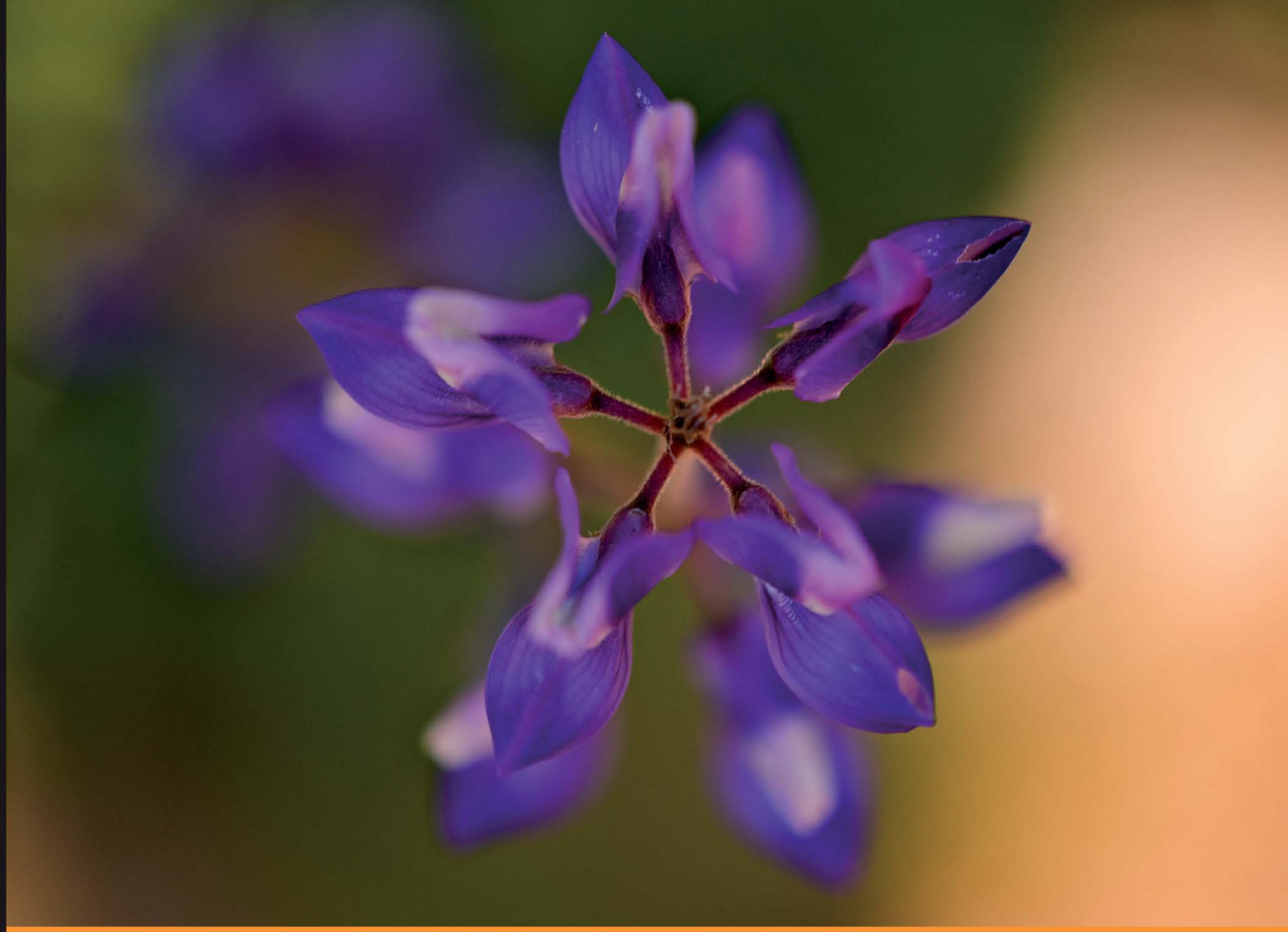


Community Experience Distilled

PHP 5 Social Networking

Create a powerful and dynamic social networking website in PHP by building a flexible framework

Michael Peacock

[PACKT] PUBLISHING open source* community experience distilled

PHP 5 Social Networking

Create a powerful and dynamic social networking website in PHP by building a flexible framework

Michael Peacock

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

PHP 5 Social Networking

First published: October 2010

Production Reference: 1181010

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-849512-38-1

www.packtpub.com

Cover Image by John M. Quick (john.m.quick@gmail.com)

Credits

Author
Michael Peacock

Reviewers
Jason Mayes
Sérgio Serra
Deepak Vohra

Acquisition Editor
Sarah Cullington

Development Editor
Wilson D'souza

Technical Editors
Chris Rodrigues
Ajay Shanker

Indexers
Hemangini Bari
Tejal Daruwale
Rekha Nair

Editorial Team Leader
Aanchal Kumar

Project Team Leader
Priya Mukherji

Project Coordinator
Srimoyee Ghoshal

Proofreader
Aaron Nash

Graphics
Nilesh R. Mohite

Production Coordinator
Melwyn D'sa

Cover Work
Melwyn D'sa

About the Author

Michael Peacock (`http://www.michaelpeacock.co.uk`) is a web developer and Zend Certified Engineer from Newcastle, UK with a degree in Software Engineering from the University of Durham. After meeting his business partner while studying at Durham, he co-founded Peacock Carter Limited (`http://www.peacockcarter.co.uk`), a creative agency based in Newcastle, where he helps run the business and manages the development team. Michael presented some of his thoughts on one particular web application architecture at the PHPNW 2010 conference.

