degrees) function 108
routes 34

S

save ($fileName, $format = null) function 108
saveUpload() method 72
Search button 191
search results
displaying 130-133
select($where = null) method 64
Sendmail transport 97
sendMessage() method 93, 94
send() method 96
ServiceManager
configuration 59, 60
existing code, migrating 61-63
key factories 61-63
ServiceManager, configuration type
abstract_factories 59
aliases 59
factories 60
invokables 60
services 60
shared 60
services, ServiceManager 60
setBody() method 96
setDestination() method 71
SetExpressCheckout 149
setFrom() method 96
setHeaders method 96
setParts() method 96
setPassword() method 52
setSubject() method 96
setTerminal() 80
setTo() method 96
shared, ServiceManager 60
shopping cart
about 140
checkout process 140
store front, creating 140-143
shoppingCartAction() action 141
Shopping Cart page 154
showImageAction() 111
SMTP transport
URL 97
speckpaypal repository 147
Sql $sql parameter 54
src component 31
src/Users/Model/ImageUpload.php file 109
src/Users/Model/ImageUploadTable.php file 109
step attribute 171
store
creating 139
StoreAdminController
about 144
addProductAction() action 144
deleteProductAction() action 144
indexAction() action 144
listOrdersAction() action 144
updateOrderStatusAction() action 144
viewOrderAction() action 144
store administration user interface
creating 144-146
key aspects 143
StoreController
about 141
indexAction() action 141
paymentCancelAction() action 141
paymentConfirmAction() action 141
productDetailAction() action 141
shoppingCartAction() action 141
store front
creating 140-143
store_orders table 140

store_products table 140
subaction parameter 110
Submit button 45
submit field 41
Submit field 42

T

TabelGateway object 76
TableGateway 50
admin UI, implementing for user management 65
form, saving 51-55
model, creating 51-55
URL 54
TableGateway class
about 64
admin UI, implementing for user management 66-70
delete($where) method 64
getLastInsertValue() method 65
getTable() method 64
insert($set) method 64
select($where = null) method 64
update($set, $where = null) method 64
TableGateway constructor
$features parameter 54
$table parameter 54
about 54
Adapter $adapter parameter 54
ResultSet $resultSetPrototype parameter 54
Sql $sql parameter 54
TableGateway object 50, 109
thumbnail filename 108
Thumbnail mode 110
time element 162
trigger() 99

U

ui-button class 88
Update operation. *See* **CRUD**
update($set, $where = null) method 64
updateOrderStatusAction() action 144
Upload entity 108
upload_id index field 127
UploadManager controller 74
UploadManagerController 72
Upload now 175
upload_sharing 76
Upload Sharing page 79
UploadTable class
about 76, 77, 81
addSharing() function 77
getSharedUploadsForUserId() function 77
getSharedUsers() function 77
removeSharing() function 77
UploadTable factory 76
UploadTable object 78
url element 164
URL helper 84
use keyword 63
user
authenticating 56, 58
User class 52
UserManagerController
about 66
deleteAction() action 67
editAction() action 66
indexAction() action 66
processAction() action 67
users
create operation 66, 67
managing, admin UI implemented 65-69
modify operation 65
UserManagerController 66
Users module 29, 34
UserTable entity 52

V

validation
adding, to registration form 47-50
view component 31
view helper
about 84
BasePath helper 85
JSON helper 85
URL helper 84
view helpers, HTML5
for input elements 168
View Image link 112
View Image page 113
view layer 30

viewOrderAction() action 144
views
about 34, 83
errror messages 41
view helpers 41
view logic 41
View Source link 89

W

WebinoImageThumb
about 105
adaptiveResize ($width, $height) function 108
crop ($startX, $startY, $cropWidth, $cropHeight) function 108
resize ($maxWidth = 0, $maxHeight = 0) function 108
rotateImage ($direction = 'CW') function 108
rotateImageNDegrees ($degrees) function 108
save ($fileName, $format = null) function 108
week element 163
where condition 64

Y

YouTube Data API
about 119
getPlaylistListFeed() method 119
getSubscriptionFeed() method 119
getTopRatedVideoFeed() method 119
getUserFavorites() method 119
getUserUploads() method 119
getVideoCommentFeed() method 119
getVideoFeed() method 119
getVideoResponseFeed() method 119
insertEntry() method 119
videos, listing for keyword 119, 121

Z

Zend\Authentication
about 55
user authentication 56-58
Zend\EventManager
Event 99
event flow 100
Event manager 99
Listener 99
module layout setting, ZF events used 100, 102
Zend\Form
about 39
registration form, creating 40-45
Zend Framework
components 71
Zend Framework 2.0
about 7, 8, 27
and PayPal 146
Git, using 11
Zend Framework 2.0 module
about 27
advantages 27
config component 31
configuring 33
creating 29
src component 31
view component 31
Zend Framework 2.0 project
folder layout 28
prerequisites 21
Zend Framework 2 module
URL 106
Zend Framework 2 ServiceManager. *See* **ZF2 ServiceManager**
Zend Framework, components
File transfer adapter 71
File upload form element 71
Zend Framework keyword 119
Zend Framework project
creating 22-26
URL 27
ZendGdata library 114
Zend GData package
installing, URL 108
Zend\Http\Response\Stream() 80
Zend\InputFilter 46
Zend\Mail 96
Zend\Mail\Message 96
Zend\Mail object 99
Zend\Mail\Transport 96
Zend\Mime\Message 96
Zend\Mime\Part 96
ZendSearch\Lucene
about 124
installing 124, 125
used, for indexing 125

used, for searching index 129-133
Zend Server CE
about 8
configuring 11-14
installing 8-10
system requisites 8
system requisites, URL 8
URL 8
Zend Server CE, configuring
Administration Interface 11
Zend Server Community Edition. *See* **Zend Server CE**
Zend Server Gateway
about 190
mobile search interface, creating 190-193
Zend_Service package 105
ZendSkeletonApplication
about 22-28
URL 22
ZendSkeletonModule 29, 34, 38
Zend Studio
about 182
CCM application, building 182-186
documentation, URL 188
URL 182
Zend Studio 10
about 178, 183
URL 178
ZF2 ServiceManager 59
ZF events
used, for setting module layout 100-102
zf_pass value 18
ZFTool
about 30
URL 30
used, for creating module 30
zf_user value 18

[PACKT] PUBLISHING open source community experience distilled

Thank you for buying
Zend Framework 2.0 by Example Beginner's Guide

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.