Michael loves working on web-related projects and new business ideas and has interests in several companies. At the moment he is working on his latest venture, Central Apps, and its flagship product Invoice Central (`http://www.invoicecentral.co.uk/`). He also takes part in amateur dramatics in his spare time, volunteering through Juniper Productions (`http://www.juniperproductions.org.uk`) in Newcastle.

He has been involved with a number of books, having written five books: *PHP 5 Social Networking*, *PHP 5 E-Commerce Development*, *Drupal 6 Social Networking*, *Selling online with Drupal e-Commerce*, *Building websites with TYPO3*, and acted as technical reviewer for two others, *Mobile Web Development* and *Drupal for Education & E-Learning*.

You can follow Michael on Twitter: `www.twitter.com/michaelpeacock`.

Acknowledgement

I'd like to thank everybody at Packt Publishing, in particular: Douglas Paterson and Sarah Cullington for working with me on building the idea of this book into a suitable structure, Srimoyee Ghoshal for helping to keep the book on track, and Wilson D'souza, the development editor, and of course the technical reviewers, Jason Mayes, Sérgio Serra, and Deepak Vohra who helped improve the quality of the book.

My thanks also go to my friends and family, in particular my fiancée Emma for her support while working on the book.

Finally, I'd like to thank you, the reader; I hope that you enjoy this book and produce a fantastic social network of your own. I look forward to hearing your feedback and seeing what you come up with!

About the Reviewers

Jason Mayes is a Web Developer, Programmer, Technical Consultant, and Strategist, with a creative twist based in the UK.

With a background in Computer Science, it was here that Jason discovered he fitted in to a rare breed of what he likes to call "hybrid developers"—those who equally enjoy being both creative and technical. Combining these two qualities he produces bespoke, usable, and well-implemented digital solutions in a number of areas.

Jason holds a first class MEng degree in Computer Science from The University of Bristol, and is a member of the BCS (British Computing Society). His final year thesis "Reality mining using mobile devices and pseudonymous social networks" was novel in its implementation, and Jason went on to be shortlisted to the final three candidates in the UK for the "Best IT Student" category in the national SET Awards, which was judged by the IET. The SET awards are established as Britain's most important awards for science and engineering undergraduates.

At the time of writing, Jason is Director of Pure42.com—his own company specializing in areas such as web development and design, digital marketing, usability, user experience, graphic design, digital advertising, social media, and technical consultancy.

Jason is also a Senior Web Development Engineer at a global semiconductor company looking after their online developments, implementations, and digital strategy. During his time there he has helped to build the company's successful online presence as it stands today. He has also worked with world leading companies such as Akamai (see `http://bit.ly/d7utAT`) in his quest for optimal solutions, and has been featured in a Computer World article related to "how to improve your website's uptime" (see `http://bit.ly/a3dnPs`).

When not pursuing a new technology or idea, Jason loves taking flying lessons, travelling, or practicing his DSLR photography skills, which he uploads to Flickr.

You can follow Jason on the following sites:

Website: `http://www.jasonmayes.com/`

Twitter: `http://twitter.com/jason_mayes`

I would like to thank all of the staff and lecturers at the University of Bristol Computer Science Department, colleagues, friends, and family who have inspired and stuck with me over the years and contributed to making me the person who I am today.

Sérgio Serra is a software engineer and an expert in business-related applications, especially ERPs.

He started working in software in 1999, first as a developer and later as a systems analyst. Over the years his work has been focused on development and deployment of large industrial applications like ERPs and production planning software. In 2004 he started developing web applications, mainly with PHP and JavaScript. He aims to someday build his own web ERP and put it into the market. In 2010 he, along with a colleague from his Computer Science Graduation, founded their own web company named Sysactum. In the same year they launched a web application for veterinary, which they have called Actumvet.

Deepak Vohra is a consultant and a principal member of the NuBean.com software company. He is a Sun Certified Java Programmer and Web Component Developer, and has worked in the fields of XML, Java programming, and J2EE for over five years. Deepak is the co-author of the Apress book *Pro XML Development with Java Technology* and was the technical reviewer for the O'Reilly book *WebLogic: The Definitive Guide*. He was also the technical reviewer for the Course Technology PTR book *Ruby Programming for the Absolute Beginner* and the technical editor for the Manning Publications book *Prototype and Scriptaculous in Action*. He is also the author of the Packt Publishing books *JDBC 4.0 and Oracle JDeveloper for J2EE Development* and *Processing XML Documents with Oracle JDeveloper 11g*.

Table of Contents

Preface

Social networking has quickly become a very popular activity on the Internet, particularly with sites such as Facebook and MySpace. When it comes to creating social networks there are many options to chose from, including off-the-shelf systems, making use of existing social networks (for example, building a Facebook application or creating a Facebook page), or building something yourself. While it may be easy to find existing solutions, the only way to have one looking and behaving exactly as you want is to build it yourself.

By initially developing a light-weight Model-View-Controller-style framework with PHP, which can easily be extended to give us a stable and solid platform to work with making common tasks easier and giving us a structure for our social networking code, we can rapidly develop a custom, powerful social networking website.

Within the first few chapters, you will have a suite of files that deal with template management, database management, user authentication management, and e-mail sending. Once this is in place, social networking-centric features can be rapidly developed and plugged into the framework, including user registration and dealing with forgotten details, user profiles, building connections with users, sending messages, sharing information, forming groups, a Developer API, and events and birthday calendars.

At the end of this book, you will have a powerful social networking platform that can take the user all the way from the signup process to forming relationships and creating groups of users. The platform is developed in a very flexible way, so the needs of any social networking site can be met, with new features easily and quickly added in as the needs of the site change.

This book doesn't just stop with how to develop a social networking platform; there are many other topics, which any developer should consider such as marketing, search engine optimization, backing up and restoring the site, and how to deal with scaling problems when the site gets popular. All of these topics are discussed too, leaving you not only with a solid social network, but with hints, tips, and advice on how to maintain it in the long term and deal with any challenges on the way.

What this book covers

Chapter 1, *PHP Social Networking*, looks into the growing popularity of social networking, including popular social networks, different ways to create or utilize social networks, and discusses what we will be creating throughout the course of the book.

Chapter 2, *Planning and Developing the Core Framework*, discusses several architectural and design patterns, including Model-View-Controller, Registry and Factory, the planning and subsequent development of our skeleton MVC-style framework with template, database, and e-mail management.

Chapter 3, *Users, Registration, and Authentication*, extends our development framework with user authentication classes, and then walks through development of registration and login features for users, as well as reminders for forgotten details.

Chapter 4, *Friends and Relationships*, looks at allowing users to connect with one another, either by adding them as friends or establishing custom relationships with one another such as a co-worker or family member.

Chapter 5, *Profiles and Statuses*, walks through the development of profiles for our users as well as a flexible status system so users can update their friends and contacts with what they are doing.

Chapter 6, *Status Stream*, discusses how to collate user statuses and activities to show a useful stream of status updates for a user's particular network, as well as for administrators to see how the network is growing.

Chapter 7, *Public and Private Messages*, enables users to communicate with one another by implementing a simple message system.

Chapter 8, *Statuses – Other Media*, allows users to share media such as images and videos with other users in their network as status updates and profile posts.

Chapter 9, *Events and Birthdays*, integrates a calendar to manage and display events created by our users and birthday notifications.

Chapter 10, *Groups*, allows users to create and maintain groups related to specific topics with their own lists of members, who opt in to be part of the group.

Chapter 11, *Developing an API*, discusses the development of an API to allow third-party websites and developers to interact with the social network, so that it can gain popularity through other applications too.

Chapter 12, *Deployment, Security, and Maintenance*, looks at steps to make the framework more secure and protect it from spam, as well as looking at how to back up the site and restoring it from a backup.

Chapter 13, *Marketing, SEO, User Retention, and Monetization Strategies*, advises on how to market and promote the social network, and gives useful tips to help develop search engine-friendly websites.

Chapter 14, *Planning for Growth*, goes through a number of potential issues that will occur when the social network becomes more popular, and advises on scalability, deployment and hosting options, caching, and content delivery networks.

What you need for this book

During the course of this book, you will need the following software to try out the various code examples:

- Apache 1.3 or above (2 recommended)
- mod_rewrite module for Apache
- MySQL 5.0 or above
- PHP 5.0 or above (5.2 or above recommended)

When working locally on your own computer, a package such as WampServer 2 for Windows is recommended, as this will install PHP, Apache, and MySQL in one, and make enabling extensions easy.

A text editor is all that is required for editing the code. However, one with syntax highlighting would be beneficial (such as Crimson Editor or Notepad++).

For deployment, an FTP application such as FileZilla will be required, and an SSH client such as PuTTY for some of the backup and restoration options would be useful.

Who this book is for

This book is primarily aimed at PHP developers, but is suitable for any web developer looking to expand their knowledge and understanding of social networking concepts. Intermediate knowledge of PHP and object-oriented programming is assumed, along with a basic knowledge of MySQL.

Conventions

In this book, you will 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 `delegateControl` method checks that the delegate controller is within the allowed delegates."

A block of code is set as follows:

```
/**
 * Is the profile valid
 * @return bool
 */
 public function isValid()
 {
   return $this->valid;
 }
```

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

```
<form action="relationship/create/{ID}" method="post">
<select name="relationship_type">
<!-- START relationship_types -->
<option value="{type_id}">{type_name}</option>
<!-- END relationship_types -->
</select>
<input type="submit" name="create" value="Connect with {name}" />
</form>
```

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: "Now if we click on the **Connect with** button on the relationship form, our relationship is created and we are shown a confirmation message".

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 via the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on `www.packtpub.com` or e-mail `suggest@packtpub.com`.

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 for this book

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/support`, 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 on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

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
PHP Social Networking

Welcome to PHP social networking! During the course of this book, we are going to build a flexible social networking site and framework using PHP, which we can easily extend to meet the needs of our social network.

In this chapter, you will learn:

- More about social networks
- About existing social networks
- Existing social networking software
- Why and when to roll your own system

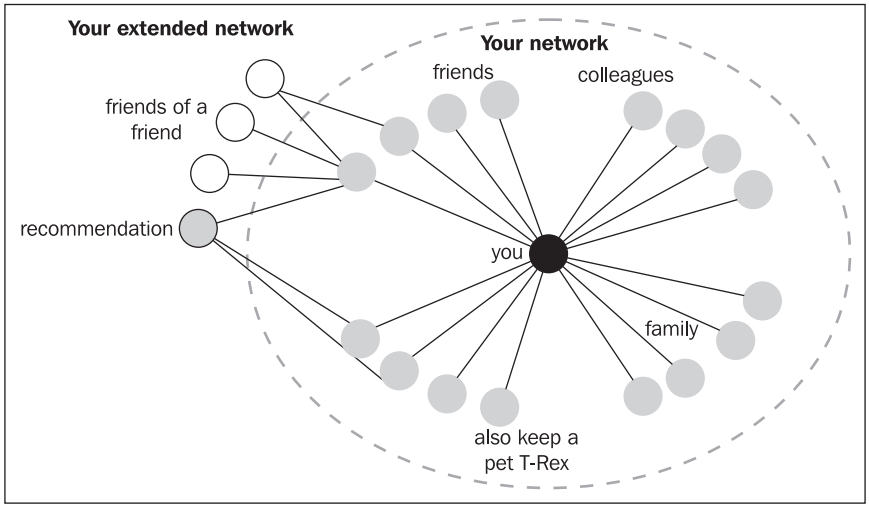
We will also discuss the social networking website that we will create during the course of this book: DinoSpace—a social network for keepers of pet dinosaurs.

Introduction to social networks

Social networks are now one of the most widely used aspects of the Web and have really taken off over the past few years. Many businesses, organizations, communities, and families are using social networking to promote themselves, to communicate better with others, and to engage with their audience.

Social networking relies upon users building up their own *network* of contacts on the site. This, in turn, introduces them to new contacts and—on many social networking websites—allows them to be found more easily. Also, this allows new contacts to be recommended or introduced, helping to grow the user's network.

Let's look at an example of how a user's network of contacts can be built up:



This social network representation shows the connections between contacts. It also illustrates how a user may be able to discover friends of a friend and friend recommendations (based on friends in common). This makes it easy for the users to build up their social network, to communicate with new people, or reconnect with lost contacts.

Social networks generally serve two primary functions. Firstly, they allow users to connect with each other and build a contact network, as we have just discussed. They provide a community with collaboration and contribution features as well. This allows the content and information within the social network to be grown by the users themselves. Later in this chapter, we will discuss some of the features available in existing social networks and social networking software, to build up a list of key features we will need to include as well as things we might like to include.

Business logic to social networks

There is some very powerful business logic to using both existing and custom social networks. Creating your own social network or social network tools gives a dedicated customer area, where feedback on products and services can be obtained, for instance, use of support forums to discuss and resolve problems. Areas that allow customers to share tips, resources, and product care tips help promote those products and services.

Examples: Businesses making use of existing social networks and their own social networks

There are some examples of businesses making great use of existing social networks and their own social networking type websites to improve their businesses. Let's have a look at three specific examples.

NameCheap: Twitter

NameCheap is a domain name registrar, and they use Twitter (`http://twitter.com/namecheap`) for two purposes. Firstly, they collect and respond to feedback from customers mentioning their company, and more prominently, they run various competitions giving away free domain names. These viral competitions encourage more users to follow them, and promote the competition, therefore increasing their brand awareness.

Dell: Twitter

Recently, Dell announced that their Twitter presence (http://twitter.com`/delloutlet`) generated $6.5 million in revenue, with orders being placed as a result of the links or discounts placed on their Twitter feed. More information is available on the Mashable website: `http://mashable.com/2009/12/08/dell-twitter-sales/`.

BT: Twitter

British Telecom uses Twitter (`http://twitter.com/btcare`) to help improve customer service and manage their reputation. In the most instances I've seen this used, it has primarily been in response to customer complaints, to try and assist them with their problems, and escalate matters such as fault testing and engineer call out. This makes them seem more caring (also emphasized by their choice of Twitter username), increases customer satisfaction by resolving problems more quickly.

Netgear: custom

While not strictly a social network, Netgear have various social aspects to their website, both through a dedicated community area (`http://www.netgear.com/community/`) and the support section of their website (`http://kb.netgear.com/app/`). The support section integrates community generated content from their discussion forums and brings this into product pages, making it easier for customers to find answers to questions staff have not answered directly. Discussion forum software is also quickly becoming social networking software to an extent, in its own right.

Existing social networks

There are many existing social networks available, some of which are already very popular and have some excellent features. Let's take a look at the most prominent features of some of these more popular sites.

Facebook

Facebook (`www.facebook.com`) is very much a global social networking website for everyone over the age of 13. It started out for students at Harvard University, branching out to all the universities, and now available for everyone. Features available include:

- A customizable profile
- Users can update their statuses
- Users can connect with other users by adding them as "friends"
- Statuses of friends can be commented upon and users can indicate that they *like* a particular status
- Friends can post messages to each other's profiles
- Photos can be posted and shared
- Events can be posted and shared, with attendees sending their RSVPs online
- Groups can be created and joined, promoting specific activities or interests
- Topics can be discussed
- Third-party developers can create their own applications for Facebook, to add more to the platform

LinkedIn

LinkedIn (`www.linkedin.com`) is a social networking site for business contacts, colleagues, and classmates, which primarily encourages business contacts to connect. Features available on LinkedIn include allowing the users to:

- Customize their profile
- Connect with colleagues
- See how users are connected to other
- Recommend other users with respect to a job
- Integrate Twitter with their account profiles
- Create and view business profiles
- Third-party developers can create their own applications too (`http://developer.linkedin.com/index.jspa`)

MySpace

MySpace (`www.myspace.com`) is a social networking website used primarily by a younger audience. It is very popular with bands, particularly because of how much profiles can be customized with HTML and how music can be embedded within profiles. Features available include:

- Customizable profiles, complete with:
  - HTML customization: allowing users to customize the colors, look, and feel of MySpace
  - Music integration
  - The user's current mood
  - Comments
- Groups: small subsets of users
- MySpace TV: video sharing
- Integration and development of third-party applications via an (a suite of) API(s). We will discuss these further in *Chapter 11*, *Developing an API*.
- Forums: for discussions.
- Polls: to get user opinion.

Twitter

Twitter (`www.twitter.com`) is a micro-blogging social networking website, which primarily deals with very short messages of 140 characters or less. Despite this, it has a large number of prominent features, including:

- Profiles can be customized, both in terms of colors and background image
- Users can update their status
- Users can reply to each other's status updates
- Users can repost another user's status update, using the ReTweet function
- Powerful searching based on users replying to each other (@replies) and tagging of tweets with #hashtags

The ease of use and small set of core features have made Twitter very popular.

Existing social networking software

Just like there a number of fantastic social networking sites, there are a number of software systems available as well. These can be used to develop unique social networking sites.

Drupal

Drupal (`http://drupal.org/`) is a popular, freely available, open source content management system. On its own, Drupal can be used to create easy-to-use, easy-to-update websites. By extending this through the thousands of modules that the communities have developed or by creating new modules, we could create almost any type of website we want, ranging from e-commerce to social networking websites.

Drupal does make an excellent candidate for social networking websites, and Packt Publishing has a book published on this subject: *Drupal 6 Social Networking* (`http://www.packtpub.com/build-social-networking-website-with-drupal-6/book`).

Elgg

Elgg (`http://elgg.org/`) is an open source social networking platform, complete with functionality for setting up profiles, sharing files, adding friends, blogging, aggregating RSS, content tagging, and social graphs. Elgg also has an API, allowing developers to extend Elgg by adding additional functionality as well as a RESTful API to allow other applications to interact with the platform.

Joomla!

Joomla! (`http://www.joomla.org/`) is another open source content management system, with a range of built-in social networking features. There is also a commercial add-on, the Jomsocial component (`http://www.jomsocial.com/overview.html`), which turns Joomla! into a truly social network.

Hybrid approaches

There are, of course, options available which combine using an off-the-shelf system and a custom system. However, these mainly facilitate extending the functionality of the existing social networking platform or by integrating some of those social aspects with our own website. Such approaches include:

- Facebook applications: creating applications that are accessed via Facebook's main site, providing additional features to users. For example, a map of dinosaur-friendly restaurants, which are hosted externally by the developer.
- Facebook connect: Allows websites to interact with Facebook, using it as an authentication protocol, pulling friend data from it, as well as pushing, and pulling status updates to and from Facebook.
- Out-of-the-box hosted solutions, such as Ning (`http://www.ning.com/`), that allow users to create and maintain a social network community direct from their web browser.
- Google OpenSocial: A set of common APIs that make applications for social networks interoperable with supporting social networking sites. It also enables site developers to integrate the API so that other developers can build applications for that site, as well.

Rolling your own

Throughout the course of this book, we are going to create our own social networking site from scratch (sometimes referred to as **rolling your own**) using PHP, as opposed to using an existing system, product, or platform (such as Drupal and its social networking modules, Elgg, or leveraging existing social networks such as Facebook).

Why roll your own?

There are a number of very popular and successful social networking websites and social networking products out there, so why would we want to create our own? Some of the benefits for us using our own social networking system are as follows:

- Easier to update and maintain: As we built it, we will know exactly how it works and so we can easily extend and maintain it.
- Licensing: Other products and options have different licenses, which dictate how the software can be used, extended, and shared with our own system. We can decide that for ourselves.
- Enhance knowledge: We can build our own system in order to learn from the process.
- Efficient code: Some existing software packages make use of third-party add-ons, which are not always well optimized for lots of users. By writing our own code, we can ensure we develop in a scalable, efficient way.
- Provide a service.
- Improve business.
- Improve communication.

Easier to update and maintain

Developers who create their own platforms are generally much more familiar with them than with other platforms. As they build them, they know exactly how the platforms work, how to improve, extend, and enhance them. With existing platforms, there is an additional learning curve to developing with them and complications, should the platforms update. With sites such as Facebook, API changes are frequently rolled out, though with existing products, such as Drupal, installing updates is optional.

Licensing

Depending on the platform or product used, there may be different licenses associated with them. Licenses restrict what can and can't be done with the product, how improvements, extensions or modifications can be released, enforcing specific copyright notices or design guidelines, and of course, with many commercial licenses, costing money.

With self-built platforms, the license is up to us. If we want to release our social networking site code to the public, we can, and we can use the license terms we choose.

Enhance knowledge

By creating a social networking website from scratch, you can enhance your knowledge of PHP, social networking, and work with various other third-party APIs along the way to create a fantastic platform.

Provide a service

There are many ways in which websites and social networks provide additional services that are relevant to the social network or the target audience, though these are often through third-party applications. For example, there are features for both Facebook and LinkedIn that can provide a list of books which a user has read. These provide links to book retailers so more information can be discovered, and the books can be purchased. Additionally, some social networks contain knowledge bases of information, which can be improved by the user.

With existing social networks, any additional service provided either directly through the social network or through third-party applications and plugins would, or could, be restricted in a number of ways. The terms and conditions of the social network would be the main restriction, followed by how the features themselves can be added.

For example, if we wanted to add a map of dinosaur-friendly restaurants to an existing social network, it would rely upon:

- Data collection: Use provisions with the social networks terms of service
- Promotion within that social network, which can be a challenge
- Provisions for third-party applications, which would most likely limit and restrict the functionality and design
- Design and user interface guidelines enforced by the social network

Improve business

By tapping into the existing user base of established social networks, we can communicate with a new group of users, increasing awareness, and hopefully, improving business. One slight flaw with existing social networks is providing extra enhancements.

Taking Facebook as an example, third-party developers create additional features and embed them as applications, and some of these applications add business functionality. One example allowed users to book a table at a restaurant. The limitation with using Facebook is that before the information is sent to the application, the user is subjected to several dialogues asking for their confirmation. These dialogues are important to prevent abuse and to ensure user data is used properly. However, it is an obstacle for developers. As more and more applications are available, there is more competition for users' attention, which recently has lead to applications requesting that users invite their friends to use it. These mass invitations have the opposite effect, and discourage users from the applications in question.

With our own social network, the data and functionality would be hosted by ourselves. This gives us the freedom to extend the functionality of the social network to help us improve business as we see fit, leading to a more relevant and user friendly social network!

Improve communication

Social networks remove most barriers to communication, such as geographical location (the only barrier which remains, is Internet access). This is the case for both existing and custom social networks. The primary advantage over using our own system is we are less restricted in how we can communicate with users. With existing social networks, you must be connected to the user and restrictions may be imposed over which communication methods you use within the social network or which external communication details are shown to you.

Why use PHP?

PHP is a popular, open source programming language. Also, unlike some other languages, it isn't a framework in its own right, which means we can structure our application however we wish.

Most modern web hosts support PHP and the database platform we will be using with it (MySQL) and although some other languages are gaining popularity (such as the Ruby on Rails framework), hosting for this isn't as common. Facebook, the world's largest social networking website, is written using PHP (albeit with countless customizations, improvements, and extras), as does Yahoo!, which operates a search engine, news portal websites, and various social websites too. Yahoo! also, until recently, employed Rasmus Lerdorf, the creator of the original PHP engine.

This book assumes we have a reasonable understanding of PHP and some knowledge of object-oriented programming, so another good reason for using PHP is skill level.

When to use something else

As we have discussed earlier, there are already a number of fully featured social networking platforms and products available, written in a variety of different programming languages. Sometimes, it is more appropriate to use one of these, such as:

- When the project has a tight deadline and a base framework isn't already in place. In the interest of time, it would be more appropriate to leverage something else.
- When there are lots of developers on the project with varying skill levels, a project or platform with plenty of existing documentation available would allow the whole team to be able to get started right away.
- If the project is for a client and they have a preferred platform.
- If an existing product has the required features and works in the way required for the project.

Our site: DinoSpace

Throughout the course of this book we are going to develop a social networking site for keepers of pet dinosaurs (of course nobody owns a real pet dinosaur, it would be too expensive, but for the sake of this book, let's pretend!), which we will call DinoSpace. The social network will enable:

- Keepers of pet dinosaurs to connect with one another
- Friendships and other custom relationships (for example, walking buddy) to be maintained with other members of the site
- Users to share stories about their pets
- Profiles of pet dinosaurs to be created:
    - Statuses to be updated
- Dinosaur-friendly places to visit to be promoted:
    - Non-keepers of dinosaurs to use the site to promote businesses and events that dinosaur keepers may find useful or interesting
- Help and support to be provided to fellow Dinosaur keepers in an interactive way

At the end of this book, we will have a flexible social network for owners of pet dinosaurs. Some screens of the final product are shown. First, we have a basic profile page:

Complete with a customizable user status stream:

As well as a range of other features, which we will discuss now.

Feature list

From looking at the features available in existing social networking platforms and products, the following features seem standard throughout most of them, and so we shall try and incorporate them into our social networking website:

- Status updates: So that users can update their network with their current status
- Commenting on status updates: So that friends and connections can comment on these status changes

- Status stream: So that changes to many contacts statuses can be viewed at a glance
- Friends and relationships: So that users can connect with one another and define the context of the connection, for example, friend, colleague, or even Dino-walking partner
- Customizable profiles: So that users can build a profile of themselves with custom information about them
- Groups: So that smaller subsets can be created and nurtured within the site, focusing on specific interests or discussions
- Messages: So that users can keep in touch with one another
- Discussions: Encouraging open discussion amongst users
- Image sharing
- Video integration and sharing
- Calendars, events and birthdays: So that users can see upcoming events, create events, and invite friends, perhaps to promote a local T-Rex immunization day at a health center

Limitations

Users of large social networks such as Facebook typically have a large network of friends (or contacts) and subsequently a large number of updates, particularly when combined with the third-party applications, which can also post status updates on their behalf. To ensure that feeds of updates don't become too cluttered, these updates go through a special service that they have developed, which allows certain applications to be filtered out and tries to ensure the user's stream is more relevant.

This is something we won't be able to implement ourselves. However, Facebook has released a number of their components as open source projects, which could be integrated into our framework, should we wish to make use of some of their solutions to large scale social networking problems.

More information can be found on the Facebook open source page:
`http://developers.facebook.com/opensource.php`.

Summary

In this chapter, we have looked into what social networking is and why we might wish to use it. Also, we discussed why we created our own site from scratch, as opposed to using an existing system. We have also discussed various existing systems and looked at their features to build a list of features, which we want to use in our site, DinoSpace!

In Chapter 2 we will plan and develop our basic development framework, which we will slowly expand over the course of the book to create a powerful social networking website.

2
Planning and Developing the Core Framework

Now that we know exactly what we are going to do in this book, and why we are going to do it, we can start building our social networking site. To ensure a speedy development process, we are going to invest some time in this chapter to carefully plan and develop a micro-framework, which will take the hassle out of many common development tasks. This will be a small, light-weight framework, as our focus is on social networking, and the purpose of the framework is purely to help us do this.

In this chapter, you will learn:

- About some common design and architectural patterns that solve common programming problems, including:
    - MVC: The Model-View-Controller architecture
    - The Registry pattern
    - The Factory pattern
    - The Front Controller pattern
- How to effectively structure files within a development framework
- How to build the framework, including:
    - How to handle user authentication
    - How to abstract database access functions
    - Template management
- Providing a single point of access to the site

Designing the framework

Before we jump in and start programming, it is important that we take some time to plan and properly design the framework.

Patterns—making life easier

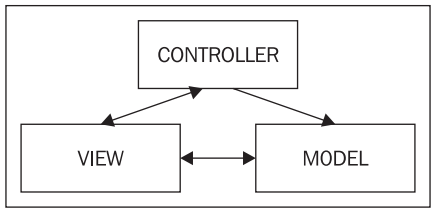
Design and architectural patterns are solutions to common programming problems, and their appropriate use can help ensure that a system is well-designed, easy to build upon, and easy for others to work with.

MVC: Model-View-Controller

The Model-View-Controller pattern is an architectural design pattern designed to separate the user interface from the business logic of an application. The user interface (view) uses the controller to interact with the logic and data of the application (model).

Let's think about our Dino Space social networking site, to see how this will work. If a user adds another user as a friend—they see the **Add as a friend** view. When they click the appropriate button to add the user as a friend, the controller processes this request from the user, and passes the request to the model, which updates the user's friends list, and where appropriate, sends any notifications. The view then updates, via instructions from the controller, to inform the user of the outcome of their request.

The following figure shows the components of the MVC architectural design pattern:



Our use of MVC won't be a religions implementation of the pattern. However, it will be an MVC style; there are numerous debates and discussions within the industry about exactly what MVC is within websites and web-frameworks, and if it is even truly applicable to web-based applications.

Model

Within our framework, the models will be PHP classes that store, manage, and process data and business logic. Access to the underlying database of data will be handled by a database access layer, which the model will make use of. The models will link closely with the database behind our social networking site, representing the data in a more suitable way, which is easier to access and manipulate than accessing the database directly.

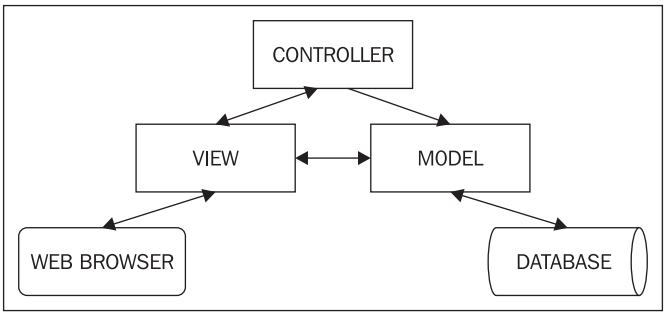
View

In our framework, the view will be made up of a combination of template files (which will contain HTML and placeholders for dynamic data), images, CSS files, and JavaScript. The templates will be merged and outputted to the user's browser on the fly by the controller.

Controller

The controllers will be a series of PHP classes, which process the user's request, and interact with the model, as well as generate views. Technically, some of our JavaScript (particularly where AJAX is used) also makes up a part of the controller, as it interacts between the view, and the model; these instances are extensions of the controller.

Because we are using the MVC pattern in a web-based environment, the architecture shown earlier can be illustrated in more detail with its interaction with the web-browser and the database. The following figure shows how the web browser and the database fit into the MVC architecture (extended MVC architecture interacting with the browser and the database):



The Front Controller pattern

The Front Controller pattern is a single file, through which all requests are routed (in our case, using Apache's `mod_rewrite`). In our case, this will almost definitely be the `index.php` file. This file will process the user's request, and pass the request to the appropriate controller.

By using a single front controller, our core includes files, core settings, and other common requirements that can all be performed, so that we know regardless of the user request these will have taken place.

If we used specific files for users to request, for example `friends.php` for friend actions, we would either have to "copy and paste" these standard features, functions, and settings, or ensure that we included a specific file that does this for us, which can be an issue if we need to re-factor the code and remove or rename this file—as we would need to ensure that we updated all the references to it.

Registry

Within most web application frameworks, there are numerous core objects, or objects containing core functionality that every aspect of the application will need to have access to. The registry pattern provides us with a means to store all of these core objects within one central object, and access them from within.

Dependency injection

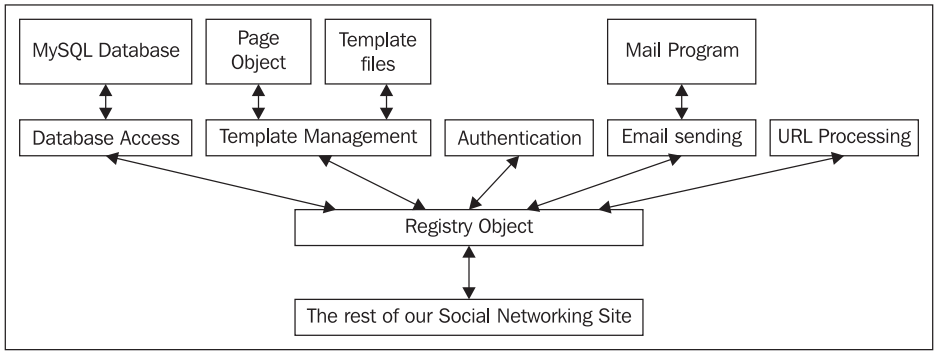
The registry pattern also makes dependency injection easier, as instead of making the object, or the objects it contains globally available—for example, through being a Singleton (which is often seen as a bad practice)—we would need to pass these objects to each of our models and controllers when we instantiate them. By storing all of the core objects within a single registry object, we only need to pass the registry object to these other objects, as opposed to having to pass six or seven objects, along with arrays of system-wide settings.

Within our social networking website, there are going to be a number of tasks that we frequently need to do, such as:

- Check to see if a user is logged in
- Get the logged in user's data
- Query the database, and perform other database-related functions
- Send e-mail notifications, for example, when a user adds another user as a friend

- Manage templates, by sending data to the views to be outputted to the user's browser
- Process the URL the user is accessing the site through, to determine which action should be performed, which controller should be used, and which method should be called

These functions will be abstracted into their own object that will be stored centrally within our registry. The rest of our social networks code can access all of these objects and features directly from our registry. The architecture of the registry is illustrated in the following screenshot:



Factory within our registry

Another design pattern that we will make use of is the Factory pattern. To save the need of creating all of the objects that our registry is going to manage, and passing them to the registry, we will simply tell the registry the name of the object we wish to create. The registry will then include the necessary class for us, and create (instantiate) the object for us. The registry then stores the newly created object in its array of objects. It is called a factory because the factory object (in our case, the registry) creates other objects.

A note on the Singleton pattern

Another pattern worth discussing is the Singleton pattern. This pattern generally involves creating a static object, for which only one instance is ever created within the application. Generally, the static nature of the Singleton means that it can be called from anywhere within our social networks code.

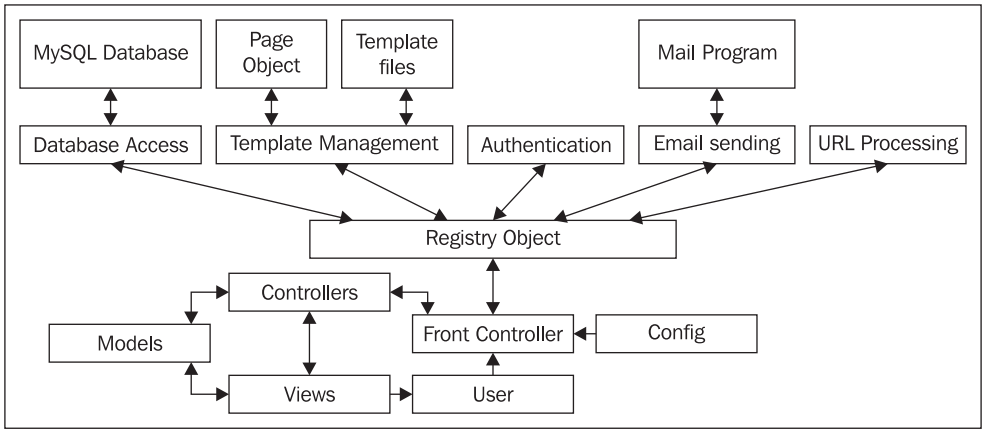
Using a Singleton for this purpose is bad practice, as it would mean our code and other objects would need to know details of the Singleton object itself. As we discussed earlier, our registry object should be passed directly to the objects in our social networks code, through their constructors, eliminating the need for the object to be globally available.

Although the registry would be useful as a Singleton, as we would only want one instance of the object to exist, we don't need to worry about this because with PHP 5 by default objects are passed by reference. This means if we pass an object to another object, instead of getting a new copy of the object (as with PHP 4), a reference to the single instance of the object is created, updating the central object, unless we were to clone the object or create a new instance of the registry class.

This is akin to pointers in the C programming language, where a pointer simply points to the space in memory used by an object or variable. When the object or variable needs to be updated, it is accessed via the pointer, saving concern for updating copies or clones of the variable or object by mistake.

Registry + MVC

By combining the MVC architecture with the registry and front controller pattern, we now have a framework where all the requests come through a central file, which creates the registry, and creates the necessary controllers. The controllers create various models where appropriate, and in some cases, pass control to other controllers, before generating and manipulating the templates to generate the views as appropriate. The following diagram shows all of these components working together:



Folder structure

Another important part of the system planning process is the directory structure that we are going to use. This will help us ensure that our files are properly organized, so that when we want to find or edit a particular file, we know exactly where to look.

Our proposed use of the MVC and Registry patterns give us a way to separate certain files, by classifying them as models, views, controllers or related to the registry; so, let's start with those folders:

- Controllers
- Models
- Registry
- Views

Within the views folder, we will have some template files, some images, some CSS, and some JavaScript. We may also allow users to switch between designs, so we would want to keep all of these, for one particular design, within a particular sub-folder. We may also utilize JavaScript libraries, as well as specific JavaScript within a particular view, so we would want to keep these separate too. If we bring this together, we get:

- Controllers
- Models
- Registry
- Views:
    - MainView
    - CSS
    - Images
    - JavaScript
    - Templates

We are also likely to have two types of uploaded files; files that we, as the administrator, upload to the site once it is live (resources), and files that users upload (uploads)—different aspects of the social network may utilize user uploads, so we should categorize this further:

- Controllers
- Models
- Registry
- Resources:
    - Images
        - Small
        - Large